GRID RESOURCE MANAGEMENT

GRID RESOURCE MANAGEMENT
State of the Art and Future Trends

Edited by
JAREK NABRZYSKI
Poznań Supercomputing and Networking Center

JENNIFER M. SCHOPF
Argonne National Laboratory

JAN WĘGLARZ
Institute of Computing Science, Poznań University of Technology

Kluwer Academic Publishers
Boston/Dordrecht/London

Contents

Preface ix
Contributing Authors xiii

Part I Introduction to Grids and Resource Management

1
The Grid in a Nutshell 3
Ian Foster and Carl Kesselman

2
Ten Actions When Grid Scheduling 15
Jennifer M. Schopf

3
Application Requirements for Resource Brokering in a Grid Environment 25
Michael Russell, Gabrielle Allen, Tom Goodale, Jarek Nabrzyski, and Ed Seidel

4
Attributes for Communication Between Grid Scheduling Instances 41
Uwe Schwiegelshohn and Ramin Yahyapour

5
Security Issues of Grid Resource Management 53
Mary R. Thompson and Keith R. Jackson

Part II Resource Management in Support of Collaborations

6
Scheduling in the Grid Application Development Software Project 73
Holly Dail, Otto Sievert, Fran Berman, Henri Casanova, Asim YarKhan, Sathish Vadhiyar, Jack Dongarra, Chuang Liu, Lingyun Yang, Dave Angulo, and Ian Foster

7
Workflow Management in GriPhyN 99
Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman

Part III State of the Art Grid Resource Management

8
Grid Service Level Agreements 119
Karl Czajkowski, Ian Foster, Carl Kesselman, and Steven Tuecke

9
Condor and Preemptive Resume Scheduling 135
Alain Roy and Miron Livny

10
Grid Resource Management in Legion 145
Anand Natrajan, Marty A. Humphrey, and Andrew S. Grimshaw

11
Grid Scheduling with Maui/Silver 161
David B. Jackson

12
Scheduling Attributes and Platform LSF 171
Ian Lumb and Chris Smith

13
PBS Pro: Grid Computing and Scheduling Attributes 183
Bill Nitzberg, Jennifer M. Schopf, and James Patton Jones

Part IV Prediction and Matching for Grid Resource Management

14
Performance Information Services for Computational Grids 193
Rich Wolski, Lawrence J. Miller, Graziano Obertelli, and Martin Swany

15
Using Predicted Variance for Conservative Scheduling on Shared Resources 215
Jennifer M. Schopf and Lingyun Yang

16
Improving Resource Selection and Scheduling Using Predictions 237
Warren Smith

17
The ClassAds Language 255
Rajesh Raman, Marvin Solomon, Miron Livny, and Alain Roy

18
Multicriteria Aspects of Grid Resource Management 271
Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Węglarz

19
A Metaheuristic Approach to Scheduling Workflow Jobs on a Grid 295
Marek Mika, Grzegorz Waligóra, and Jan Węglarz

Part V Data-Centric Approaches for Grid Resource Management

20
Storage Resource Managers 321
Arie Shoshani, Alexander Sim, and Junmin Gu

21
NeST: A Grid Enabled Storage Appliance 341
John Bent, Venkateshwaran Venkataramani,, Nick LeRoy, Alain Roy, Joseph Stanley, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny

22
Computation Scheduling and Data Replication Algorithms for Data Grids 359
Kavitha Ranganathan and Ian Foster

Part VI Quality of Service: QoS

23
GARA: A Uniform Quality of Service Architecture 377
Alain Roy and Volker Sander

24
QoS-Aware Service Composition for Large-Scale Peer-to-Peer Systems 395
Xiaohui Gu and Klara Nahrstedt

Part VII Resource Management in Peer-to-Peer Environments

25
A Peer-to-Peer Approach to Resource Location in Grid Environments 413
Adriana Iamnitchi and Ian Foster

26
Resource Management in the Entropia System 431
Andrew A. Chien, Shawn Marlin, and Stephen T. Elbert

27
Resource Management for the Triana Peer-to-Peer Services 451
Ian Taylor, Matthew Shields, and Ian Wang

Part VIII Economic Approaches and Grid Resource Management

28
Grid Resource Commercialization 465
Chris Kenyon and Giorgos Cheliotis

29
Trading Grid Services within the UK e-Science Grid 479
Steven Newhouse, Jon MacLaren, and Katarzyna Keahey

30
Applying Economic Scheduling Methods to Grid Environments 491
Carsten Ernemann and Ramin Yahyapour

References 507

Index 567

Preface

Grid computing can be defined as coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations. More simply, Grid computing typically involves using many resources (compute, data, I/O, instruments, etc.) to solve a single, large problem that could not be performed on any one resource. Often Grid computing requires the use of specialized middleware to mitigate the complexity of integrating of distributed resources within an Enterprise or as a public collaboration.

Generally, *Grid resource management* is defined as the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Grid applications compete for resources that are very different in nature, including processors, data, scientific instruments, networks, and other services. Complicating this situation is the general lack of data available about the current system and the competing needs of users, resource owners, and administrators of the system.

Grids are becoming almost commonplace today, with many projects using them for production runs. The initial challenges of Grid computing—how to run a job, how to transfer large files, how to manage multiple user accounts on different systems—have been resolved to first order, so users and researchers can now address the issues that will allow more efficient use of the resources.

While Grids have become almost commonplace, the use of good Grid resource management tools is far from ubiquitous because of the many open issues of the field.

- **Multiple layers of schedulers.** Grid resource management involves many players and possibly several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources. Other layers may be in between these, for example one to

handle a set of resources specific to a project. At every level additional people and software must be considered.

- **Lack of control over resources.** Grid schedulers aren't local resource management systems; a Grid-level scheduler may not (usually does not) have ownership or control over the resources. Most of the time, jobs will be submitted from a higher-level Grid scheduler to a local set of resources with no more permissions than the user would have. This lack of control is one of the challenges that must be addressed.

- **Shared resources and variance.** Related to the lack of control is the lack of dedicated access to the resources. Most resources in a Grid environment are shared among many users and projects. Such sharing results in a high degree of variance and unpredictability in the capacity of the resources available for use. The heterogeneous nature of the resources involved also plays a role in varied capacity.

- **Conflicting performance goals.** Grid resources are used to improve the performance of an application. Often, however, resource owners and users have different performance goals: from optimizing the performance of a single application for a specified cost goal to getting the best system throughput or minimizing response time. In addition, most resources have local policies that must be taken into account. Indeed, the policy issue has gained increasing attention: How much of the scheduling process should be done by the system and how much by the user? What are the rules for each?

These issues have been addressed only in part by the relevant literature in the field of Grid computing. The first book on Grid computing, *The Grid: Blueprint for a New Computing Infrastructure* by Foster and Kesselman, and its updated second edition, available in 2004, are a good starting point for any researcher new to the field. In addition, the recent book by Berman, Fox, and Hey entitled *Grid Computing: Making the Global Infrastructure a Reality*, presents a collection of leading papers in the area, including the "Anatomy of the Grid" and the "Physiology of the Grid", two papers that provide an overview of the shape, structure, and underlying functionality of Grid computing. Research results on the topic of resource management in Grid environments are presented regularly in selected sessions of several conferences, including SuperComputing (SC), the IEEE Symposium on High-Performance Distributed Computing (HPDC), and the Job Scheduling Strategies for Parallel Processing workshop, as well as in the Global Grid Forum, a standards body for the field. We present the first book devoted entirely to the open research questions of Grid resource management.

Goals of This Book

Our goal in *Grid Resource Management* is to present an overview of the state of the field and to describe both the real experiences and the current research available today. Other work has discussed the larger Grid community; this work focuses on the important area of resource management.

A secondary goal of this book is to elucidate the overlap with related areas of work. Toward this end, we include discussions of work with peer-to-peer computing, economic approaches, and operations research.

Grid computing is a rapidly developing and changing field, and the chapters presented here point toward the future work need to make practical resource management commonplace. A third goal is to help guide this research.

Plan of the Book

In this book we cover the current problems facing users, application developers, and resource owners when working with Grid resource management systems. The book is divided into eight topical parts and consists of 30 chapters overall.

Part 1 gives an overview of Grid computing and the general area of resource management in Grid environments. The chapters survey the field of Grid computing, Grid resource management from a user's point of view, ways of describing Grid schedulers, and security issues.

Part 2 examines resource management approaches in two collaborative projects, GrADS and GriPhyN, and how these projects provide Grid capabilities to researchers.

Part 3 gives an overview of existing resource management systems that are more production oriented than research oriented. This includes a review of the current resource management approaches in the Globus Toolkit, Legion, Condor, PBS, LSF, and Maui/Silver.

Part 4 discusses current research for matching and prediction techniques. It shows how prediction can be used in Grid resource management and describes techniques that are used to map the resources to jobs.

Part 5 presents many aspects of scheduling data in a Grid environment. The issue is important to data-driven Grid applications, where large files (up to petabyte scale) are needed by jobs in remote locations, where the job is to be executed either by creating a storage appliance, such as in the NeST approach, or by using more efficient replication of data.

Part 6 describes the use of quality of service (QoS) for Grid systems. In a Grid environment, QoS applies not just to networks but to all resources.

Part 7 presents some of the issues related to resource management on Peer-to-Peer (P2P) Grids, that is, those that combine the complexity of Grids with

the scale and dynamism of peer-to-peer communities. Discussions of Entropia and Triana follow a resource location approach for small-world environments.

Part 8 focuses on several economic approaches to Grid resource management. Many researchers have begun to make comparisons between the open commodities market and ways of sharing resources in a distributed community, and these are discussed in the context of several projects.

Audience

We intend this book for a broad audience, including practitioners and researchers interested in scheduling and also graduate students and advanced undergraduate students in computer science interested in Grid computing, operations research, management sciences, distributed computing, and networks.

We believe that this book will be especially useful as a college text for a senior undergraduate- or graduate-level course, one lecture spent on each chapter. In a semester-length course, some topics can be covered in greater depth, and many chapters include recommendations for further reading. This book will also be useful for a full-year course if combined in the first semester with one of the general overview texts listed earlier for the first semester.

Thanks

With great pleasure we acknowledge the efforts of the many people who have contributed to the development of this book. First and foremost we thank the contributing authors, who took time out of their busy schedules to contribute their work and comments to this volume.

We are also very grateful for the expert help we received during the preparation of the manuscript. Special thanks to Bogdan Ludwiczak, PSNC, for assistance with editing and LaTex, and Gail Pieper, ANL, for amazing technical editing. Both went out of their way to help us meet seemingly impossible deadlines.

JENNIFER M. SCHOPF, JAREK NABRZYSKI, AND JAN WĘGLARZ

Contributing Authors

Gabrielle Allen
Max Planck Institute for Gravitational Physics
allen@aei-potsdam.mpg.de

Dave Angulo
Department of Computer Science, The University of Chicago
dangulo@cs.uchicago.edu

Andrea C. Arpaci-Dusseau
Department of Computer Science, University of Wisconsin-Madison
dusseau@cs.wisc.edu

Remzi H. Arpaci-Dusseau
Department of Computer Science, University of Wisconsin-Madison
remzi@cs.wisc.edu

John Bent
Department of Computer Science, University of Wisconsin-Madison
johnbent@cs.wisc.edu

Fran Berman
San Diego Supercomputer Center, University of California, San Diego
Department of Computer Science and Engineering, University of California, San Diego
berman@cs.ucsd.edu

James Blythe
Information Sciences Institute, University of Southern California
blythe@isi.edu

Henri Casanova
San Diego Supercomputer Center, University of California, San Diego
Department of Computer Science and Engineering, University of California, San Diego
casanova@sdsc.edu

Giorgos Cheliotis
IBM Zürich Research Lab
gic@zurich.ibm.com

Andrew A. Chien
Department of Computer Science and Engineering, University of California, San Diego
achien@cs.ucsd.edu

Karl Czajkowski
Information Science Institute, University of Southern California
karlcz@isi.edu

Holly Dail
San Diego Supercomputer Center, University of California, San Diego
hdail@cs.ucsd.edu

Ewa Deelman
Information Sciences Institute, University of Southern California
deelman@isi.edu

Jack Dongarra
Department of Computer Science, University of Tennessee
dongarra@cs.utk.edu

Stephen T. Elbert
International Business Machines
selbert@us.ibm.com

Carsten Ernemann
Computer Engineering Institute, University Dortmund
carsten.ernemann@uni-dortmund.de

Ian Foster
Department of Computer Science, The University of Chicago
Mathematics and Computer Science Division, Argonne National Laboratory
foster@mcs.anl.gov

Yolanda Gil
Information Sciences Institute, University of Southern California
gil@isi.edu

Tom Goodale
Max Planck Institute for Gravitational Physics
goodale@aei-potsdam.mpg.de

Andrew S. Grimshaw
Department of Computer Science, University of Virginia
agrimshaw@avaki.com

Junmin Gu
Lawrence Berkeley National Laboratory
jgu@lbl.gov

Xiaohui Gu
Department of Computer Science, University of Illinois at Urbana-Champaign
xgu@cs.uiuc.edu

Marty A. Humphrey
Department of Computer Science, University of Virginia
humphrey@cs.virginia.edu

Adriana Iamnitchi
Department of Computer Science, The University of Chicago
anda@cs.uchicago.edu

David B. Jackson
Cluster Resources, Inc.
jacksond@supercluster.org

Keith R. Jackson
Lawrence Berkeley National Laboratory
krjackson@lbl.gov

Katarzyna Keahey
Mathematics and Computer Science Division, Argonne National Laboratory
keahey@mcs.anl.gov

Chris Kenyon
IBM Zürich Research Lab
chk@zurich.ibm.com

Carl Kesselman
Information Science Institute, University of Southern California
carl@isi.edu

Krzysztof Kurowski
Poznań Supercomputing and Networking Center
kikas@man.poznan.pl

Nick LeRoy
Department of Computer Science, University of Wisconsin-Madison
nleroy@cs.wisc.edu

Chuang Liu
Department of Computer Science, The University of Chicago
chliu@cs.uchicago.edu

Miron Livny
Department of Computer Science, University of Wisconsin-Madison
miron@cs.wisc.edu

Ian Lumb
Platform Computing Inc.
ilumb@platform.com

Jon MacLaren
Manchester Computing, The University of Manchester
jon.maclaren@man.ac.uk

Shawn Marlin
Science Application International Corporation
shawn.r.marlin@saic.com

Marek Mika
Institute of Computing Science, Poznań University of Technology
mika@cs.put.poznan.pl

Lawrence J. Miller
Department of Computer Science, University of California, Santa Barbara
ljmiller@cs.ucsb.edu

Jarek Nabrzyski
Poznań Supercomputing and Networking Center
naber@man.poznan.pl

Klara Nahrstedt
Department of Computer Science, University of Illinois at Urbana-Champaign
klara@cs.uiuc.edu

Anand Natrajan
Department of Computer Science, University of Virginia
an4m@cs.virginia.edu

Steven Newhouse
London e-Science Centre, Imperial College London
sjn5@doc.ic.ac.uk

Bill Nitzberg
Altair Grid Technologies
bill@computer.org

Graziano Obertelli
Department of Computer Science, University of California, Santa Barbara
graziano@cs.ucsb.edu

Ariel Oleksiak
Poznań Supercomputing and Networking Center
ariel@man.poznan.pl

James Patton Jones
Altair Grid Technologies
jjones@pbspro.com

Rajesh Raman
Department of Computer Science, University of Wisconsin-Madison
dr_rajesh_raman@yahoo.com

Kavitha Ranganathan
Department of Computer Science, The University of Chicago
kavitha@uchicago.edu

Alain Roy
Department of Computer Science, University of Wisconsin-Madison
roy@cs.wisc.edu

Michael Russell
Max Planck Institute for Gravitational Physics
russell@aei-potsdam.mpg.de

Volker Sander
Central Institute for Applied Mathematics, Forschungszentrum Jülich GmbH
v.sander@fz-juelich.de

Jennifer M. Schopf
Mathematics and Computer Science Division, Argonne National Laboratory
jms@mcs.anl.gov

Uwe Schwiegelshohn
Computer Engineering Institute, University Dortmund
uwe.schwiegelshohn@udo.edu

Ed Seidel
Max Planck Institute for Gravitational Physics
eseidel@aei-potsdam.mpg.de

Matthew Shields
Department of Computer Science and Physics and Astronomy, Cardiff University
matthew.shields@astro.cf.ac.uk

Arie Shoshani
Lawrence Berkeley National Laboratory
shoshani@lbl.gov

Otto Sievert
Department of Computer Science and Engineering, University of California, San Diego
otto@cs.ucsd.edu

Alexander Sim
Lawrence Berkeley National Laboratory
asim@lbl.gov

Chris Smith
Platform Computing Inc.
csmith@platform.com

Warren Smith
Computer Sciences Corporation
NASA Ames Research Center
wwsmith@nas.nasa.gov

Marvin Solomon
Department of Computer Science, University of Wisconsin-Madison
solomon@cs.wisc.edu

Joseph Stanley
Department of Computer Science, University of Wisconsin-Madison
jass@cs.wisc.edu

Martin Swany
Department of Computer Science, University of California, Santa Barbara
swany@cs.ucsb.edu

Ian Taylor
Department of Computer Science, Cardiff University
i.j.taylor@cs.cardiff.ac.uk

Mary R. Thompson
Lawrence Berkeley National Laboratory
mrthompson@lbl.gov

Steven Tuecke
Mathematics and Computer Science Division, Argonne National Laboratory
tuecke@mcs.anl.gov

Sathish Vadhiyar
Department of Computer Science, University of Tennessee
vss@cs.utk.edu

Venkateshwaran Venkataramani
Oracle
veeve@cs.wisc.edu

Jan Węglarz
Institute of Computing Science, Poznań University of Technology
Poznań Supercomputing and Networking Center
jan.weglarz@cs.put.poznan.pl

Grzegorz Waligóra
Institute of Computing Science, Poznań University of Technology
grzegorz.waligora@cs.put.poznan.pl

Ian Wang
Department of Computer Science and Physics and Astronomy, Cardiff University
i.n.wang@cs.cardiff.ac.uk

Rich Wolski
Department of Computer Science, University of California, Santa Barbara
rich@cs.ucsb.edu

Ramin Yahyapour
Computer Engineering Institute, University Dortmund
ramin.yahyapour@uni-dortmund.de

Lingyun Yang
Department of Computer Science, The University of Chicago
lyang@cs.uchicago.edu

Asim YarKhan
Department of Computer Science, University of Tennessee
yarkhan@cs.utk.edu

I

INTRODUCTION TO GRIDS AND RESOURCE MANAGEMENT

Chapter 1

THE GRID IN A NUTSHELL

Ian Foster[1] and Carl Kesselman[2]
[1]*Mathematics and Computer Science Division, Argonne National Laboratory*
[2]*Information Sciences Institute, University of Southern California*

Abstract The emergence and widespread adoption of Grid computing has been fueled by continued growth in both our understanding of application requirements and the sophistication of the technologies used to meet these requirements. We provide an introduction to Grid applications and technologies and discuss the important role that resource management will play in future developments.

1. INTRODUCTION

The term "the Grid" was coined in the mid-1990s to denote a (then) proposed distributed computing infrastructure for advanced science and engineering. Much progress has since been made on the construction of such an infrastructure and on its extension and application to commercial computing problems. And while the term "Grid" has also been on occasion applied to everything from advanced networking and computing clusters to artificial intelligence, there has also emerged a good understanding of the problems that Grid technologies address, as well as a first set of applications for which they are suited.

Grid concepts and technologies were initially developed to enable resource sharing within scientific collaborations, first within early Gigabit testbeds [CS92, Cat92, Mes99] and then on increasingly larger scales [BCF+98, GWWL94, BJB+00, SWDC97, JGN99, EH99]. At the root of these collaborations was the need for participations to share not only datasets but also software, computational resources, and even specialized instruments such as telescopes and microscopes. Consequently, a wide range of application types emerged that included distributed computing for computationally demanding

data analysis (pooling of compute power and storage), the federation of diverse distributed datasets, collaborative visualization of large scientific datasets, and coupling of scientific instruments such as electron microscopes, high-energy x-ray sources, and experimental data acquisitions systems with remote users, computers, and archives (increasing functionality as well as availability) [BAJZ98, Joh99, HKL+00].

We have argued previously that underlying these different usage modalities there exists a common set of requirements for *coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations* [FKT01]. The term *virtual organization* is often applied to these collaborations because of their distributed and often ephemeral nature. This same requirement for resource sharing across cross organizational collaborations arises within commercial environments, including enterprise application integration, on-demand service provisioning, data center federation, and business-to-business partner collaboration over the Internet. Just as the World Wide Web began as a technology for scientific collaboration and was adopted for e-business, we see a similar trajectory for Grid technologies.

The success of the Grid to date owes much to the relatively early emergence of clean architectural principles, de facto standard software, aggressive early adopters with challenging application problems, and a vibrant international community of developers and users. This combination of factors led to a solid base of experience that has more recently driven the definition of the service-oriented Open Grid Services Architecture that today forms the basis for both open source and commercial Grid products. In the sections that follow, we expand on these various aspects of the Grid story and, in so doing, provide context for material to be presented in later chapters.

2. VIRTUAL ORGANIZATIONS

We have defined Grids as being concerned with enabling coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing that we are concerned with is not primarily file exchange, as supported by the Web or peer-to-peer systems, but rather direct access to computers, software, data, services, and other resources, as required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a *virtual organization* (VO) [FKT01].

VOs can vary greatly in terms of scope, size, duration, structure, distribution, and capabilities being shared, community being serviced, and sociology. Examples of VOs might include

- the application service providers, storage service providers, cycle providers, and consultants engaged by a car manufacturer to perform scenario evaluation during planning for a new factory;
- members of an industrial consortium bidding on a new aircraft;
- a crisis management team and the databases and simulation systems that they use to plan a response to an emergency situation; and
- members of a large international high-energy physics collaboration.

These examples only hint at the diversity of applications enabled by cross-organizational sharing. In spite of these differences, however, study of underlying technology requirements leads us to identify a broad set of common concerns and requirements. We see a need to establish and maintain highly flexible sharing relationships capable of expressing collaborative structures such as client-server and peer-to-peer, along with more complex relationships such as brokered or sharing via intermediaries. We also see requirements for complex and high levels of control over how shared resources are used, including fine-grained access control, delegation, and application of local and global policies. We need basic mechanisms for discovering, provisioning, and managing of varied resources, ranging from programs, files, and data to computers, sensors, and networks, so as to enable time critical or performance-critical collaborations, and for diverse usage modes, ranging from single user to multi-user and from performance sensitive to cost sensitive and hence embracing issues of quality of service, scheduling, co-allocation, and accounting.

3. GRID APPLICATIONS

Various Grid application scenarios have been explored within both science and industry. We present here a representative sample of thirteen such applications that collectively introduce the broad spectrum of usage scenarios that are driving Grid adoption and development. These applications include compute-intensive, data-intensive, sensor-intensive, knowledge-intensive, and collaboration-intensive scenarios and address problems ranging from multi-player video gaming, fault diagnosis in jet engines, and earthquake engineering to bioinformatics, biomedical imaging, and astrophysics. Further details on each application can be found in a set of case studies collected in a recent book [FK04].

Distributed Aircraft Engine Diagnostics. The U.K. Distributed Aircraft Maintenance Environment (DAME) project is applying Grid technologies to

the challenging and broadly important problem of computer-based fault diagnosis, an inherently distributed problem in many situations because of the range of data sources and stakeholders involved [AJF+04]. In particular, DAME is working to diagnose faults in Rolls Royce aircraft engines, based on sensor data recorded at the rate of one gigabyte per engine per transatlantic flight.

NEESgrid Earthquake Engineering Collaboratory. The U.S. Network for Earthquake Engineering Simulation (NEES) is an ambitious project to enable remote access to, and collaborative use of, the specialized equipment used to study the behavior of structures, such as bridge columns when subjected to the forces of an earthquake. NEESgrid uses Grid technology to provide remote access to experimental access (i.e., teleobservation and telecontrol); to couple physical experiments with numerical simulation; and to archive, discover, and analyze simulation, experimental, and observational data [KPF04, PKF+01].

World Wide Telescope. Advances digital astronomy enable the systematic survey of the heavens and the collection of vast amounts of data from telescopes over the world. New scientific discoveries can be made not only by analyzing data from an individual instruments but also by comparing and correlating data from different sky surveys [SG04, SG01]. The emerging "World Wide Telescope," or virtual observatory, uses Grid technology to federate data from hundreds of individual instruments, allowing a new generation of astronomers to perform analysis of unprecedented scope and scale. While the most immediate challenges relate to data formats and data management, the need to manage the computation resources consumed by such data analysis tasks is a looming issue.

Biomedical Informatics Research Network. (BIRN). The goal of this U.S. project is to federate biomedical imaging data for the purpose of research and, ultimately, improved clinical case [EP04]. To this end, BIRN is deploying compute-storage clusters at research and clinical sites around the United States and deploying Grid middleware to enable the integration of image data from multiple locations.

In silico Experiments in Bioinformatics. The U.K. myGrid project is applying Grid technologies to the semantically rich problems of dynamic resource discovery, workflow specification, and distributed query processing, as well as provenance management, change notification, and personalization [GPS04].

story, a group of U.S. physicists and computer scientists completed a challenging data generation and analysis task for a high energy physics experiment, harnessing computing and storage resources at six sites to generate 1.5 million simulated events during a two-month run [GCC+04].

Virtual Screening on Desktop Computers. In this drug discovery application, an intra-Grid composed of desktop PCs was used for virtual screening of drug candidates [Chi04, CFG02, CCEB03]. An existing molecular dock-

ing application was integrated into a commercial Grid environment to achieve a significant increase in processing power over what drug companies would typically have dedicated to compound screening.

Enterprise Resource Management. GlobeXplorer, a provider of online satellite imagery, is an example of an enterprise that uses advanced resource management techniques to improve the efficiency and flexibility of intra-Enterprise resource usage [Gen04].

Infrastructure for Multiplayer Games. Butterfly.net, a service provider for the multiplayer videogaming industry, is using Grid technologies to deliver scalable hosting services to game developers [LW04, LWW03]. A potentially huge number of game participants and a need for interactive response lead to challenging performance requirements

Service Virtualization. As we discuss in greater detail below, virtualization is playing an increasingly important role in enterprise IT infrastructures [XHLL04]. In a recent success story, the deployment of a resource and service virtualization solution at a global investment bank resulted in significant performance improvements.

Access Grid Collaboration System. High-end collaboration and conferencing environments represent an application domain for Grid technologies that is rapidly growing in importance [CDO+00, Ste04]. The delivery of rich group collaboration capabilities places heavy demands on Grid technologies.

Collaborative Astrophysics. An enthusiastic community of computational astrophysicists has been working for some years to apply Grid technologies to a range of problems in high-end collaborative science [ABH+99, ADF+01, AS04].

4. GRID TECHNOLOGY

The development of Grids has been spurred and enabled by the staged development of increasingly sophisticated and broadly used technologies. As illustrated in Figure 1.1, early experiments with "metacomputing" [CS92, Cat92, EH99, GWWL94, Mes99] worked primarily with custom tools or specialized middleware [GRBK98, FGN+96, FGT96, Sun90] that emphasized message-oriented communication between computing nodes.

The transition from metacomputing to Grid computing occurred in the mid-1990s with the introduction of middleware designed to function as wide-area infrastructure to support diverse online processing and data-intensive applications. Systems such as the Storage Resource Broker [BMRW98], Globus Toolkit® [FK97], Condor [FTF+02, LLM88], and Legion [GW97, GFKH99] were developed primarily for scientific applications and demonstrated at various levels of scale on a range of applications. Other developers attempted to leverage the middleware structure being developed for the World Wide Web by

using HTTP servers or Web browsers as Grid computing platforms [BKKW96, BSST96, GBE+98]. These systems did not gain significant use, however, partly because the middleware requirements for distributed information systems such as the Web are different from those for Grid applications.

Globus Toolkit. By 1998, the open source Globus Toolkit (GT2) [FK97] had emerged as a de facto standard software infrastructure for Grid computing. GT2 defined and implemented protocols, APIs, and services used in hundreds of Grid deployments worldwide. By providing solutions to common problems such as authentication, resource discovery, resource access, and data movement, GT2 accelerated the construction of real Grid applications. And by defining and implementing "standard" protocols and services, GT pioneered the creation of interoperable Grid systems and enabled significant progress on Grid programming tools. This standardization played a significant role in spurring the subsequent explosion of interest, tools, applications, and deployments, as did early success stories such as a record-setting 100,000-entity distributed interactive simulation scenario in 1998 [BDG+98, BCF+98] and the solution in June 2000 of nug30 [ABGL02], a previously open problem in optimization theory.

The GT2 protocol suite leveraged heavily existing Internet standards for security, resource discovery, and security. In addition, some elements of the GT2 protocol suite were codified in formal technical specifications and reviewed within standards bodies: notably, the GridFTP data transfer protocol (for which multiple implementations exist) [ABB+02b] and elements of the Grid Security Infrastructure [FKTT98, TEF+02]. In general, however, GT2 "standards" were not formal, well documented, or subject to public review. They were not in themselves a sufficient basis for the creation of a mature Grid ecosystem. Similar comments apply to other important Grid technologies that emerged during this period, such as the Condor high throughput computing system.

This period also saw significant development of more user-oriented tools, most building on the Globus Toolkit. For example, MPICH-G [KTF03], a Grid-aware version of the public-domain Message Passing Interface (MPI) [GLS94], provided a standards-based message-passing environment. Tools such as NetSolve and Ninf [TNS+02] sought to deliver Grid-enabled software to a nonexpert user community, while portal toolkits [AC02, Nov02, PLL+03, RAD+02, TMB+01] allowed Grid-enabled applications to be delivered to end users via a standard Web browser. Workflow systems such as DAGman [DAG] and Chimera [FVWZ02], and scripting languages such as pyGlobus [Jac02], focus on coordination of components written in traditional programming languages.

Open Grid Services Architecture. As interest in Grids continued to grow, and in particular as industrial interest emerged, the importance of true stan-

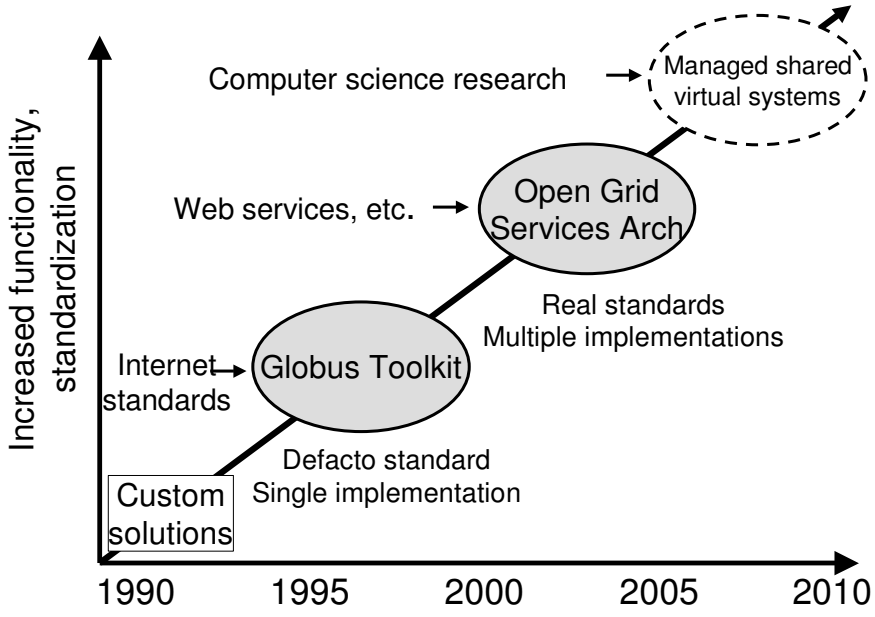


Figure 1.1. The evolution of Grid technology.

dards increased. The Global Grid Forum, established in 1998 as an international community and standards organization, appeared to be the natural place for such standards to be developed, and indeed multiple standardization activities are now under way. In particular, 2002 saw the emergence of the Open Grid Services Architecture (OGSA) [FKNT02], a true community standard with multiple implementations–including the OGSA-based Globus Toolkit 3.0 [WSF+03], released in 2003. Building on and significantly extending GT2 concepts and technologies, OGSA firmly aligns Grid computing with broad industry initiatives in service-oriented architecture and Web services.

In addition to defining a core set of standard interfaces and behaviors that address many of the technical challenges introduced above, OGSA provides a framework within which can be defined a wide range of interoperable, portable services. OGSA thus provides a foundation on which can be constructed a rich Grid technology ecosystem comprising multiple technology providers. Thus, we see, for example, major efforts under way within the U.K. eScience programme to develop data access and integration services [ACK+04, PAD+02].

Concurrent with these developments we see a growing recognition that large-scale development and deployment of Grid technologies is critical to future success in a wide range of disciplines in science, engineering, and the humanities [ADF+03], and increasingly large investments within industry in related

areas. While research and commercial uses can have different concerns, they also have much in common, and there are promising signs that the required technologies can be developed in a strong academic-industrial partnership. The current open source code base and emerging open standards provide a solid foundation for the new open infrastructure that will result from this work.

Managed, Shared Virtual Systems. The definition of the initial OGSA technical specifications is an important step forward, but much more remains to be done before the full Grid vision is realized. Building on OGSA's service-oriented infrastructure, we will see an expanding set of interoperable services and systems that address scaling to both larger numbers of entities and smaller device footprints, increasing degrees of virtualization, richer forms of sharing, and increased qualities of service via a variety of forms of active management. This work will draw increasingly heavily on the results of advanced computer science research in such areas as peer-to-peer [CMPT04, FI03], knowledge-based [BLHL01, GDRSF04], and autonomic [Hor01] systems.

5. SERVICE ORIENTATION, INTEGRATION, AND VIRTUALIZATION

Three related concepts are key to an understanding of the Grid and its contemporary technologies and applications: service orientation, integration, and virtualization.

A service is an entity that provides some capability to its clients by exchanging messages. A service is defined by identifying sequences of specific message exchanges that cause the service to perform some *operation*. By thus defining these operations only in terms of message exchange, we achieve great flexibility in how services are implemented and where they are located. A service-oriented architecture is one in which all entities are services, and thus any operation that is visible to the architecture is the result of message exchange.

By encapsulating service operations behind a common message-oriented service interface, service orientation isolates users from details of service implantation and location. For example, a storage service might present the user with an interface that defines, among other things, a store file operation. A user should be able to invoke that operation on a particular instance of that storage service without regard to how that instance implements the storage service interface. Behind the scenes, different implementations may store the file on the user's local computer, in a distributed file system, on a remote archival storage system, or in free space within a department desktop pool-or even to choose from among such alternatives depending on context, load, amount paid, or other factors. Regardless of implementation approach, the user is aware only that the requested operation is executed-albeit

with varying cost and other qualities of service, factors that may be subject to negotiation between the client and service. In other contexts, a *distribution framework* can be used to disseminate work across service instances, with the number of instances of different services deployed varying according to demand [AFF+01, GKTA02, XHLL04].

While a service implementation may directly perform a requested operation, services may be *virtual*, providing an interface to underlying, distributed services, which in turn may be virtualized as well. Service virtualization also introduces the challenge, and opportunity, of *service integration*. Once applications are encapsulated as services, application developers can treat different services as building blocks that can be assembled and reassembled to adapt to changing business needs. Different services can have different performance characteristics; and, in a virtualized environment, even different instances of the *same* service can have different characteristics. Thus new distributed system integration techniques are needed to achieve end-to-end guarantees for various qualities of service.

6. THE FUNDAMENTAL ROLE OF RESOURCE MANAGEMENT

Fundamental to both service virtualization and integration is the ability to discover, allocate, negotiate, monitor, and manage the use of network-accessible capabilities in order to achieve various end-to-end or global qualities of service. Within a service-oriented architecture, these capabilities may include both traditional *resources* (computational services offered by a computer, network bandwidth, or space on a storage system) and virtualized *services* (e.g., database, data transfer, simulation), which may differ in the function they provide to users but are consistent in the manner in which they deliver that function across the network. Nevertheless, for historical reasons, and without loss of generality, the term *resource management* is commonly used to describe all aspects of the process of locating various types of capability, arranging for their use, utilizing them, and monitoring their state.

In traditional computing systems, resource management is a well-studied problem. Resource managers such as batch schedulers, workflow engines, and operating systems exist for many computing environments. These resource management systems are designed and operate under the assumption that they have complete control of a resource and thus can implement the mechanisms and policies needed for effective use of that resource in isolation. Unfortunately, this assumption does not apply to the Grid. We must develop methods for managing Grid resources across separately administered domains, with the resource heterogeneity, lose of absolute control, and inevitable differences in policy that result. The underlying Grid resource set is typically heterogeneous

even within the same class and type of resource. For example, no two compute clusters have the same software configuration, local resource management software, administrative requirements, and so forth. For this reason, much of the early work in Grid resource management focused on overcoming these basic issues of heterogeneity, for example through the definition of standard resource management protocols [CFK+98b, CFK+02] and standard mechanisms for expressing resource and task requirements [RLS98].

More important than such issues of plumbing, however, is the fact that different organizations operate their resources under different policies; the goals of the resource user and the resource provider may be inconsistent, or even in conflict. The situation is further complicated by the fact that Grid applications often require the concurrent allocation of multiple resources, necessitating a structure in which resource use can be coordinated across administrative domains [CFK99, FFR+02]. Much current research in Grid resource management is focused on understanding and managing these diverse policies from the perspective of both the resource provider and the consumer [BWF+96, AC02, KNP01, CKKG99] with the goal of synthesizing end-to-end resource management in spite of the fact that the resources are independently owned and administered.

The emergence of service-oriented architecture, the increased interest in supporting a broad range of commercial applications, and the natural evolution of functionality are collectively driving significant advances in resource management capabilities. While today's Grid environment is primarily oriented toward best-effort service, we expect the situation to become substantially different in the next several years, with end-to-end resource provisioning and virtualized service behavior that is indistinguishable from nonvirtualized services becoming the rule rather than the exception.

We possess a good understanding of the basic mechanisms required for a provisioned Grid. Significant challenges remain, however, in understanding how these mechanisms can be effectively combined to create seamless virtualized views of underlying resources and services. Some of these challenges lie strictly within the domain of resource management, for example, robust distributed algorithms for negotiating simultaneous service level agreements across a set of resources. Other issues, such as expression of resource policy for purposes of discovery and enhanced security models that support flexible delegation of resource management to intermediate brokers are closely tied to advances in other aspects of Grid infrastructure. Hence, the key to progress in the coming years is to create an extensible and open infrastructure that can incorporate these advances as they become available.

7. SUMMARY

We have provided both a historical and a technological introduction to Grid computing. As we have discussed, the dynamic and cross-organizational nature of the Grid is at the root of both the opportunities and challenges that are inherent in Grid infrastructure and applications. These same issues also have a profound effect on resource management. While many of the resource management techniques developed over the past 40 years have applicability to the Grid, these techniques must be reassessed in the context of an environment in which both absolute knowledge of system state and absolute control over resource policy and use are not possible.

The development of reasonably mature Grid technologies has allowed many academic and industrial application groups to achieve significant successes. Nevertheless, much further development is required before we can achieve the ultimate goal of virtualized services that can be integrated in flexible ways to deliver strong application-level performance guarantees. Advances in resource management will be key to many of these developments.

Acknowledgments

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, Office of Science, under Contract W-31-109-Eng-38, and by the NSF Middleware Initiative.

Chapter 2

TEN ACTIONS WHEN GRID SCHEDULING

The User as a Grid Scheduler

Jennifer M. Schopf
Mathematics and Computer Science Division, Argonne National Laboratory

Abstract In this chapter we present a general architecture or plan for scheduling on a Grid. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated. These interactions are also closely tied to the functionality of the Grid Information Services. This Grid scheduling approach has three phases: resource discovery, system selection, and job execution. We detail the steps involved in each phase.

1. INTRODUCTION

More applications are turning to Grid computing to meet their computational and data storage needs. Single sites are simply no longer efficient for meeting the resource needs of high-end applications, and using distributed resources can give the application many benefits. Effective Grid computing is possible, however, only if the resources are scheduled well.

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. We define a *job* to be anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep). We use the term *resource* to mean anything that can be scheduled: a machine, disk space, a QoS network, and so forth. In general, for ease of use, in this chapter we refer to resources in terms associated with compute resources; however, nothing about the approach is limited in this way.

In general we can differentiate between a Grid scheduler and a *local resource scheduler*, that is, a scheduler that is responsible for scheduling and

managing resources at a single site, or perhaps only for a single cluster or resource. These are the lowest-level of lower-level scheduling instances discussed in Chapter 4 One of the primary differences between a Grid scheduler and a local resource scheduler is that the Grid scheduler does not "own" the resources at a site (unlike the local resource scheduler) and therefore does not have control over them. The Grid scheduler must make best-effort decisions and then submit the job to the resources selected, generally as the user. Furthermore, often the Grid scheduler does not have control over the full set of jobs submitted to it, or even know about the jobs being sent to the resources it is considering use of, so decisions that tradeoff one job's access for another's may not be able to be made in the global sense. This lack of ownership and control is the source of many of the problems to be solved in this area.

In this chapter we do not address the situation of speculative execution - submitting a job to multiple resources and, when one begins to run, canceling the other submissions. We do, however, discuss resource selection (sometimes termed resource discovery [Ber99]), assignment of application tasks to those resources (mapping [Ber99]), and data staging or distribution.

Historically, the most common Grid scheduler is the user, and that is the point of view presented in this chapter. Many efforts are under way, however, to change this situation [Nab99, Zho92, IBM01, GP01, BWF+96] and work detailed in Chapters 11, 13, and 9 among others. Many of these are discussed later in this book, but it can be argued that no single system addresses all the needed features yet. In Section 2 we briefly discuss the related Grid Information System interactions expected by a Grid-level scheduler. In Section 3 we describe the three phases a user goes through when scheduling a job over resources on multiple administrative domains–resource discovery, selection, and job execution. Most implemented systems follow a similar pattern of execution. For each step we define the work involved and distinguish it from the work of a common parallel scheduler.

2. GRID INFORMATION SERVICE

The decisions a scheduler makes are only as good as the information provided to it. Many theoretical schedulers assume one has 100 percent of the information needed, at an extremely fine level of detail, and that the information is always correct. In Grid scheduling, this is far from our experience. In general we have only the highest level of information. For example, it may be known that an application needs to run on Linux, will produce output files somewhere between 20 MB and 30 MB, and should take less than three hours but might take as long as five. Or, it may be known that a machine is running Linux and has a file system located at a certain address that ten minutes ago

had 500 MB free, but there is no information about what will be free when one's application runs there.

In general, Grid schedulers get information from a general *Grid Information System* (GIS) that in turn gathers information from individual local resources. Examples of these systems are the Globus Monitoring and Discovery Service (MDS2) [CFFK01, MDS] and the Grid Monitoring Architecture (GMA), developed by the Global Grid Forum performance working group [TAG+03], which has several reference implementations under development [pyG, Smi01, CDF+01, GLM], and is being deployed as part of the European Data Grid project. These two approaches emphasize different pieces of the monitoring problem although both address it as a whole: MDS2 concentrates on the resource discovery portion, while GMA concentrates on the provision of data, especially streaming data.

While different in architecture, all Grid monitoring systems have common features [ZFS03]. Each deals with organizing sets of sensors (information providers in MDS2 or producers in GMA) in such a way that an outside system can have easy access to the data. They recognize that some data is more statically oriented, such as type of operating system or which file systems are accessible; and this static data is often cached or made more rapidly available. They serve dynamic data in very different ways (streaming versus time-out caches, for example) but recognize the need for a heavier-weight interaction for dealing with data that changes more often. All of these systems are extensible to allow additional monitoring of quantities, as well as higher-level services such as better predictions or quality-of-information metrics [VS03, WSH99a, SB99].

Typically, Grid monitoring systems must have an agreed-upon *schema*, or way to describe the attributes of the systems, in order for different systems to understand what the values mean. This is an area of on-going work and research [GLU, DAM, CGS] with considerable debate about how to represent a schema (using LDAP, XML, SQL, CIM, etc.) and what structure should be inherent to the descriptions.

3. STAGES OF GRID SCHEDULING

Grid scheduling involves three main phases: *resource discovery*, which generates a list of potential resources; *information gathering* about those resources and selection of a best set; and *job execution*, which includes file staging and cleanup. These phases, and the steps that make them up, are shown in Figure 2.1.

3.1 Phase 1: Resource Discovery

The first stage in any scheduling interaction involves determining which resources are available to a given user. The resource discovery phase involves

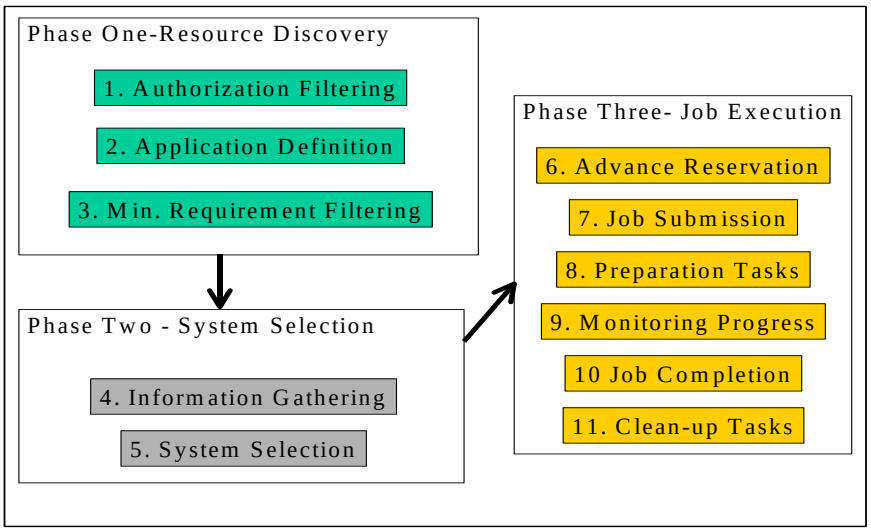


Figure 2.1. Three-phase plan for Grid scheduling.

selecting a set of resources to be investigated in more detail in Phase 2, information gathering. i At the beginning of Phase 1, the potential set of resources is the empty set; at the end of this phase, the potential of resources is some set that has passed a minimal feasibility requirements. The resource discovery phase is done in three steps: authorization filtering, job requirement definition, and filtering to meet the minimal job requirements.

3.1.1 Step 1: Authorization Filtering

The first step of resource discovery for Grid scheduling is to determine the set of resources that the user submitting the job has access to. In this regard, computing over the Grid is no different from remotely submitting a job to a single site: without authorization to run on a resource, the job will not run. At the end of this step the user will have a list of machines or resources to which he or she has access. The main difference that Grid computing lends to this problem is sheer numbers. It is now easier to get access to more resources, although equally difficult to keep track of them. Also, with current GIS implementations, a user can often find out the status of many more machines than where he or she has accounts on. As the number of resources grows, it simply does not make sense to examine those resources that are not authorized for use.

A number of recent efforts have helped users with security once they have accounts, but very little has been done to address the issues of accounting and account management [SN02]. When a user is performing scheduling at the

Grid level, the most common solution to this problem is to simply have a list of account names, machines, and passwords written down somewhere and kept secure. While the information is generally available when needed, this method has problems with fault tolerance and scalability.

3.1.2 Step 2: Application Requirement Definition

To proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources (see Step 3).

The set of possible job requirements can be very broad and will vary significantly between jobs. It may include static details (the operating system or hardware for which a binary of the code is available, or the specific architecture for which the code is best suited) as well as dynamic details (for example, a minimum RAM requirement, connectivity needed, or /tmp space needed). The more details that are included, the better the matching effort can be.

Currently, the user specifies job requirement information as part of the command line or submission script (in PBS [PBS] or LSF [Xu01], for example), or as part of the submitted ClassAd (in approaches using Condor's matchmaking, as detailed in Chapter 23, such as the Cactus work [ADF+01] or the EDG broker [GP01]). Many projects have emphasized the need for job requirement information as well, for example with AppLeS [COBW00, BWC+03] and the Network Weather Service [Wol98]. It is generally assumed in most system work that the information is simply available.

On a Grid system this situation is complicated by the fact that application requirements will change with respect to the systems they are matched to. For example, depending on the architecture and the algorithm, memory requirements may change, as may libraries needed, or assumptions on available disk space.

Very little work has been done to automatically gather this data, or to store it for future use. This is in part because the information may be hard to discover. Attempts to have users supply this information on the fly has generally resulted in data that has dubious accuracy - for example, notice how almost every parallel scheduler requires an expected execution time, but almost every system administration working with these schedulers compensates for the error in the data, by as much as 50% in some cases.

3.1.3 Step 3: Minimal Requirement Filtering

Given a set of resources to which a user has access and at least a small set of job requirements, the third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. At the end of this

step, the user acting as a Grid scheduler will have a reduced set of resources to investigate in more detail.

Current Grid Information Services are set up to contain both static and dynamic data, described in Section 2. Many of them cache data with an associated time-to-live value to allow quicker response times of long-lived data, including basic resource information such as operating system, available software, and hardware configuration. Since resources are distributed and getting data to make scheduling decisions can be slow, this step uses basic, mostly static, data to evaluate whether a resource meets some basic requirements. This is similar to the discovery stage in a monitoring and discovery service.

A user doing his or her own scheduling will simply go through the list of resources and eliminating the ones that do not meet the job requirements (as much as they are known), for example, ruling out all the non-AFS-accessible resources for applications requiring AFS.

Because the line between static and dynamic data is often one drawn for convenience and not for science, most automatic systems incorporate this feasibility searching into Step 4, where full-fledged queries are made on the system. We maintain that as systems grow, this stage will be an important one for continued scalability of other Grid-level schedulers.

3.2 Phase 2: System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This selection is generally done in two steps: gathering detailed information and making a decision. We discuss these two steps separately, but they are inherently intertwined, as the decision process depends on the available information.

3.2.1 Step 4: Dynamic Information Gathering

In order to make the best possible job/resource match, detailed dynamic information about the resources is needed. Since this information may vary with respect to the application being scheduled and the resources being examined, no single solution will work in all, or even most, settings. The dynamic information gathering step has two components: what information is available and how the user can get access to it. The information available will vary from site to site, and users currently have two man sources–a GIS, as described in Section 2, and the local resource scheduler. Details on the kind of information a local resource scheduler can supply are given in Chapter 4.

A more recent issue when interacting with multiple administrative domains is the one of local site policies, and the enforcement of these policies. It is becoming common for a site to specify a percentage of the resources (in terms

of capacity, rime, or some other metric) to be allocated specifically for Grid use. These details must also be considered as part of the dynamic collection of data.

In general, on the Grid, scalability issues as well as consistency concerns significantly complicate the situation. Not only must more queries be made, but also if resources are inaccessible for a period of time, the system must evaluate what to do about the data needed for those sites. Currently, this situation is generally avoided by assuming that if dynamic data is not available, the resources should be ignored; however, in larger systems, another approach should be considered.

A large body of predictive work exists in this area (for example Chapters 14, 15, and 16) but most of it requires additional information not available on current systems. And even the more applied work [Wol98, GTJ+02, SFT98, Dow97] has not been deployed on most current systems.

3.2.2 Step 5: System Selection

With the detailed information gathered in Step 4, the next step is to decide which resource (or set of resources) to use. Various approaches are possible, and we give examples of three in this book, Condor matchmaking in Chapter 17, multi-criteria in Chapter 18, and meta-heuristics in Chapter 19.

3.3 Phase 3: Job Execution

The third phase of Grid scheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources.

3.3.1 Step 6: Advance Reservation (Optional)

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, an advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost.

One issue in having advance reservations become more common is the need for the lower-level resource to support the fundamental services on the native resources. Currently, such support is not implemented for many resources, although as service level agreements become more common (see Chapter 8), this is likely to change.

3.3.2 Step 7: Job Submission

Once resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated

as running a series of scripts and may or may not include setup or staging (see Step 8).

In a Grid system, the simple act of submitting a job can be made very complicated by the lack of any standards for job submission. Some systems, such as the Globus GRAM approach [CFK+98b, GRAc], wrap local scheduling submissions but rely heavily on local-parameter fields. Ongoing efforts in the Global Grid Forum [GGF, SRM] address the need for common APIs [DRM], languages [SD03], and protocols [GRAa], but much work is still to be done.

3.3.3 Step 8: Preparation Tasks

The preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at NASA was considered unsuccessful because it did not address the need to stage files automatically.

Most often, a user will run scp, ftp or a large file transfer protocol such as GridFTP [ABB+02a] to ensure that the data files needed are in place. In a Grid setting, authorization issues, such as having different user names at different sites or storage locations, as well as scalability issues, can complicate this process.

3.3.4 Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing.

Historically, such monitoring is typically done by repetitively querying the resource for status information, but this is changing over time to allow easier access to the data. If a job is not making sufficient progress, it may be rescheduled (i.e., returning to Step 4). Such rescheduling is significantly harder on a Grid system than on a single parallel machine because of the lack of control involved - other jobs may be scheduled and the one of concern pre-empted, possibly without warning or notification. In general, a Grid scheduler may not be able to address this situation. It may be possible to develop additional primitives for interactions between local systems and Grid schedulers to make this behavior more straight-forward.

3.3.5 Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter.

For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envi-

sion situations in which a completion state cannot be reached. And of course, end-to-end performance monitoring to ensure job completion is a very open research question.

3.3.6 Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Any of the current systems that do staging (Step 8) also handle cleanup. Users generally do this by hand after a job is run, or by including clean-up information in their job submission scripts.

4. CONCLUSION

This chapter defines the steps a user currently follows to make a scheduling decision across multiple administrative domains. This approach to scheduling on a Grid comprises three main phases: (1) resource discovery, which generates a list of potential resources; (2) information gathering and choosing a best set of resources; and (3) job execution, which includes file staging and cleanup.

While many schedulers have begun to address the needs of a true Grid-level scheduler, none of them currently supports the full range of actions required. Throughout this chapter we have directed attention to complicating factors that must be addressed for the next generation of schedulers to be more successful in a complicated Grid setting.

Acknowledgments

Thanks to the Grid Forum Scheduling Area participants for initial discussions, especially the co-chair, Bill Nitzberg (PBS, Altair Engineering). Many thanks also to Alan Su (UCSD, AppLeS), Dave Jackson (Maui/Silver), Alain Roy (University of Wisconsin, Condor), Dave Angulo (University of Chicago, Cactus), and Jarek Nabrzyski (Poznan, GridLab). This work was supported in part by the Mathematical Information and Computational Sciences Division Subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

Chapter 3

APPLICATION REQUIREMENTS FOR RESOURCE BROKERING IN A GRID ENVIRONMENT

Michael Russell,[1] Gabrielle Allen,[1] Tom Goodale,[1] Jarek Nabrzyski,[2] and Ed Seidel[1]
[1]*Max Planck Institute for Gravitational Physics*
[2]*Poznan Supercomputing and Networking Center*

Abstract We discuss the problem of resource brokering in a Grid environment from the perspective of general application needs. Starting from a illustrative scenario, these requirements are broken down into the general areas of computation, data, network, security and accounting. Immediate needs for applications to profitably start using the Grid are discussed, along with case studies for applications from astrophysics and particle physics.

1. INTRODUCTION

In this chapter we identify some of the key application resource brokering requirements that Grids must be able to satisfy, based on our experiences in working with teams of distributed researchers studying complex physics phenomena. In our research we are confronted daily with the need for a global computing infrastructure and with our own dreams of building more powerful applications. To better illustrate our point, we begin the next section with a scenario that details one of the many ways we envision using Grids in the future. In Section 3, we detail the general requirements highlighted by this scenario and others. Section 4 discusses some of the key challenge problems that remain to be solved before such a scenario can be realized. Section 5 describes some of our own work in developing Grid solutions for the astrophysics community and a second use case illustrating some of the general requirements of high-energy physics applications. Finally, we conclude our chapter with a brief look at important issues related to the resource brokering requirements.

2. MOTIVATING SCENARIO

Now consider the following scenario, illustrated in Figure 3.1, motivated by the very real and pressing needs of the gravitational physics community spurred by the imminent arrival of data from a world wide network of laser interferometric gravitational wave detectors [Gib02].

The gravitational wave detector network collects a TByte of data each day that must be searched using different algorithms for possible events such as black hole and neutron star collisions, pulsar signals or other astronomical phenomena. Routine real-time analysis of gravitational wave data from the detector identifies a burst event, but this standard analysis reveals no information about the burst source or location. To obtain this location, desperately required by astrophysicists for turning their telescopes to view the event before its visual signal fades, a large series of pre-computed possible gravitational wave signals, called *templates*, must be cross-correlated against the detector data.

Gwen, an Italian astrophysicist monitoring the real-time analysis, accesses the detector's Portal, where the performance tool reports that 3 TFlops/s will be needed to analyze the 100GB of raw data within an hour and that the data will need to be transfered to the machines before analysis can begin. Local Italian resources are insufficient, so using the brokering tool, she locates the fastest available machines around the world that have networks to the gravitational wave data repository that are fast and stable enough for the transfer of the necessary data in a matter of minutes. She selects five machines that together are able to perform the data analysis in the required hour, and with scheduling and data management tools, data is moved, executables created, and the analysis begins.

The analysis proceeds smoothly until it is determined that a custom template is required that must be created from a full scale numerical black hole simulation. The portal, aware that Gwen is still online and working, informs her of this development by instant message. Gwen is immediately able to use the brokering tool to search for large resources capable of running such a simulation. No single machine is available, but two machines are located that together form a large virtual machine connected by sufficient bandwidth and as a unit they will be able to complete the distributed simulation in an additional hour. Gwen then uses the application tools on the portal to assemble the correct executable for this run and stages it to run immediately across the two machines. Before leaving for lunch she uses the broker to extend the reservations on the original resources required for the data analysis to allow for this set back.

An hour later, the simulation finishes and the needed template is automatically delivered to the template repository. The original data analysis can now continue.

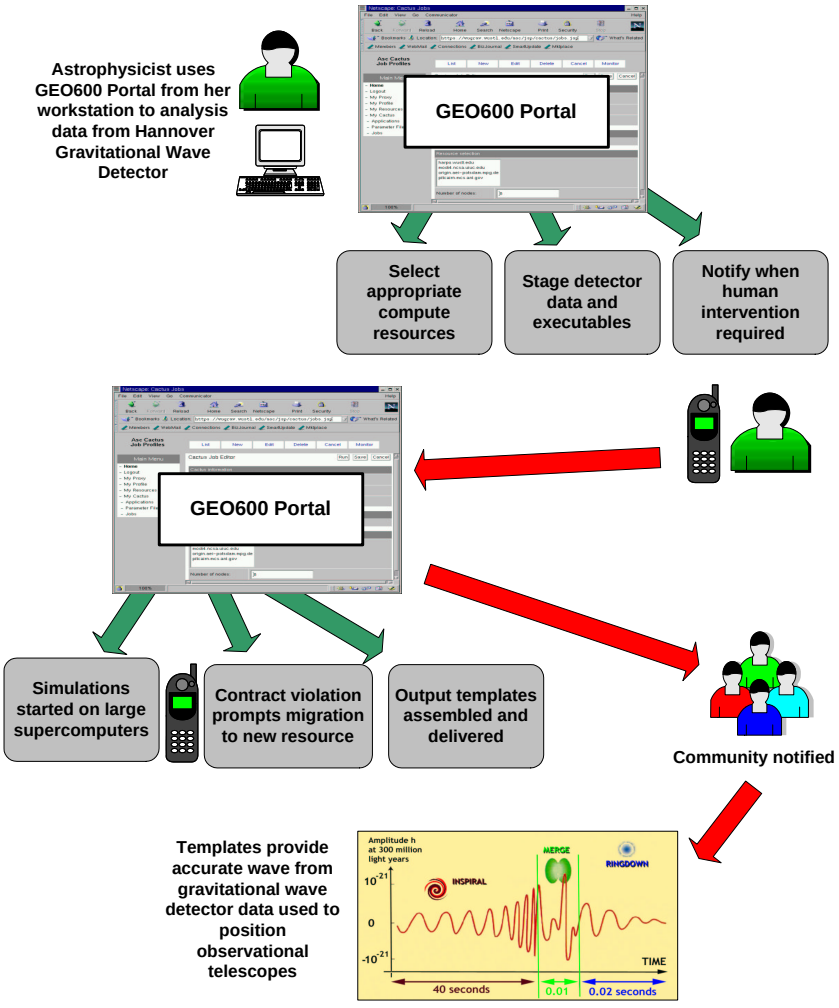


Figure 3.1. A Grid scenario showing the potential use of resource brokering in a gravitational wave data analysis.

In an cafe twenty minutes later, an urgent text message on her mobile phone from the Portal's notification tool informs her that one machine is now overloaded, breaking its runtime contract. The Portal confirmation to make use of a different, more expensive, machine in order to fulfill the contract. She judges that time is of the essence, connects with her PDA to the Portal, and instructs the migration tool to move this part of the analysis to the indicated machine. Within the specified hour, a second text message tells her and her collaborators that the analysis is finished, and provides the location of the resulting data, which is available for all to view. Using this data, observatories are now correctly instructed to position their telescopes, and astrophysicists are able to find and view an exceptionally strong gamma-ray burst, characteristic of a supernovae.

This scenario illustrates how, given the appropriate resource brokering mechanisms, a researcher need not be concerned with where and how computing resources are acquired and used. It may be, that the researcher wishes to make a final call as to which resources are used, as was in the case in our example. Even then this decision process should be relatively straightforward given a user's preferences and application resource requirements. Indeed, Grids are motivated by these requirements.

While the complete scenario is still futuristic, many elements of the required infrastructure are already in place, and research projects such as application-focused GridLab [GL, ADD+03], GrADS [BCC+01], and GriPhyN [DKM+02] are all working to provide missing components and to provide working environments and tools to make such scenarios a reality. However, many challenges, both technical and sociological, remain in developing both the Grid infrastructure and the applications that will successfully run in such environments. Resource brokering is the fundamental component that will move us much closer to making this a reality.

3. GENERAL REQUIREMENTS

The gravitational wave detection scenario above provides a useful backdrop in which to discuss the basic resource brokering needs many applications exhibit, but by no means are the requirements discussed here limited to that example. There are at least three major requirements areas that have direct bearing on the performance needs of applications and we address them here – *compute-related*, *data-related*, and *network-related* requirements.

3.1 Compute-Related Requirements

Computationally-intensive applications drive the need for brokering services that can quickly and effectively locate high-performance computational resources given the particular requirements of those applications and end-users.

In locating resources, brokering services need to take into account not only the computational needs of an application (available CPU, memory, disk space, etc.), but also key factors such as the time at which the application is required to run, the financial cost of the resource, and the efficiency of application on that particular resource. There are many other factors that may also be taken into account, such as the machine failure rate, the network characteristics for transferring results to mass storage or the firewall security policy that could prevent the use of remote interactions. In our example scenario, results were urgently needed, and the throughput was the overriding consideration in choosing resources.

Grids present a challenge for applications in that they are heterogeneous, so may consist of many different types of computing platforms and operating systems (and versions of operating systems). Resources also have varying software installations and environments, such as shells, compilers, libraries. A given application may be compatible with only a limited set of computing architectures and software. Even when an application is portable across many platforms its particular characteristics may dictate that it will run more efficiently on a given architecture. These needs must not only be taken into account by the brokering services, but also dictate the need for applications to be able to sufficiently describe their requirements.

3.2 Data-Related Requirements

Most applications have at least some basic file management needs that place constraints on how the computational resources are brokered in a Grid environment. Consider, for example, a simulation or data transformation application that operates on a set of input files and produces output files to be analyzed or processed by a second application to visualize the resulting data. The size of the input and output files and the amount of overall storage space allotted to a user necessarily has some bearing on the scheduling of such applications. If the input files must be retrieved from a remote location, then the time required to transfer the files must be taken into consideration when scheduling compute resources for the given application. Moreover, many computing sites require users to archive their data files in some mass secondary storage or tertiary storage immediately after the termination of a computing job. Ideally, an end-user should not be concerned where their files are, so long as file operations are taken care of efficiently and with minimal trouble to the end-user.

As we saw in our gravitational wave detection scenario, the need for access to distributed datasets is a powerful motivation for building and using Grids. Some applications must perform analysis on datasets geographically-distributed among many files, and this requires an efficient means to locate and access datasets irrespective of their location and file organization. In order to

provide this level of data access at least two mechanisms are required. First, it must be possible to associate meta-data with files that describes their content in as much detail as is required for their given applications. Second, it must be possible to access the contents of files efficiently according to the data model. By combining dataset access capabilities with the file replica management capabilities (described in detail in Chapter 22, application programmers can develop more powerful and useful applications for visualizing and analyzing datasets.

Alternatively, datasets may be distributed among several relational database management systems (RDBMs), as is the case for many bio-informatics applications that perform extensive database queries for discovering particular gene patterns in proteins. In a Grid environment, where applications may be run on any given set of resources, the means for how databases are advertised, located, and accessed becomes an important, and complicated, issue. Additionally, the need for co-allocation mechanisms across computational and database resources becomes more critical in a Grid environments.

In order to begin addressing these problems the OGSA Distributed Access and Integration (OGSA-DAI) project [OGSb] is currently designing and building front-end services to relational databases and XML repositories that conform to OGSA [FKNT02]. These front-end services can then be employed to develop higher-level, federated databases across the Grid that offer a virtual database system to application developers.

3.3 Network-Related Requirements

In Grid environments the communication requirements of an application are subject to numerous network constraints. These considerations have a direct bearing on the performance of the application, and hence on how resources need to be brokered for it. As in our scenario, it is typical for an application to require access to significant amounts of input data from remote locations. Applications may also need to write data to a remote file, provide analysis data for remote visualization or provide a remote steering interface. Distributed applications, including tightly coupled simulations as in our scenario, may need to regularly exchange data between distant machines.

While communication-intensive applications may require high-speed networks for transferring large amounts of data, some applications simply require dependable networks. At minimum, it must be possible to discover the conditions of networks on which applications are to be run. This prescribes the need for network monitoring and prediction services that can provide applications, resource brokering services, and users alike with the ability to query the characteristics of networks associated with the target computing resources. For an end user, such characteristics may include transfer times and reliability,

whereas developers designing communication patterns for applications may also require bandwidth and latency information. Such services may also provide applications with details about how to best use the network, for example the optimal packet size to use for sending data.

Some applications, especially communication intensive and real-time applications, require certain minimum guarantees on the network bandwidth or network speed of communication links between network nodes, hence must rely on some type of Quality-of-Service (QoS) brokering services providing guaranteed reservation of network capacity along with computational resources. This approach is discussed in more detail in Chapters 23 and 24.

External networks may also place constraints on the communication patterns of an application. Firewalls can prevent communication between any two machines, system administrators may close down services if they misinterpret remote accesses, and machines or internal processors of clusters without external IP address present fundamental problems.

4. CHALLENGE PROBLEMS

There remain many issues to resolve to meet basic resource brokering requirements in today's supercomputing and Grid environments. This section describes the main challenge problems to be addressed to make out earlier example scenario a reality, including issues of application deployment, support for metacomputing, performance prediction and adaptive brokering.

4.1 Application Deployment

One challenge problem that we touched upon earlier is how to broker the appropriate resources for an application given that some applications can only run on a limited set of computing architectures, operating systems, and in some cases specific machines. It makes sense to associate metadata with an application to describe the computing platforms on which it can execute. In addition to the machine architectures and operating systems with which it is compatible, this metadata can be used to specify the libraries, tools, or services it requires on target resources. Such metadata could then be used by a resource brokering service to allocate computational resources that are appropriate for an application.

There is a need for a Grid *application deployment descriptor* standard that all Grid resource brokering services could utilize to broker resources for applications. This would simplify the use of Grids from the perspective of application developers and increase the portability of their applications across different Grids. In the general case, a resource brokering service needs to be able to find information that describes how to obtain and deploy an applica-

tion's executable onto target resources, as well as information about how these activities will affect the resource requirements.

For example, an application may exist in an repository of executables or as source code in concurrent-version-service (CVS) repositories from where it can be checked-out onto target resources and compiled into executables on demand, or it may already be installed on the target machine. It can take a significant amount of time to fetch and build an executable, and this process is also machine dependent. For example, the Cactus binary used to create Figure 3.2 takes several hours to compile on some machines, but only minutes on others. This can be addressed by a version tracking service that stages an application onto potential resources of interest when new versions of the application become available.

The resource brokering service may need to fetch input data files before executing the application. Depending on the size of the input files and the network characteristics, this process may also last long enough to become a factor in negotiating what resources are best for running the application. This data should be included in the application meta-data.

4.2 Metacomputing

There are various motivations and methods for metacomputing, that is, distributing tightly coupled applications across multiple resources. Simulations, such as the black hole simulations in our example scenario, are limited by the size of the machine they can run on. First, by harnessing multiple machines simultaneously, much larger—and hence more accurate or detailed—simulations can be run. Second, as in the scenario, if more processors are needed for a job, but no single machine can provide them right away, a resource broker can find several machines that can each provide some of the processors, thereby increasing throughput dramatically.

If applications are distributed across multiple machines, the issue of network latency must be taken into account. Computationally intensive applications, such as black holes simulations, are often programmed using a parallel programming model, for example domain decomposition with message-passing using MPI [SOHL+98]. MPICH-G2 [KTF03], from the Globus Project, provides a communication layer for MPI codes to run across multiple machines connected by a WAN. The efficiency for applications in such an environment has always been limited by the latency in the connections between sites. Depending on the nature of the problem and its communication needs, adaptive techniques can be applied to compensate for that latency [ADF+01].

If the potential of metacomputing is realized, the job of a Grid scheduler increases dramatically. A scheduler will have to be able to coordinate the execution of a job between the different job schedulers and job queues that exist

at each computing site. An evaluation of feasible resources must take place, as some job queues may have a long delay in job execution while others may not provide the minimum amount of processors or memory to make a distributed job worth considering. Finding the best configuration given these limitations may result in costly searches. Worse, without some means of coordinating job submissions at the Grid level, the problem is likely to be intractable.

4.3 Predicting Performance

The ability to predict an application's performance across different computing platforms and use-case scenarios can better enable brokering services to negotiate appropriate resources for that application, given its deployment descriptor and the user's preferences for that application. There are three ways to approach this problem: *theoretical prediction*, or making a prediction based on an analysis of an application's programming model and problem domain, *history-based predication*, or making a prediction based on previous runs, and *testcase-based prediction*, where the application itself is tested for a limited set of cases on representative machines.

In the case of theoretical predictions, the solution for any application depends in part on how well understood the computing model is for that application and whether it makes sense to predict performance based on that model. Still, there are many applications for which the programming model is sufficiently well known and the application problem domain lends itself to predictability.

History-based predictions make the most sense for applications that are not likely to change over time, are dedicated to solving well known problems, or are typically executed on a given set of resources where past performance is a strong indicator of future performance of the application on those same resources.

Testcase-based prediction may be the only method for dealing with applications that have unknown characteristics or applications that change over time, for example due to ongoing development. Such an approach is most suitable for the black hole simulation code in our example since the computational characteristics depend dynamically on the initial data used and it may not be possible to determine *a priori* whether computational and communication intensive elliptic solvers will be needed at each iteration of the simulation. Prediction techniques in Grid environments are described in more detail in Chapter 14, Chapter 15, and Chapter 16.

4.4 Adaptive Brokering

Given the complexities encountered in brokering resources in a Grid environment where it may not be possible to reliably predict the future performance

of computational resources and the networks on which they reside, especially under heavy usage, then the need for adaptive brokering services becomes more apparent. Adaptive brokering allows for the selection of resources to change over time if they stop meeting the needs of an application when it is executing. The CACTUS group, as part of the GrADS project [BCC+01], has developed one approach to this problem that allows for the migration of tasks. This is discussed in detail in Chapter 6.

5. CASE STUDIES

We now take a look at two case studies that put the above requirements discussion into a real-world context. Our first study details our experience in supporting the computing needs of numerical physics applications in Grid environments. Here we investigate resource brokering requirements of massively parallel, high-performance programs such as those developed with the Cactus Toolkit. In our second case study, we describe the Compact Muon Solenoid (CMS) Collaboration, a worldwide virtual organization of researchers and scientists preparing for the upcoming CMS experiment to take place at CERN. This overview sheds light on the some of the complex computational, data management, and network problems that must be solved for many high-energy physics applications.

5.1 Numerical Relativity

Numerical relativists make daily use of supercomputers to run large-scale simulations that model black holes, neutron stars, gravitational waves, and other objects in the universe with Einstein's General Theory of Relativity. Their results, as in our initial scenario, are crucial to the success of gravitational wave astronomers that use the new generation of laser interferometric detectors.

These simulations (see Figure 3.2), as for many other fields of science, are limited in the accuracy they can model nature by the size of the computational resources available to them. The rate at which the relativists can study physics is restricted by the amount of resources available as well as by the throughput of simulations that can be obtained. The daily working environment of the relativists is greatly impacted by the ease with which they can (1) find information about the state of the resources, data, and simulations, (2) access and stage jobs, and (3) interact with the resulting distributed data.

Collaborative issues are also very important. The numerical relativity community consists of diverse and geographically distributed researchers making use of multiple resources from many different centers and funding agencies, spanning different countries, certificate authorities, and Grid information

servers. The researchers need to be able to efficiently share and coordinate their codes, results, expertise, and resources.

The field of numerical relativity can thus highlight many requirements for Grid technologies. In particular, resource brokering is a fundamental service that can aid their work by simplifying their day-to-day working practices, where today they need to master many different operating systems, file systems and job submission procedures.

Core requirements from this community include that Grid technologies be stable, robust and usable on *all* their important resources, that there be accessible information about the status and location of the running simulations and result data, and that the end-user can easily control when and where their simulations are deployed, if so desired.

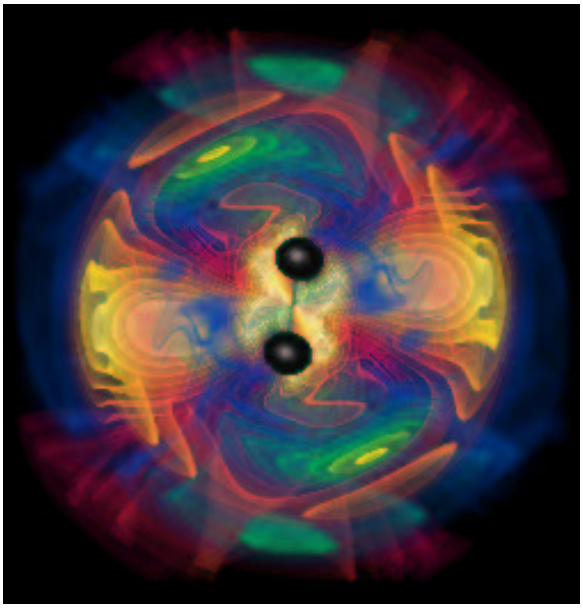


Figure 3.2. Two orbiting black holes, at center, are about to collide in this visualization of one the largest simulations to date of this process. The emitted gravitational waves, hoped to be detected this decade, are shown as swirls.

5.1.1 Cactus Code

Several numerical relativity groups, including researchers at the University of Texas (USA) [UT], RIKEN (Japan) [RIK], and National Autonomous University of Mexico (Mexico) [UNA], now use the Cactus Code [GAL+03] as an application programming framework. Cactus, developed at the Max Planck In-

stitute for Gravitational Physics, consists of a core *flesh*, which coordinates the activities of modules, and the modules, which are referred to as *thorns*. Thorns contain both computational infrastructure (e.g. parallel drivers, I/O, coordinates) and application components. For convenience, thorns are packaged into toolkits. The Cactus Computational Toolkit provides core computational infrastructure, and the Cactus Einstein Toolkit provides the basic infrastructure and functionality for numerical relativity simulations. Other disciplines can construct their own toolkits.

Cactus thorns interoperate with the flesh and each other via standard interfaces. This makes the development and use of thorns independent of one another, which in turn eases the coordination of geographically dispersed research groups. It also allows each group to concentrate on its own area of expertise. For example, physicists may focus on the development of physics thorns, while computer scientists may develop Grid-enabling thorns that support interoperability with Grid services (e.g. GRAM-based job schedulers, MDS-based information services).

The modularity, flexibility and computational infrastructure of Cactus, coupled with the urgent needs of the numerical relativity community, has made it a good application for prototyping Grid scenarios. In particular, it is a prime application for the GridLab Project that is developing application-oriented environments to provide Grid infrastructure, including a Grid Application Toolkit [AAG+02] and a collaborative Grid portal framework called GridSphere [NRW03].

Several prototype approaches have been shown to greatly increase the usability of Grid environments for numerical relativity simulations for real black hole simulations in Cactus. First, job migration [AAF+01] can move a simulation between resources, either when a job has exhausted its queue allocation at one machine, if a broker determines that a different resource is more appropriate (e.g. cheaper, faster or larger), or if a runtime contract has been violated. Automatic and intelligent task farming of independent or loosely coupled simulations greatly eases the burden of parameter surveys. As any simulation runs, independent tasks can be automatically spawned to alternative resources, allowing the main simulation to run faster and making use of possibly cheaper resources. Simulations too big for any individual machine can be staged on large virtual machines constructed from a set of distributed resources [ADF+01].

5.2 High-Energy Physics

Applications that perform data-analysis on high-energy physics (HEP) experiments provide excellent case-studies of requirements for Grid environments. Consider, for example, the Compact Muon Solenoid (CMS) data-

analysis application. The CMS Collaboration [Hol01] consists of over 2300 people from 159 institutes in 36 countries, and is part of several large-scale Grid projects, including GriPhyN [GRIb], PPDG [PPD], and EU Data Grid [EDGa] in order to tackle its complex computational, data management, and network resource needs.

The CERN CMS detector is one of two general purpose detectors of the Large Hadron Collider (LHC) accelerator and will begin taking data in 2006. The LHC accelerator circulates a large number of particle bunches, containing 10^{11} protons, at a rate of 40,000,000 times per second, allowing two bunches of particles coming from opposite directions to cross through each other inside the CMS detector. CMS has a high event rate because physicists are looking for phenomena that have extremely low probability of occurring in any single event. Of the 40,000,000 events per second detected, some 100 are selected for storage and later analysis, where the size of one CMS event is approximately 1MB.

One of the prime objectives of CMS is to confirm the existence of the Higgs boson, a particle that cannot be observed directly but rather will be inferred through the existence of four instances of another type of particle, a positively-charged lepton. There are a number of events that produce charged leptons, of which only a small number can be used to study properties of the Higgs boson particle. The data reduction requirements for this type of analysis are quite extreme. Only a few hundred events selected from 4×10^{14} events may be selected in the first year of the CMS detector going online. Some of the data reduction will happen before any data is stored, but the rest must be conducted on the data both offline and collaboratively via Grid resources.

5.2.1 Key Requirements

CMS analysis illustrates Grid requirements for HEP applications quite well. Researchers distributed around the globe will be analyzing CMS experiment data within a year after it becomes available. Typically, the data for each event is small, 1KB-1MB. However, CMS analysis is a continuous data reduction process. Moreover many kinds of CMS applications will need to analyze terabyte-to-petabyte samples drawn from multi-petabyte data stores, requiring large-scale transfers that correspond to very high throughput requirements. This requires not just rapid advances in existing network infrastructures but services for brokering network and data resources to the many HEP and other large-scale data-oriented applications that will be competing for those resources.

On the other hand, CMS analysis can be done in a step-wise fashion, by successively applying algorithms for event selection and data processing and scheduling a workflow of jobs corresponding to each stage in the analysis. This also means that theoretical and historical predictive techniques can be applied

to improve resource allocation. Furthermore, since the computational workloads of data analysis applications tend to be read-oriented, the replication and caching requirements are simpler than for some other applications. Nevertheless, effective replication and caching strategies are integral parts of the overall resource brokering solutions for HEP applications.

One attempt to demonstrate the potential value of a Grid-enabled R&D system for Monte Carlo analysis of the CMS experiment was developed by researchers in the US CMS Collaboration [USC]. The CMS Monte Carlo Production (MOP) system [MOP], demonstrated at SuperComputing 2001, uses Condor-G [FTF+02], to distribute production jobs and the Grid Data Mirroring Package (GDMP) [GDM] to replicate files in a Data Grid environment. MOP scripts, developed at FermiLab, define a series of jobs to be scheduled for submission by Condor-G, while GDMP relies upon the Globus Replica Catalog and GridFTP to securely replicate files as required for later analysis. GDMP is part of the EU DataGrid software package. Condor-G is described in more detail in Chapter 9.

6. RELATED ISSUES

Although the new functionality described here will provide tremendous advantages for applications over the current use of Grid resources, the future that we envision holds far greater promises. Reaching further will require fundamental changes in today's technologies and policies, and in the way in which applications and the resources on which they run are viewed.

6.1 Application Frameworks

In order to make the most effective use of Grid environments, applications need to be able to take advantage of any brokered resources, so they need to be able to be highly portable, self-configuring and self-optimizing. This suggests the need for *application frameworks* that support multiple computing architectures and configurations. Beyond portability, however, application frameworks should provide developers with higher-level operations. For example, a framework could include checkpointing or migration operations to allow an application to be easily moved between resources should its resource requirements change over time. This suggests that applications should be self-describing in order to be able to provide information about their requirements and capabilities to resource brokers.

Indeed, these ideas are what help drive the development of the Cactus Toolkit. Its portability means that users are able to develop and test applications on their laptops, and then easily stage and run them on the worlds largest and newest supercomputers. Cactus includes many features that make it particularly suitable for exploiting Grid environments and Grid paradigms — support

for remote parameter steering, platform independent I/O, a switchable parallel layer, robust checkpointing and restart, portability, a flexible make system, and the ability to monitor and control interactions through any web browser. These are the kinds of features that will make the scenarios like the gravitational wave detection example truly possible.

6.2 Virtual Organizations

Perhaps most importantly, providing better support for virtual organizations (VOs) at all levels will enable collaborative groups to better control and leverage their collective resources. Today, VOs simply can't exist without a great of deal of effort from administrators and project managers. This is because by definition VOs may consist of multiple administrative domains, each of which may have a unique set of security requirements and accounting policies.

6.3 Security Requirements

Security is an increasingly important consideration as applications begin to routinely make use of networks. In our motivating scenario, the physicist needed to authenticate to a portal, access and move protected community data, provide the resource broker with credentials to use compute resources, and so on. The user also needed to be able to trust that the services she was using would not compromise her own credentials. A persistent current problem for applications attempting to interact with, and provide, remote services is the existence of firewalls around resources. Such security requirements are often the last thing on a user's mind, and viewed as a hindrance imposed on their working. But addressing security, as discussed in Chapter 5, is one of most difficult problems engineers have to face when building Grid solutions.

6.4 Accounting Policies

The cost of resources is generally an important factor for end-users in deciding where to run their applications. In general, a limited number of *computing units* at various computing centers are generally provided by funding agencies, where a computing unit is usually associated with some unit cost for each processor per hour. In addition, there are often file system quotas on the amount of data a user can store at any given time in different file systems. At any one site there may be queues with different associated costs, so that an application may be completed faster if more units are paid during each processor hour.

6.5 User Preferences

No matter how intelligent applications and resource brokering services become, we can never fully anticipate what a user's needs will be. This means enabling users to balance their application requirements with other requirements for cost, throughput time and efficiency. In order to build more robust solutions, we can try to incorporate user preferences by studying their behavior and usage patterns. For example, currently large computer centers frown upon individuals who submit too many jobs, or who take advantage of existing scheduler logic to jump up a queue and gain faster throughput. However, as applications become more dynamic, we can expect that they will begin to find new ways to exploit scheduler logic and such. Therefore, we must find new to support dynamic behavior so brokers can quickly and efficiently allocate extra resources or deallocate unused resources.

7. SUMMARY

This chapter provides an introduction to some of the major problems that Grid resource brokering services must solve in order to make Grids viable and practical to users. Additionally, we have discussed some of the key challenge problems and issues that if overcome will pave the way for new and more powerful applications scenarios. Finally, the case studies provided here emphasize the very real needs of scientists and research communities today and in the near future.

Acknowledgments

The work and ideas described in this paper were contributed to by many colleagues, and we thank in particular the physicists in the numerical relativity group at Max Planck Institute in Potsdam, Germany, who have contributed many requirements for their real needs for large scale computing. We are pleased to acknowledge support from the EU GridLab project (IST-2001-32133).

Chapter 4

ATTRIBUTES FOR COMMUNICATION BETWEEN GRID SCHEDULING INSTANCES

Uwe Schwiegelshohn and Ramin Yahyapour
Computer Engineering Institute, University Dortmund

Abstract

Typically, Grid resources are subject to individual access and usage policies because they are provided by different owners. These policies are usually enforced by local management systems that maintain control of the resources. However, few Grid users are willing to deal with those management systems directly in order to coordinate the resource allocation for their jobs. This leads to a Grid scheduling architecture with several layers. In such an architecture, a higher-level Grid scheduling layer and the lower-level layer of local scheduling systems must efficiently cooperate in order to make the best use of Grid resources. In this chapter we describe attributes characterizing those features of local management systems that can be exploited by a Grid scheduler.

1. INTRODUCTION

Some computational Grids are based on compute and network resources of a single owner, for example in the case of Enterprise systems. But many computational Grids consist of resources with different owners in order to provide a high degree of flexibility and to allow efficient sharing of resources. However, in these cases, most owners are not willing to devote their resources exclusively for Grid use. For instance, a computer may temporarily be removed from a Grid to work solely on a local problem. In order to react immediately in such situations, owners typically insist on local control over their resources, which is achieved by implementing a local management system.

On the other hand, it can be a cumbersome and tedious for a potential Grid user to manually find, reserve and allocate all the resources needed to run an application. To automate this process, a specific Grid management system is needed. Ideally, such a Grid management system includes a separate schedul-

ing layer that collects the Grid resources specified in a job request, checks the considerations of all requirements, and interacts with the local scheduling systems of the individual Grid resources. Hence, the scheduling paradigm of such a Grid management system will significantly deviate from that of local or centralized schedulers that typically have immediate access to all system information. Although it seems obvious that a Grid scheduler may consist of more than a single scheduling layer, many details of an appropriate scheduling architecture have not yet been established.

Nevertheless, it is clear that some layers are closer to the user (*higher-level scheduling instance*) while others are closer to the resource (*lower-level scheduling instance*). And of course those different layers must exchange information.

The information that passes between the scheduling layers of a Grid management system is subject of this chapter. Clearly, the flow of information between the layers is not symmetric: A query flows from a higher-level to a lower-level scheduling instance, while a confirmation takes the opposite direction. However, this observation does not necessarily indicate a fixed or static hierarchy of schedulers. Also the concept of scheduling layers is not restricted to two levels of scheduling instances in general. For instance, consider a large computational Grid consisting of several Virtual Organizations (VOs). Each of those VOs may have a separate Grid management system. A user request needs not to be executed in the VO in which the request was issued. Therefore, the Grid scheduler of one VO passes the request to the Grid scheduler of the second VO. Then this second Grid scheduler interacts with the local scheduling systems of the resources in the VO it works for. Clearly, three scheduling instances are involved in this situation. The Grid scheduler of the second VO is a lower-level scheduling instance with respect to the Grid scheduler of the first VO, while it is a higher-level scheduling instance in the communication process with the local scheduling systems of the second VO. Of course the scheduling instance in the lowest layer is always a local scheduling system.

Nowadays, a variety of different local scheduling systems, for example PBS [PBS], LoadLeveler [IBM01], LSF [Xu01], or Condor [LLM88], are installed on the individual computer systems. Some of these local scheduling systems are presented in this book. See Chapter 11 for information on Maui/Silver, Chapter 12 for information on LSF, and Chapter 13 for information on PBS.

A Grid scheduling layer must exploit the capabilities of these local scheduling systems to make efficient use of the corresponding Grid resources. However, those capabilities are not the same for all local scheduling systems due to system heterogeneity. Therefore, we introduce attributes to describe, in general, the features of a lower-level scheduling instance that can be used by a higher-level scheduling instance. The attributes are also the first step towards a

classification of Grid-relevant properties of local schedulers. Further, they can be used to indicate what future enhancements of local scheduling systems may improve their Grid-readiness.

Note that attributes can be combined: if more than two levels of scheduling instances are used as described above, then a scheduling instance in the middle collects attributes from possibly several lower-level scheduling instances, combines them, and provides those combined attributes to a higher-level scheduling instance. As the combination of attributes is not subject of this chapter, we restrict ourselves to two scheduling instances in the following sections.

Although we have stated above that the attributes can be used to evaluate Grid-readiness of a scheduling system, we want to strongly emphasize that the features described by the attributes are neither an obligation nor a limitation for the design of a lower-level scheduling system. For instance, some features are not relevant for certain resource types and need not to be considered in the corresponding local schedulers. The presence of an attribute simply indicates that the specified feature is provided by the lower-level scheduling instance and can be used by a higher-level scheduling instance. On the other hand, if such an attribute is not present for a lower-level scheduling instance then the higher-level scheduling instance cannot use the specified feature.

Also note that it is not the purpose of this chapter to address mechanisms for the communication between scheduling layers. Similarly, we do not define the structure and syntax for the description of resources. This kind of information can be accessed through other Grid information services which are currently subject to a standardization process in the Information Systems and Performance area [ISP] and the Architecture area [ARC] of the Global Grid Forum [GGF]. Consequently, there are no attributes in this description to determine, for instance, the set of resources for which a lower-level scheduler is responsible.

In the next section we present an example a Grid application. This example is used to illustrate the meaning of some of the attributes which are listed in the following sections. These are classified into 4 groups:

1 Attributes used when accessing the available scheduling information (Section 3),

2 Attributes used when requesting the resources (Section 4),

3 Attributes used when querying for allocation properties (Section 5), and

4 Attributes used when manipulating the allocation execution (Section 6).

The concepts described in this chapter are a result of the work done by the Scheduling Attributes Working Group of the Global Grid Forum, and they have been published as a Grid Forum Document [SY01]. The Working Group stated however, that this list of attributes while sufficient may not yet be complete.

2. TYPICAL USE OF A GRID

For a purpose of illustrating some of the attributes we consider a simple example where a job is executed on a computational Grid that includes the following resources with different owners:

- Several clusters of workstations,
- A visualization cave [LJD+99],
- A database, and
- A bandwidth broker for the network that connects other resources.

Our example job is a workflow job consisting of three stages:

1. Data transfer from the database to a workstation cluster,
2. Preprocessing of the basic data on the workstation cluster in batch mode and generation of some temporary data, and
3. Processing of the temporary data on the workstation cluster and, in parallel, online visualization of the results in the visualization cave.

A user generates a request and submits it to the Grid scheduler that tries to find a suitable allocation of resources. As not all of the requested resources are available at a single site, the job requires multi-site processing in Stage 3. To this end the Grid scheduler requires information about the properties of the involved local scheduling systems. Scheduling attributes described in this chapter provide this information.

Before addressing the task of the Grid scheduler in detail we define an *allocation* to be an assignment of resources to a request. An allocation is tentative until it is executed, that is, until resources are actually consumed. A *schedule* gives information about planned or guaranteed allocations. Such a guarantee of an allocation means that there is a guaranteed assignment of resources. This does not necessarily guarantee the completion of a job, as the actual job processing time may exceed the requested allocation time.

Note that there is a strict dependence between the stages in the example above. In Stage 1 the database, the workstation cluster and sufficient bandwidth in the network must be available at the same time. The length of this stage is mainly determined by the network bandwidth. However, this step is not necessary if the database is deployed on the same cluster that is used in Stages 2 and 3. The efficient execution of Stage 3 requires concurrent access to the workstation cluster, the visualization cave, and the network. For Stage 3, the visualization cave and the network require concurrent access to the workstation cluster. Moreover, the network must provide the bandwidth required by the job.

The Grid scheduler used in the example tries to obtain an allocation with a *guaranteed completion time* for Stage 2 in order to make an *advance reservation* for the visualization cave and the network in the next stage. In Stage 3, the Grid scheduler requests an *exclusive allocation* that will *run-to-completion* on the processing resources to prevent any negative influence of other independent jobs on the visualization. Finally, the Grid scheduler asks for a *tentative schedule* of the visualization cave and the network in order to find a sufficiently large time slot during which all the resources required for Stage 3 are available.

This example shows how a Grid scheduler can use certain features of local scheduling systems to generate an efficient schedule. The availability of those features is described by attributes. To this end, a Grid scheduler needs to know which features are available for a local scheduling system.

3. ACCESS TO AVAILABLE SCHEDULING INFORMATION

The first set of attributes addresses local schedule information that is made accessible to a higher-level scheduling instance.

3.1 Access to a Tentative Schedule

Some local management systems are able to return on request the complete schedule of current and future allocations. This information allows a Grid scheduler to efficiently determine suitable timeslots for co-allocation of resources in multi-site computing, as needed for Stage 3 of our example. Note that a local management system may not return the complete schedule due to architectural limitations or due to system policy restrictions.

Even if the complete schedule of a local resource cannot be made available, the local management system may return some information that can be used by a higher-level scheduler. The type and the amount of this information can be specified with additional options of this attribute. As an example, a local management system may provide an authorized higher-level scheduling system with the projected start time of a specified allocation.

In other systems, the returned schedule information is subject to change as there may be a different access to those local resources that is not controlled by the lower-level scheduling instance.Then this information has a limited reliability, which can also be described by an option of this attribute.

3.2 Exclusive Control

The local scheduler has exclusive control over its resources, that is, no allocations can be scheduled on those resources without using this scheduler instance. In this case, the reliability of schedule information from the local

scheduler is very high as there can be little outside interference. Then, a Grid scheduler can better exploit the knowledge about a local schedule to generate efficient allocations for complex job requests.

3.3 Event Notification

Some unforeseeable events may influence a local schedule, for example a sudden failure of a resource or the early termination of a job. A local scheduler may pass information on to a higher-level scheduling instance if this scheduling instance has subscribed to the appropriate event notification. If so, the higher-level scheduling instance can quickly react and reschedule some allocations. Those events may also include notification of an allocation change, for example cancellation or changed execution time, as well as information about modified resource conditions. This attribute states that the local scheduling system supports event notification. However, note that it does not specify any event types nor define interfaces to query supported event types.

4. REQUESTING RESOURCES

Local scheduling systems do not only differ in the type of functionality they provide but also in the type of information they need when resources are requested. This information is characterized by the second set of attributes.

4.1 Allocation Offers

Local management systems may support the generation of potential resource allocations on request. In our example, several workstation clusters with different characteristics may be available for Stage 1. A Grid scheduler can determine the best suited cluster to use by considering different offers returned from the corresponding local schedulers, especially with regards to the location of the database and the available bandwidth on the network. To this end, the local scheduling instance may offer different resource allocations to the Grid scheduler.

Local management systems can further be classified according to the number of offers they generate for a request. For instance, local management systems may provide several offers for a request with possibly overlapping allocations. It is then up to the higher-level scheduler to select a suitable offer from them. In our example, the local scheduling system may generate different offers that differ in price or execution time. The Grid scheduler may use this feature for the multi-site allocation in our example in Stage 3 where corresponding allocations must be found for different resources. If several allocation offers are provided, the Grid scheduler has more flexibility to select a suitable offer to allow for an efficient and concurrent execution of a job.

4.2 Allocation Cost or Objective Information

The local management system returns cost or objective information for an allocation. In case of several allocation offers, a Grid scheduler can, for instance, use this information for the evaluation of different offers. The cost for a specified allocation usually relates to the policy that is applied by the lower-level scheduling instance. This represents the scheduling objective of the owner of the resource. It is obvious that costs for an allocation will be an important criterion for a Grid scheduler in selecting a suitable allocation for a job.

4.3 Advance Reservation

This feature is of great help for any Grid scheduler that must schedule multi-stage or multi-site jobs. The Grid scheduler in the example can ensure that the network connection between visualization cave and workstation cluster is sufficient in Stage 3 of our example by obtaining a reservation for appropriate bandwidth.

An Advance Reservation API is already subject of a document from the Global Grid Forum [RS02]. With such an interface a higher-level scheduling instance is able to access the Quality of Service features of Grid resources.

4.4 Requirement for Providing Maximum Allocation Length in Advance

As already mentioned, some local management systems require that a maximum allocation length is provided together with the resource request. In our example, the application in Stage 2 does not require any user interaction and is therefore a batch job. For this type of job, an efficient scheduler needs information about the maximum execution length. Historically, resource requests have often been submitted without additional information about the amount of time that the resources would be used. These jobs are started and run until completion. However, current scheduling algorithms, for example backfilling [Lif96], require additional information on the maximum allocation length in order to generate an efficient schedule.

4.5 Deallocation Policy

A deallocation policy for pending allocations applies to some local management systems. Such systems establish requirements that must be met by the user or the higher-level scheduling instance to keep the allocation valid. The requirement that an allocation must be repeatedly confirmed until the start of its execution is an example of such a policy. For instance, in our example the visualization cave might require complying to such a policy to ensure

a high degree of utilization of the cave itself. Of course, these policies must be further specified to allow the higher-level scheduler to keep its allocations. Attribute 5.1: Revocation of an Allocation, in comparison, describes the reliability of an allocation as given by the local management system.

4.6 Remote Co-Scheduling

A local management system may allow for co-scheduling where the actual resource allocation and the schedule are generated by a higher-level scheduling instance. In this scenario, the local management system provides interfaces to the higher-level scheduling instance to delegate certain modifications to the local schedule. This includes the generation and cancellation of allocations on the local schedule by the higher-level scheduler. In our example, the Grid scheduler can use this property to quickly co-allocate resources for the multi-site allocation in Stage 3. In this case, some part of the authority of a lower-level scheduling instance is delegated to the higher-level scheduling instance.

4.7 Consideration of Job Dependencies

A local scheduler may take dependencies between allocations into account if they are provided by the higher-level scheduling instance. For instance, in case of a complex job request, the lower-level scheduler will not start an allocation if the completion of another allocation is required and still pending. These dependencies are sometimes also referred to as the workflow of a job. For Grid jobs, different steps often depend on each other. In our example, the dependencies of Stage 1 and 2 may be considered by a local management system as precedence relations.

Note that the complete workflow graph of the job may not be initially available to the Grid scheduler since the results of some job steps may influence the following steps. In this case, some of the tasks of a higher-level scheduling instance are delegated and managed by local management systems.

5. ALLOCATION PROPERTIES

Allocations of resources may differ with respect to timing and reliability. For instance, a reliable allocation is important for multi-stage and multi-site jobs. Therefore, we use a third set of attributes to describe allocation properties.

5.1 Revocation of an Allocation

Some resource owners may assign a low priority for Grid use. In such a case local management systems may reserve the right to revoke an existing allocation until the resources are actually being used. Here we address only a revo-

cation of an allocation that cannot be prevented by actions of the higher-level scheduling instance. Therefore, the revocation is independent of any process that must be executed by the user system or a higher-level scheduling instance to prevent deallocation according to the deallocation policy of a local scheduling system, see Attribute 4.5: Deallocation Policy. For our example, the cluster in Stage 3 may be withdrawn from Grid use or may be used to execute another job with a higher priority. In this case, the allocation was not guaranteed. Note that a local management system may support both revocable and irrevocable allocations.

5.2 Guaranteed Completion Time of Allocations

Some local management systems guarantee the completion time of an allocation. While the system reserves the right to individually determine the actual start and end time of an allocation within a given time frame, it guarantees that the requested resource allocation will be executed before a given deadline. For instance, scheduling systems that are based on the backfilling strategy [Lif96, FW98] can provide information on the maximum completion time of a newly submitted resource request while not being able to predict the actual starting time. In those cases it is necessary for the user to provide information in advance on the maximum job run-time, see also Attribute 4.4: Maximum Allocation Length. With this information, an upper bound for the completion time of the jobs can be calculated by the local scheduling system. In our example, the completion time of Stage 2 is available to the Grid scheduler and used to coordinate the start of Stage 3.

5.3 Guaranteed Number of Attempts to Complete a Job

Some tasks, for instance, the transfer of data over a network link, cannot always be completed on the first try. In those situations it is useful if the local management system guarantees a minimum number of attempts before informing the user of the failure or requiring a re-scheduling from the higher-level scheduling system. This reduces the work of the higher-level scheduling instance. In our example, this feature can be used in Stage 1 to ensure that data has been successfully transferred to the workstations cluster.

5.4 Allocations Run-to-Completion

The local management system will not preempt, stop, or halt a job after it has been started. In particular, once started, the allocation will stay active on the given resources until the end of the requested allocation time frame or the completion of the job. This information about the way a job is executed helps a higher-level scheduling instance to more reliably predict the completion time of a running job, and is therefore useful for the coordination of concurrent

allocations on different resources. In our example, the Grid scheduler selects a workstation cluster with this property for Stage 3 to ensure the uninterrupted execution of the job concurrently with the allocation of the visualization cave.

5.5 Exclusive Allocations

The allocation runs exclusively on the provided set of resources, that is, the resources are not time-shared, and this allocation is not affected by the execution and resource consumption of another allocation running concurrently. Similar to the previous attributes, this information may be helpful to estimate the required run-time of a job on a specific resource. In a time-shared scenario, the run-time of a job may significantly deviate from an estimate due to the interference of other jobs on the same resource. In our example, the efficient use of the visualization device depends on the reliable performance of the workstation cluster in Stage 3. Therefore, the Grid scheduler can require that the workstation cluster is exclusively available during the allocation in this stage.

5.6 Malleable Allocations

Some local management systems support the addition or removal of resources to/from applications during run time [FRS+97]. In this case, the size of an allocation can change during the execution of a job. If such a modification of the allocation is not controlled by the higher-level scheduling instance it has a similar impact on the reliability of the estimated run-time of a job as time-sharing of resources. In addition not all applications may be able to handle a reduced allocation size. Therefore, those resources are not suitable for all job requests.

This problem is less severe if the local management can only increase the resource set of an allocation during run time, that is, if resources are not taken from a job. Feitelson et al. [FRS+97] describe such an allocation by the term *moldable*.

6. MANIPULATING THE ALLOCATION EXECUTION

For multi-stage or multi-site Grid jobs, unexpected changes of an allocation may influence other allocations and require quick re-scheduling. In this case it may be beneficial if the higher-level scheduling instance can directly modify other running allocations in order to better coordinate the execution of the jobs. The fourth set of attributes describes actions that a higher-level scheduling instance may directly initiate on resources without involving the local management system.

6.1 Preemption

Some local management systems allow for temporary preemption of an allocation by a higher-level scheduling instance. In this case, the corresponding application is stopped but remains resident on the allocated resources and can be resumed at a later time [WFP+96]. Such preemption is not synonymous with the preemption in a multitasking system that typically happens in the time range of milliseconds. It indicates only that the local management system offers the ability to remotely initiate a preemption of an allocation, for example to temporarily free resources for other use or to synchronize two allocations on different resources. Moreover, this kind of preemption does not necessarily require checkpointing. For instance, if machines of the allocated resource set go down, it is not guaranteed that the preempted allocation can be resumed on other resources.

6.2 Checkpointing

Other local management systems support the checkpointing of a job. In this case, a checkpoint file of the job is generated to allow for a later continuation of the job from the checkpoint. The generation of the checkpoint file can be independent of any actual preemption of the job. The checkpoint file may also be migratable to other resources, but this feature is not mandatory. Note that different types of checkpointing paradigms exist, and on some systems not all kinds of jobs can be checkpointed. Here, the application must cooperate and support the checkpointing feature.

6.3 Migration

Some local management systems support the migration of an application or part of an application from one resource set to another set. Hence, an application can be stopped at one location and the corresponding data is packed such that the application can be moved to another location and be restarted there. This migration process is only of relevance to a higher-level scheduling instance if it can initialize and control this process. Therefore, the attribute does not include any migration of allocations within the domain of the lower-level scheduling instances that is not influenced by the higher-level scheduling instance, see Attribute 5.6: Malleable Allocations. Also, the migration attribute does not necessarily require the presence of the Attribute 6.2: Checkpointing. However, similar to the remark on checkpointing, explicit support for migration by the application may be required to use this feature.

6.4 Restart

Here, the local management system supports the receiving and the restart of a stopped and packaged application from another resource set.

On some systems, the continuation of a job may be supported as so called Checkpoint Restart. Then, a restart is only possible from a checkpoint file, see Attribute 6.2: Checkpointing. That is, the system does not support migration on the fly.

7. CONCLUSION

Ideally, computational Grids consist of many different resources but look like a single system from a user's point of view. A similar structure can also be found in a Grid scheduler, an important component of any computational Grid. This has led to the concept of a layered scheduler consisting of higher-level scheduling instances that are close to the user and lower-level scheduling instances that are close to the resources. Due to the heterogeneity of available local scheduling systems that form the lowest layer of a Grid scheduler, information about the properties of those systems must be provided to the higher-level scheduling instances. This is done with the help of attributes which are presented in this chapter. Those attributes may also be helpful for the future development of local scheduling systems for Grid resources. Ideally, the existence of an attribute should not be determined by the features of a local scheduling system but by properties of the resource or by system policies.

Some chapters in the book address actual implementations of resource management systems with Grid support. More specifically, the MAUI/Silver scheduler is described in Chapter 11, Platform's LSF is presented in Chapter 12, and the Portable Batch System (PBS) can be found in Chapter 13. These systems represent the current state of the art for resource management systems that can be found in today's computational Grids.

Acknowledgments

To a large degree this chapter describes the results of discussions in the Scheduling Attributes Working Group of the Global Grid Forum, see also [SY01]. The authors would like to thank all contributors to this effort.

Chapter 5

SECURITY ISSUES OF GRID RESOURCE MANAGEMENT

Mary R. Thompson and Keith R. Jackson
Lawrence Berkeley National Laboratory

Abstract Secure management of Grid resources presents many challenges. This chapter will examine the security requirements that are essential to Grids and some of the software that is available to meet them. We will discuss how well these security tools have been utilized and review some of the existing and proposed security standards that may be the foundations of the next generation of Grid security tools.

1. INTRODUCTION

Resource management in Computational and Data Grid environments offers many security challenges. Some of these challenges exist in many computing environments, but others are unique to Grid systems. The goal of this chapter is to highlight the security issues of providing and using resources on a Grid; to itemize the current state of the art for handling Grid security and to suggest future directions for Grid security middleware.

We will begin by offering a brief overview of the main security requirements for resource management on the Grid. Most of these arise from the geographically and organizationally distributed nature of Grids and the types of resources that are being referenced. This will be followed by a brief introduction to the basic concepts of cryptography as they are applied in a Grid environment.

We will then look in detail at the differing requirements and solutions for authentication, authorization and trust relationships. Enforcement of authorization policies and auditing of resource use will be covered more briefly. We will conclude the chapter by examining some of the open issues in Grid security.

2. BACKGROUND

2.1 Unique Grid Requirements

Many of the unique problems in Grid security arise because of the distributed nature of Grid systems. A Grid offers uniform access to resources that may be widely distributed geographically and organizationally. These different organizations may have radically different security policies that must be reconciled to allow for the coordinated usage of resources across these sites.

A Grid will contain a variety of resources each of which may have different security requirements. The most common types of resources on the Grid are computational and storage resources. High-end computational resources require a high-level of investment, and their usage is tightly controlled. Administrators may wish to control the use of CPU cycles on the machine, disk space usage, load factors, network bandwidth to and from the machine, etc. (see Chapter 18). There are also issues about who can access or control a job after it has started running. Any security system must allow for the propagation of identity information for use in accounting and auditing. Storage resources may simply be disks attached to a compute host or may be stand-alone storage repositories that are remotely accessible from many hosts. Controlling storage devices involves the control of the raw storage space as well as the data stored in it.

Another type of Grid resource is a scientific instrument. Today, more and more scientific instruments are being shared over the network. A high-energy light source such as the Advanced Light Source at LBNL, or a high-end electron microscope, is a very expensive and unique instrument. Most universities and research facilities are unable to own such resources on their own. In the past, researchers would travel to the location of the instrument to perform their experiments, and then return to their home institutions for data analysis. Today this cycle is accelerated by providing remote access to these one-of-a-kind instruments. A sample may be sent to the instrument location for preparation by a local technician. From that point, it can be controlled on the instrument by the investigator from his or her home institution. Clearly for these expensive instruments it is important to be able to enforce fine-grained control over the actions that a user might perform. In particular, it should not be possible for a remote user to damage the instrument.

The Grid also differs from other environments, in that most applications require the coordinated usage of multiple resources at multiple sites. Even the simplest Grid usage scenarios require several resources. For example, a simple climate modeling simulation will require at least the usage of storage resources for input and output data sets, network resources to transfer the data, and compute resources to run the simulation. These resources must be coordinated in order to allow the job to complete. It is not sufficient to have time on the su-

per computer if the researcher cannot move needed data from the mass storage system to the computer. This kind of tight coupling of resources may also require the ability to make advanced reservations for these resources so as to ensure their availability. This is especially true for uses that involve a high-end scientific instrument, which are typically scheduled months in advance. See Chapter 3 for a more elaborate scientific use case which illustrates the need for reserved resources and treating executing jobs as resource to which controlled access is needed.

Grid users as well as resources come from many different real and *Virtual Organizations* (VO) [FKT01]. A Grid user may be part of more than one VO, and have different access rights depending on the role they are performing.

2.2 Cryptography Overview

There are several important concepts from cryptography that are necessary to understand before attempting to understand the current solutions for distributed security. The first concept is a *message digest*. A message digest is also known as a *hash function* or *message authentication code* and is used to create a unique *fingerprint* of arbitrary bit streams. Another concept is *encryption*. An encryption algorithm takes some data and transforms it so that the original data cannot be discovered by anyone looking at the encrypted data. Any encryption algorithm must use some type of key so that only the intended recipient can decrypt the data. There are two main types of encryption keys: *symmetric* and *asymmetric*.

A message digest is a one-way function that takes an arbitrary length input and produces a fixed length output with certain properties. The fact that this is a one-way function means it is nearly impossible to take the output from a message digest and calculate what the input was. Another important property of hash functions is collision resistance. This means that for a given input to a cryptographic hash it should be computationally hard to find another input that will hash to the same output. Common examples of message digests are MD5 [Riv92] or SHA1 [EJ01].

In symmetric encryption a single key is used to both encrypt and decrypt the data. Symmetric encryption algorithms take as input a key and an arbitrary length bit stream and produce an arbitrary length bit stream as output. A good encryption algorithm should produce output that makes it very hard to calculate what the original input was. For example, an algorithm like AES [Sta01] used properly with sufficient keying material would take longer then the known lifetime of the universe to break with all of the computing power on the planet today. One obvious problem with symmetric key encryption is that both parties must have the same keying material. This presents the problem of how both parties securely acquire this keying material.

Asymmetric cryptography offers one solution to this problem. In asymmetric cryptography a pair of keys are used. One key is called the public key and can be publicly available. The other is the private key and must be kept secret by its owner. This key pair has the important property that information encrypted with one key can only be decrypted with the other key. Thus, the sender of a message can find the public key for the person they wish to communicate with from a public source like a web server or directory server. The sender then uses the recipient's public key to encrypt a message that only the recipient can decrypt using the corresponding private key. One important problem arises when doing this. How do you know the key you found on the web server belongs to the intended recipient? Different types of *Public Key Infrastructures* or *PKI*'s have solved this problem in a variety of ways. One common solution is to have a trusted third party attest to the fact that this key belongs to the intended recipient. Typically, this third party is called a *Certification Authority* or *CA*. Several other solutions to this problem will be discussed later in the chapter.

3. AUTHENTICATION

Cryptographic primitives are of interest to Grid resource management because they can be used to enable two important operations: authentication and authorization. These operations allow a Grid resource to identify who is requesting access to the resource and if they are allowed to use the resource.

Authenticating a request is the usually first step in getting access to a Grid resource. Authentication mechanisms in Grids tend to differ from single system authentication because of the need to support both Grid-wide and local identities, the need to authenticate a user of delegated credentials and the relatively recent development of public key cryptography. This section will examine the distinctive needs of a Grid and the technology to satisfy them.

3.1 What is Authentication and Why Do It?

Authentication is the process of proving your identity to someone. In the real world this is most commonly done through the possession of a photographic ID issued by a government body, e.g., a passport or drivers license. Similarly in the computer world, proof of possession of some cryptographic key can be used to prove your identity to a remote resource.

Authentication in a computer environment is the process of associating a real-world identity with a request to a machine. An underlying assumption of authentication is that a unique machine-readable *unique-id* will always be assigned to the same unique real world person or computer. Authentication takes place when the connecting entity provides a credential containing such a unique-id and the authenticating agent can verify that the credential legiti-

mately represents the connecting entity. This is normally done by requiring the connecting entity to prove knowledge of a secret that only the owner of the credential knows. In the case of a remote connection the authentication agent needs to secure the communication channel once the remote entity has been verified so that no one but the authenticated entity can use it. This is usually done by creating a symmetric session key that is known by both ends of the authenticated connection and used in all subsequent communications. At this point there may be a local user id and/or running job associated with the channel. From then on, all actions performed as a result of requests coming from that channel are assumed to be done by the authenticated entity. In a Grid environment there are several types of entities that need to be authenticated. The most common are individual persons consuming resources, usually referred to as a user or client, and hosts providing resource and services. Sometimes roles, such a system administrator, or organization or accounts are thought of as entities in their own right and can be the acting entity.

There are two important Grid security functions enabled by authentication. One is *authorization*, which is the checking of what actions are allowed for this user id. The second is *accountability*, which is accomplished by auditing the actions taken on a host by jobs running under each user id. Authorization is the most common reason for needing authenticated connections. If individual auditing is not required group or role based authentication may be sufficient.

This rest of this section covers the types of credentials commonly used in distributed computing environments, the challenges in using these credentials and some new approaches to credential management that may alleviate some of the existing problems.

3.2 Types of Credentials

The common underlying bases for cryptographically useful credentials are either a user name and shared secret such as a password, or a user name and public/private key pair. Shared secrets must be known by the target resource in advance of a user's attempt to authenticate. When public/private keys are used for authentication, the resource only needs to know the public key of the user. Only the user has possession of the private key. With group or role credentials the private key is shared among all the members of the group, contributing to the belief that such credentials are less trustworthy.

The Kerberos [MNSS87] authentication server was an early development to facilitate authentication in a network of hosts based on symmetric key encryption. It defined a new type of credential and a distribution scheme for them. It is based on Roger Needham and Michael Schroeder's authentication and key exchange protocol published in 1978 [NS78]. Each entity (host or user) in a Kerberos domain has a shared secret with the Kerberos server. There is then

a three-way protocol between the user, the resource and the Key Distribution Server (KDC) which proves to the target host that the user is who she claims to be. The Kerberos tokens that are passed across the network can be viewed as a new sort of credential, but they are based on the user and target passwords.

Public/private key cryptography, introduced by Whitfield Diffie and Martin Hellman in 1976 [DH77], offers an alternative to shared secret keys. In authentication protocols based on public/private keys, each entity's public key is shared with the resource targets to which it is authenticating. In the authentication protocol, the target host challenges the connecting entity to decrypt a one-time phrase with the user's private key, thus proving that it possesses the private key and is in fact the user that it claims to be.

The most common credentials used in Grids are *X.509 certificates*. These certificates bind a multi-component meaningful name, called a *Distinguished Name* (*DN*), to a public key and are signed by a CA. X.509 certificates were introduced in 1989 as part of the ISO Directory-Authenticating Framework [CCI88]. Since then they have been adopted as an IETF standard [HFPS02]. Most production CAs will guarantee that the DN legitimately represents the individual who requested the certificate and will issue certificates containing the same public key and DN to only one individual. Thus an X.509 certificate used with a challenge protocol, such as *Secure Socket Layer* (SSL) [FKK96], [DR99], can prove that the parties on both ends of a communication channel are uniquely identified by a public key, and the names in the certificates legitimately describe the entities.

Grids based on Globus [FK97, GLO] middleware use X.509 certificates to identify users and a slightly extended version of SSL, called the *Grid Security Infrastructure* (GSI) [FKTT98], to authenticate connections.

There was considerable discussion in the Grid Forum's Certificate Policy working group [GCP] as to the recommended content of a DN. While strictly random unique names would satisfy the cryptographic needs, it was decided that meaningful, unique names assigned to unique individuals would be useful in audit logs and cross site authorization policies.

So far all the Grid CAs have rejected the idea of issuing certificates that would be shared between groups of users or for services that run on more than one host. The rationale behind this policy is that strict accountability outweighs the convenience of shared credentials.

One of the essential requirements of Grid computing is the ability of a user to delegate some or all of his rights to programs running on his behalf. The level of risk involved of handing a long-term password or private key to a program tends to invalidate that credential. Thus, it is necessary to create a *delegated credential* with some substantial restrictions to hand off to programs. Grid authentication and authorization mechanisms must be able to recognize a

delegated certificate, authenticate the presenter of that certificate and trace its privileges back to the original delegator.

So far only simple delegation mechanisms have been widely used. The Kerberos TGT (Ticket Granting Ticket) is a short lived ($\sim$24hrs) credential to grant access to the holder. The GSI *proxy certificate* is a short-lived ($\sim$12hrs) X.509 certificate with its own unencrypted private key signed by the delegator. These proxies can be used to perform any operation the delegator is allowed to perform. Currently the GSI originators have proposed a new delegated credential format that uses an extension to restrict the rights of the credential [TEF+02].

3.3 Key Distribution

Both secret and public key authentication requires an infrastructure for distributing keys. In distributing public keys the only security issue is to ensure that the resource target has the correct public key for a user. In the case of secret keys, such as passwords, it is essential that no one other than trusted parties learn the key. Passwords were originally distributed on a one to one basis between the owner and a resource, where they needed to be stored securely. If the same password is used for many resources, a password compromise on one could compromise all resources. Neither having different passwords for each resource, nor sharing the same passwords with many resources, scales well. The Kerberos authentication system was developed to solve this problem.

A variety of schemes exist to distribute public keys and to associate them with user names. Typically, there is some sort of object that contains a user name and a public key that is distributed as a public credential. An X.509 certificate is such a credential signed by a CA. Thus if a relying party has the public key of the CA which it has retrieved and stored in a secure manner, it can verify an X.509 certificate that it gets through an untrusted channel, for example from the user who is trying to be authenticated. This reduces the secure distribution of public keys to just distributing the trusted CAs' public keys.

One of the major advantages of PKI is that it does not require an on-line authentication server to distribute keys in real time. In a Grid environment where the users and resource servers are more transitory and widely dispersed than within a single Enterprise, it is problematical to keep up a network of secure connections between the users, the resources and a key distribution server. This is one of the reasons that PKI is used more frequently than Kerberos in Grid environments. One of the major steps in building production Grids has been the establishment of high quality CAs dedicated to supporting one or more scientific Grids. These are currently run by organizations contracted to support the Grid [EDGb, DOEa].

3.4 Credential Management

An issue that has surfaced with the wider use of public/private key credentials is how private keys can both be safely stored and conveniently used. Creation of a short-term proxy is part of the solution as it allows a user to sign-on once per day using a passphrase to unlock the long-term credential and then allows agent programs to use the short-term unencrypted key for each subsequent authentication operation. There still remains the problem of users choosing a strong credential passphrase. Since these keys will be stored on local workstations, which are often very insecure, there is a reasonable risk that the private key file could be stolen, at which point the passphrase could be subject to a dictionary attack.

Another problem that arises with private keys is that a user may want to make a Grid access from a machine on which the private key is not stored. There are some solutions to these problems such as keeping a private key on a pin-protected smart card or storing proxy or real certificates on a secure server. Another possible solution is to use the Kerberos CA [KHDC01] server to issue a short lived PKI credential based on your Kerberos credential.

3.5 Other Distributed Authentication Solutions

There are several recent distributed service infrastructures that have introduced on-line authentication servers to facilitate single-sign with a single password to many hosts. One is the .NET Web Services infrastructure from Microsoft that uses the Passport [PAS] server, another is Liberty Alliance's [LAP] specification for federated authentication servers, and a third is the Internet2 middleware Shibboleth server [CE]. Each of these on-line trusted authentication servers supply a set of attributes about a user to a recognized resource. These attributes may or may not include an official name for the user. The user authenticates to the authentication server and then passes an authentication handle to the resource. That resource can then use the handle to query the authentication server to discover if the user has the attributes required to grant access. These models allow for single-sign on to a group of co-operating hosts using a password and allow the users to determine what information to give to a target host. They do require an on-line trusted authentication server.

In the evolving Open Grid Services Architecture [FKNT02, WSF+03], the intent is to support more than one type of identity credential. The pieces of the middleware that are responsible for the higher level uses of authentication such as trusted connections and resource access authorization should be able to authenticate users with either X.509 certificates, Kerberos tickets or possibly some handle to a trusted authentication server.

4. AUTHORIZATION

Authorization is the process of deciding if a person is allowed to perform some operation. For example, at my bank I would first use my drivers license to prove my identity and then the bank would match that identity to the given account number to ensure that I am allowed to remove the money. Similarly a Grid resource such as a supercomputer will have policy that says who is allowed to run what jobs on the machine.

This section will describe Grid authorization requirements; review some of the authorization mechanisms currently in use by Grid middleware and point to emerging standards for authorization.

4.1 Requirements

Grid applications that require access to many resources, which may be in a different organizational domains, and may have different stakeholders, present several authorization challenges. The resource provider may need to enforce access defined by policy from multiple stakeholders. The stakeholder needs to identify the people (in different domains) that are allowed to use the resource. The policy for a resource may be set by multiple stakeholders, for example, the owner of an application, and the administrator for the host on which it is running. A resource scheduler needs to determine the user's access to the resources that the job will use, but the exact resources may not be known in advance. As the job progresses, a process running in one domain may need to make a controlled access to resources in a different domain. Some of the requirements these scenarios impose on an authorization framework are:

- Multiple stakeholders need to see and set access policy for resources provided by remote servers.
- Access policy at various resources needs to have a common way to identify users, roles and other attributes.
- Grid resource providers and stakeholders would like to be able to group individual users by their access rights, so that access policies can be simpler and less subject to change.
- Various agents (e.g., schedulers, policy enforcement points) need to be able to determine a user's rights to a resource.

In addition to authorization requirements there are also some policy tensions that are specific to Grids. The one that has had a severe impact on current Grid implementations is between complete local control over compute resources and Grid-wide authorization policies to allow users seamless use of Grid resources. In experimental and developing Grids there is conflict between actually getting work done on the Grid and securely protecting all of the resources.

As long as a Grid is used by a small number of trusted and authenticated users, fine-grained access to resources can be self-policed. Finally there is always tension in access control systems between policies that allow fine grained access control based on a rich set of user and system attributes, and an efficient and understandable access control mechanism.

4.2 State of the Art

This section will examine some of the current Grid middleware tools to discover what authorization mechanisms are supported today. First, we introduce some terminology common to authorization models that will be used in this section and the next. A client requests access to a resource through a *Policy Enforcement Point* (PEP), which may call out to a *Policy Decision Point* (PDP) sometimes called an *authorization service*. *Policy statements* describe the access policies for resources and are defined by the resource *stakeholders* or a *Source of Authority* (SOA) for the resource domain. *Attributes* describe characteristics of users that can be used as the basis for granting actions on a resource. They are attested to by *Attribute Authorities* (AA). A *credential* is a token that can be used to prove an identity or possession of an attribute. A *certificate* is a digitally signed and dated document that can be verified by a *relying party*.

4.2.1 Grid Middleware

The Globus [FK97, GLO], Condor [CON] and Legion [FKH+99] projects provide much of the Grid middleware used by the scientific community. Each of these systems has been introduced in previous chapters in this book. In this section we summarize and compare the authorization mechanisms used by these systems.

The resources in a Condor pool are the workstation cycles and the stakeholders are the "owners" of the workstation. The stakeholder decides when and to whom the workstation will be available as part of a Condor pool, e.g,. certain times of day, when system is idle, and limited to certain sets of users. These policies are included in the *ClassAd* that a workstation owner provides when the workstation is added to the pool. The Condor scheduler daemon will enforce the policy before allowing a job to start running on a workstation. The job on the workstation is run in a sandboxed environment where critical system calls are trapped and redirected back to the job owner's home machine, thus protecting the filesystem on the pool machine and providing the user job with a consistent environment across all the pool machines.

The original Condor implementation took advantage of the fact that the machines in a Condor pool were all Unix systems running within the same institution. Thus they could use the local user ids to identify people, could often share

a common AFS or NFS file system, or perform remote system calls that were needed in sandboxing a job. However, since then, Condor has implemented flocking by mapping user ids across organization boundaries and integrating with Globus to leverage off its global ids and secure connections. One of the strengths of the Condor system is to provide the local workstation owner with strong control over when the workstation could be used and how quickly it could be reclaimed.

In the Globus Toolkit 2 (GT2) implementation the emphasis is also on local control by the sites that are providing the resources. A gridmap-file maps all the authorized users' Grid Ids (an X.509 distinguished name) to a local user id. Since the current implementation of the Gatekeeper allows an authorized user to run any binary on the system or upload a binary to run, being a Globus user is the equivalent of having a login account on the host. Currently, Globus jobs are not sandboxed in any manner other than running under the userid that was found in the gridmap-file for the authorized user. Thus in most systems the site administrator requires each Globus user to have his own account.

There have been several experimental modifications to allow either the gatekeeper [CCO+03] or job-manager [KW02] to have finer-grained access control over what the user can do. It is only when the Globus user can be strictly limited, that system administrators are likely to relax the requirement that each user have his own login id and Globus can hope to provide the desired "run-anywhere" goal of Grid computing.

Legion represents all entities: users, processing resources, storage, as Legion objects. The names of each Legion object (its LOID) contains its public key, so that two objects can authenticate to each other by connecting and encrypting a session key with the target public key and the client's private key. The target party can get the other entity's public key from its name and use it and it's own private key to decrypt the key. Getting access in a Legion system is done by calling a method on an object. Each Legion object goes through a "mayI" level to check if the requested access is allowed before executing the method. MayI can be implemented in multiple ways. The default implementation that comes as part of the Legion runtime library implements access control lists and credential checking. Access control lists can be inherited from the user on whose behalf the objects are created or can be added by the Class Manager for the particular kinds of objects that it manages. Legion allows for very powerful per object, per method access control based on a user's Grid credentials. This level of access control is also supported by CORBA and Java, but it often proves hard to group the access policies in ways to make them both meaningful and manageable.

4.2.2 Grid Authorization Services

The Community Authorization Server [PWF+02] is a recent addition to Globus Toolkit 2. It is based on the premise that Grid users are often members of a Virtual Organization and that those organizations may be the most logical place to set Grid access use policy. The underlying premise is that a resource provider would grant blocks of resources to a VO, and then the VO would redistribute access among its own members. When a user wants to use a resource, he will first go to the CAS server and request to be granted the rights that are needed to use the resource. If the CAS has been granted those rights by the resource, and if the CAS policy states that this user may have these rights, the CAS will sign an assertion that the user has these rights. The user then includes this rights assertion in the proxy certificate used to connect to the resource server. The resource gatekeeper can verify that the certificate belongs to the user, and that the CAS server has authorized his use of the resource. The resource server may also need to locally confirm that the CAS server was granted these rights. Currently the CAS server has been used to control GridFTP access [ABB+02b] and may be used in the future by the gatekeeper to verify the rights of a user to run particular executables. Akenti is an authorization server/library under development by Lawrence Berkeley National Laboratory since 1997 [TMEC02]. It implements policy based access control where the policy can be written by multiple stakeholders and stored in distributed digitally signed XML documents. Akenti (a Policy Decision Point) provides a simple interface to a resource server (a Policy Enforcement Point) that takes a user identity (DN or X.509 certificate), the name of a resource, and the actions that are requested and returns accept, deny or a conditional response. The response is in the form of a signed capability certificate (a signed authorization assertion) that may include a list of runtime conditions (such as disk availability or load factor) that the resource server must evaluate. Akenti has been used in several collaboratories [Fus, PRY99] to control access to Web pages, job execution in a CORBA derived job framework, and with the Globus job-manager to do fine-grained access of job execution and control.

VOMS is a Virtual Organization Management Server that provides an authorization service similar to the CAS model [VOM]. It is currently under development within the European DataGrid. The policy information is stored in a database accessible to the VOMS server that issues Globus proxy certificates that include attribute information. The policy enforcement points must be able to interpret this access information.

4.3 Authorization Related Standards

Most of the more sophisticated Grid authorization schemes are not widely used. Instead, Grid resource providers have chosen to use the Globus gridmap-

file mechanism, some other local map file, or to write some ad-hoc scheme tailored to their specific site. Also, unlike the authentication standards where X.509 certificates have been accepted as a standard, none of the proposed authorization standards, such as the IETF PKIX [pki] Policy Management Infrastructure (PMI) and the related X.509 Attribute Certificates [FH02], Simple Public Key Infrastructure [Ell99], Policy Maker and Keynote, [BFIK99] or GAA [RN99] have had wide acceptance. One of the reasons for this lack of accepted standards is that a general solution to authorization is a very complicated problem, while a local authorization mechanism at a single resource site can be quite simple. While this may initially satisfy the resource provider, it usually does not scale well to many transient remote users. It also does not scale well for users that are trying to use a variety of Grid sites.

The Grid community is currently addressing this challenge by defining a framework and language to express authorization policy, queries and decisions along with the supporting elements of such as attribute assertions, trust policy and management. One of the assumptions of this effort is that Grid middleware will be evolving to the Grid services model based on developing industry Web Services standards [FKNT02].

The Web Services [W3C, IM02] community is currently developing standards for expressing the entire range of security issues in distributed systems. As part of this effort, various groups are developing standard XML vocabularies for expressing authorization decisions, policy and queries. SAML, *Security Assertion Markup Language* [HBM02], and XACML, *eXtensible Access Control Markup Language* [Ope02], are two related XML schemas proposed as standards by the OASIS consortium [OAS]. SAML focuses on queries and replies about security related assertions. XACML is designed to express authorization policy.

When robust, open source Grid middleware is using a standard authorization and trust vocabulary and authorization servers have been implemented that can interpret this vocabulary, authorization will be at the point that authentication was when the first open source implementation of SSL and then GSI was released. This is the point when we are likely to see standard Grid resource authorization mechanisms developed.

5. TRUST RELATIONSHIPS

Trust relationships define what entities may be trusted to set policy or authorize actions. In traditional single host or single enterprise systems, trust is often implicitly granted on the basis of where the information is found. For example, the /etc/passwd and /etc/shadow files are the trusted source of user identification on Unix systems. A slightly more sophisticated trust document is the ypservers file that defines the host which runs the trusted NIS server, which defines all the authorized users. File access policy is held in the access control lists associated with the files. Access to such files and meta-data is tightly controlled, so the practice is to trust whatever information is found there.

On the other hand, in the real world, organizations usually have many policy documents that explicitly state what departments, positions or individuals have responsibility for what actions. These policies may include who is allowed to delegate responsibilities and limits on the people to whom rights can be delegated.

A cross-organizational Grid has the same need to explicitly state what entities have the rights to perform actions. It is often preferable to make such declarations in terms of VOs and roles rather than individuals in order to scale to many users and privileges.

Some of the common trust statements used in distributed systems are: list of trusted CAs; list of AAs who may issue certificates attesting to possession of specified attributes; list of stakeholders for a resource who may issue access policy for the resource; identification of authorization servers who are trusted to make authorization decisions; and identification of VO servers which define which users are a members of a VO. In a PKI these trusted entities are usually identified by their public keys, so that any signatures generated by them can be cryptographically verified.

The first step that a resource provider must take in putting resources on the Grid is to designate the trusted parties for authentication and authorization. If he chooses a system where all policy and authorization information must reside on files at his site, it will not scale well. Currently Grids specify the CAs who will be trusted to sign public key certificates and possibly VO servers who can define roles or privileges for its users. Systems such as Akenti also explicitly state who can sign Attribute Certificates and use policies. XACML policy statements also include who can sign for attributes.

6. POLICY ENFORCEMENT

Policy enforcement is an area where practice is well short of what is desired. Many of the more sophisticated distributed authorization systems have been used mainly by application resource servers that directly control all access to a resource, for example, GridFTP servers or Web servers. In these examples

all actions on a resource can be mediated by the resource server, which in turn may call out to an authorization server.

The common Grid scenario of running code on a remote host is harder to effectively police. Enforcing the name (or even the hash code) of the code to be executed is easy enough, but once binary code is executing only the operating system has control of its actions. Most operating systems have very limited control over executing code, based solely on user and/or group id. If the code is written in Java a finer-grain level of control is possible. It is also possible to trap system calls as Condor does and enforce policy at the level of system calls. While there have been a variety of experimental operating systems and extensions to Unix systems that allow more control over executing jobs, most operating systems today do not implement these. Since Grid middleware strives to connect many heterogeneous systems, it is disinclined to take advantage of system specific policy enforcement mechanisms.

This does not mean that authorization policy cannot be written that specifies fine-grain control, but only that the policy must be cognizant of what the target systems can enforce.

7. AUDITING REQUIREMENTS

Auditing the use of Grid resources is required by both the resource provider and the users of the resource. The resource provider needs a record of who is using its resources both for charging purposes and to discover any misuse. Thus, there must be accurate records of the amount and dates of usage linked to the person (or at least account number) who used it.

In a Grid environment where there are service commitments by resource providers to projects or VOs, the accounting record may also be important to prove to the various projects that each one got its fair share of the resource. This level of auditing may be accomplished by simply keeping a log of the time jobs were started, the Grid id of the person starting the job, how much CPU was used and the time they ended. File storage typically keeps the userid of the user that created a file along with the file sizes. In a Grid it may be necessary to map the local userids back to Grid ids for auditing purposes. The information in these logs should be available to the user whose information it is, but will usually need to be protected from other users.

More detailed logging of long-running job execution is desired to allow remote users to monitor the progress of their jobs. This data needs to be available in real-time and on a controlled basis to the remote user or people that he has designated. There should be enough information in these logs to allow a remote user to know what has happened if his job failed to complete successfully.

Finally, some users will have the need for intermediate results from long running jobs, perhaps in order to steer the computation or to abort a computation that is going in the wrong direction.

The Grid middleware for job-management needs to be responsible for logging job control information, but only the application program can log intermediate results. In the case of Globus, both the gatekeeper and job-manager log a level of information determined by configuration files. The queuing job-manager keeps some information about the state of the job that the remote user can query via the job handle. Other log information is mostly only available by logging into the server machine and reading log files.

The method of obtaining intermediate results in real-time is up to the application. Various event management systems have be designed to help with this task. One of the future challenges of Grid middleware is make implementations of such tools easily usable by Grid applications.

8. CONCLUSION

In this chapter we have looked at some of the most common security issues arising in Grid resource management. We have attempted to identify those issues that are unique to a Grid environment, and examine the current state of the art in solving them. We have seen the need for emerging security standards for middleware that controls access to Grid resources. Although working solutions exist to many of the security issues examined, others still require research. We do not yet have adequate ways to support multiple levels of authentication where the usage of X.509 certificates is too heavyweight. Nor are we able to easily integrate different methods of authentication.

The larger open issues are related to authorization. There is a tension between the ease of use of an authorization system and the level of granularity at which authorization decisions can be made. It is also an open question as to what level of authorization is best done by the VOs and what should be done by the resource providers. Simple forms of authorization exist in the Grid today, but are not sufficient to capture complex resource policy. Current efforts at solving this problem often end up creating systems that are too complex to be widely accepted.

Current research in Grid security is focusing on defining common ways of expressing such diverse items as message security (without tying it to a particular transport protocol), authentication (while supporting multiple types of authentication schemes), trust policies, authorization policies and assertions about these items. Once these standards have been established, standards compliant tools need to be written to implement required functionality. At that point, writers of new Grid applications should be able to choose security policy and authorization components as easily as they can choose tools like SSL

secure messaging or Kerberos authentication today. Meanwhile, writers of Grid applications should try to localize actions on resources that require authorization and whenever possible provide callout hooks at these sites. If the current efforts at standardization succeed, the whole landscape of Grid security functionality will change.

Acknowledgments

This work was supported in part by U.S. Department of Energy, Office of Science, Office of Advanced Science, Mathematical, Information and Computation Sciences under contract DE-AC03-76SF00098.

II

RESOURCE MANAGEMENT IN SUPPORT OF COLLABORATIONS

Chapter 6

SCHEDULING IN THE GRID APPLICATION DEVELOPMENT SOFTWARE PROJECT

Holly Dail,[1] Otto Sievert,[2] Fran Berman,[1,2] Henri Casanova,[1,2] Asim YarKhan,[3] Sathish Vadhiyar,[3] Jack Dongarra,[3] Chuang Liu,[4] Lingyun Yang,[4] Dave Angulo,[4] and Ian Foster[4,5]

[1]*San Diego Supercomputer Center, University of California, San Diego*
[2]*Department of Computer Science and Engineering, University of California, San Diego*
[3]*Department of Computer Science, University of Tennessee*
[4]*Department of Computer Science, The University of Chicago*
[5]*Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract

Developing Grid applications is a challenging endeavor that at the moment requires both extensive labor and expertise. The Grid Application Development Software Project (GrADS) provides a system to simplify Grid application development. This system incorporates tools at all stages of the application development and execution cycle. In this chapter we focus on application scheduling, and present the three scheduling approaches developed in GrADS: development of an initial application schedule (launch-time scheduling), modification of the execution platform during execution (rescheduling), and negotiation between multiple applications in the system (metascheduling). These approaches have been developed and evaluated for platforms that consist of distributed networks of shared workstations, and applied to real-world parallel applications.

1. INTRODUCTION

Computational Grids can only succeed as attractive everyday computing platforms if users can run applications in a straightforward manner. In these collections of disparate and dispersed resources, the sum (the Grid) must be greater than the parts (the machines and networks). Since Grids are typically

built on top of existing resources in an ad-hoc manner, software layers between the user and the operating system provide useful abstraction and aggregation.

In the past decade, many useful software products have been developed that simplify usage of the Grid; these are usually categorized as *middleware*. Successful middleware efforts have included, for example, tools for collecting and disseminating information about Grid resources (e.g., Network Weather Service [WSH99a]), all-encompassing meta operating systems that provide a unified view of the Grid (e.g., Legion [GFKH99]), and collections of relatively independent Grid tools (e.g., the Globus Toolkit [FK98a]). Middleware has simplified Grid access and usage, allowing the Grid to be useful to a wider range of scientists and engineers.

In order to build application-generic services, middleware developers have generally not incorporated application-specific considerations in decisions. Instead, application developers are expected to make any decisions that require application knowledge; examples include data staging, scheduling, launching the application, and monitoring application progress for failures. In this environment Grid application development remains a time-consuming process requiring significant expertise.

The Grid Application Development Software Project (GrADS) [BCC+01, KMMC+02] is developing an ambitious alternative: replace the discrete, user-controlled stages of preparing and executing a Grid application with an end-to-end software-controlled process. The project seeks to provide tools that enable the user to focus only on high-level application design without sacrificing application performance. This is achieved by incorporating application characteristics and requirements in decisions about the application's Grid execution. The GrADS architecture incorporates user problem solving environments, Grid compilers, schedulers, performance contracts, performance monitors, and reschedulers into a seamless tool for application development. GrADS builds on existing middleware and tools to provide a higher-level of service oriented to the needs of the application.

Scheduling, the matching of application requirements and available resources, is a key aspect of GrADS. This chapter focuses on scheduling needs identified for the GrADS framework. Specifically, we discuss the following distinct scheduling phases.

- **Launch-time scheduling** is the pre-execution determination of an initial matching of application requirements and available resources.
- **Rescheduling** involves making modifications to that initial matching in response to dynamic system or application changes.
- **Meta-scheduling** involves the coordination of schedules for multiple applications running on the same Grid at once.

Initial prototype development in GrADS revealed that existing application scheduling solutions were inadequate for direct application in the GrADS system. System-level schedulers (e.g., LoadLeveler [IBM01] and Maui Scheduler [Mau]) focus on throughput and generally do not consider application requirements in scheduling decisions. Application-specific schedulers [Ber99] have been very successful for individual applications, but are not easily applied to new applications. Other projects have focused on easy-to-characterize classes of applications, particularly master-worker applications [AGK00, COBW00]; these solutions are often difficult to use with new classes of applications.

Consequently, we worked to develop scheduling strategies appropriate to the needs of GrADS. We began by studying specific applications or application classes and developing scheduling approaches appropriate to those applications or application classes [PBD+01, AAF+01, DBC03, SC03, VD03b, VD02]. Concurrently, the GrADS team has been developing GrADSoft, a shared implementation of GrADS. Proven research results in all areas of the GrADS project are incorporated in GrADSoft with appropriate modifications to allow integration with other components. The resulting system provides an end-to-end validation of the GrADS research efforts. Many of the scheduling approaches discussed in this chapter have been incorporated in GrADSoft [MMCS+01].

This chapter is organized as follows. Section 2 describes the GrADS project in greater detail and provides an overview of key Grid technologies we draw upon for our scheduling work. Section 3 describes the applications we have used to validate our designs. Section 4 presents our launch-time scheduling approach, Section 5 presents our rescheduling approach, and Section 6 presents our metascheduling approach. Section 7 concludes the chapter.

2. GRADS

Figure 6.1 provides a high-level view of the GrADS system architecture [KMMC+02]. This architecture provides the framework for our scheduling approaches. The following is a conceptual view of the function of each component in the architecture. Note that the view given in Figure 6.1 and in the following discussion is an idealized view of how GrADS components should function and interact with one another. In reality, the system is under development and does not yet function exactly as described. This chapter will describe the current state of the scheduling components, but we will not have space to describe the current status of all GrADS components.

The *Program Preparation System* (PPS) handles application development, composition, and compilation. To begin the development process, the user interacts with a high-level interface called a problem solving environment (PSE)

to assemble their Grid application source code and integrate it with software libraries. The resulting application is passed to a specialized GrADS compiler. The compiler performs application analysis and partial compilation into intermediate representation code and generates a configurable object program (COP). This work may be performed off-line as the COP is a long-lived object that may be re-used for multiple runs of the application. The COP must encapsulate all results of the PPS phase for later usager, for example, application performance models and partially compiled code, also konwn as intermediate representation code.

The *Program Execution System* (PES) provides on-line resource discovery, scheduling, binding, application performance monitoring, and rescheduling. To execute an application, the user submits parameters of the problem (such as problem size) to the GrADS system. The application COP is retrieved and the PES is invoked. At this stage the scheduler interacts with the Grid run-time system to determine which resources are available and what performance can be expected of those resources. The scheduler then uses application requirements specified in the COP to select an application-appropriate schedule (resource subset and a mapping of the problem data or tasks onto those resources).

The binder is then invoked to perform a final, resource-specific compilation of the intermediate representation code. Next, the executable is launched on the selected Grid resources and a real-time performance monitor is used to track program performance and detect violation of performance guarantees. Performance guarantees are formalized in a performance contract [VAMR01]. In the case of a performance contract violation, the rescheduler is invoked to evaluate alternative schedules.

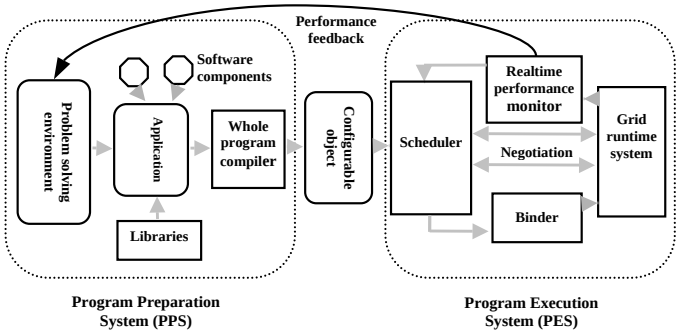


Figure 6.1. Grid Application Development Software Architecture.

2.1 Grid Technology

The previous section outlined the GrADS vision; as components of that vision are prototyped and validated, they are incorporated in a shared prototype implementation of GrADS called GrADSoft. Many of the scheduling approaches described in this chapter have been integrated into this prototype. We utilize the following Grid technologies to support our research efforts.

The Globus Toolkit Monitoring and Discovery Service (MDS2) [CFFK01] and the Network Weather Service (NWS) [WSH99a] are two on-line services that collect and store Grid information that may be of interest to tools such as schedulers. The MDS2 is a Grid information management system that is used to collect and publish system configuration, capability, and status information. Examples of the information that can typically be retrieved from an MDS server include operating system, processor type and speed, and number of CPUs available. The NWS is a distributed monitoring system designed to track and forecast dynamic resource conditions; Chapter 14 describes this system in detail. Examples of the information that can typically be retrieved from an NWS server include the fraction of CPU available to a newly started process, the amount of memory that is currently unused, and the bandwidth with which data can be sent to a remote host. Our scheduling approaches draw on both the MDS and NWS to discover properties of the Grid environment.

The ClassAds/Matchmaking approach to scheduling [RLS99] was pioneered in the Condor high-throughput computing system [LLM88]. Refer to Chapter 9 for a discussion of Condor and Chapter 17 for a discussion of ClassAds. *ClassAds* (i.e., Classified Advertisements) are records specified in text files that allow resource owners to describe the resources they own and resource consumers to describe the resources they need. The ClassAds/Matchmaking formalism includes three parts: The *ClassAds language* defines the syntax for participants to create ClassAds. The *advertising protocol* defines basic conventions regarding how ClassAds and matches must be communicated among participants. The *matchmaker* finds matching requests and resource descriptions, and notifies the advertiser of the result. Unfortunately, at the time of our research efforts the ClassAds/Matchmaker framework only considered a single resource request at a time. As will be described in Section 4.3.1, our work has extended the ClassAds/Matchmaker framework to allow scheduling of parallel applications on multiple resources.

To provide focus and congruency among different GrADS research efforts we restricted our initial efforts to particular execution scenarios. Specifically, we focus on applications parallelized in the Message Passing Interface (MPI) [For94]. This choice was based on both (1) the popularity and ease of use of MPI for the implementation of parallel scientific applications and (2) the recent availability of MPICH-G2 [KTF03] for execution of MPI applica-

tions in the wide-area. We use the Globus Toolkit [FK98a] for authentication and job launching and GridFTP [ABB+02a] for data transfers.

3. FOCUS APPLICATIONS

The GrADS vision to build a general Grid application development system is an ambitious one. To provide context for our initial research efforts, we selected a number of real applications as initial focus points. Our three primary selections were ScaLAPACK, Cactus, and FASTA; each of these applications is of significant interest to a scientific research community and has non-trivial characteristics from a scheduling perspective. Additionally, we selected three iterative applications (Jacobi, Game of Life, and Fish) for use in rapid development and testing of new scheduling techniques. These applications were incorporated into GrADS by the following efforts: ScaLAPACK [PBD+01], Cactus [AAF+01], Jacobi [DBC03], Game of Life [DBC03], and Fish [SC03]. The FASTA effort has not been described in a publication, but was led by Asim YarKhan and Jack Dongarra. Our focus applications include implementations in Fortran, C, and C++.

3.1 ScaLAPACK

ScaLAPACK [BCC+97] is a popular software package for parallel linear algebra, including the solution of linear systems based on LU and QR factorizations and the determination of eigenvalues. It is written in a single program multiple data (SPMD) style and is portable to any computer that supports MPI or PVM. Linear solvers such as those provided in ScaLAPACK are ubiquitous components of scientific applications across diverse disciplines.

As a focus application for GrADS, we chose the ScaLAPACK right-looking LU factorization code based on 1-D block cyclic data distribution; the application is implemented in Fortran with a C wrapper. Performance prediction for this code is challenging because of data-dependent and iteration-dependent computational requirements.

3.2 Cactus

Cactus [ABH+99] was originally developed as a framework for finding numerical solutions to Einstein's equations and has since evolved into a general-purpose, open source, problem solving environment that provides a unified, modular, and parallel computational framework for scientists and engineers. The name Cactus comes from the design of central core (or "flesh") which connects to application modules (or "thorns") through an extensible interface. Thorns can implement custom applications, as well as computational capabilities (e.g. parallel I/O or checkpointing). See Chapter 3 for an extended discussion of Cactus.

We focus on a Cactus code for the simulation of the 3D scalar field produced by two orbiting sources; this application is implemented in C. The solution is found by finite differencing a hyperbolic partial differential equation for the scalar field. This application decomposes the 3D scalar field over processors and places an overlap region on each processor. In each iteration, each processor updates its local grid points and then shares boundary values with neighbors. Scheduling for Cactus presents challenging workload decomposition issues to accommodate heterogeneous networks.

3.3 FASTA

In bio-informatics, the search for similarity between protein or nucleic acid sequences is a basic and important operation. The most exacting search methods are based on dynamic programming techniques and tend to be very computationally expensive. FASTA [PL88] is a sequence alignment technique that uses heuristics for fast searches. Despite these optimizations, due to the current size of the sequence databases and the rate at which they are growing (e.g. human genome database), searching remains a time consuming process. Given the size of the sequence databases, it is often undesirable to transport and replicate all databases at all compute sites in a distributed Grid.

The GrADS project has adapted a FASTA sequence alignment implementation [FAS] to use remote, distributed databases that are partially replicated on some of the Grid nodes. When databases are located at more than one worker, workers may be assigned only a portion of a database. The application is structured as a master-worker and is implemented in C. FASTA provides an interesting scheduling challenge due to the database locality requirements and large computational requirements.

3.4 Iterative Applications

We selected three simple iterative applications to support rapid development and testing of new scheduling approaches. We chose these applications because they are representative of many important science and engineering codes and they are significant test cases for a scheduler because they include various and significant computation, communication, and memory usages. All three applications are implemented in C and support non-uniform data distribution.

The *Jacobi method* is a simple linear system solver. A portion of the unknown vector x is assigned to each processor; during each iteration, every processor computes new results for its portion of x and then broadcasts its updated portion of x to every other processor.

Conway's *Game of Life* is a well-known binary cellular automaton [Fla98]. A two-dimensional mesh of pixels is used to represent the environment, and each pixel of the mesh represents a cell. In each iteration, the state of every cell is updated with a 9-point stencil and then processors send data from their edges (ghost cells) to their neighbors in the mesh.

The *Fish* application models the behavior and interactions of fish and is indicative of many particle physics applications. calculates Van der Waals forces between particles in a two-dimensional field. Each computing process is responsible for a number of particles, which move about the field. Because the amount of computation depends on the location and proximity of particles, this application exhibits a dynamic amount of work per processor.

4. LAUNCH-TIME SCHEDULING

The launch-time scheduler is called just before application launch to determine how the current application execution should be mapped to available Grid resources. The resulting *schedule* specifies the list of target machines, the mapping of virtual application processes to those machines, and the mapping of application data to processes. Launch-time scheduling is a central component in GrADSoft and is key to achieving application performance; for this reason, we have experimented with a variety of launch-time scheduling approaches including those described in [LYFA02, DBC03, PBD+01]. Through this evolution we have identified an application-generic launch-time scheduling architecture; in this section we describe that architecture and key contributions we have made to each component of the design. Many of the approaches described here have been incorporated in GrADSoft; the resulting unified system has been tested with all of the applications described in Section 3.

4.1 Architecture

Figure 6.2 shows our launch-time scheduling architecture. This architecture is designed to work in concert with many other GrADS components as part of the general GrADS architecture described in Section 2. Launch-time scheduling proceeds as follows; we describe each component in detail in the next section. A user submits a list of machines to be considered for this application execution (the machine list). The list is passed to the GrADS information services client; this component contacts the MDS2 and the NWS to retrieve information about Grid resources. The resulting Grid information is passed to the filter resources component. This component retrieves the application requirements and filters out resources that can not be used for the application. The remaining Grid information is then passed to the search procedure. This

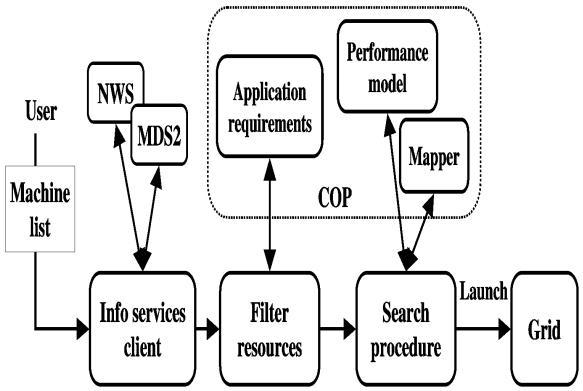


Figure 6.2. GrADS launch-time scheduling architecture.

procedure searches through available resources to find application-appropriate subsets; this process is aided by the application performance model and mapper. The search procedure eventually selects a final schedule, which is then used to launch the application on the selected Grid resources.

4.2 GrADS Information Services Client

To enable adaptation to the dynamic behavior of Grid environments, Grid resource information must be collected in real time and automatically incorporated in scheduling decisions. The MDS2 and NWS, described in Section 2.1, collect and store a variety of Grid resource information. The GrADS information services client retrieves resource information of interest to the scheduler and stores that information in a useful format for the scheduler.

4.3 Configurable Object Program

As described in Section 2, the output of the GrADS program preparation system is a configurable object program, or COP. The scheduler obtains the application requirements specification, the performance model, and the mapper from the COP. As the GrADS program preparation system matures, the COP will be automatically generated via specialized compilers. In the meantime, we use various approaches, described below, to develop hand-built COP components for our focus applications. These hand-built components have been useful in studying scheduling techniques and have provided guidance as to what sorts of models and information should be automatically generated by the PPS in the future.

4.3.1 Application Requirements Specification

A reasonable schedule must satisfy application resource requirements. For example, schedules typically must include sufficient local memory to store the entire local data set. See Chapter 3 for an additional discussion of specifications of application resources requirements.

We have explored two ways to specify application resource requirements. The first is a primarily internal software object called an abstract application resource and topology (AART) model [KMMC+02]. Ultimately, the GrADS compiler will automatically generate the AART; however, we do not yet have an automated GrADS compiler in GrADSoft. Currently, application developers must provide a specification of the application requirements before utilizing GrADSoft for a new application. For this purpose, we have developed a second approach based on ClassAds (see Section 2.1). With ClassAds, resource requirements can be easily specified in a human-readable text file. Furthermore, some developers are already familiar with ClassAds semantics.

The original ClassAds language is not sufficient for parallel applications. To support parallel application description, we developed a number of extensions to ClassAds.

- A *Type* specifier is supplied for identifying set-extended ClassAds. The expression *Type="Set"* identifies a set-extended ClassAd.
- Three aggregation functions, *Max*, *Min*, and *Sum*, are provided to specify aggregate properties of resource sets. For example,

$$Sum(other.memory) > 5Gb \tag{6.1}$$

  specifies that the total memory of the set of resources selected should be greater than 5 Gb.
- A boolean function *suffix(V, L)* returns *True* if a member of list *L* is the suffix of scalar value *V*.
- A function *SetSize* returns the number of elements within the current resource ClassAd set.

Our extensions, called *Set-extended ClassAds*, are described in full in [LYFA02].

Our usage of ClassAds in GrADSoft is illustrated by the resource request for the Cactus application shown in Figure 6.3. The parameters of this resource request are based on performance models developed for Cactus in [RIF01b]. In this example, we use

$$Sum(other.MemorySize) >= (1.751 + 0.0000138 * z * x * y) \tag{6.2}$$

to express an aggregate memory requirement, and

$$suffix(other.machine, domains) \tag{6.3}$$

[
type = "*Set*";
service = "*SynService*";
iter = "100"; α = 100; x = 100; y = 100; z = 100;
domains = {"*cs.utk.edu*", "*ucsd.edu*"};
requirements = $Sum(other.MemorySize) \geq$
$(1.757 + 0.0000138 \times x \times y \times z)$ &&
$suffix(other.machine, domains)$;
computetime = $x \times y \times \alpha / other.cpuspeed \times 370$;
comtime = $other.RLatency + x \times y \times 254 / other.Lbandwidth$;
exectime = $(computetime + comtime) \times iter + startup$;
mapper = [*type* = "*dll*"; *libraryname* = "*cactus*";
function = "*mapper*"];
rank = $Min(1/exectime)$;
]

Figure 6.3. Example of a set-extended ClassAds resource requirements specification.

to constrain the resources considered to those within domain *cs.utk.edu* or *ucsd.edu*. The execution progress of the Cactus application is generally constrained by the subtask on the slowest processor; we therefore specify $rank = Min(1/exectime)$ so that the rank of a resource set is decided by the longest subtask execution time. The $exectime$, $computetime$, and $comtime$ entries define the performance model for Cactus.

To better express the requirements of a variety of Grid applications, we recently developed a new description language, called *Redline* [LF03], based on Set-extended ClassAds. This language is able to describe resource requirements of SPMD applications, such as Cactus, as well as MIMD applications where different application components may require different types of resources. For example, with Redline one can specify unique requirements for a storage resource, computation resources, and for the network connecting storage and computation resources. We use this added expressiveness to better satisfy application requirements in the scheduling process.

4.3.2 Performance Modeling

The performance model is used by the launch-time scheduler to predict some metric of application performance for a given schedule. For ease of discussion we assume the metric of interest is application execution time, but one could easily consider turnaround time or throughput instead. The performance

model object takes as input a proposed schedule and the relevant Grid information and returns the predicted execution time for that schedule.

Other GrADS researchers are developing approaches for automatically generating performance models in the GrADS compiler [Grab]. In lieu of such technologies, we have experimented with several methods for developing and specifying performance models in GrADS. We generally used the following approach: (i) develop an analytic model for well-understood aspects of application or system performance (e.g. computation time prediction is relatively straightforward), (ii) test the analytic model against achieved application performance in real-world experiments, and (iii) develop empirical models for poorly-understood aspects of application or system behavior (e.g. middleware overheads for authentication and startup may be difficult to predict). We used this approach for the iterative focus applications [DBC03], Cactus [RIF01b], and FASTA [PL88].

For the ScaLAPACK application, we pursued a different approach [PBD+01]. The ScaLAPACK LU factorization application is an iterative application consisting of three important phases: the factorization phase involves computation, the broadcast phase involves communication, and the update phase consists of both computation and communication. As the application progresses, the amount of computation and communication in each phase and in the iteration as a whole varies. Analytical models based on simple mathematical formulas can not provide accurate predictions of the cost of the ScaLAPACK application on heterogeneous systems. Instead, we developed a simulation model of the application that simulates the different phases of the application for a given set of resources. The resulting application simulation functions as an effective performance model.

We implemented most of our performance models as shared libraries; GrADSoft dynamically loads the application-appropriate model as needed. This approach allows application-specific performance model code to be used by our application-generic software infrastructure. Using ClassAds, GrADSoft also supports an easier to build alternative: the performance model can be specified as an equation within the ClassAd (see Figure 6.3). While some performance dependencies can not be described in this ClassAds format, we believe this approach will be very attractive to users due to its ease-of-use.

4.4 Mapper

The mapper is used to find an appropriate mapping of application data and/or tasks to a particular set of resources. Specifically, the mapper takes as input a proposed list of machines and relevant Grid information and outputs a mapping of virtual application processes to processors (ordering) and a mapping of application data to processes (data allocation).

Some applications allow the user to specify machine ordering and data allocation; examples include FASTA, Cactus, and our three iterative applications. For these applications, a well-designed mapper can leverage this flexibility to improve application performance. Other applications define ordering and data allocation internally, such as ScaLAPACK. For these applications, a mapper is unnecessary.

4.4.1 Equal Allocation

The most straightforward mapping strategy takes the target machine list and allocates data evenly to those machines without reordering. This is the default GrADSoft mapper and can easily be applied to new applications for which a more sophisticated strategy has not been developed.

4.4.2 Time Balancing

The Game of Life and Jacobi are representative of applications whose overall performance is constrained by the performance of the slowest processor; this behavior is common in many applications with synchronous inter-processor communication. In [DBC03] we describe our time balance mapper design for the Game of Life and Jacobi in detail; we briefly describe this design here.

The goal of the ordering process is to reduce communication costs. We reorder machines such that machines from the same site are adjacent in the topology. Since the Game of Life is decomposed in strips and involves neighbor-based communication, this ordering minimizes wide-area communications and reduces communication costs. For Jacobi there is no clear effect since its communication pattern is all-to-all.

The goal of the data allocation process is to properly load the machines such that all machines complete their work in each iteration at the same time; this minimizes idle processor time and improves application performance. To achieve this goal we frame application memory requirements and execution time balance considerations as a series of constraints; these constraints form a constrained optimization problem that we solve with a linear programming solver [LP]. To use a real-valued solver we had to approximate work allocation as a real-valued problem. Given that maintaining a low scheduling overhead is key, we believe that the improved efficiency obtained with a real-valued solution method justifies the error incurred in making such an approximation.

4.4.3 Data Locality

FASTA is representative of applications with data locality constraints: reference sequence databases are large enough that it may be more desirable to co-locate computation with the existing databases than to move the databases from machine to machine. The mapper must therefore ensure that all reference

databases are searched completely and balance load on machines to reduce execution time.

As with the time balance mapper, a linear approximation was made to the more complicated nonlinear FASTA performance model; we again used [LP], but here the result is an allocation of portions of reference databases to compute nodes. Constraints were specified to ensure that all the databases were fully handled, that each compute node could hold its local data in memory, and that execution time was as balanced as possible. The load balancing is complicated by the fact that the databases are not fully distributed. For example, a given database may only be available at one compute node, requiring that it be computed there regardless of the execution time across the rest of the compute nodes.

4.5 Search Procedure

The function of the search procedure is to identify which subset of available resources should be used for the application. To be applicable to a variety of applications, the search procedure must not contain application-specific assumptions. Without such assumptions, it is difficult to predict a priori which resource set will provide the best performance for the application. We have experimented with a variety of approaches and in each case the search involves the following general steps: (i) identify a large number of sets of resources that may be good platforms for the application, (ii) use the application-specific mapper and performance model to generate a data map and predicted execution time for those resource sets, and (iii) select the resource set that results in the lowest predicted execution time.

This *minimum execution-time multiprocessor scheduling problem* is known to be NP-hard in its general form and in most restricted forms. Since launch-time scheduling delays execution of the application, maintaining low overhead is a key goal. We have developed two general search approaches that maintain reasonable overheads while providing reasonable search coverage: a resource-aware search and a simulated annealing search.

4.5.1 Resource-Aware Search

Many Grid applications share a general affinity for certain resource set characteristics. For example, most Grid applications will perform better with higher bandwidth networks and faster processing speeds. For our resource-aware search approach, we broadly characterize these general application affinities for particular resource characteristics; we then focus the search process on resource sets with these broad characteristics.

Over the evolution of the project we have experimented with three search approaches that fall in this category: one developed for the ScaLAPACK appli-

```
ResourceAware_Search:
sites ← FindSites( machList )
siteCollections ← ComputeSiteCollections( sites )
foreach (collection ∈ siteCollections )
  foreach (machineMetric ∈ (computation, memory, dual) )
    for (r ← 1:size(collection) )
      list ← SortDescending( collection, machineMetric )
      CMG ← GetFirstN( list, r )
      currSched ← GenerateSchedule( CMG,
            Mapper, PerfModel )
      if (currSched.predTime < bestSched.predTime)
        bestSched ← currSched
return (bestSched)
```

Figure 6.4. A search procedure that utilizes general application resource affinities to focus the search.

cation [PBD+01], one developed as part of the set-extended ClassAds framework [LYFA02], and one developed for the GrADSoft framework [DBC03]. We describe the last search procedure as a representative of this search type.

The goal of the search process is to identify groups of machines that contain desirable individual machines (i.e., fast CPUs and large local memories) and have desirable characteristics as an aggregate (i.e., low-delay networks); we term these groups of machines candidate machine groups (CMGs). Pseudo-code for the search procedure is given in Figure 6.4. In the first step, machines are divided into disjoint subsets, or sites. Currently, we group machines in the same site if they share the same domain name; we plan to improve this approach so that sites define collections of machines connected by low-delay networks. The *ComputeSiteCollections* method computes the power set of the set of sites. Next, in each *for* loop the search focus is refined based on a different characteristic: connectivity in the outer-most loop, computational and/or memory capacity of individual machines in the second loop, and selection of an appropriate resource set size in the inner-most loop. The *SortDescending* function sorts the current *collection* by *machineMetric*. For example, if *machineMetric* is *dual*, the *sortDescending* function will favor machines with large local memories *and* fast CPUs. *GetFirstN* simply returns the first *r* machines from the sorted *list*. Next, to evaluate each CMG, the *GenerateSchedule* method (i) uses the *Mapper* to develop a data mapping for the input *CMG*, (ii) uses the *Performance model* to predict the execution time for the given schedule (*predtime*), and (iii) returns a schedule structure which contains the CMG, the map, and the predicted time. Finally, predicted execution time is used to select the best schedule, which is then returned.

The complexity of an exhaustive search is 2^p, where p is the number of processors. The complexity of the above search is $3p2^s$ where s is the number of sites. As long as $s << p$, this search reduces the search space as compared to an exhaustive search. We have performed in-depth evaluation of this search procedure for Jacobi and Game of Life [DBC03], and validated it for Cactus, FASTA, and ScaLAPACK.

4.5.2 Simulated Annealing

Our resource-aware search assumes general application resource affinities. For applications which do not share those affinities, the resource-aware search may not identify good schedules. For example, FASTA does not involve significant inter-process communication and so does not benefit from the resource-aware procedure's focus on connectivity. Instead, for FASTA it is important that computation be co-scheduled with existing databases. For this type of application, we have developed a simulated annealing search approach that does not make assumptions about the application, but is a costlier search approach.

Simulated annealing [KGJV83] is a popular method for statistically finding the global optimum for multivariate functions. The concept originates from the way in which crystalline structures are brought to more ordered states by an *annealing process* of repeated heating and slowly cooling the structures. Figure 6.5 shows our adaptation of the standard simulated annealing algorithm to support scheduling. For clarity we have omitted various simple heuristics used to avoid unnecessary searches.

In our procedure, we perform a simulated annealing search for each possible resource subset size, from 1 to the full resource set size. This allows us to decrease the search space by including only machines that meet the memory requirements of the application, and to avoid some of the discontinuities that may occur in the search space. For example, when two local minima consist of different machine sizes, the search space between them is often far worse than either (or may not even be an eligible solution).

For each possible machine size r we create a filtered list of machines that can meet resource requirements. The initial CMG (candidate machine group) is created by randomly selecting r machines out of this list. A schedule is created from the CMG using the *GenerateSchedule* method, which provides the predicted execution time of the schedule.

At each temperature T, we sample the search space repeatedly to ensure good coverage of the space. To generate a new CMG, the current CMG is perturbed by adding one machine, removing one machine, and swapping their ordering. The new CMG is evaluated using *GenerateSchedule* and the difference between the predicted execution time and that of the current CMG is calculated as dE. If the predicted execution time is smaller ($dE < 0$), then the new CMG becomes the current CMG. If $dE > 0$ then we check if the new

```
SimulatedAnnealing_Search:
for r ← 1:size(machList)
  filteredList ← check local memory
  if size(filteredList) < r, continue
  currCMG ← pick r machines randomly from filteredList
  currSched ← GenerateSchedule(currCMG, Mapper, PerfModel)
  foreach temperature ← max:min
    while energy has not stabilized
      % remove machine, add machine and swap
order
      newCMG ← randomly perturb currCMG
      newSched ← GenerateSchedule( newCMG,
            Mapper, PerfModel )
      if newSched.predTime < currSched.predTime
        currCMG ← newCMG
      else if random number < exp(-dE/kT)
        % Probabilistically allow increases in
energy
        currCMG ← newCMG
      if currSched.predTime < bestSched.predTime
        bestSched ← currSched
return bestSched
```

Figure 6.5. A search procedure that uses simulated annealing to find a good schedule without making application assumptions.

CMG should be probabilistically accepted. If a random number is less than the Boltzmann factor $exp(-dE/T)$, then accept the new CMG, else reject the new CMG. Using this distribution causes many upward moves to be accepted when the temperature is high, but few upward moves to be accepted when the temperature is low. After the space has been sampled at one temperature, the temperature is lowered by a constant factor α ($T \leftarrow \alpha T, \alpha < 1$). This search procedure allows the system to move to lower energy states while still escaping local minima. The schedule with the best predicted execution time is returned. We have performed a validation and evaluation of this search procedure for the ScaLAPACK application in [YD02].

5. RESCHEDULING

Launch-time scheduling can at best start the application with a good schedule. Over time, other applications may introduce load in the system or application requirements may change. To sustain good performance for longer

running applications, the schedule may need to be modified during application execution. This process, called *rescheduling*, can include changing the machines on which the application is executing (*migration*) or changing the mapping of data and/or processes to those machines (*dynamic load balancing*).

Rescheduling involves a number of complexities not seen in launch-time scheduling. First, while nearly all parallel applications support some form of launch-time scheduling (selection of machines at a minimum), very few applications have built-in mechanisms to support migration or dynamic load balancing. Second, for resource monitoring we have to differentiate between processors on which the application is running and processors on which the application is not running. Measurements from resource monitors such as NWS CPU sensors can not be directly compared between active and inactive processors. Third, the overheads of rescheduling can be high: monitoring for the need to reschedule is an ongoing process and, when a rescheduling event is initiated, migration of application processes or reallocation of data can be very expensive operations. Without careful design, rescheduling can in fact hurt application performance.

We have experimented with a variety of rescheduling approaches to address these issues, including an approach developed for the Cactus application [AAF+01], an approach called application migration that has been validated for the ScaLAPACK application [VD03b], and an approach called process swapping that has been validated for the Fish application [SC03]. Through this evolution we have identified a rescheduling architecture. Note that our rescheduling efforts were all conducted for iterative applications, allowing us to perform rescheduling decisions at each iteration. In the following sections we describe this architecture and key contributions we have made to each component of the design.

5.1 Architecture

Figure 6.6 shows our rescheduling architecture and we depict an application already executing on N processors. We also assume that a *performance contract* [VAMR01] has been provided by earlier stages of GrADS; this contract specifies the performance expected for the current application execution. While the application is executing, *application sensors* are co-located with application processes to monitor application progress. Progress can be measured, for example, by generic metrics such as flop rate or by application-intrinsic measures such as number of iterations completed. In order that these sensors have access to internal application state, we embed them in the application itself. We work to minimize the impact of these sensors on the application's footprint and execution progress. Meanwhile, *resource sensors* are located on

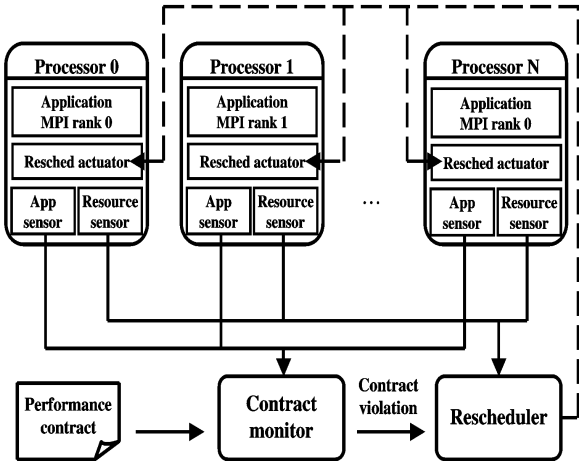


Figure 6.6. GrADS rescheduling architecture.

the machines on which the application is executing, as well as the other machines available to the user for rescheduling. For evaluating schedules, it is important that measurements taken on the application's current execution machines can be compared against measurements taken on unused machines in the testbed.

Application sensor data and the performance contract are passed to the *contract monitor*, which compares achieved application performance against expectations. When performance falls below expectations, the contract monitor signals a violation and contacts the rescheduler. An important aspect of a well-designed contract monitor is the ability to differentiate transient performance problems from longer-term issues that warrant modification of the application's execution. Upon a contract violation, the *rescheduler* must determine whether rescheduling is profitable, and if so, what new schedule should be used. Data from the resource sensors can be used to evaluate various schedules, but the rescheduler must also consider the cost of moving the application to a new execution schedule and the amount of work remaining in the application that can benefit from a new schedule.

To initiate schedule modifications, the rescheduler contacts the *rescheduling actuators* located on each processor. These actuators use some mechanism to initiate the actual migration or load balancing. Application support for migration or load balancing is the most important part of the rescheduling system. The most transparent migration solution would involve an external migrator that, without application knowledge, freezes execution, records important state such as register values and message queues, and restarts the execution on a new

processor. Unfortunately, this is not yet feasible as a general solution in heterogeneous environments. We have chosen application-level approaches as a feasible first step. Specifically, the application developer simply adds a few lines to their source code and then links their application against our MPI rescheduling libraries in addition to their regular MPI libraries.

5.2 Contract Monitoring

GrADS researchers have tried a variety of approaches for application sensors and contract monitoring. The application sensors are based on Autopilot sensor technology [RVSR98] and run in a separate thread within the application's process space. To use these sensors, a few calls must be inserted into the application code. The sensors can be configured to read and report the value of any application variable; a common metric is iteration number. Alternatively, the sensors can use PAPI [LDM+01] to access measurements from the processor's hardware performance counters (i.e., flop rate) and the MPICH profiling library for communication metrics (i.e., bytes sent per second).

The performance contracts specify predicted metric values and tolerance limits on the difference between actual and predicted metric values. For example, contracts can specify predicted execution times and tolerance limits on the ratio of actual execution times to predicted execution times. The contract monitor receives application sensor data and records the ratio of achieved to predicted execution times. When a given ratio is larger than the upper tolerance limit, the contract monitor calculates an average of recently computed ratios. If the average is also greater than the upper tolerance limit, the performance monitor signals a contract violation to the rescheduler. Use of an average reduces unnecessary reactions to transitory violations.

In the following two sections, we present two approaches to rescheduling for which contract monitoring provides a fundamental underpinning.

5.3 Rescheduling Via Application Migration

Application rescheduling can be implemented with application migration, based on a stop/restart approach. When a running application is signaled to migrate, all application processes checkpoint important data and shutdown. The rescheduled execution is launched by restarting the application, which then reads in the checkpointed data and continues mid-execution.

To provide application support for this scenario we have implemented a user-level checkpointing library called **S**top **R**estart **S**oftware (SRS) [VD03a]. To use SRS, the following application changes are required.

- `SRS_Init()` is placed after `MPI_Init()` and `SRS_Finish()` is placed before `MPI_Finalize()`.

- The user may define conditional statements to differentiate code executed in the initial startup (i.e., initialization of an array with zeros) and code executed on restart (i.e., initialization of array with values from checkpoint). SRS_Restart_Value() returns 0 on start and 1 on restart and should be used for these conditionals.

- The user should insert calls to `SRS_Check_Stop()` at reasonable stopping places in the application; this call checks whether an external component has requested a migration event since the last `SRS_Check_Stop()`.

- The user uses `SRS_Register()` in the application to register the variables that will be checkpointed by the SRS library. When an external component stops the application, the SRS library checkpoints only those variables that were registered through `SRS_Register()`. `SRS_Read()` is used on startup to read checkpointed data.

- `SRS_Read()` also supports storage of data distribution and number of processors used in the checkpoint. On restart, the SRS library can be provided with a new distribution and/or a different number of processors; the data redistribution is handled automatically by SRS.

- The user must include the SRS header file and link against the SRS library in addition to the standard MPI library.

At run-time, a daemon called *Runtime Support System* (RSS) is started on the user's machine. RSS exists for the entire duration of the application, regardless of migration events. The application interacts with the RSS during initialization, to check if the application needs to be stopped during `SRS_Check_Stop()`, to store and retrieve pointers to the checkpointed data, and to store the processor configuration and data distribution used by the application.

We use a modified version of the contract monitor presented in Section 5.2. Specifically, we added (i) support for contacting the migration rescheduler in case of a contract violation, (ii) interfaces to allow queries to the contract monitor for remaining application execution time, and (iii) support for modifying

the performance contract dynamically. For the third issue, when the rescheduler refuses to migrate the application, the contract monitor lowers expectations in the contract. Dynamically adjusting expectations reduces communication with the contract monitor.

The migration rescheduler operates in two modes: *migration on request* occurs when the contract monitor signals a violation, while in *opportunistic migration* the rescheduler checks for recently completed applications (see [VD03b] for details) and if one exists, the rescheduler checks if performance benefits can be obtained by migrating the application to the newly freed resources. In either case, the rescheduler contacts the NWS to obtain information about the Grid; our application migration approach does not involve special-purpose resource sensors. Next, the rescheduler predicts remaining execution time on the new resources, remaining execution time on the current resources, and the overhead for migration and determines if a migration is desirable. We have evaluated rescheduling based on application migration in [VD03b] for the ScaLAPACK application.

5.4 Rescheduling Via Process Swapping

Although very general-purpose and flexible, the stop/migrate/restart approach to rescheduling can be expensive: each migration event can involve large data transfers and restarting the application can incur expensive startup costs. Our process swapping approach [SC03] provides an alternative tradeoff: it is light-weight and easy to use, but less flexible than our migration approach.

To enable swapping, the MPI application is launched with more machines than will actually be used for the computation; some of these machines become part of the computation (the *active* set) while some do nothing initially (the *inactive* set). The user's application sees only the active processes in the main communicator (`MPI_Comm_World`); communication calls are hijacked and work is performed behind the scenes to convert the user's communication calls in terms of the active set to real communication calls in terms of the full process set. During execution, the system periodically checks the performance of the machines and swaps loaded machines in the active set with faster machines in the inactive set. This approach is very practical: it requires little application modification and provides an inexpensive fix for many performance problems. On the other hand, the approach is less flexible than migration: the processor pool is limited to the original set of machines and we have not incorporated support for modifying the data allocation.

To provide application support for process swapping we have implemented a user-level swapping library. To use swapping, the following application changes are required.

- The iteration variable must be registered using the `swap_register()` call.
- Other memory may be registered using `mpi_register()` if it is important that their contents be transferred when swapping processors.
- The user must insert a call to `MPI_Swap()` inside the iteration loop to exercise the swapping test and actuation routines.
- The user must include `mpi_swap.h` instead of `mpi.h` and must link against the swapping library in addition to the standard MPI library.

In addition to the swap library, swapping depends on a number of run-time services designed to minimize the overheads of rescheduling and the impact on the user. The *swap handler* fulfills the role of application sensor, resource sensor, and rescheduling actuator in Figure 6.6. The swap handler measures the amount of computation, communication, and barrier wait time of the application. The swap handler also measures passive machine information like the CPU load of the machine and active information requiring active probing like current flop rate.

The *swap manager* performs the function of the rescheduler in Figure 6.6. A swap manager is started for each application and is dedicated to the interests of that application only. Application and resource information is sent by the swap handlers to the swap manager. Acting in an opportunistic migration mode, the swap manager analyzes performance information and determines when and where to swap processes according to a swap policy. When a swap is necessary, the swap manager triggers the swap through the swap handlers.

When a swap request is received, the swap handler stores the information. The next time the application checks if swapping is necessary by calling `MPI_Swap()`, the swap is executed. From the application's perspective, the delay of checking for a swap request is minimal: the information is located on the same processor and was retrieved asynchronously.

We have evaluated our process swapping implementation with the Fish application and experimental results are available in [SC03].

6. METASCHEDULING

Our launch-time scheduling architecture (Figure 6.2) schedules one application at a time and, except for the usage of dynamic NWS data on resource availability, does not consider the presence of other applications in the system. There are a number of problems with this application-centric view. First, when two applications are submitted to GrADS at the same time, scheduling decisions will be made for each application ignoring the presence of the other. Second, if the launch-time scheduler determines there are not enough resources

for the application, it can not make further progress. Third, a long running job in the system can severely impact the performance of numerous new jobs entering the system. The root cause of these and other problems is the absence of a metascheduler that possesses global knowledge of all applications in the system and tries to balance the needs of the applications.

The goal of our metascheduling work [VD02] is to investigate scheduling policies that take into account both the needs of the application and the overall performance of the system. To investigate the best-case benefits of metascheduling we assume an idealized metascheduling scenario. We assume the metascheduler maintains control over all applications running on the specified resources and has the power to reschedule any of those applications at any time. Our metascheduling architecture is shown in Figure 6.7. The metascheduler is implemented by the addition of four components, namely database manager, permission service, contract negotiator and rescheduler to the GrADS architecture.

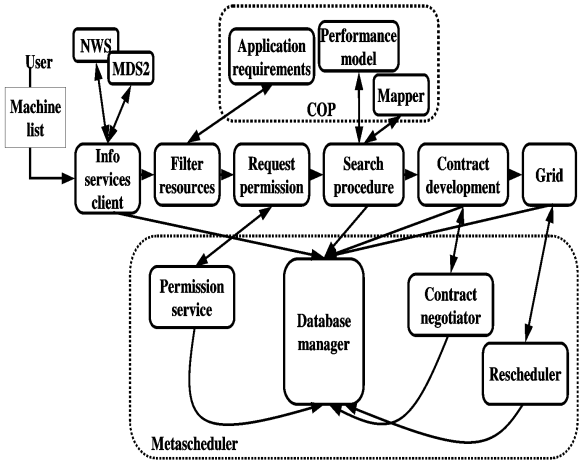


Figure 6.7. GrADS metascheduler architecture.

The *database manager* acts as a repository for information about all applications in the system. The information includes, for example, the status of applications, application-level schedules determined at application launch, and the predicted execution costs for the end applications. As shown in Figure 6.7, the application stores information in the database manager at different stages of execution. Other metascheduling components query the database manager for information to make scheduling decisions.

In metascheduling, after the filter resources component generates a list of resources suitable for problem solving, this list is passed, along with the problem parameters, to the *permission service*. The permission service determines if the

filtered resources have adequate capacities for problem solving and grants or rejects permission to the application for proceeding with the other stages of execution. If the capacities of the resources are not adequate for problem solving, the permission service tries to proactively stop a large resource consuming application to accommodate the new application. Thus, the primary objective of the permission service is to accommodate as many applications in the system as possible while respecting the constraints of the resource capacities.

In the launch-time scheduling architecture, shown in Figure 6.2, after the search procedure determines the final schedule, the application is launched on the final set of resources. In the metascheduling architecture shown in Figure 6.7, before launching the application, the final schedule along with performance estimates are passed as a performance contract to the *contract developer*, which in turn passes the information to the metascheduling component, *contract negotiator*. The contract negotiator is the primary component that balances the needs of different applications. It can reject a contract if it determines that the application has made a scheduling decision based on outdated resource information. The contract negotiator also rejects the contract if it determines that the approval of the contract can severely impact the performance of an executing application. Finally, the contract negotiator can also preempt an executing application to improve the performance contract of a new application. Thus the major objectives of the contract negotiator is to provide high performance to the individual applications and to increase the throughput of the system.

More details about our metascheduling approach and an initial validation for the ScaLAPACK application are available in [VD02].

7. STATUS AND FUTURE WORK

In this chapter we have described approaches developed in GrADS for launch-time scheduling, rescheduling, and metascheduling in a Grid application development system. In developing these approaches we have focused on a Grid environment consisting of distributed clusters of shared workstations; unpredictable and rapidly changing resource availabilities make this a challenging and interesting Grid environment. Previous Grid scheduling efforts have been very successful in developing general scheduling solutions for embarrassingly parallel applications (e.g. [COBW00]) or serial task-based applications (e.g. [RLS99]). We focus instead on data-parallel applications with non-trivial inter-processor communications. This application class is challenging and interesting because to obtain reasonable performance, schedulers must

incorporate application requirements and resource characteristics in scheduling decisions.

We have incorporated many of our launch-time scheduling approaches in GrADSoft and the resulting system has been tested for all of the focus applications described in Section 3. Currently, we are working to generalize our rescheduling results and incorporate them in GrADSoft. Similarly, researchers at UIUC are integrating more sophisticated contract development technologies and researchers at Rice University are working to integrate initial GrADS compiler technologies for automated generation of performance models and mappers. GrADSoft will be used as a framework for testing these new approaches together and extending our scheduling approaches as needed for these new technologies.

In collaboration with other GrADS researchers we plan to extend our approaches to support more Grid environments and other types of applications. Rich Wolski and Wahid Chrabakh, GrADS researchers at the University of California, Santa Barbara, have recently developed a reactive scheduling approach for a Satisfiability application. This application's resource requirements dynamically grow and shrink as the application progresses and the reactive scheduling approach dynamically grows and shrinks the resource base accordingly. As this work evolves, their results can be used to extend our rescheduling approaches to consider changing application requirements. Anirban Mandal and Ken Kennedy, GrADS researchers at Rice University, are working to develop compiler technologies for work-flow applications; we plan to extend our scheduling approaches for this application type.

Acknowledgments

The authors would like to thank the entire GrADS team for many contributions to this work. This material is based upon work supported by the National Science Foundation under Grant 9975020.

Chapter 7

WORKFLOW MANAGEMENT IN GRIPHYN

Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman
Information Sciences Institute, University of Southern California

Abstract This chapter describes the work done within the NSF-funded GriPhyN project in the area of workflow management. The targeted workflows are large both in terms of the number of tasks in a given workflow and in terms of the total execution time of the workflow, which can sometimes be on the order of days. The workflows represent the computation that needs to be performed to analyze large scientific datasets produced in high-energy physics, gravitational physics, and astronomy. This chapter discusses issues associated with workflow management in the Grid in general and also provides a description of the Pegasus system, which can generate executable workflows.

1. INTRODUCTION

As the complexity of Grid applications increases, it becomes ever more important to provide a means to manage application composition and the generation of executable application workflows. This chapter describes the GriPhyN project, which explores, among others, issues of workflow management in Grids in support of large-scale scientific experiments.

GriPhyN (Grid Physics Network [GRIb]) is an NSF-funded project that aims to support large-scale data management in physics experiments such as high-energy physics, astronomy, and gravitational wave physics in the wide area. GriPhyN puts data both raw and derived under the umbrella of *Virtual Data*. A user or application can ask for a virtual data product using application-specific metadata without needing to know whether the data is available on some storage system or if it must to be computed. To enable the computation, the system needs to have a recipe (possibly in a form of a workflow) that describes the data products.

To satisfy the user's request, GriPhyN will schedule the necessary data movements and computations to produce the requested results. To perform the scheduling, resources need to be discovered, data needs to be located and staged, and the computation needs to be scheduled. The choice of particular computational resources and data storage systems will depend on the overall performance and reliability of entire computation. GriPhyN uses the Globus Toolkit [FK97, GLO] as the basic Grid infrastructure and builds on top of it high-level services that can support virtual data requests. These services include request planning, request execution, replica selection, workflow generation, and others.

The following section describes the workflow generation process in general, starting from the application level, which contains application component descriptions, down to the execution level, where the refined workflow is ready for execution. Section 3 provides a brief overview of the issues involved in workflow management in the Grid in general. Particular aspects of the problem such as workflow performance and workflow fault tolerance are addressed. Section 4 describes the Pegasus (Planning for Execution in Grids) [DBG+03b] framework that was developed to provide an experimental platform for workflow management research. Two specific configurations of Pegasus are discussed. The first takes a description of the workflow in terms of logical filenames and logical application component names and produces an executable workflow. The second builds an executable workflow based on metadata that describes the workflow at the application level, allowing scientists to request data in terms that are scientifically meaningful. The chapter concludes with related work on the topic and future research directions.

2. GENERATING APPLICATION WORKFLOWS

Scientists often seek specific data products that can be obtained by configuring available application components and executing them on the Grid. As an example, suppose that the user's goal is to obtain a frequency spectrum of a signal S from instrument Y and time frame X, placing the results in location L. In addition, the user would like the results of any intermediate filtering steps performed to be available in location I, perhaps to check the filter results for unusual phenomena or perhaps to extract some salient features to the metadata of the final results. The process of mapping this type of user request onto jobs to be executed in a Grid environment can be decomposed into two steps, as shown in Figure 7.1.

1 **Generating an abstract workflow:** Selecting and configuring application components to form an abstract workflow. The application components are selected by examining the specification of their capabilities and checking to see if they can generate the desired data products. They

are configured by assigning input files that exist or that can be generated by other application components. The abstract workflow specifies the order in which the components must be executed. More specifically, the following steps need to be performed:

(a) Find which application components generate the desired data products, which in our example is a frequency spectrum of the desired characteristics. Let one such component be C_n. Find which inputs that component needs, check if any inputs are available and if so let the corresponding files be $I_1 \ldots I_j$. If any input is required in formats that are not already available, then find application components that can produce that input, and let one such component be C_{n-1}. This process is iterated until the desired result can be generated from a composition of available components that can operate over available input data, namely $C_1 \ldots C_n$ and $I_1 \ldots I_m$ respectively.

(b) Formulate the workflow that specifies the order of execution of the components $C_1 \ldots C_n$. This is what we call an *abstract workflow*. Please note that at this level the components and files are referred to by their *logical names*, which uniquely identify the components in terms of their functionality and the data files in terms of their content. A single logical name can correspond to many actual executables and physical data files in different locations.

2 **Generating a concrete workflow:** Selecting specific resources, files, and additional jobs required to form a concrete workflow that can be executed in the Grid environment. In order to generate a *concrete workflow*, each component in the abstract workflow is turned into an executable job by specifying the locations of the physical files of the component and data, as well as the resources assigned to the component in the execution environment. Additional jobs may be included in the concrete workflow, for example, jobs that transfer files to the appropriate locations where resources are available to execute the application components. Specifically, the following steps need to be performed:

(a) Find the physical locations (i.e., physical files) of each component $C_1 \ldots C_n$: C_1-$pf \ldots C_n$-pf.

(b) Check the computational requirements of C_1-$pf \ldots C_n$-pf and specify locations $L_1 \ldots L_n$ to execute them according to the required and available resources.

(c) Determine the physical locations of the input data files I_1-$pf \ldots I_m$-pf, and select locations are more appropriate given $L_1 \ldots L_n$.

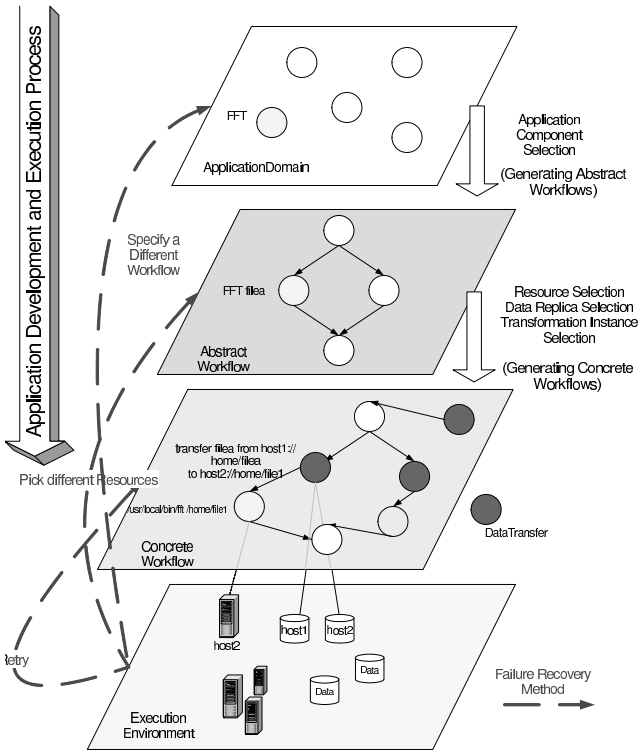


Figure 7.1. The process of developing data intensive applications for Grid environments.

(d) Augment the workflow description to include jobs $K_1 \ldots K_{m+n}$ to move component and input data files (C_1-$pf \ldots C_n$-pf and I_1-$pf \ldots I_m$-pf) to the appropriate target locations $L_1 \ldots L_n$.

Although Grid middleware allows for discovery of the available resources and of the locations of the replicated data, users are currently responsible for carrying out all of these steps manually. There are several important reasons that automating this process not only desirable but necessary:

- Usability: Users are required to have extensive knowledge of the Grid computing environment and its middleware functions. For example, the user needs to understand how to query an information service such as the Monitoring and Discovery Service (MDS2) [CFFK01], to find the available and appropriate computational resources for the computational requirements of a component (step 2b). The user also needs to query the Replica Location Service (RLS) [CDF+02] to find the physical locations of the data (step2c).

- Complexity: In addition to requiring scientists to become Grid-enabled users, the process of workflow generation may be complex and time consuming. Note that in each step the user makes choices when alternative application components, files, or locations are available. The user may reach a dead end where no solution can be found, which would require backtracking to undo some previous choice. Many different interdependencies may occur among components, and as a result it may be difficult to determine which choice of component and or resource to change and what would be a better option that leads to a feasible solution.

- Solution cost: Lower cost solutions are highly desirable in light of the high cost of some of the computations and the user's limitations in terms of resource access. Because finding any feasible solution is already time consuming, users are unlikely to explore alternative workflows that may reduce execution cost.

- Global cost: Because many users are competing for resources, minimizing cost within a community or a virtual organization (VO) [FKT01] is desirable. This requires reasoning about individual user choices in light of all other user choices. It may be desirable to find possible common jobs that can be included across user workflows and executed only once.

While addressing the first three points above would enable wider accessibility of the Grid to users, the last point, minimizing global cost, simply cannot be handled by individual users and will likely need to be addressed at the architecture level. In addition, there are many policies that limit user access to resources, and that needs to be taken into account to accommodate as many users as possible as they contend for limited resources.

An additional issue is the reliability of execution. When the execution of the job fails, in today's Grid framework the recovery consists of resubmitting that job for execution on the same resources. (In Figure 7.1 this is shown as retry.) However, it is also desirable to be able to choose a different set of resources when tasks fail. This choice needs to be performed at the abstract workflow level. Currently, there is no mechanism for opportunistically rerunning the remaining tasks in the workflow to adapt to the dynamic situation of the environment. Moreover, if any job fails repeatedly it would be desirable for the system to assign an alternative component to achieve the same overall user goals. This approach would need to be performed at the application level where there is an understanding of how different application components relate to each other.

3. WORKFLOW MANAGEMENT ISSUES IN GRIDS

Grid computing has taken great strides in the last few years. The basic mechanisms for accessing remote resources have been developed as part of the Globus Toolkit and are now widely deployed and used. Among such mechanisms are: Information services, which can be used to find the available resources and select the resources which are the most appropriate for a task; Security services, which allow users and resources to mutually authenticate and allow the resources to authorize users based on local policies; Resource management, which allows for the scheduling of jobs on particular resources; and Data management services, which enable users and applications to manage large, distributed, and replicated data sets.

With the use of the above mechanisms, one can manually find out about the resources and schedule the desired computations and data movements. However, this process is time consuming and can potentially be complex. As a result, it is becoming increasingly necessary to develop higher-level services that can automate the process and provide an adequate level of performance and reliability. Among such services are workflow generators, which can construct concrete or abstract workflows or both, and executors, which given a concrete workflow execute it on the specified Grid resources.

When generating workflows, one can take into account metrics such as the overall runtime of the workflow, and the probability that the workflow will be executed successfully. An important aspect of obtaining desired results from workflow execution is the relationship between workflow generator and workflow executor. In the following section we explore this interaction.

3.1 Planning for Optimal Performance

In this section, we focus on the behavior of a workflow generator (here referred to as the planner) and its interaction with an executor, such as Condor-G/DAGMan [FTF+02]. The relationship and interfaces between the planner and the executor are important both from the standpoint of planning and fault tolerance. For this discussion, we assume that multiple requests to the system are handled independently.

We first describe two ways that the process of workflow generation and execution can vary, and use them to characterize the different interactions between the planner and executor. Decisions about the workflow, such as the resource on which to run a particular task, can be made with reference to information about the whole workflow or just the task at hand. The former we refer to as a *global decision* and the latter as a *local decision*. Typically, executors make local decisions about tasks, while some planners make global decisions. Decisions can be made as soon as possible (*early*) or just before the task is executed (*in-time*). Typically, an executor makes any decisions just before task execu-

tion. To reason globally, a planner needs to make decisions before any tasks are executed, but they may be revisited nearer the execution time.

One can imagine two extremes for how work is divided between the planner and executor: on one hand, the planner can make a fully detailed workflow based on the current information about the system that includes where the tasks need to execute, the exact location from where the data needs to be accessed for the computation, etc., leaving very few decisions to the executor. We call this approach *full-plan-ahead*, and it can be seen to result in an early, global decision-maker. At the other extreme, the planner can leave many decisions up to the executor. It can, for example, give the executor a choice of compute platforms to use, a choice of replicas to access, etc. At the time the executor is ready to perform the computation or data movements, the executor can consult the information services and make local decisions (here called *in-time local scheduling*).

The benefit of the full-plan-ahead approach is that the planner can aim to optimize the plan based on the entire structure of the workflow. However, because the execution environment can be very dynamic, by the time the tasks in the workflow are ready to execute, the environment may have changed so much that the execution is now far from the expected performance. Additionally, the data may no longer be available at the location assumed by the planner, leading to an execution-time error. Clearly, if the planner constructs fully instantiated plans, it must be able to adapt to the changing conditions and be able to quickly re-plan.

Faults due to the changing environment are far less likely to occur when the executor is given the freedom to make decisions as it processes the abstract workflow, as in the in-time-scheduling approach. At the time a task is to be scheduled, the executor can use the information services to find out about the state of the resources and the location of the data and make a locally optimal decision. However, because the executor does not use global information about the request, it can make potentially expensive decisions.

Another approach is *in-time global scheduling*. Here, the planner provides an abstract workflow description to the executor, which in turn contacts the planner when it is ready to schedule a task and asks for the execution location for the task. The planner can at that time make a decision that uses global information about the workflow. In the meantime, the planner can continually plan ahead, updating its views of the state of the Grid. If the resources available to the planner itself are not able to support continuous planning, the planner can run periodically or at a time when a new decision needs to be made. Because the planner can make decisions each time a task is scheduled, it can take many factors into consideration and use the most up-to-date information. The drawback, however, is that the control may be too fine, and may result

in high communication overheads and a large amount of computation due to re-planning.

Clearly, there is no single best solution for all applications, since these approaches can have very different characteristics. For example, consider a data-intensive application, where the overall runtime is driven by data movement costs, in which some task A requires file A_i as input and produces output file B_i that is input for task B. If B can only run in a few locations, and B_i is large relative to A_i, then a local decision-maker such as the in-time local scheduler is likely to create a very poor plan, because it will first choose a location to run A that is close to its required input A_i, without regard to the requirement to then move the output B_i to a location where B can run. Global reasoning is required to find the best workflow.

To decide on the best strategy, however, we also need to talk about how quickly the Grid environment changes. If the availability of the needed resources changes quickly enough that an early decision for where to execute B may be poor or invalid by the time A completes, and if B can run at many locations or B_i is relatively small, then in-time local scheduling would be better than full-ahead-planning. If the domain changes quickly and requires global scheduling, then in-time global scheduling is probably the best strategy. However, it is also possible that global scheduling itself will take significant time to complete compared with the savings it produces. So if the planning time is large compared with the execution savings, in-time local scheduling or full-ahead-planning may still be preferred. Similar arguments can be made for compute-intensive applications.

Another factor in workflow management is the use of reservations for various resources such as compute hosts, storage systems, and networks. As these technologies advance, we believe that the role of full-plan-ahead and in-time global scheduling systems will increase.

Up to now, we have considered only the case where the workflow management system handles only one request at a time. The problem becomes more complex when the system is required to find efficient workflows for multiple requests and to accommodate various usage policies and community and user priorities. In this case, full-plan-ahead planners have the the advantage of being able to compare and in some cases optimize end-to-end workflows (through exhaustive search.) However, full-plan-ahead planners still face the challenge of needing to be able to react to the changing system state.

The nature of this problem seems to indicate that the workflow management system needs to be flexible and adaptable in order to accommodate various applications behavior and system conditions. Because an understanding of the application's behavior is crucial to planning and scheduling the execution, application performance models are becoming ever more necessary.

3.2 Planning for Fault Tolerance

Performance is not the sole criterion for workflow generation. Fault tolerance of the workflow execution also needs to be considered. The abstract workflow generator can construct a few different workflows, allowing the lower levels to decide which abstract workflow to implement. For example one workflow can rely on data that is available only from very unreliable resources, whereas another can use resources that are expected to be reliable. In that case, the concrete workflow generator may decide to instantiate the second abstract workflow.

The concrete workflow generator itself may provide multiple concrete workflows, leaving it up to the executor to choose between them, or allowing it to execute alternative workflows in cases of a failure of a particular concrete workflow. The generator may also include "what if" scenarios in the concrete workflows. The executor would then execute the dependent nodes in the workflow based on the success or failure (or possibly some other metric) of the execution of the parent nodes. The generator can also instantiate the abstract workflow based on the information it has about the reliability of the system components. To achieve this, monitoring of the behavior of various system components needs to be performed. This information can also be used by the executor, which can replicate tasks if it is scheduling jobs on compute resources viewed as unreliable or on compute resources connected by a poorly performing network. The execution environment itself can incorporate fault tolerance measures, by checkpointing the state and providing restart capabilities. The executor may also want to adapt the checkpointing interval for the jobs based on which resources the job is running.

So far we have discussed the general issues in workflow management in GriPhyN. In the following sections we focus on the first implementation of a workflow generation system.

4. PEGASUS-PLANNING FOR EXECUTION ON GRIDS

Pegasus, which stands for Planning for Execution in Grids, was developed at ISI as part of the GriPhyN project. Pegasus is a configurable system that can map and execute complex workflows on the Grid. Currently, Pegasus relies on a full-ahead-planning to map the workflows. As the system evolves, we will incorporate in-time local scheduling and in-time global scheduling as well.

Pegasus and its concrete workflow generator has been integrated with the GriPhyN Chimera system [FVWZ02]. Chimera provides a catalog that can be used to describe a set of application components and then track all the data files produced by executing those applications. Chimera contains the mechanism to locate the recipe to produce a given logical file in the form of an abstract work-

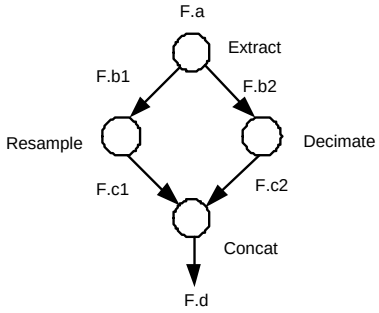


Figure 7.2. An example abstract workflow.

flow. In the Chimera-driven configuration, Pegasus receives an abstract workflow (AW) description from Chimera, produces a concrete workflow (CW), and submits the CW to Condor-G/DAGMan for execution. The workflows are represented as Directed Acyclic Graphs (DAGs). The AW describes the transformations and data in terms of their logical names; it has information about all the jobs that need to be performed to materialize the required data.

The Pegasus *Request Manager* sends this workflow to the *Concrete Workflow Generator* (CWG.) The latter queries the Globus Replica Location Service (RLS) to find the location of the input files, as specified by their logical filenames in the DAG. The RLS returns a list of physical locations for the files.

The information about the available data is then used to optimize the concrete workflow from the point of view of Virtual Data. The optimization is performed by the *Abstract DAG Reduction* component of Pegasus. If data products described within the AW are found to be already materialized, Pegasus reuses them and thus reduces the complexity of the CW.

The following is an illustration of the reduction algorithm. Figure 7.2 shows a simple abstract workflow in which the logical component *Extract* is applied to an input file with a logical filename $F.a$. The resulting files, with logical filenames $F.b1$ and $F.b2$, are used as inputs to the components identified by logical filenames *Resample* and *Decimate*, respectively. Finally, the results are *Concatenated*.

If we assume that $F.c2$ is already available on some storage system (as indicated by the RLS), the CWG reduces the workflow to three components, namely *Extract*, *Resample*, and *Concat*. It then adds the transfer nodes for transferring $F.c2$ and $F.a$ from their current locations. It also adds transfer nodes between jobs that will run on different locations. For example, if the executables for *Resample* and *Concat* are available at two different locations, *Fc.1* will have to be transferred. Finally, the CWG adds output transfer nodes to stage data out and registration nodes if the user requested that the resulting data be published and made available at a particular location. The concrete

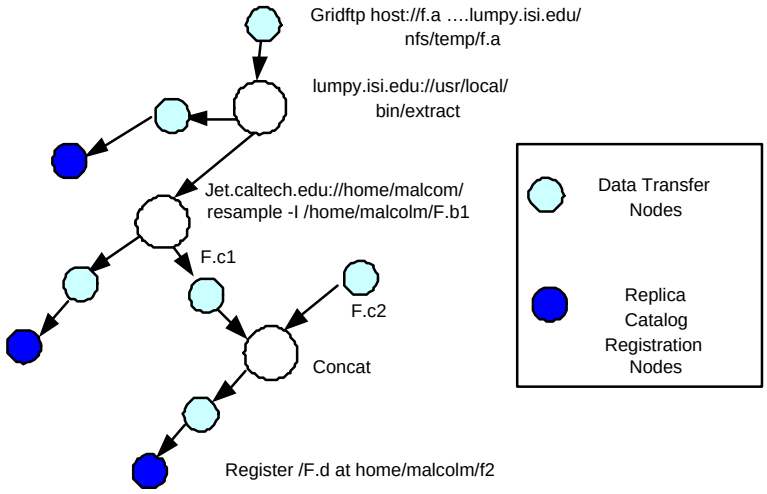


Figure 7.3. An example reduced, concrete workflow.

workflow for this scenario is shown in Figure 7.3. Once the workflow is instantiated in the form of a DAG, software such as DAGMan and Condor-G is used to schedule the jobs described in the nodes of the DAG onto the specified resources in their specified order.

In general, the reduction component of Pegasus assumes that it is more costly to execute a component (a job) than to access the results of the component if that data is available. For example, some other user in the VO may have already materialized part of the entire required data set. If this information is published into the RLS, then Pegasus can utilize this knowledge and obtain the data thus avoiding possible costly computations. As a result, some components that appear in the abstract workflow do not appear in the concrete workflow.

During the reduction, the output files of the jobs that do not need to be executed are identified. Any antecedents of the redundant jobs that do not have any unmaterialized descendants are removed. The reduction algorithm is performed until no further nodes can be reduced.

The CWG component of Pegasus maps the remaining abstract workflow onto the available resources. Currently the information about the available resources is statically configured. However, at this time, we are incorporating the dynamic information provided by the Globus Monitoring and Discovery Service (MDS2) into the decision-making process.

The CWG also checks for the feasibility of the abstract workflow. It determines the root nodes for the abstract workflow and queries the RLS for the existence of the input files for these components. The workflow can only be

executed if the input files for these components can be found to exist somewhere in the Grid and are accessible via a data transport protocol, such as GridFTP [ABB+02b]. The *Transformation Catalog* [DKM01] is queried to determine if the components are available in the execution environment and to identify their locations. The Transformation Catalog performs the mapping between a logical component name and the location of the corresponding executables on specific compute resources. The Transformation Catalog can also be used to annotate the components with the creation information as well as component performance characteristics. Currently the CWG picks a random location to execute from among the possible locations identified by the catalog.

Transfer nodes are added for any of these files that need to be staged in, so that each component and its input files are at the same physical location. If the input files are replicated at several locations, the CWG currently picks the source location at random.

Finally transfer nodes and registration nodes, which publish the resulting data products in the RLS, are added if the user requested that all the data be published and sent to a particular storage location.

In order to be able to execute the workflow, Pegasus' *Submit File Generator* generates submit files which are subsequently sent to DAGMan for execution. Their execution status is then monitored within Pegasus.

In the abstract-workflow-driven configuration, Pegasus has been shown to be successful in mapping workflows for very complex applications such as the Sloan Digital Sky Survey [AZV+02] and the Compact Muon Source [DBG+03a]. Pegasus took the abstract workflow with hundreds of nodes, instantiated it, and oversaw its successful execution on the Grid.

4.1 Planning at the Level of Metadata

Pegasus can also be configured to generate a workflow from a metadata description of the desired data product. This description is application-specific and is meaningful to the domain expert. This configuration of Pegasus was applied to one of the physics projects that are part of GriPhyN, the Laser Interferometer Gravitational-Wave Observatory, (LIGO [LIG, AAD+92, BW99]). LIGO is a distributed network of interferometers whose mission is to detect and measure gravitational waves predicted by general relativity, Einstein's theory of gravity. One well-studied source of gravitational waves is the motion of dense, massive astrophysical objects such as neutron stars or black holes. Other signals may come from supernova explosions, quakes in neutron stars, and pulsars.

LIGO scientists have developed applications that look for gravitational waves possibly emitted by pulsars. The characteristics of the search are the position in the sky, frequency range of the expected signal, and time interval

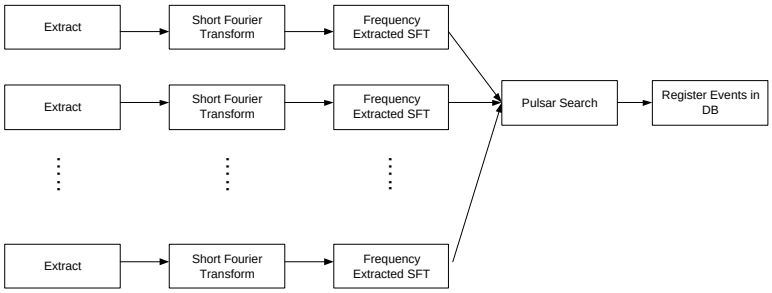


Figure 7.4. The stages in the LIGO pulsar search.

over which to conduct the search. LIGO scientists would like to make requests for data using just such metadata.

To support this type of request, the pulsar application is decomposed into several stages (each an application in itself). Some of these stages may contain thousands of independent applications. The first stage (Figure 7.4) consists of extracting the gravitational wave channel from the raw data. Some processing geared towards the removal (cleaning) of certain instrumental signatures also needs to be done at that time. The pulsar search is conducted in the frequency domain; thus, Fourier Transforms, in particular Short Fourier Transforms (SFTs), are performed on the long duration time frames (second stage). Since the pulsars are expected to be found in a small frequency range, the frequency interval of interest is extracted from the STFs (third stage). The resulting power spectra are used to build the time-frequency image, which is analyzed for the presence of pulsar signatures (fourth stage). If a candidate signal with a good signal-to-noise ratio is found, it is placed in LIGO's event database (final stage). Each of the stages in this analysis can be described uniquely by LIGO-specific metadata, as seen in Table 7.1. For example, the SFT is described by the channel name and the time interval of interest. These descriptions need to made available to the workflow generation system, here Pegasus, which translates them into abstract and/or concrete workflows.

Table 7.1. Metadata describing the LIGO pulsar search stages.

Stage	Metadata
Extract	Channel name, time interval
SFT	Channel name, time interval
Frequency Extracted SFT	Channel name, time interval, frequency band
Pulsar Search	Channel name, time interval, frequency band, location in sky

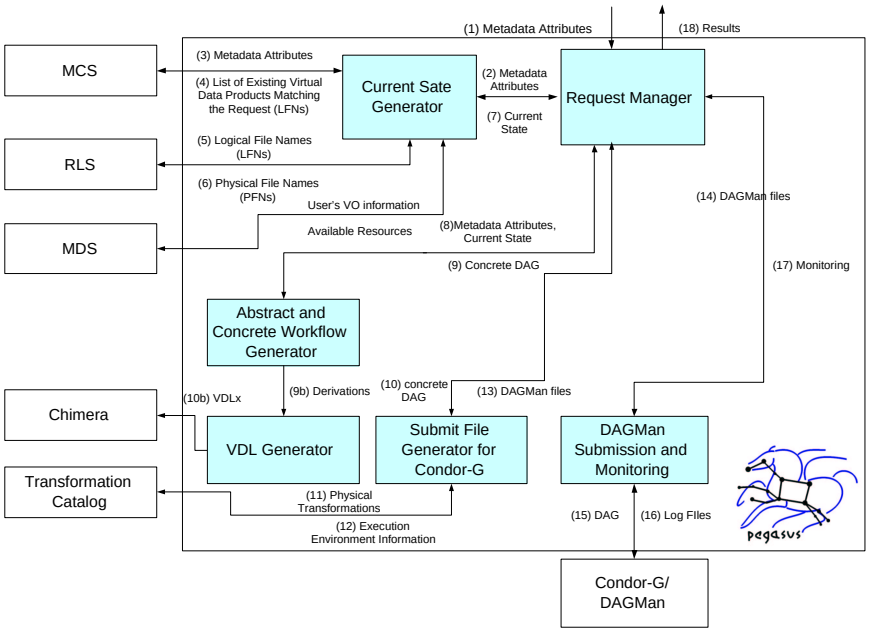


Figure 7.5. Pegasus configured to perform the LIGO pulsar search, based on application-specific metadata.

4.2 Metadata-driven Pegasus

Given attributes such as time interval, frequency of interest, location in the sky, etc., Pegasus was configured to support the LIGO pulsar search. Details about the search can be found in [DKM+02]. Pegasus is currently able to produce any virtual data products present in the LIGO pulsar search. Figure 7.5 shows Pegasus configured for such a search.

Pegasus receives the metadata description from the user. The *Current State Generator* queries the *Metadata Catalog Service* (MCS) [CDK+02], to perform the mapping between application-specific attributes and logical file names of existing data products (step 3 in Figure 7.5). The MCS, developed at ISI, provides a mechanism for storing and accessing metadata, which is information that describes data files or data items. The MCS allows users to query based on attributes of data rather than names of files containing the data. In addition to managing information about individual logical files, the MCS provides management of logical collections of files and containers that consist of small files that are stored, moved, and replicated together.

The MCS can keep track of metadata describing the logical files such as:

- Information about how data files were created or modified, including the creator or modifier, the creation or modification time, what experimental

apparatus and input conditions produced the file, or what simulation or analysis software was run on which computational engine with which input parameters, etc.

- A description of what the data stored in a file represent, for example, time series data collected at a particular instrument.

Once the metadata for a product is specified, MCS determines the corresponding logical file and determines the metadata and logical filenames for all other sub-products that can be used to generate the data product, and returns them to the Current State Generator as shown in Figure 7.5. The Current State Generator then queries the RLS to find the physical locations of the logical files (step 5). We are also interfacing the Current State Generator to MDS2 to find the available Grid resources. Currently, this information is statically configured. The metadata and the current state information are then passed to the *Abstract and Concrete Workflow Generator* (ACWG).

AI-based planning technologies are used to construct both the abstract and concrete workflows [BDG+03, BDGK03]. The AI-based planner models the application components along with data transfer and data registration as operators. The parameters of the planner operators include the location where the component can be run. Some of the effects and preconditions of the operators can capture the data produced by components and their input data dependencies. State information used by the planner includes a description of the available resources and the files already registered in the RLS. The input goal description can include (1) a metadata specification of the information the user requires and the desired location for the output file, (2) specific components to be run, or (3) intermediate data products. The planner generates the concrete workflow (in the form of a DAG) necessary to satisfy the user's request. The planner takes into account the state of the network to come up with an efficient plan, and in some domains finds an optimal plan using an exhaustive search that is made feasible by careful pruning. This planner also reuses existing data products when applicable.

The plan generated specifies the sites at which the job should be executed and refers to the data products in terms of metadata. This plan is then passed to the submit file generator for Condor-G (step 10). The submit file generator determines first the logical names for the data products by querying the MCS and then the physical names by querying the RLS. In addition, it queries the Transformation Catalog to get the complete paths for the transformations at the execution locations described in the concrete DAG.

Pegasus also contains a Virtual Data Language generator that can populate the Chimera catalog with newly constructed derivations (step 9b). This information can be used later to obtain provenance information about the derived data products.

In this configuration, Pegasus similarly generates the necessary submit files and sends the concrete workflow to DAGMan for execution (step 15).

As a result of execution of the workflow, the newly derived data products are registered both in the MCS and RLS and thus are made available to subsequent requests.

The metadata-driven Pegasus configuration that supported the LIGO pulsar search was first demonstrated at the SC 2002 conference held in November in Baltimore, MD. For this demonstration the following resources were used:

- Caltech (Pasadena, CA): Data Storage Systems and the LIGO Data Analysis System (LDAS), which performed some of the stages of the LIGO analysis shown in Figure 7.4.

- ISI (Marina del Rey, CA): Condor Compute Pools, Data Storage Systems, Replica Location Services and Metadata Catalog Services.

- University of Wisconsin (Milwaukee, WI): Condor Compute Pools and Data Storage Systems.

The requests for pulsar searches were obtained using an automatic generator that produced requests both for approximately 1300 known pulsars as well as for random point-searches in the sky. A user was also able to request a specific pulsar search by specifying the metadata of the required data product through a web-based system. Both the submission interfaces as well as all the compute and data management resources were Globus GSI (Grid Security Infrastructure) [WSF+03] enabled. Department of Energy Science Grid [DOEb] issued X509 certificates were used to authenticate to all the resources.

During the demonstration period and during a subsequent run of the system, approximately 200 pulsar searches were conducted (both known as well as random), generating approximately 1000 data products involving around 1500 data transfers. The data used for this demonstration was obtained from the first scientific run of the LIGO instrument. The total compute time taken to do these searches was approximately 100 CPU hrs. All the generated results were transferred to the user and registered in the RLS; the metadata for the products placed in the MCS as well as into LIGO's own metadata catalog. Pegasus also generated the corresponding provenance information using the Virtual Data Language and used it to populate in the Chimera Virtual Data Catalog.

The job execution was monitored by two means. For each DAG, a start and end job was added, which logged the start time and the end time for the DAG into a MySQL database. This information was then published via a web interface. Another script-based monitor parsed the Condor log files at the submit host to determine the state of the execution and published this information onto the web.

5. RELATED WORK

In the area of AI planning techniques, the focus is on choosing a set of actions to achieve given goals, and on scheduling techniques that focus on assigning resources for an already chosen set of actions. Some recent approaches in scheduling have had success using iterative refinement techniques [SL94] in which a feasible assignment is gradually improved through successive modifications. The same approach has been applied in planning and is well suited to ACWG, where plan quality is important [AK97]. Some work has been done on integrating planning and scheduling techniques to solve the joint task [MSH+01].

Central to scheduling large complex workflows is the issue of data placement, especially when the data sets involved are very large. In CWG, we give preference to the resources where the input data set is already present. Ranganathan and Foster, in [RF02, RF01] and Chapter 22, study the data in the Grid as a tiered system and use dynamic replication strategies to improve data access, and achieve significant performance improvement when scheduling is performed according to data availability while also using a dynamic replication strategy.

While running a workflow on the Grid makes it possible to perform large computations that would not be possible on a single system, it leads to a certain loss of control over the execution of the jobs, as they may be executed in different administrative domains. To counter this, there are other systems try to provide Quality of Service guarantees required by the user while submitting the workflow to the Grid. Nimrod-G [ABG02, BAG00] uses the information from the MDS to determine the resource that is selected to meet the budget constraints specified by the user, while Keyani et al. [KSW02] monitors a job progress over time to ensure that guarantees are being met. If a guarantee is not being met, schedules are recalculated.

Each of the systems mentioned above are rigid because they use a fixed set of optimization criteria. In GriPhyN we are developing a framework for a flexible system that can map from the abstract workflow description to its concrete form and can dynamically change the optimization criteria.

6. FUTURE DIRECTIONS

The efforts in workflow planning presented in this chapter lay the foundation for many interesting research avenues.

As mentioned in Section 3.2, fault tolerance is an important issue in the Grid environment, where runtime failures may result in the need to repair the plan during its execution. The future work within Pegasus may include planning strategies that will design plans to either reduce the risk of execution failure or to be salvageable when failures take place. Current AI-based planners can ex-

plicitly reason about the risks during planning and searching for reliable plans, possibly including conditional branches in their execution. Some planners delay building parts of the plan until execution, in order to maintain a lower commitment to certain actions until key information becomes available.

Another area of future research is the exploration of the use of ontologies to provide rich descriptions of Grid resources and application components. These can then be used by the workflow generator to better match the application components to the available resources.

Acknowledgments

We would like to thank Gaurang Mehta, Gurmeet Singh, and Karan Vahi for the development of the Pegasus system. We also gratefully acknowledge many helpful and stimulating discussions on these topics with our colleagues Ian Foster, Michael Wilde, and Jens Voeckler. The Chimera system mentioned in this paper has been jointly developed by the University of Southern California Information Sciences Institute, the University of Chicago, and Argonne National Laboratory. Finally, we would like to thank the following LIGO scientists for their contribution to the development of the Grid-enabled LIGO software: Kent Blackburn, Phil Ehrens, Scott Koranda, Albert Lazzarini, Mary Lei, Ed Maros, Greg Mendell, Isaac Salzman, and Peter Shawhan. This research was supported in part by the National Science Foundation under grants ITR-0086044(GriPhyN) and EAR-0122464 (SCEC/ITR).

III

STATE OF THE ART
GRID RESOURCE MANAGEMENT

Chapter 8

GRID SERVICE LEVEL AGREEMENTS

Grid Resource Management with Intermediaries

Karl Czajkowski,[1] Ian Foster,[2,3] Carl Kesselman,[1] and Steven Tuecke[2]

[1] *Information Science Institute, University of Southern California*
[2] *Mathematics and Computer Science Division, Argonne National Laboratory*
[3] *Department of Computer Science, The University of Chicago*

Abstract We present a reformulation of the well-known GRAM architecture based on the Service-Level Agreement (SLA) negotiation protocols defined within the Service Negotiation and Access Protocol (SNAP) framework. We illustrate how a range of local, distributed, and workflow scheduling mechanisms can be viewed as part of a cohesive yet *open* system, in which new scheduling strategies and management policies can evolve without disrupting the infrastructure. This architecture remains neutral to, and in fact strives to mediate, the potentially conflicting resource, community, and user policies.

1. INTRODUCTION

A common requirement in distributed computing systems such as Grids [FK99b, FKT01] is to coordinate access to resources located within different administrative domains than the requesting application. The coordination of Grid resources is complicated by the frequently competing needs of the resource consumer (or *application*) and the *resource owner*. The application needs to understand and affect resource behavior and often demands assurances as to the level and type of service being provided by the resource. Conversely, the resource owner may want to maintain local control and discretion over how their resource(s) are used.

A common means of reconciling these two competing demands is to establish a procedure by which the parties can negotiate a *service-level agreement* (SLA) that expresses a resource provider *contract* with a client to provide some measurable capability or to perform a specified task. An SLA allows a client

to understand *what to expect from resources* without requiring detailed knowledge of competing workloads nor of resource owners' policies.

The Globus resource management architecture that we present here is based on the notion of negotiation of SLAs between a client and a resource provider. A resource provider can be directly associated with a resource, or alternatively may be a service that virtualizes multiple resources, i.e., a *broker* or *super-scheduler* in which case negotiation proceeds hierarchically. The SLAs are defined such that it is meaningful to construct hierarchies of negotiating entities acting as both providers and clients.

We have previously explored Grid resource management methods in our work with the Grid Resource Allocation and Management (GRAM) [CFK+98b] component of the Globus Toolkit, which supports remote access to job-submission systems; DUROC [CFK99], a co-allocation system that uses GRAM functions to compose jobs that use more than one resource; and the General-purpose Architecture for Reservation and Allocation (GARA) [FRS00, FFR+02], which extends GRAM to support immediate and advance reservation. Building on these experiences, and in particular on the decoupling of reservation and task creation introduced in GARA, we defined the Service Negotiation and Access Protocol (SNAP) framework [CFK+02], an abstract architecture that defines operations for establishing and manipulating three distinct types of SLA, as follows:

1 *Resource service level agreements* (RSLAs) that represent a commitment to provide a resource when claimed by a subsequent SLA.

2 *Task service level agreements* (TSLAs) that represent a commitment to perform an activity or task with embedded resource requirements.

3 *Binding service level agreements* (BSLAs) that represent a commitment to apply a resource to an existing task, i.e. to extend the requirements of a task after submission or during execution.

As illustrated in Figure 8.1, these three kinds of SLA support an interactive resource management model in which one can independently submit tasks to be performed, obtain promises of capability, and bind a task and a capability. These three types of agreement can be combined in different ways to represent a variety of resource management approaches including batch submission, resource brokering, adaptation, co-allocation, and co-scheduling.

Our presentation here first reviews the SNAP design and then describes a concrete realization of the framework in the GRAM-2 next-generation Globus resource management architecture. The GRAM-2 design addresses various technical issues that arise when the SNAP protocol building blocks are incor-

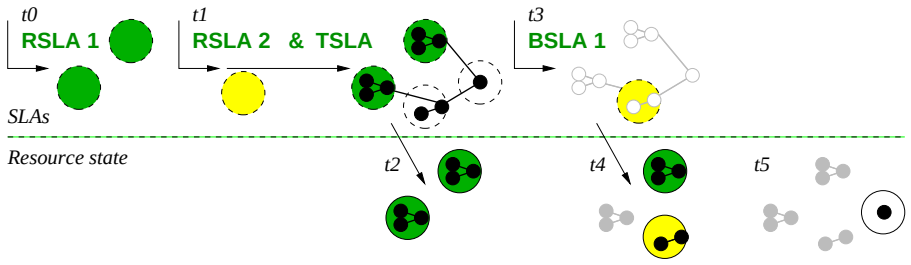


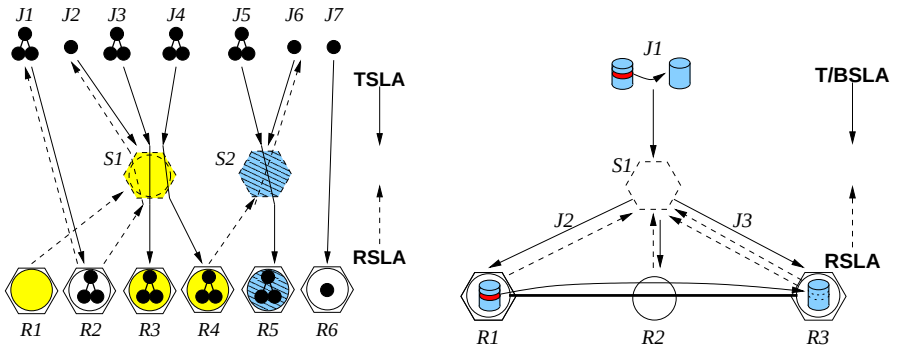
Figure 8.1. Three types of SLA—RSLA, TSLA, and BSLA—allow a client to schedule resources as time progresses from t_0 to t_5. In this case, the client acquires two resource promises (RSLAs) for future times; a complex task is submitted as the sole TSLA, utilizing RSLA 1 to get initial portions of the job provisioned; later, the client applies RSLA 2 to accelerate execution of another component of the job via BSLA 1; finally, the last piece of the job is provisioned by the manager according to the original TSLA.

porated into an operational resource management system. It provides a complete solution to the problem of hierarchical negotiation of SLAs between a resource consumer and a resource provider. (A resource provider can be directly associated with a resource, or alternatively may be a service that virtualizes multiple resources, e.g., a broker or scheduler.)

2. MOTIVATING SCENARIOS

The SNAP SLA model is designed to address a broad range of applications through the aggregation of simple SLAs. We motivate its design by examining three scenarios: a Grid in which *community schedulers* mediate access to shared resources on behalf of different client groups; a file-transfer service that uses advance reservations to perform data staging under deadline conditions; and a job-staging system that uses co-allocation to coordinate functions across multiple resource managers.

The schedulers in our scenarios are all examples of a class of resource management intermediaries that are variously referred to as brokers, agents, distributed schedulers, meta-schedulers, or super-schedulers [FRS00, MBHJ98]. What distinguishes each kind of intermediary are the policies that are supported between users, intermediaries, and underlying resource schedulers. For example, an intermediary may support a large community of users (e.g., a typical resource broker), or act on behalf of a single user (an agent). A similarly wide range of policies can exist between the intermediary and its resource(s)—some intermediaries may have exclusive access to resources while others may have no more rights than a typical user. The intermediary may not even be a distinguished entity for policy but instead may simply act via rights delegated from the client.



(a) Community scheduler scenario. Community schedulers (S1–S2) mediate access to the resources (R1–R6) by their users (J2–J6) through resale of SLAs suitable to community criteria.

(b) File transfer scenario. Transfer scheduler coordinates disk reservation before co-scheduling transfer endpoint jobs to perform transfer jobs for clients.

Figure 8.2. SLA architecture scenarios. Persistent intermediate scheduling services form SLAs with users and underlying resources to help coordinate user activity in the Grid. This supports scalable negotiation and also identifies points where mapping from one request or policy language to another is likely to occur.

2.1 Community Scheduler Scenario

A community scheduler is an entity that acts as a policy-enforcing intermediary between a community of users and a set of resources. Activities are submitted to the community scheduler rather than to end resources. The scheduler then works to schedule those activities onto community resources so as to meet community policies regarding use of the resource set: for example, to optimize response time or throughput, or to enforce allocations. We are thus faced with a two-tiered SLA negotiation process, from client to scheduler and then from scheduler to client. The strictness of the policy environment in enforcing this multi-tier SLA negotiation will affect the predictability and efficiency of the resulting schedules. Even in an open resource environment in which the community scheduler is easily bypassed by aggressive clients, lightweight clients may benefit from the sharing of resource discovery processes performed by the scheduler.

As depicted in Figure 8.2(a), a Grid environment may contain many resources (R1–R6), all presenting both an RSLA and a TSLA interface. First, the scheduler negotiates capacity guarantees (via RSLAs) with its underlying resources. With these capacity guarantees in hand, it can then negotiate RSLAs or TSLAs with its clients, fulfilling its commitments by negotiating further

SLAs with resources to map the requested user activities to the previously negotiated capacity. Depending on the community, workload, and other factors, the scheduler may variously negotiate capacity before receiving user requests (as suggested in the preceding discussion), or alternatively, may do so only *after* receiving requests. In either case, the ability to negotiate agreements with underlying resources abstracts away the impact of other community schedulers as well as any non-Grid local workloads, assuming the resource managers will enforce SLA guarantees at the individual resources.

Community scheduler services (S1 and S2 in Figure 8.2(a)) present a TSLA interface to users. Users in this environment interact with community and resource-level schedulers as appropriate for their goals and privileges. The privileged client with job J7 in Figure 8.2(a) may not need RSLAs nor the help of a community scheduler, because the goals are expressed directly in the TSLA with resource R6. The client with job J1 acquires an RSLA from R2 in anticipation of its requirements and utilizes that reservation in a TSLA.

Jobs J2 to J6 are submitted to community schedulers S1 and S2 which might utilize special privileges or domain-specific knowledge to efficiently implement their community jobs across the shared resources. Note that R2 is running job J1 while guaranteeing future availability to S1 (which is in turn guaranteeing J2 a place to run based on that reservation). Similarly, R4 is running job J4 from S1 while guaranteeing a future slot to J6 by way of S2. Scheduler S1 also maintains a speculative RSLA with R1 to more rapidly serve future high-priority job requests.

2.2 File Transfer Service Scenario

We next consider a scenario in which the user activities of interest are concerned with the transfer of a file from one storage system to another. A transfer requires multiple resources—storage space on the destination resource, plus network and endpoint I/O bandwidth during the transfer—and thus the scheduler needs to manage multiple resource types and perform co-scheduling of these resources through their respective managers.

As depicted in Figure 8.2(b), the file transfer scheduler S1 presents a TSLA interface, storage systems provide TSLA/RSLA interfaces, and a network resource manager R2 presents an RSLA/BSLA interface. A user submitting a transfer job (J1) to the scheduler negotiates a TSLA that includes a deadline. The scheduler works to meet this deadline by: (1) obtaining a storage reservation on the destination resource R3 to ensure that there will be space for the data; (2) obtaining bandwidth reservations from the network and the storage devices, giving the scheduler confidence that the transfer can be completed within the user-specified deadline; (3) submitting transfer endpoint jobs J2 and J3 to implement the transfer using the previously negotiated space and band-

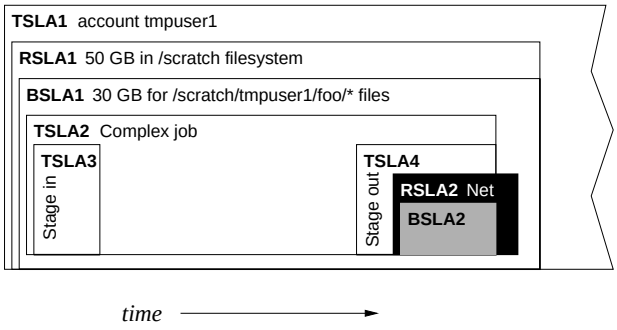


Figure 8.3. Dependent SLAs for file transfers associated with input and output of a job with a large temporary data space. BSLA2 is dependent on TSLA4 and RSLA2, and has a lifetime bound by those two. All job components depend on an outermost account sandbox assigned temporarily for the purpose of securely hosting this job.

width promises; and finally, establishing a BSLA with R2 to utilize the network reservation for the sockets created between J1 and J2.

2.3 Job Staging with Transfer Service Scenario

SLAs can be linked to address more complex resource co-allocation situations. We illustrate the use of linking by considering a job that consists of a sequence of three activities: data is transferred from a storage system to an intermediate location; some computation is performed using the data; and the result is transferred to a final destination. Such a sequence would typically be treated monolithically by the job system, but this approach is inappropriate when data transfers involve significant resource utilization that spans resource domains, as in the previous data transfer scenario, in which source and destination storage are under separate control.

As in the other examples, the computation in question is to be performed on resources allocated to a community of users. However, for security reasons, the computation is not performed using a group account, but rather, a temporary account is dynamically created for the computation. The SLA model facilitates the decomposition of staging and computation activities that is required for these functions to be integrated with dynamic account management functions.

In Figure 8.3, TSLA1 results from a negotiation with the resource to establish a temporary user account, such as might be established by a resource for a client who is authorized through a Community Authorization Service [PWF+02]. All job interactions performed by that client on the resource become linked to this long-lived TSLA, as in order for the account to be reclaimed safely, all dependent SLAs must be destroyed. The figure illustrates how the individual SLAs associated with resources and tasks can be combined

to address the end-to-end resource and task management requirements of the entire job. Of interest in this example are:

TSLA1 is the TSLA negotiated to establish the above-mentioned temporary user account.

RSLA1 promises the client 50 GB of storage in a particular file-system on the resource.

BSLA1 binds part of the promised storage space to a particular set of files within the file-system.

TSLA2 runs a complex job that will subsequently spawn subjobs for staging input and output data.

TSLA3 is the TSLA negotiated for the first file transfer task, which stages the input to the job site (without requiring any additional QoS guarantees in this case).

TSLA4 is the TSLA negotiated for the second file transfer task, to stage the large output from the job site, under a deadline, before the local file-system space is lost.

RSLA2 and BSLA2 are used by the file transfer service to achieve the additional bandwidth required to complete the (large) transfer before the deadline.

The job for which TSLA2 is negotiated might have built-in logic to establish the staging jobs TSLA3 and TSLA4, or this logic might be incorporated within the entity that performs task TSLA2 on behalf of the client. In Figure 8.3, the nesting of SLA "boxes" is meant to illustrate how the lifetime of these management abstractions is linked in practice. Such linkage can be forced by a dependency between the subjects of the SLAs, e.g., BSLA2 is meaningless beyond the lifetime of TSLA4 and RSLA2, or alternatively can be added as a management convenience, e.g., by triggering recursive destruction of all SLAs from the root to hasten reclamation of application-grouped resources.

3. RESOURCE VIRTUALIZATION THROUGH INTERMEDIARIES

The Community Scheduler introduced above can be viewed as virtualizing a set of resources for the benefit of its user community. Resource virtualiza-

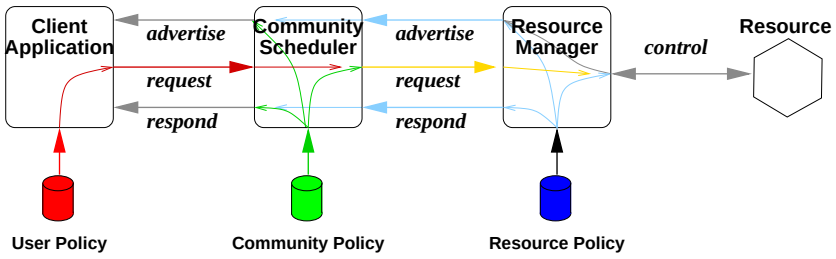


Figure 8.4. SLA negotiation with intermediaries. A negotiation pipeline between a user, community scheduler, and resource manager permits policies to be introduced at each stage which affect the outcome and are illustrated using color mixing. User policy affects what requests are initiated, community policy affects how user requests are mapped to resource-level requests, and resource policy affects how resources may be utilized. Thus policy from each source mixes into the stream of requests going to the right, and into the streams of advertisements and request-responses going to the left.

tion can serve to abstract details of the underlying resources or to map between different resource description domains. In our initial community scheduler example, the scheduler provides the same sort of resource and task description as the underlying resources, only insulating the user community from task-placement decisions. However, with the file transfer scenario, the scheduler accepts requests in a more application-level file transfer description language. In this case, the scheduler insulates the user community from the more complicated resource interactions necessary to implement a file transfer with deadline guarantees in a distributed environment.

Each such intermediary scheduler delineates a boundary between resource domains and may map from more abstract user terminology to underlying resource mechanisms, as well as bridging policy domains. Such mappings can complicate the Grid resource management problem, as important scheduling information can be lost. However, we believe that there is also the opportunity to introduce intuitively-framed policies at each such boundary.

Figure 3 illustrates how policies from different domains mix into SLA negotiation with intermediaries, each of which can potentially contribute to resource management decisions. Community policies may affect relative priority of different user tasks with a community scheduler. At the resource, owner policies may affect relative priority of different communities. In order to implement the community policy, the community scheduler must negotiate dynamic policies (SLAs) to differentiate tasks on the same physical resources.

4. UNDERSTANDING SERVICE LEVEL AGREEMENTS

The preceding sections describe the role of automated intermediaries in negotiating SLAs for complex Grid scenarios. Automated SLA negotiation requires that we be able to represent, in machine-processable terms, what is *offered* by a service and what is *desired* or requested by a client. Following discovery and negotiation, our three kinds of SLA represent what is to be *performed* by the resources.

This leads to several questions. If all three agreements capture what is to be performed, what distinguishes the three kinds of agreement? How are the SLAs represented? What does it mean for a scheduler to agree or "promise" to do something? In the remainder of this section, we answer these questions.

4.1 Different Kinds of Agreement

There are three promises we can capture in our SLAs. Resource managers can promise that a resource will be available for future use, promise that a resource will be consumed in a certain manner, and promise that a certain task will be performed. The three kinds of agreement introduced in Section 1 capture important combinations of these promises. What makes these combinations important is how they provide for the multiple negotiation phases exploited in our scenarios.

All three kinds of SLA provide a promise of future resource availability. What distinguishes RSLAs from the others is that *RSLAs only promise resource availability* without any associated plan for utilization. As illustrated in Figure 8.5, the remaining two kinds of SLA are formulated by the introduction of the other promises. Our *BSLAs add a plan for resource utilization*, without performing any new tasks, and *TSLAs capture all three promises at once*. There is not a fourth kind of SLA capturing a task promise without utilization. Such an agreement would not be meaningful, because performing a task requires resource utilization.

In the simplest case, these SLAs are negotiated in sequential order, as specified above. More generally, there is a partial order based on references made in the resulting SLAs. For example, an RSLA may be referenced in a TSLA, or it may only be referenced in a subsequent BSLA that augments the task. In the latter case, the TSLA and RSLA can be negotiated in any order. General many-to-many references are meaningful, though the range of possible scenarios may be limited by policy in a given negotiation.

RSLAs correspond to (immediate or advance) reservations, i.e. they represent a promise that can be employed in future SLA negotiation. No RSLA has any effect unless it is claimed through a follow-up TSLA or BSLA negotiation. This split-phase negotiation is useful when attempting to synchronize

interactions with multiple resource providers, because it allows a client to obtain commitments for future availability of capability before details of the use of that capability have been decided [DKPS97, FGV97, HvBD98, FRS00].

By binding a resource reservation to a task, a BSLA allows for control of the allocation of resources to tasks (*provisioning*), independent of task creation. Negotiation of a BSLA does not initiate any task: the task must be created separately (either before or after the BSLA, depending on the task naming mechanisms employed in BSLA). This decoupled provisioning can be used to provision a task created outside the SLA negotiation framework, e.g., a network socket or local UNIX process created through interactive mechanisms. BSLA negotiation can also be used to separate the provisioning decisions from basic task management with TSLAs; if the task has variable requirements, these can be expressed by shorter duration BSLAs associated with a long-lived TSLA that only represents the baseline requirements of the task. A BSLA can add to, but not diminish, the resource allocation requirements expressed during task creation.

Finally, the negotiation of an TSLA represents a commitment by a resource manager (task scheduler) to perform the described task with the described levels of resource provisioning. In all cases, there is a need for complex SLA meta-data to denote whether or how implied requirements are addressed.

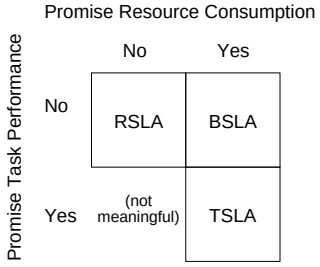


Figure 8.5. Three kinds of SLA. All three SLAs promise resource availability, but TSLAs and BSLAs add additional promises of task completion and/or resource utilization.

4.2 Representing SLAs

To build a system using these three kinds of SLA, we need to represent them in some machine-processable form. For integration with XML-based systems such as Web Services [CCMW01] or Grid Services [TCF+03], we would want XML schema definitions for the SLA content. However, these definitions are cumbersome to present and should be developed in a community standards body.

We can describe the content of the SLAs, both to help envision the scope of the terms and for input into standards processes. Our descriptions make use of the following elemental descriptions:

- *SLA references*, which allow the newly negotiated SLA to be associated with pre-existing SLAs;
- *Resource descriptions*, which are the main subject of RSLA negotiations and may also appear within a TSLA or BSLA (potentially accompanied by RSLA references);
- *Resource metadata*, which qualify the capability with time of availability, reliability, etc.;
- *Task descriptions*, which are the main subject of TSLA negotiations and may also appear within a BSLA;
- *SLA metadata*, which qualify the SLA content with degree of commitment (see Section 4.2), revocation policies, SLA lifetime, etc.

The content of an RSLA includes resource descriptions and metadata, as well as SLA metadata. The description captures resource capabilities such as storage space, nodes in a multicomputer, or processing throughput during a certain interval of time. The SLA metadata might capture the level of commitment *promised* by the resource manager to the client.

The content of a BSLA includes TSLA references or task descriptions, resource descriptions, optional RSLA references, and SLA metadata. The TSLA references or task descriptions identify tasks which will consume resources. The resource descriptions describe what resources will be provided to the tasks and the RSLA references identify existing resource promises to be exploited. Example tasks might be job processes, network flows, or filesystem accesses.

The content of a TSLA includes a task description, resource descriptions, optional RSLA references, and SLA metadata. The task description describes what task will be completed. The resource description describes requirements of the task and the RSLA references identify existing resource promises to be exploited.

The ability to negotiate SLAs can be beneficial regardless of how much credence one is able to put in the agreements. At one extreme, an SLA may represent simply a guess as to what may be possible; at another, it may be accompanied by a strict contractual agreement with defined financial penalties for noncompliance. Even if this degree of commitment is formalized in SLA meta-data, it is always possible for a manager to be error-prone or untrustworthy. Licensing mechanisms might be used to allow users to judge which managers are to be trusted [LMN94].

5. SLA CONSTRAINT-SATISFACTION MODEL

The SNAP model is concerned primarily with the protocols used to negotiate SLAs. However, it is also useful to provide some informal discussion

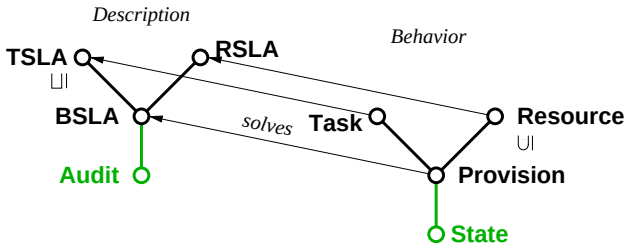


Figure 8.6. Constraint domain. Lower items in the figure conservatively approximate higher items. The solution spaces on the right are ordered as subsets, e.g., Provisioning$\subseteq$Reserves because provisioning constrains a resource promise to a particular task. Solution ordering maps to the model relation for constraints, e.g., BSLA$\sqsubseteq$RSLA on the left.

of the semantics associated with SLA negotiation. Given a particular task description and resource description language, the purpose of a resource provider (whether resource owner or scheduler) is to attempt to *satisfy* SLAs specified in requests. An SLA is satisfied if the resource provider can produce a non-empty solution set of possible resource and task schedules that delivers the capabilities and performs the directives encoded in the SLA content. A self-contradictory or unsatisfiable SLA has an empty solution set. We have previously explained [CFK$^+$02] how satisfaction of SLA terms by resource behavior is related to the notion of problem solving by refinement of plans. As application goals are translated through a sequence of intermediaries, the terms are refined until a concrete resource schedule is reached and application goals satisfied.

A TSLA says that a manager will run a job according to its self-expressed performance goals and provisioning requirements. A RSLA says that a manager will provide a resource capability when asked by the client. A corresponding BSLA encompasses both of these agreements and says the manager will apply a resource to help satisfy requirements while performing a job.

5.1 Descriptive and Behavioral Concept Domains

Figure 8.6 illustrates this ordering of SNAP concepts in terms of refinement and satisfaction. Descriptive or behavioral elements at the bottom of the figure are more concrete realizations of the concepts connected above them in the figure. This domain diagram partially orders abstract concepts by vertical position, with more abstract concepts on top. Concepts are only ordered if there is a connecting path of lines representing a transitive ordering relationship. The thick lines between the right-hand *behavioral* concepts represent the subset relationship, e.g., actual resource states are a subset of possible states fitting a provisioning plan (State$\subseteq$Provision) and a concrete provisioning solutions are a subset of task solutions.

The left-hand SLA terms in Figure 8.6 are ordered by the *satisfaction* relationship that we introduced previously [CFK+02], in which, for example, the more specific BSLA terms can be thought to satisfy the more general TSLA or RSLA terms while also introducing new details absent from the more general SLAs. Note the interesting parallel between the two halves of the figure, shown by thin arrows that link behavioral concepts to descriptive terms. The behavioral concept *solves* a constraint specified with the descriptive terms. For a particular BSLA b and TSLA t, there are corresponding provisioning and task solution sets $S_{\mathbf{B}}(b)$ and $S_{\mathbf{T}}(t)$, respectively, such that $b \sqsubseteq t$ if and only if $S_{\mathbf{B}}(b) \subseteq S_{\mathbf{T}}(t)$.

6. APPROACHES TO IMPLEMENTATION

The SNAP architectural model serves to frame our understanding of existing resource management mechanisms and how they fit together in a Grid environment. Adoption of the abstract model does not require much change to existing systems, though it does help identify limitations and policy-biased features of existing systems. Just as GRAM [CFK+98b] *adapts* localized schedulers to a Grid-wide protocol, we believe that the SLA-based SNAP architecture can be deployed as GRAM-2 with adapters to local schedulers. However, the best implementation approach would be for vendors to natively support new interoperable negotiation protocols in the local schedulers.

6.1 Toward Standards for Interoperability

Within the Global Grid Forum, the GRAAP working group is chartered to standardize negotiation protocols in the form of Open Grid Service Architecture *portTypes* (interfaces). Further work needs to be chartered to produce SLA content standards.

GRAM-2 implementation is underway using similar but proprietary portTypes in anticipation of this work; we will migrate to use standard interfaces when they are available. Multiple commercial scheduler vendors are interested in implementing GRAAP interfaces—and interim Globus GRAM-2 interfaces while GRAAP standardization progresses.

Multiple products, including Globus Toolkit 3 (GT3), will provide OGSA infrastructure implementation within which a scheduler implementation can be hosted.

6.2 SLA Implementation Through Policy Mapping

A TSLA transfers specific task-completion responsibilities from the user to a manager. The scheduler then becomes responsible for reliably planning and enacting the requested activity, tracking the status of the request, and perhaps notifying the user of progress or terminal conditions. A RSLA similarly

delegates specific resource capacity from a manager to a user. The manager might implement this delegation via hidden operational policy statements that enforce the conditions necessary to deliver on the guarantee. For example, a CPU reservation might prevent further reservations from being made, or an internal scheduling priority might be adjusted to steal resources from a best-effort pool when necessary.

Tasks may make resource requests dynamically during their execution: for example, I/O requests or low-level memory allocation requests. Thus, we can configure the task-resource binding expressed by a BSLA so that task resource requests are interpreted as claims against the RSLA promise. In the general case, a BSLA binding may include capability descriptions to restrict what claims can be made, as well as more fine-grained resource-to-task mapping information. When such restriction and mapping information is expressible, it is possible to create complex many-to-many binding relationships between RSLAs and TSLAs without ambiguity or over-subscription to capabilities.

In the face of ambiguous or over-committed binding, fall-back policies must still resolve conflicts at runtime. Traditional first-come, first-serve or fair-share job schedulers can be seen as implementing such fall-back policies in an environment where every TSLA is bound to the same machine-wide RSLA. The addition of fine-grained binding information simply partitions these conflicts into smaller logical resource domains, thus allowing the scheduler to be guided with application and administrator goals.

6.3 Security Considerations

Whether SLA guarantees are enforced via such policy-mapping or not, the negotiation of SLAs is easily seen as a form of distributed policy management. As such, work needs to be started to get SLA proponents involved with existing security standards activities.

With the Global Grid Forum, the OGSA working group is chartered to guide narrowly-targeted groups providing OGSA-relevant standards. The OGSA-SEC working group is specifically chartered to develop approaches complementary of basic Web Service standards coming from other communities such as W3C and OASIS.

7. RELATED WORK

Resource management in networks and wide area computing systems is receiving increased attention (for overviews, see [Ber99, GS99]). However, there has been little work on the specific problems addressed here, namely general-purpose architectural constructs for reservation and co-allocation of heterogeneous collections of resources. Here we review briefly some relevant work; space constraints prevent a complete survey.

Various proposals have been made for advance reservations in the Internet [WS97, FGV97, DKPS97, HvBD98, BL98]. These capabilities are typically encapsulated into the function of the network, in the form of cooperating sets of servers that coordinate advance reservations along an end-to-end path. Such efforts have proposed solutions for various network reservation challenges, but do not address problems that arise when an application requires co-allocation of multiple resource types.

The theory of co-allocation is well understood, and sophisticated techniques exist for determining resource requirements (e.g., identifying the CPU and network resources required to meet application QoS requirement [MIS96, NS96, NCN98]) and for scheduling scarce resources in the face of competing demands. However, the mechanics of co-allocation in a distributed computing environment have received less attention. RSVP [BZB+97] and Beagle [CFK+98a] can be used to signal resource QoS requirements through a network, but they focus on networks and do not address directly how to discover and select (perhaps under application control) appropriate resources from potentially large populations.

Multimedia applications have motivated techniques for allocating both memory and CPU for channel handlers [MIS96] and CPU, bandwidth, and other resources for video streams [NS96, NCN98]. However, these specialized approaches do not extend easily to other resource types. Many approaches deal with collection of network resources only [FGV97, DKPS97, WG98]. Other related work presents generic methods which could be used for heterogeneous resource types [HvBD98, SP98].

The policy issue is investigated within admission control [WG98] and resource sharing [SP98]. For example, in [WG98] effective admission control policy is proposed for booking ahead network services. Admission control is based on a novel application of effective bandwidth theory to the time domain.

In decentralized, wide area systems, a lack of exclusive control of resources, reduced resource reliability, and a larger resource base from which to select candidate resources introduces the problem of co-allocating multiple resources while individual allocation requests may fail. We believe that effective strategies for discovering alternative resources subject to policy variation and reservation failure will be highly application specific, and any solution that embeds this strategy into the basic infrastructure will fail to meet QoS requirements.

The Darwin project at CMU has built a system with some similarities to SNAP [CFK+98a]. However, Darwin deals with network resources only. It assumes that the network is controlled via Darwin-specific protocols (i.e., Beagle) and hence does not accommodate independently administered resources.

In summary, previous work has not focused on the integration of heterogeneous collections of locally administered resources. Furthermore, much of the

effort has been to support network-centric applications, such as multimedia, rather than the more general applications that we seek to support here.

8. CONCLUSIONS

We have presented a new model and protocol for managing the process of negotiating access to, and use of, resources in a distributed system. In contrast to other architectures that focus on managing particular types of resources (e.g., CPUs or networks), our Service Negotiation and Acquisition Protocol (SNAP) defines a general framework within which reservation, acquisition, task submission, and binding of tasks to resources can be expressed for any resource in a uniform fashion.

Our SLA-based model, with hierarchies of intermediaries, emphasizes multi-phase negotiation across the policy domain boundaries that structure the Grid. By identifying three important types of SLA needed for multi-phase negotiation, we show how generic brokering patterns can be deployed and extended with details of new resource types. The use of generic negotiation patterns and the allowance for resource virtualization—by which intermediaries translate from one set of negotiable terms to another—allows for evolution in a resource management architecture. Evolution is an important step toward realizing a permanent global Grid, in which new resource capabilities and application modes must be incorporated into a running distributed system.

Acknowledgments

We are grateful to many colleagues for discussions on the topics discussed here, in particular Larry Flon, Jeff Frey, Steve Graham, Bill Johnston, Miron Livny, Jeff Nick, Alain Roy and Volker Sander. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

Chapter 9

CONDOR AND PREEMPTIVE RESUME SCHEDULING

Alain Roy and Miron Livny
Department of Computer Science, University of Wisconsin-Madison

Abstract Condor is a batch job system that, unlike many other scheduling systems, allows users to access both dedicated computers and computers that are not always available, perhaps because they are used as desktop computers or are not under local control. This introduces a number of problems, some of which are solved by Condor's preemptive resume scheduling, which is the focus of this paper. Preemptive resume scheduling allows jobs to be interrupted while running, and then restarted later. Condor uses preemption in several ways in order to implement the policies supplied by users, computer owners, and system administrators.

1. INTRODUCTION

Condor is a batch job system that allows users to take advantage of both dedicated and non-dedicated computers. Traditionally, batch job systems have allowed users to submit jobs only to dedicated local computers, such as multiprocessor supercomputers and clusters of computers. In addition to such dedicated computers, Condor allows users to submit jobs to computers that are only occasionally available for Condor to access, such as desktop computers belonging to other users or distant computers under someone else's control. We see such computers not as a problem, but as an opportunity, so we call them opportunistic resources.

The ability to use these non-dedicated computers provides users with extra computing power, but also adds complexity to Condor in two ways. The first is the need to remove jobs, called *preemption*, before they are have completed executing in order to meet the needs of owners, users, and administrators and to deal with unplanned outages. The second is the need to deal with the inevitable heterogeneity of computers available to Condor.

1.1 Preemption

In our experience [LL90], computer owners will only allow their computers to run Condor jobs if Condor does not negatively impact their activities. Therefore, Condor will checkpoint and preempt jobs when an owner needs their computer. This could either be when an owner returns to a computer after an absence, or when the computer is busy with some other activity, or for another reason based on the owner's policies. When another computer is available to run the job, Condor will resume the job on that computer. Note that the job can be resumed without loss of work because checkpointing saves the state of the job. This is *preemptive resume scheduling*, and the focus of this paper.

In order for preemption to be most useful, Condor needs to be able to checkpoint jobs so that work is not lost when the jobs are preempted. Condor can checkpoint most jobs without requiring the jobs to be modified if they can be relinked with libraries provided by Condor [LS92]. There are some restrictions on jobs that can use Condor's checkpointing, particularly jobs that spawn new processes or use kernel threads. In practice, many jobs can be relinked to use Condor's checkpointing, and thereby gain the benefits of preemptive resume scheduling.

Preemptive resume scheduling has been instrumental in Condor's success. Jobs can be preempted in order to meet the needs of any of the three types of stakeholders in a Condor system: users who submit jobs, owners of computers, and system administrators. Condor can preempt jobs on the behalf of users when better resources become available. Condor can preempt jobs on the behalf of computer owners to ensure that the owner's policy on sharing the computers is met. Finally, Condor can preempt jobs to meet the policy of the system administrators, who are concerned about the efficiency of the entire Condor pool of computers. In the remainder of this paper, we will discuss how Condor uses preemption in order to meet the goals of these three types of stakeholders.

1.2 Heterogeneity

While it is relatively easy to build a cluster of dedicated computers using identical or very similar types of computers, it is generally not possible to build homogeneous Condor pools which include non-dedicated computers. Computers within a department are often bought at different times and for different purposes, and it is common for them to be a variety of architectures and to have a variety of characteristics. Condor needs to be able to cope with this variety in order to provide the maximum amount of computational power to its users.

In order to deal effectively with this heterogeneity, Condor uses *matchmaking* to match user's jobs with appropriate computers. Both jobs and computers are described with the *ClassAd* (short for classified advertisements) language.

ClassAds provide schema-free descriptions of jobs and resources that is easy to use effectively, yet powerful enough to allow for sophisticated policies and constraints to be expressed by users, owners, and administrators. ClassAds and matchmaking are described in Section 3.

2. THE ADVANTAGES OF PREEMPTIVE RESUME SCHEDULING

Preemptive resume scheduling has several important properties.

- Preemptive resume scheduling allows the scheduler to take advantage of resources that may only be available occasionally. Because jobs can be checkpointed, preempted, and run elsewhere, work done on such a non-dedicated computer is not lost. Instead, this computer has provided an opportunity to accomplish extra work.

- Preemptive resume scheduling relieves the need to do backfilling, which is commonly done in schedulers. Backfilling allows a scheduler to take advantage of holes in the schedule to run more jobs and thereby increase its efficiency. With a preemptive resume scheduler, backfilling becomes much simpler, or even unimportant. It is not critical to schedule certain tasks during specific time-slices. Instead, any task can fill a time-slice, and it can be preempted later. Note that Condor does not use backfilling at all, since it does not expect users to specify time limits for their jobs, nor does it enforce any time limits.

- Preemptive resume scheduling helps to fairly share computers between users. For instance, as we show below, jobs can be preempted when a user that has a higher priority needs access to a computer. When combined with a system that provides dynamic calculation of user priorities, this can ensure that users are treated fairly.

- Preemptive resume scheduling allows high priority jobs to run when a high-priority user demands it by suspending lower priority jobs temporarily while the high priority jobs run. Because of the ability to resume preempted jobs, they can be easily resumed when the high priority job finishes. This is referred to as *computing on demand*.

3. SCHEDULING IN CONDOR

3.1 The Triumvirate

When users submit jobs to Condor, they do not submit to global queues, as they would in many other batch systems [BHKL00]. Instead, Condor has a decentralized model where users submit to a local queue on their computer,

and Condor processes on that computer interact with the Condor matchmaker and the computers that run the job. Interaction with the matchmaker is called matchmaking, and interaction with other computers is called claiming. Both of these interactions are essential steps to run a job.

Note that each computer in a Condor pool runs only a single job at a time, not multiple jobs. Computers with multiple CPUs may run one job per CPU.

Three distinct entities are involved in running a job: the owner of the job, the owner of the computer or computers that run the job, and the administrator of the entire Condor pool. Because each of these entities may have complex policies about running jobs, we can consider them a sort of triumvirate that controls how jobs are run. (The word "triumvirate" comes from the groups of three people that used to rule the Roman empire, such as Caesar, Pompey, and Crassus.) These individual triumvirates arise and fall as jobs are submitted and run.

To understand how matchmaking and claiming work, it is necessary to understand what ClassAds are. Please refer to Chapter 17 or [RLS98] for more information about ClassAds. From those descriptions, recall two important features of ClassAds: ClassAds are schema-free associations between names and expressions. Expressions can be evaluated in the context of another ClassAd, and this is the basis of matchmaking. For instance, a job ClassAd that contains an expression:

```
Requirements =    other.Type == "Machine"
               && other.RAM  >  500
```

says that the job must be matched with something that is a machine with at least 500 units of RAM. ClassAds do not define the units.

3.2 Matchmaking and Claiming

In order to run a job submitted to Condor, there are interactions between three components, as shown in Figure 9.1: the *user agent*, the *owner agent*, and the *matchmaker*. These components represent the triumvirate and ensure that their wishes are followed.

Note that in other Condor literature, you may find the user agent referred to as the *schedd*, the owner agent referred to as the *startd*, and the matchmaker referred to as two programs, the *collector* and the *negotiator*. These are not obvious names, but they persist for historical reasons.

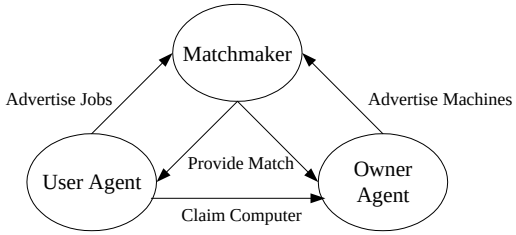


Figure 9.1. The Condor triumvirate which governs where jobs are run and when they are preempted.

3.2.1 Participants in Matchmaking and Claiming

There are three components to the triumvirate, and they are responsible for representing the needs of the people or groups they represent.

- *User Agent* When a user submits a job, it is the user agent's responsibility to make sure that the job runs to completion, assuming no errors in the user's code. It maintains a persistent queue of jobs and a history of past jobs. If other components in Condor fail, it is responsible for retrying the job and informing users of problems that occur.
- *Owner Agent* When a owner decides to add a computer to the Condor pool, the owner agent ensures the owner's policy for how the computer can be used is enforced. It is also responsible for starting jobs that are submitted to the computer.
- *Matchmaker* The matchmaker is responsible for finding matches between user and owner agents, and is also responsible for implementing pool-wide policies that affect the overall performance and stability of the entire pool.

3.2.2 The Process of Matchmaking and Claiming

When a user submits a job to the user agent, it is immediately stored in a persistent queue. This persistent queue allows the user agent to recover should the agent or the computer on which it is running fail. The job is uniquely identified by the name of the user agent, which is unique within the pool, plus a number unique to the user agent. The user agent then sends a ClassAd that informs the matchmaker that it has jobs to run. As long there are jobs that are not running, the ClassAd is sent to the matchmaker every five minutes. The user agent does not rely on stability or persistence in the matchmaker.

Similarly, every five minutes each owner agent in a Condor pool submits a ClassAd that describes the computer it is responsible for. ClassAds are also

sent whenever the state of a computer changes significantly. For instance, the state of a computer can be "idle", meaning that it is available to run jobs, or "owner", meaning that the owner of the computer is using it; when the state changes from idle to owner or vice-versa, the owner agent will inform the matchmaker.

The matchmaker accepts ClassAds from user and owner agents, but does not store them persistently. Instead, the matchmaker discards ClassAds if the job or computer ClassAds are not resubmitted for a while—they are presumed to be unavailable for running jobs if they do not regularly report themselves as available. Should the matchmaker ever crash, it will simply relearn the state of the Condor pool within five minutes. This soft-state mechanism enables reliable operation without complicated persistence mechanisms.

Whenever a job is submitted, or every five minutes, the matchmaker attempts to find matches between jobs and computers. This is called the negotiation cycle. It contacts each user agent that has jobs to run, and obtains the ClassAds that represent the job. It then matches the jobs against each machine ClassAd in the pool to see if it can run the jobs. In order to match, each job's requirements (as given in the job's ClassAd) are evaluated in the context of the machine and the machine's requirements are evaluated in the context of the job. These both must evaluate to true in order for the job to match.

When a match is found, the matchmaker informs both the user agent and the owner agent of the match, then continues matchmaking. It is up to the user and owner agents to claim the match independently of the matchmaker. To do this, the user agent contacts the owner agent. Because the matchmaker was operating on information that may have been out of date, the owner checks to make sure that the job and machine ClassAds still match. This is useful because, for instance, the machine's owner may have reclaimed the machine for his own use since the matchmaker performed that match, and therefore no jobs are able to run on the machine.

If the match can still be made, then the user agent sends the job directly to the owner agent and it begins executing. The user agent monitors the progress of the job, and should any problems outside of the program scope [TL02] occur, such as failure of the owner agent, the user agent will go through the matchmaking process again. If the job fails on its own accord, such as a segmentation fault due to a pointer problem, the user agent records the error and informs the user.

When a job begins running on a computer, a new process on the same computer as the user agent begins running. This process is known as a *shadow*, and it is responsible for implementing Condor's remote I/O capabilities. If a job has been relinked with the Condor libraries in order to be able to be checkpointed, as described above, the shadow will perform two functions. First, it will assist in checkpointing if necessary, either saving checkpoints directly on

the computer the job was submitted from or redirecting them to a specialized checkpoint server. Second, it will redirect I/O on behalf of the application. That is, all I/O performed by the remote application will be performed on the computer that the job was submitted from. This allows jobs to have access to their files no matter where they execute, and also makes them less dependent on free disk space from the remote computer.

Note that running jobs continue even if the matchmaker fails for any reason because all communication is between the computers that submitted the jobs and the computers that are running the jobs. While the matchmaker is unavailable no new jobs can be matched, but all old jobs continue on with no difficulty. When the matchmaker becomes available, jobs will continue running within five minutes.

This is just a sketch of the matchmaking process, and there are several complicating factors. In particular, there are several ways in which jobs can be preempted while running. These are described below.

4. PREEMPTION IN THE TRIUMVIRATE

As described above, running a job in a Condor pool is a cooperative process between three agents that represent the concerns of three people, or groups of people. The user agent represents the person who wishes to run a job, the owner agent represents the person who owns a machine, and the matchmaker represents the maintainer of the Condor pool. The interests of all the people involved must be respected. In part this happens by allowing users and owners to specify requirements for a successful match. Another important aspect is preemption: each of these three parties is allowed to preempt a running job in order to satisfy their requirements.

In order for preemption to be the most effective, checkpointing should be used to allow the state of the program to be saved. When programs meet certain requirements (such as not using long-lived network communication) and can be relinked with Condor libraries, they can be checkpointed so that they can be restarted after they are preempted. Condor can checkpoint jobs when they are preempted from a machine, users can manually request that checkpoints be made, and Condor can periodically checkpoint jobs to ensure against future failures. Not all jobs can be checkpointed, so preemption must be used cautiously. But from a user's perspective, preemption is an inevitable fact of life: even in a dedicated cluster of computers, there may be crashes and disruptions that cause jobs to fail during execution. Note that in a Grid environment, this may happen even more frequently. Being able to checkpoint and react to preemption, for whatever reason it might occur, is an essential part of Condor's approach to reliability.

4.1 User Preemption

There are three ways that a user can preempt a job.

The first way is manually: a user can always remove a job that is either waiting to run or running by executing a command. The job can be checkpointed then rerun, or it can be forcefully terminated, which does not allow the job to be checkpointed and restarted from where it was terminated.

The second way is an automation of the above process. If the user knows some conditions under which the job should be removed—perhaps the user knows that when the job has run for more than 12 hours it must be doing something incorrectly—these conditions can be specified when the job is submitted. Condor's user agent will monitor this condition, and should it ever become true, it will stop the job on behalf of the user. As of Condor version 6.4, the job is not checkpointed, though this is likely to change in future versions.

The last, and most interesting way, is that Condor could periodically rematch the job in order to see if could run on a better machine. If a better machine would be found, Condor could preempt the job on the machine on which it is running, then restart it on the better machine. This is not currently implemented, but Condor is structured to make this easy to implement. Most of the changes would take place within the owner agent which would continue to advertise a job while it was running, but would change the Requirements expression in the ClassAd to add the requirement that "Rank > CurrentRank" to state that it will only run better machines. The ease of this implementation shows the usefulness of the structure of Condor—the owner agent can easily implement a user's policies for jobs. We expect this feature to be implemented in the near future.

Note that, although adding the mechanisms to do this is not difficult, it needs to be done carefully in order to avoid problems. For instance, it is important to ensure that jobs do not "thrash" between computers, always looking for greener grass elsewhere. To do this, a job's requirements either need to specify features that are not affected by running on a machine, such as total memory instead of system load, or a job's requirements need to be updated to restrict a job from running on machines it has already run on.

4.2 Owner Preemption

If an owner wants to remove a Condor job running on his machine, there are two ways for it to happen.

First, Condor can automatically preempt jobs. Owners can write flexible policies that state when jobs should be preempted. For example, an owner could write a policy that when he begins typing at the keyboard, jobs with a small image size are suspended and other jobs are preempted. (Condor calculates the image size for each job and puts it into the job ClassAd, so this

information is available.) If a job is suspended for more than ten minutes (the owner has continued typing on the keyboard), the job must be preempted. The owner can choose to allow or disallow checkpointing before the job is preempted. Of course, various alternative strategies can be used: all jobs could be suspended, all jobs could be preempted, other criteria can be used to decide when it should occur and so on. The owner gets to decide how Condor jobs are allowed to use his resource and Condor enforces his desires.

Second, a user can always request that Condor preempt jobs running on his machine by running a command. This is not usually necessary because of Condor's mechanisms for automatically preempting jobs.

4.3 The Matchmaker: Preemption for Everyone

When the matchmaker is looking for locations to run a job, it can enforce the desires of the pool administrator by attempting to place jobs in order to increase the efficiency of the pool or to favor some users instead of others. The matchmaker also helps owners run jobs that they prefer by preempting other jobs already running. To understand how both of these work, we will describe the matchmaking process in a bit more detail than we described it above.

When the matchmaker is looking for a match for a particular job, it examines the ClassAd for each available computer one at a time, in order. Note, however, that computers continue to advertise themselves while they are already running a job, because they are willing to run "better jobs". This advertising allows currently running jobs to be preempted.

After checking that a particular job and machine match, the matchmaker evaluates several expressions:

- How does the job rank the machine? Job ClassAds specify a rank expression that, when evaluated, can differentiate between different machines. For instance, a job might prefer a faster machine, a machine with more memory, or a machine belonging to the research group that the job's owner is part of. This machine rank allows Condor to meet the user's needs.

- Is there already a job running on the machine? If so, the matchmaker evaluates the machine's rank of the job and compares this rank to the machine's rank of the currently running job. If the new rank is higher, the matchmaker may consider preempting the job. The relies on a rank expression supplied by the machine's owner in the machine's ClassAd to describe which jobs the owner prefers to be run on his machine. This job rank allows Condor to meet the owner's needs.

- If there is a job running on the machine, but it does not have a higher rank than the currently running job, does the owner of the job looking

for a match have a higher user priority than the owner of the currently running job? If so, the matchmaker evaluates the pool administrator's requirements for preempting to see if they are true. The pool administrator may only allow preemption based on user priority when certain conditions are met, such as not preempting jobs when they have not run for very long, in order to avoid thrashing. This allows Condor to meet the system administrator's needs.

This machine will be a candidate for running a job if:

- the job's rank of the machine is better than any seen so far,
- the job's rank is the same as the previous candidate, but running the job on this machine would do "less" preemption (either no preemption, or preemption based on rank instead of preemption based on priority), or
- the rank is the same as the previous candidate, and the same kind of preemption would be performed, but the pool administrator ranks this as a better preemption than the previous candidate. The pool administrator may rank a job preemption as better for any reason, but common reasons might be that a job is smaller (so there will be less network traffic caused by the preemption) or that the job is being run by a more important user.

If this machine is the best candidate so far, it is used for future comparison with machines as matching for a job continues.

As described above, this matchmaking algorithm helps enforce the policies of the pool administrators, the owners of machines, and the users who submit jobs. For owners of machines, if a machine is running a job but finds another job that is better (perhaps it is a job submitted by a collaborator of the machine owner), it can choose to run the new job. Also note that Condor will strongly prefer not to preempt jobs at all, if it can run a job on an idle machine. For pool administrators, Condor lets the administrator impose policies on when jobs can be preempted, and which jobs should be preferentially preempted over other jobs.

5. CONCLUSIONS

By using all of these different types of preemption together, Condor can balance the desires of the users, machine owners, and pool administrators. Preemption is supported throughout Condor by providing both checkpointing mechanisms and opportunities for preemption. Because of this support, Condor can both take advantage of sporadically available resources such as desktop computers and can react to problems such as machines that fail while jobs are running. This flexibility and robustness has been essential to Condor's success.

Chapter 10

GRID RESOURCE MANAGEMENT IN LEGION

Anand Natrajan, Marty A. Humphrey, and Andrew S. Grimshaw
Department of Computer Science, University of Virginia

Abstract Grid resource management is not just about scheduling jobs on the fastest machines, but rather about scheduling all compute objects and all data objects on machines whose capabilities match the requirements, while preserving site autonomy, recognizing usage policies and respecting conditions for use. In this chapter, we present the Grid resource management of Legion, an object-based Grid infrastructure system. We argue that Grid resource management requires not a one-size-fits-all scheduler but an architectural framework that can accommodate different schedulers for different classes of problems.

1. INTRODUCTION

The Legion Project began in late 1993 with the recognition of the dramatic increases in wide-area network bandwidth, truly low-cost processors, and cheap disks looming on the horizon. Given the expected changes in the physical infrastructure, we asked what sorts of applications would people want, and what system software infrastructure would be needed to support those applications. As a result of this analysis, we designed and implemented the Legion Grid Computing system, which is reflective, object-based to facilitate encapsulation, extensible, and is in essence an operating system for Grids. Whereas Globus is a collection of tools from a toolkit [FK99a], Legion provides standard operating system services–process creation and control, interprocess communication, persistent storage, security and resource management–on a Grid. By doing so, Legion abstracts the heterogeneity inherent in distributed resources and makes them look like part of one virtual machine. We feel strongly that having a common underlying architecture and set of necessary services built over it is critical for success in Grids, particularly as the line between computational Grids and data Grids blurs [AVD01]. In this sense, the Legion architecture anticipates the drive to Web Services and the Open Grid Systems Architecture (OGSA) [FKNT02]. There are many papers describ-

ing Legion's core architecture and use (e.g., [GW97, GFKH99, NCWD+01, LFH+03]); in this chapter, we focus on the Legion resource management system.

2. OBJECT PLACEMENT IN A GRID

The scheduling process in Legion broadly translates to placing objects on processors. Scheduling is invoked not just for running users' jobs but also to create any object on a Grid, such as a Grid file, a Grid directory, a Grid application or even a Grid scheduler. After an object is created on a processor, it can perform its tasks, for example, respond to read/write calls if the object is a Grid file, or respond to status requests if it is a Grid job. Therefore, object placement is crucial to the design of the Legion run-time system because it can influence an object's run-time behavior greatly. An improper placement decision may impede an object from performing its tasks, for example, because it cannot start on any processor of a given architecture or because the processor is no longer available. Even if a placement decision ensures that an object starts correctly, it does not guarantee that the decision is beneficial to the user. A good placement decision is certain to vary depending on the object being placed and the user's goals as well as resource usage policies.

Determining good object placements in a large distributed, heterogeneous environment, such as a Grid, is difficult because the underlying system is complex, and because object behavior can be influenced by many different factors, such as system status (number, type, and load of components), hardware capabilities (processor, network, I/O, memory, etc.), interactions between objects and object-specific characteristics (size, location of persistent state, estimated performance, etc.). Factors such as security concerns, fault-tolerance objectives and special resource requirements may place hard restrictions on where an object can be placed. These factors are usually expressed as constraints on the placement decision. In general, finding an optimal placement is prohibitively expensive. Several research efforts, such as Utopia [ZWZD93], NOW [ACPtNt95], Condor [LL90, PL95], Zoom [WASB95], Prophet [Wei95] and others [Cof76, FC90, GY93, WKN+92], have focused on algorithms and systems for near-optimal solutions or optimal solutions to very restricted problem sub-types or user goals [Kar96].

In Legion, we designed a scheduling framework that can accommodate different placement strategies for different classes of applications. In addition to the expected Grid goals of support for heterogeneity and multi-organizational control, the goals included [Kar96]:

- *Support for multiple placement algorithms.* The framework must be flexible enough to be able to incorporate placement algorithms developed by others.

- *Support for user selection of placement.* Users must be permitted to choose the placement approach that best matches their goals.
- *Ease of use.* It should be easy for developers to add new placement algorithms as schedulers in the framework. Additionally, it should be easy for end-users to access the schedulers for performing their tasks. Default placement mechanisms ease the use of the framework for inexperienced users.
- *Ability to cope with uncertain, outdated or partial information.* We expect that in Grids, information may be missing or inaccurate. The scheduling framework in general and the schedulers that are part of it must continue to perform acceptably even when the available system information is less than perfect.
- *Ability to resolve conflicts.* In a system that supports shared objects and resources, conflicts may arise over their use. The framework must have a well-defined resolution behavior in the case of conflicts.
- *Scalability.* The framework should not degrade significantly (or at least degrade gracefully) when the number of processors becomes large or when the requests for placement become more frequent.
- *Low overhead.* The framework should not impose penalties on users who choose not to use it. For users who do choose to use it, the overheads involved in invoking the scheduling process should be small compared to the duration of the task performed.
- *Integration with other Legion services.* As a direct relation to the Legion philosophy of providing an integrated Grid infrastructure, the scheduling framework must cooperate and communicate with other frameworks in Legion, such as those for security and fault-tolerance. The framework must also take into account persistent storage associated with the shared object space in Legion.

In the following sub-sections, we describe the main tasks of the Legion scheduling framework. The goal of this description is not to advocate one placement policy over another. Although we did select a particular placement policy in order to validate (and populate) our framework, we did not and do not claim that placement policy to be optimal or best-suited for all objects.

2.1 Initiating Placement

Placement decisions can be initiated in two ways. In the first case, an object can request the underlying infrastructure explicitly to place (or schedule) other

objects with which it must interact. For example, a user (represented as an object in Legion) may request Legion to run a particular job on a particular machine. In this case, the placement initiation is explicit in two senses: the request to place is a direct implication of issuing the run command, and the placement location is provided by the user directly. Legion does not require a user to initiate placement in the latter sense – an undirected run command transfers the burden of finding a placement location from the user to Legion.

In the second case, placement initiation may be implicit, and therefore must be automatic. In other words, an object, say a user, may access another object without realizing that the latter is currently inactive. In this case, Legion will re-activate the second object automatically, which in turn may require it to be placed on an available processor. The processor chosen in this case may or may not be the same processor on which that object was active previously. Implicit or automatic placement initiation occurs frequently in Legion; in order to conserve resources, Legion may deactivate infrequently-used objects. When a subsequent action does require such objects to be available, Legion will re-activate them. Seemingly mundane Grid operations such as logging in, checking the contents of a Grid directory and issuing a run may cause several objects to be re-activated.

2.2 Preparing for Placement

Regardless of how placement is initiated, preparing for placement involves three tasks. The first task is selecting an appropriate scheduler from the framework. To be most effective, the scheduler chosen must factor in criteria that are important to the user [BW96, Ber99, LYFA02]. Since the scheduler itself may require time and CPU cycles to make a decision, its performance and cost must be weighed against its anticipated benefits. This selection may be made automatically by Legion, or may be specified by sophisticated users who choose to indicate which scheduler, or even which processor, must be used. When placement is initiated automatically, there exists a mechanism for indicating which scheduler to use. This mechanism is captured in attributes of *class objects*, which are managers or factories for creating instances of different kinds of objects. For example, when users decide to *port* an application to Legion, they use a tool that essentially creates an application class object in the Grid. A class object may be associated with any of the schedulers available in the Grid. When a user requests that this application be run, the class object consults its attributes, determines the associated scheduler and invokes this scheduler to perform a placement decision for an instance of the application, namely the run desired by the user.

The second task in preparing for placement is sending the selected scheduler a placement request. Each scheduler may implement a different algorithm and

may require different system information for performing placement. Designing a format for the placement request is a non-trivial task; some may argue that if this problem can be solved the problem of designing a general-purpose scheduler for all classes of applications is made much easier. One approach for designing a placement request format is to design a general description language that is flexible and extensible enough to express most placement requests. The challenge with this approach is actually being able to design a scheduler that takes all possible *programs* that can be written in this language and do something useful. Another approach is to develop a standard interface for all schedulers. Unfortunately, a standard interface often implies being able to express only a small subset of functionality possible just so that the more simplistic schedulers can be accommodated. In Legion, we incorporated both approaches. The scheduling framework required conforming to a standard interface, but we also provided a language for querying the database object that collected information on all processors in a Grid so that other schedulers could be written.

The third task is to specify object-specific placement constraints to the scheduler. In Legion, specific placement constraints are specified as attributes on the associated class objects. Typically, these constraints permit specifying either processors that are suited (or unsuited) for this class object or arbitrary attributes that a processor is expected to possess as well in order to qualify as a match. When a class object receives a request to create an instance, it passes these constraints to the scheduler as part of the placement request. The default scheduler we provided with the system takes these constraints into account when making a decision; however, we do not require all schedulers that were part of the framework to take those constraints into account.

2.3 Performing Placement

Placement is performed by the selected scheduler. The scheduler is clearly the heart of the placement process; however, we recognized that others were better at writing schedulers than we were. We provided a framework wherein experts could write schedulers and plug them into our framework. Naturally, in order to validate the framework as well as provide default placement, we wrote our own scheduler. The main tasks of this scheduler are what we expected of any scheduler:

- *Determine the application requirements.* These are available as constraints passed in by the class object.

- *Determine the resources available.* These are available from a database object, called a *collection*, which can be accessed programmatically as well as by using a query language.

- *Invoke a scheduling algorithm.* Invoking the algorithm results in a schedule. For our default scheduler, we employed a random algorithm. Given that this scheduler was a default, we did not expect it to be used frequently. Moreover, given that we could not predict which objects in a Grid would end up using the default scheduler as opposed to a more appropriate scheduler, we felt that *random* was as good or as bad an algorithm as any.

- *Enforce the schedule.* A schedule is of academic interest unless it results in the object actually being created on some suitable processor. As part of the placement process, the framework must ensure that the schedule generated results in object creation, or if it does not, invoke the scheduling algorithm again, perturbing it so that it generates a different schedule. Alternatively, the framework must communicate its failure clearly back to the class object or the user so that other actions can be taken.

2.4 Gathering System Information

A key component in making any placement decision is gathering the necessary information, such as processor types, OS types and versions, attached devices, available disk space, memory and swap size, CPU load, run queue length, security and reservation policies, network bandwidth, network latency, packet drop percentages, etc. Earlier, we alluded to this step, but assumed that the information was already available when preparing for placement. However, when designing a scheduling framework, we had to design mechanisms to ensure that this kind of information was available to the scheduler. We designed a new object, called a *collection* (similar in spirit to MDS [CFFK01]), which functioned as a database for this information. We felt a collection object was necessary so that a scheduler could find reasonably-current information about available processors in one place instead of contacting every processor in a Grid. In turn, either the collection object polled every processor periodically for system information or processors themselves pushed this data into the collection. Collections are generic repositories of object attributes; collections that specifically store information about processors are associated with schedulers in order to aid placement.

2.5 Gathering Application Information

Accurate and detailed information about the behavioral characteristics of different objects can aid scheduling. In Legion, application-specific information can be specified in a class object, as we discussed earlier. The class object can be given a set of placement constraints that essentially restricts the processors on which its instances can run. Also arbitrary attributes, typically called

desired host properties, can be used to restrict the choice of processors; only processors that possess those properties may be selected. Setting these constraints and attributes can be done at any time in the lifetime of the Grid. An additional manner in which the choice of processors can be constrained is by controlling the platforms on which instances of the class object can run. For example, if a user provides only Solaris and Windows binaries for a particular class object, then instances of that class can never be scheduled on, say, a Linux or SGI machine. Furthermore, the user can instruct a particular run – which creates an instance of the class object – to run on any machine of a particular architecture. Thus, Legion provides users with mechanisms to control the scheduling process with application-level information.

3. MECHANICS OF RESOURCE MANAGEMENT

Legion is both an infrastructure for Grids as well a collection of integrated tools constructed on top of this infrastructure. The basic infrastructure enables secure, dataflow-based, fault-tolerant communication between objects. Communicating objects could be diverse resources, such as applications, jobs, files, directories, schedulers, managers, authentication objects (representations of users in a Grid), databases, tools, etc. The Legion scheduling framework acts as a mediator to find a match between placement requests and processors. The scheduling process in Legion is one of negotiation between resource consumers, i.e., autonomous agents acting on behalf of applications or users or objects, and resource providers, i.e., autonomous agents acting on behalf of processors or machines or resources. By providing mechanisms for specifying security and usage policies, resource providers can control who runs what and when on their processors. Likewise, by specifying constraints and choosing schedulers, users can control how their applications run.

The scheduling framework that exists between the providers and consumers attempts to satisfy the expectations of both the providers and the consumers. In the context of Grid resource management, the main contribution of the Legion project is not the algorithm used by the default scheduler, but the surrounding infrastructure that takes security, fault-tolerance, matching, etc. into account for every single object created in a Grid. The infrastructure enables creating objects, jobs included, on any appropriate processor in a Grid, whether across a room or across the globe. The location transparency gained is a deep-rooted part of the Legion philosophy of providing a single virtual machine abstraction for the disparate resources in a Grid.

The components of the Legion resource management framework are: *class objects*, resource objects (*hosts* and *vaults*), information database objects (*collections*), *scheduler objects*, schedule implementor objects (*enactors*) and *implementation objects* [CKKG99]. Before we examine each component in de-

tail, we will examine their interactions at a higher level (Figure 1). A typical chain of events in a Grid could involve a user initiating a tool to start an instance of an application on a machine. This chain results in a tool object contacting an application class object (to create an instance of this application); which in turn contacts a scheduler (to generate schedules for running this application on a machine); which contacts a collection (to procure information about machines), an enactor (to reserve time on the target machine) and a host object (to start the job on the machine). After the scheduler selects a host object, it contacts the application class object with enough information to start a job instance on the machine.

In the rest of this section, we describe the different objects that participate in resource management. The implementation and interaction of these objects echoes the philosophy we discussed above. However, we regard this set of objects as only one of many possible implementations of that philosophy.

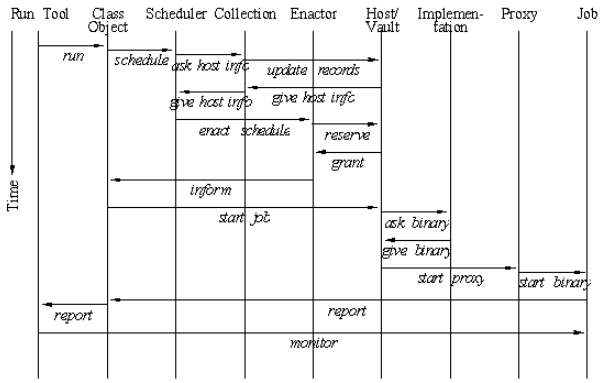


Figure 10.1. The scheduling process in Legion.

3.1 Class Objects

In Legion, class objects define the type of their instances, as in other object-oriented systems, but in addition are also active entities, acting as managers for their instances. A class is the final authority in controlling the behavior of its instances, including object placement. When a Legion Grid is deployed, a variety of class objects are pre-created already. These class objects can create instances of commonly-used Grid objects such as directories, files, schedulers, collections and even other class objects. Later, other class objects may be added to the Grid. For example, a developer may add a new class object that creates instances of an entirely new kind of object, such as a network object.

Alternatively, a developer may refine an existing class object, such as the file class object in order to create a new class object that can create specialized instances, such as matrix or two-dimensional files. Finally, users who port their applications to a Grid typically create, unbeknownst to them, application class objects that are managers for every single run of that application.

All class objects define a *create instance* method, which is invoked during placement initiation. This method may take parameters for an explicit placement or may be called with minimum parameters for an implicit placement. If the placement is explicit, Legion bypasses schedulers, enactors and collections and attempts to start objects on host-vault pairs directly. If the placement is implicit, the scheduling framework is invoked with as much information as available.

3.2 Scheduler and Enactor Objects

A scheduler objects maps requests to resources. As part of this process, the scheduler is given information by the class object about how many instances to create, as well as what constraints apply. Application-specific schedulers may demand and may be supplied with more information about the resource requirements of the individual objects to be created. In addition, a scheduler also requires information about the platforms or architectures on which instances of this class can run. All of this information is procured from the class object.

A scheduler obtains resource information by querying a collection, and then computes a schedule for placing the requested objects. This schedule is passed to an enactor that bears the responsibility of ensuring that the schedule is successful. Each schedule has at least one master version and a list of variant versions. Master and variant versions contain a list of mappings, with each mapping indicating that an instance of the class should be started on the indicated host/vault pair. The master version of a schedule contains the scheduler's best attempt to schedule the requested objects. A variant version differs from a master schedule slightly in terms of the resources selected, representing a poorer scheduling decision to which the enactor can resort if the master fails.

Upon receiving a schedule from a scheduler, the enactor attempts to determine whether the schedule will be successful. In order to do so, it extracts the mappings from the master version, contacts each host/vault pair involved and inquires whether the sub-request on it will be successful. A host/vault pair may choose to reject this sub-request based on its current situation – such a rejection is part and parcel of the negotiation philosophy. If the master version cannot be satisfied because of such rejections, the enactor resorts to the variant versions to schedule successfully. If no version can be made successful, the enactor reports an error and cancels the rest of the scheduling process. If a successful version can be found, the enactor procures reservations from the

host/vault (if the host/vault support it) and reports back to the class object with the successful version.

3.3 Collection Objects

A collection is an object that acts as a repository for information describing the state of the resources in a Grid. Each record is stored as a set of Legion object attributes. Collections provide methods to join them and update records. Typically, host and vault objects join collections, although other objects may also join. Members of a collection may supply their attributes in either a *pull* model or a *push* model. In a pull model, the collection takes on the responsibility of polling its members periodically for updates. In a push model, the members periodically initiate updates to the collection (Legion authenticates the member to ensure it is allowed to update the data in the collection). A push model is more appropriate in a scenario in which the members of a collection may lose and regain connectivity with the rest of the Grid. A pull model is more appropriate in a scenario in which we wish to avoid the update implosion of several members updating a single collection.

Users, or their agents such as schedulers, obtain information about resources by issuing queries to a collection. A collection query is a string conforming to some grammar. Currently, a collection is a passive database of static information, queried by schedulers. Collections can be extended to support the ability for users to install code to compute new description information dynamically and integrate it with existing description information for a resource. This capability is especially important to users of the Network Weather Service [WSH99a], which predicts future resource availability based on statistical analysis of past behavior.

Another use of collections is to structure resources within the Legion system. Having a few, global collections can reduce scalability. Therefore, collections may receive data from, and send data to, other collections. Making collections be members of other collections gives us the flexibility to have a collection for each administrative domain and thus achieve hierarchical structuring of Grid resources.

3.4 Host and Vault Objects

Host and vault objects represent two basic resource types in Legion, processors and disk space respectively. Typically, these objects are started on the same machine, but they are not required to be co-located. A host object encapsulates processor capabilities (e.g., a processor and its associated memory) and is responsible for instantiating objects on the processor. Thus, the host object acts as an arbiter for the processor's capabilities. A host object can represent single-machine systems as well as a queue management system such

as LoadLeveler [IBM01], NQS [Kin92], PBS [BHL+99], or LSF [Zho92]. A vault object encapsulates storage capabilities (e.g., available disk space) and is responsible for storing the persistent state of objects running on that machine. Every Legion object must have a vault to hold its *Object Persistent Representation* (OPR). The OPR holds the persistent state of the object, and is used for migration and for shutdown/restart purposes. When requested by an enactor, a host object grants reservations for future service. The exact form of the reservation may vary by implementation of the host object, but it must be non-forgeable tokens; the host object must recognize these tokens when they are passed in with subsequent requests from the class.

There are three broad groups of host/vault functions: reservation management, object management, and information reporting. Reservation functions are used by an enactor to obtain a reservation token for each sub-request in a schedule. When asked for a reservation, a host is responsible for ensuring that its vault is accessible, that sufficient resources are available, and that its local placement policy permits instantiating the object. A host/vault pair is responsible for managing an object during its lifetime. Object management may involve de-activation and re-activation if requested as well as migration. Migrating an object involves collecting its OPR and transmitting it to some other host/vault pair. Hosts and vaults repopulate their meta-data after reassessing their local state periodically. This reassessment is done by invoking local resource management tools or calls on the underlying machine or queuing system. The resultant meta-data, also called attributes, may be pushed into or pulled by a collection object.

3.5 Implementation Objects

Implementation objects may be viewed as representations of the actual binaries required to run objects on a processor. Every object, whether it be a user's job or a Legion object, requires a binary of the appropriate architecture to run on a processor. Registering these binaries with a class object is the porting process in Legion; the crux of Legion's support for running legacy applications as well as Legion-aware applications is registering binaries with class objects. A class object tracks the implementation objects associated with itself when initiating placement. Therefore, a class that has only Solaris and Windows implementations will never request a schedule containing Linux or SGI machines. When a class receives a viable schedule from an enactor, it communicates with the host/vault objects in order to start objects. Host/vault objects in turn receive the names of the implementation objects for that class, and contact the implementation objects to download the associated binary for running the instance.

Since implementations are objects themselves, they are created in much the same way as any other object. Implementations for hosts and vaults, however, are more basic than implementations of most other class objects. Therefore, host/vault implementations are procured by looking in well-known directories in the Legion installation. Once the host/vault pairs are running, implementations for any other object can be procured from anywhere in the Grid. A minor but interesting point about implementations is that it is perfectly possible and reasonable that the Linux implementation of a particular application class object may actually be started on a Solaris machine. The distinction to remember is that the application's Linux binary happens to be stored on a Solaris machine; therefore the implementation runs on a Solaris machine, but the application binary, when desired, will run only on a Linux machine.

3.6 Proxy Objects

Proxy objects are used to execute legacy application binaries on host and vault pairs and do not play a role in scheduling. However, they are a resource management component because they enable users to employ Legion tools to monitor the progress of a job on a remote machine even though the original job does not respond to Legion requests. Instead, the proxy responds to Legion requests about the status of the job. Since the proxy is not the job itself, it cannot give application-specific status of the job. However, the proxy can provide information such as the name of the machine on which the job runs, the current working directory of the job, the files present in that directory as well as contents of those files at any time, etc.

4. LESSONS LEARNED FROM THE LEGION RESOURCE MANAGEMENT SYSTEM

Several of the key lessons we learned about Grid resource management are captured in the design decisions we incorporated in the scheduling framework. First, in many ways, scheduling should be treated no differently than the other parts of the Grid infrastructure. Although not shown in Figure 1, every object-to-object communication in the scheduling sequence requires the reliability, efficiency, and privacy/integrity of those object interactions not related to scheduling. We chose to implement the scheduling framework using the same policies and mechanisms available to all object interactions – every communication between any pair of objects must go through the Legion protocol stack (see Figure 2 for an example stack), which involves constructing program graphs, making method invocations, checking authorization, assembling or disassembling messages, encrypting messages, retransmitting messages, and so on. Since every communication goes through such a stack, Le-

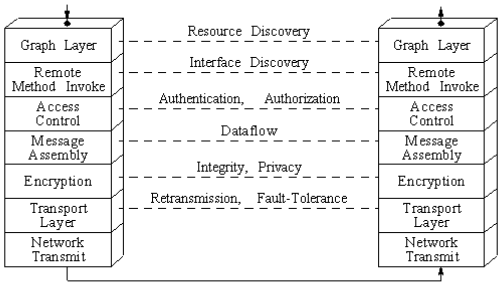


Figure 10.2. The protocol stack in Legion.

gion provides security and fault-tolerance as well as scheduling as part of an integrated resource management framework.

Second, scheduling in Legion is a process of negotiation. Most schedulers view CPU cycles as passive resources waiting to be utilized by the next available job. However, in a multi-organizational framework, a CPU is not necessarily available simply because it is idle. The owner of the CPU – the organization that controls the machine – may impose restrictions on its usage. Therefore, when matching a job to an available CPU, Legion initiates a negotiation protocol which respects the requirements of the job as well as the restrictions imposed by the CPU owner. In other words, we consider site autonomy an important part of the scheduling, or more correctly, the resource management process. Even if a scheduler selects a particular host for running a job, the host may reject the job based on its current policies. Depending on the implementation, the scheduler may investigate variant schedules or may inform the user of the failure to run the job.

Third, the scheduler can be replaced. Each and every component of a Legion Grid is replaceable. Thus the scheduler in the figure can be replaced by a new one that employs any algorithm of choice. Not just the scheduler, but the toolset that uses the scheduler can be changed as well. For example, we wrote a queue object that uses a similar chain of events to mimic the operation of a queuing system. Also, we wrote a parameter-space tool (similar in spirit to Nimrod [ASGH95]) that can start jobs instantaneously or send them to our queue. A Legion Grid can have multiple schedulers or even multiple instances of a particular scheduler. Applications can be configured to use a specific scheduler. Thus, the Legion Grid resource management framework explicitly allows for different schedulers for different classes of applications.

Of course, users can bypass the entire scheduling mechanism, by specifying machines directly or using some non-Legion tool for constructing a schedule for their applications. Bypassing the scheduling mechanism does not mean bypassing security and fault-tolerance, because those functions are at lower levels in the stack. Naturally, if desired, lower levels can be replaced or eliminated as well with the attendant implications.

Fourth, the scheduling infrastructure can be used as a meta-scheduling infrastructure as well. The host object shown in Figure 1 could be running on the front-end of a queuing system or the master node of an MPI cluster. Thus, Legion could be used to select such a host, but subsequent scheduling on the queue or the cluster could be delegated to the queuing system or the MPI system.

When designing the Legion Grid resource management framework, we had a wider definition of resource management than most other distributed systems. We tried to construct a framework within which other parties could write schedulers for different classes of applications. We consciously did not design for only the *classic* applications – long-running, compute-intensive, parallel applications, requiring high performance. Naturally, we did provide a single reference implementation of a scheduler in order to perform resource management on a Legion Grid immediately upon installation. However, we intended this scheduler to be a default – a catch-all scheduler for users who wished to use a Legion Grid as-is. We always intended permitting other schedulers to be part of any Legion Grid.

We did make mistakes in the design of our Grid infrastructure; some of those mistakes were in the scheduling framework. Some of these mistakes are technical, whereas others are psychological. If we were to re-design Legion, here are some lessons we would keep in mind:

People are reluctant to write schedulers. We could not rely on Grid scheduling experts to write schedulers for Legion. Once we learned this lesson, we wrote two new schedulers to complement the default scheduler that already came with every Legion installation. One was a round-robin scheduler for creating instances of files, directories and other objects on a Grid. The round-robin scheduler made quick decisions based on a machine file that was part of its state, thus avoiding expensive scheduling decisions for simple object creation. The second scheduler was a performance-based scheduler for parameter-space studies. This scheduler took CPU speeds, number of CPUs and loads into account for choosing machines on which to run parameter-space jobs.

Writing schedulers deep into a framework is difficult. While we did provide a framework for writing schedulers, a mistake we made was requiring scheduler writers to know too much about Legion internals. Typically, in addition to the scheduling algorithm of interest, a scheduler writer would have to know about schedulers, enactors, hosts, classes and collections; their internal data

structures; the data they packed on the wire for several method calls; and Legion program graphs. The effort required to write such a *deep scheduler* was too much. In essence, we had violated one of our own principles: ease of use. Our mistake lay in making Legion easy-to-use for end-users, but not necessarily so for developers. Once we recognized our error, we wrote a *shallow scheduler*, i.e., a scheduler that was about as complex as the default scheduler but did not require knowing too much about Legion internals. The performance-based scheduler for parameter-space studies mentioned earlier is an example of a shallow scheduler. This scheduler is a self-contained Perl script that requires knowing about the collection object (a database of attributes) and the one command to access it. Not having to know Legion details was a significant advantage in the design of this scheduler.

The lesson we learned from this experience was that a high cost of constructing new schedulers is a deterrent to development. Another lesson we learned was that a high cost of running a scheduler can hurt a Grid as well. Put differently, we learned that a quick and acceptable scheduler is much better than a slow but thorough scheduler.

High scheduler costs can undermine the benefits. In Legion, a scheduler is invoked every time an object must be placed on some machine on a Grid. Given the Legion view of scheduling as a task for placing any object not just a compute object, creating files and directories, implementations and queue services, consoles and classes, all require an intermediate scheduling step. For long, the scheduler that would be invoked for any creation was the default scheduler. While we fully understood the need for different schedulers for different kinds of objects, an artifact of our implementation was that we created only one scheduler – the default one.

The default scheduler's algorithm was complex in two respects. One, the actual processing time took long, especially as the number of machines in a Grid grew. Moreover, the scheduler constructed variant versions for every request just in case the master version did not meet with success. Two, the process invoked methods on too many remote objects. Each method call (or outcall) was a relatively expensive operation. Therefore, even a simple schedule would take too long to generate. Accordingly, we built faster schedulers which perhaps did not find near-optimal and variant schedules, but were far quicker than the default. The round-robin scheduler made fewer outcalls and had a simple algorithm for choosing hosts, but was adequate for scheduling files and directories. Likewise, the shallow scheduler we wrote for performance-based scheduling scheduled parameter-space jobs quickly [NHG02]. It initially spent a few seconds building a schedule, but re-used the schedule for the duration of the application.

Over-complex schedulers are unnecessary. In Legion, we created a sophisticated scheduling framework, but we also implemented this framework

in a complicated manner. In particular, splitting the scheduling process from the reservation process (the scheduler and enactor objects respectively), was overkill. The added flexibility this split gave us was never used, and we believe that it will not be used for a while because complex scheduling techniques, such as co-scheduling, that require reservations are useful for a small subset of applications only [SFT02]. Too many objects were involved in the scheduling process, making it feel like the process had too many moving parts. The failure of any one object could derail the scheduling process, making it hard to create new objects – files, directories, implementations, jobs, etc. – on a Grid.

5. SUMMARY

In this chapter, we discussed the philosophy and mechanisms of the Legion resource management framework. In Legion, resource management is invoked not just for running jobs but also to place other Grid components, such as files, directories, databases, etc. The key element in resource management is placement, i.e., determining on which machine to start running an object. In Legion, placement is a negotiation process between the requirements of users and the policies of resource managers. This negotiation process is carried out by a scheduler which also employs an algorithm to determine which resources of the available ones is most suited for starting the requested object. Every scheduler in Legion implements the negotiation process, although different schedulers may employ different algorithms.

As Grids mature, diverse resources will be included in Grids and Grid resource management will be central to the working of a Grid. We hope that our experience will serve to guide the design of resource managers. In particular, we believe that the pressing challenges that face the Grid community are the design of rich and flexible resource specification languages in order to match resources with requests, and the design of a framework that can incorporate different solutions for different aspects of Grid resource management.

Acknowledgments

We thank the members of the Legion Team at the University of Virginia for their hard work over the years. In particular, we thank John Karpovich, who developed much of the initial philosophy underlying resource management in Legion. This work was partially supported by DARPA (Navy) contract N66001-96-C-8527, DOE grant DE-FG02-96ER25290, DOE contract Sandia LD-9391, Logicon (for the DoD HPCMOD/PET program) DAHC 94-96-C-0008, DOE D459000-16-3C, DARPA (GA) SC H607305A, NSF-NGS EIA-9974968, NSF-NGS ACI-0203960, NSF-NPACI ASC-96-10920, and a grant from NASA-IPG.

Chapter 11

GRID SCHEDULING WITH MAUI/SILVER

David B. Jackson
Cluster Resources, Inc.

Abstract This chapter provides an overview of the interactions of and services provided by the Maui/Silver Grid scheduling system. The Maui Scheduler provides high performance scheduling for local clusters including resource reservation, availability estimation, and allocation management. Silver utilizes the underlying capabilities of Maui to allow multiple independent clusters to be integrated and intelligently scheduled.

1. INTRODUCTION

This chapter touches only briefly on the overall design and capabilities of the Maui Scheduler [Jac03a] and the Silver Grid Scheduler [Jac03b]. It focuses primarily on an item by item review of various Grid scheduling requirements and the facilities provided by Maui to satisfy these needs. On occasion, references to Silver are provided to demonstrate how these facilities may be used in conjunction to provide production quality Grid scheduling. In this chapter we describe our approach using the attributes detailed in Chapter 4. Further information regarding philosophy, design, capabilities, and usage of either Maui or Silver can be found at [Mau, Sil].

1.1 Overview of Maui

Maui is an advanced HPC scheduling system in use at a large percentage of the world's largest and most advanced computing centers. It provides a comprehensive suite of job and resource management tools and policies that reflect the needs of these sites.

Maui is an *external* scheduler, meaning that its use does not mandate use of a specific resource manager. Rather it extends the capabilities of a site's existing resource manager, enhancing the resource manager's capabilities and effectiveness. Key areas of enhancement include job preemption, fairshare, prior-

ity management, scheduling optimization including backfill, quality of service guarantees, resource utilization tracking and limit enforcement, and others. Maui can operate with virtually all of the major HPC resource management systems including OpenPBS [PBS], PBSPro [Jon03b], LoadLeveler [IBM01], LSF [PG], SGE [Sun], SSS [SSS], and BProc [bpr]. End users can continue to interact directly with the resource manager without needing to learn any new concepts or commands. On the other hand, they may also take advantage of various Maui features which either offer them greater information about their jobs and available resources or provides them with greater control over their jobs.

In the Grid scheduling arena, Maui is popular due to its capabilities in the areas of advance reservations, extensive resource availability query support, external job migration, resource charging and quality of service support.

1.2 Overview of Silver

Silver was designed to be a real-world enterprise level Grid scheduler meeting key criteria of being highly efficient, non-intrusive, reliable, and transparent. It takes advantage of the rich capabilities found within the Maui Scheduler to provide load balancing, co-allocation, data scheduling and quality of service guarantees at the Grid level. It allows enterprises to organize geographically distributed compute resources and share these resources in a well-managed manner controlling how, when, and where Grid resources are to be made available to participating users.

Silver provides interfaces which allow end users to submit and track Grid jobs using intuitive and familiar job scripting languages and commands. Support for PBS and LoadLeveler style interfaces are available and additional interfaces are on the way. With these interfaces, users are able to immediately utilize Grid resources without requiring additional training.

At a high level, Silver operates by receiving and queueing Grid level job requests. Silver organizes these job requests and evaluates resource availability by querying the Grid interface of the Maui Scheduler running at each feasible participating cluster. Maui's Grid interface incorporates workload, resource, policy, and priority information to determine if and when a particular request could be satisfied and reports the results back to Silver. Integrating information from all feasible clusters, Silver makes an optimal selection, reserves local resources through Maui, and stages the Grid job and Grid data as needed to the destination clusters.

2. ALLOCATION PROPERTIES

Maui uses the concept of a *reservation* to maintain resource allocations. Support for future allocations, or advance reservations, is probably the single cluster scheduler feature which has attracted the most attention in the realm of Grid scheduling. In this regard, Maui possesses a very mature, optimized, and flexible capability. In general, Maui advance reservations allow a site to set aside a block of resources for various purposes such as cluster maintenance, special user projects, guaranteed job start time, or as a means of enforcing various policies. Internally, a reservation consists of three primary attributes, a *resource expression*, an *access control list*, and a *timeframe*.

The resource expression indicates both resource quantity and type conditions which must be met by resources to be considered for inclusion in the reservation. The expressions may request particular compute nodes by name or may specify a set of conditions which must be met by candidate resources. In addition to identifying which resources to reserve, the resource expression can also specify constraints indicating whether or not the scheduler is allowed to optimize or otherwise modify that reservation's placement in space.

The access control list (ACL) specifies which *consumers* (i.e., jobs, other reservations, etc.) may actually utilize the reserved resources. This ACL allows resource access based on the consumer's credentials and attributes. For example, an ACL may specify that reserved resources may only be utilized by jobs which are owned by user john or user steve and are either submitted to class *lowpri*, are preemptible, or request less than 20 minutes of walltime.

The reservation timeframe indicates the time period over which the reservation should actually block the resources. It may also include optional timeframe constraints which control whether or not the scheduler is allowed to optimize the placement of the reservation in time. It is important to note that all Maui reservations are time based. Thus, when a reservation is created, if an *allocation length* is not explicitly specified, a default allocation length will be selected and utilized in evaluating the feasibility of the reservation request.

2.1 Revocation of an Allocation

Maui reservations can be configured to support revocable or irrevocable allocations depending upon need. In cases where there are strict time constraints on data availability or job completion (i.e. weather modeling for the 5:00 news), an absolute resource availability guarantee is required. With an irrevocable reservation, the resources will be available at the requested time regardless of existing or future workload. Only in the event of significant system failures will the scheduler be unable to fulfill the request. However, in most cases, allocation timeframes need not be absolute and can be supported with a revocable reservation. If an allocation is made with a revocable reser-

vation, the reservation would only be released if it precludes a higher priority request from being granted. Because Maui supports dynamic prioritization, it is possible for a higher priority request to arrive *after* the initial allocation has been granted. Reservation revocability can be configured on a system wide or individual reservation basis and policies can be put in place to make the likelihood of reservation revocation extremely small.

By default, reservations are irrevocable; they are created and maintained until the reservation timeframe has expired or the reservation is explicitly removed by the reservation owner. However, Maui can be configured to allow the reservation to be preempted by higher priority local or remote needs. The reservation can also be configured with a non-time based *deallocation policy* so as to automatically terminate based on a number of triggers described later in the *Deallocation Policy* portion of Section 4.5.

2.2 Guaranteed Completion Time of Allocations

Using reservation time constraints, an allocation can be locked to an exact time, guaranteed to complete before a certain time, or guaranteed to start after a given time. In each case, resources for the allocation are actually reserved for a specific time and then the scheduler attempts to regularly optimize this reservation according to the specified time constraints and current environmental conditions. Note that the degree of freedom on job execution time can be controlled at the Grid scheduler level. Tight constraints provide predictable results; loose constraints allow the scheduler additional freedom to optimize.

2.3 Guaranteed Number of Attempts to Complete a Job

Maui will not attempt to start a job until its prerequisites are satisfied. Silver, as a Grid scheduler, will coordinate the staging of each job with its data needs by intelligently prestaging input data and executables in a *just in time* manner. These features minimize the number of job failures because a job will not even be staged to a given cluster unless its prerequisite conditions are already verified to have been met.

However, Maui also supports the concept of guaranteed job start retry. Using Maui's *defer* mechanism, sites can specify how many times Maui should try to locate resources for and/or start a job before giving up and placing a *hold* on the job. Sites can specify different limits for retries depending on whether it is the scheduler which is unable to start the job or the job which is unable to execute. Sites may also indicate the amount of time they would like to transpire between subsequent retries.

2.4 Allocations Run-to-Completion

Maui can be configured to disable all or a subset of job preemption actions based on job attributes. Clearly, this can include disabling all forms of preemption for all Grid-based workload thus guaranteeing that these jobs run to completion without interference.

2.5 Exclusive Allocations

Jobs can request *dedicated resources* to guarantee exclusive access to resources allocated. These resources will not be shared in any way with other jobs. Dedicated resource access can be requested on a job by job basis or configured to be applied to all jobs of a particular type.

2.6 Malleable Allocations

All aspects of Maui reservations can be dynamically modified including changes to reserved resources in both the time and space dimensions. The feature can be used to support MPI-2 dynamic jobs (where the underlying resource manager also supports this capability) or other forms of dynamic workload. When a request to add resources to an allocation is received, Maui will determine resource availability both on and off of allocated nodes. If satisfactory resources are found, Maui will grow the allocation as needed and return the new resource information to the requestor. When a request to remove resources is received, Maui will currently oblige in all cases, determining the best resources to release, reducing the allocation, and returning the updated resource information.

All allocation modifications are originated by the job, by administrators, or by peer services such as a Grid scheduler. In no case will Maui mandate that a job relinquish resources nor will it initiate a request that a job consume additional resources.

If a job begins consuming resources which exceed its initial resource request, Maui may preempt or even cancel the job depending on the *resource utilization* policy configured. This policy can specify different limits and actions depending on the exact resource limit violated. Alternatively, this policy can be completely disabled allowing unlimited resource limit violations.

3. ACCESS TO AVAILABLE SCHEDULING INFORMATION

Maui will not reveal information regarding existing workload, reservations, policies, or total resources to the Grid scheduler. Instead, when a Grid scheduler considers a particular cluster for use, it formulates a context sensitive resource request incorporating account mapping, minimum duration, and per

cluster type and quantity requirement translations. Maui receives and evaluates this request and incorporates resource, workload, reservation, policy, and priority factors. The result returned to the Grid scheduler is a list of *tuples* with each tuple including a feasible job start time, the total quantity of resources available at this time slot, an availability duration, the cost of these resources, and the quality of information of this report. These tuples reflect all possible start times which can satisfy the job's minimum duration constraint and reflect only those resources which are accessible by the job and guaranteed to be available.

3.1 Access to the Tentative Schedule

The start time tuples returned by Maui provide information regarding all possible availability times for the resources requested. As such, this tuple list provides a complete *tentative schedule* and is very useful in reservation and co-allocation based Grid scheduling.

As an alternative to this model, the Grid scheduler may also request a single *estimated start time* for a given job. The approach incorporates all of the factors listed above but also includes statistical modifications due to *over the horizon jobs* (anticipated jobs yet to be submitted), *wall clock limit inaccuracies*, and *workload profiles*. This query returns a time zone independent estimated job start time and a *quality of information* factor. This query is more applicable to Grid scheduling systems attempting to load balance a number of clusters where time constraints do not mandate resource reservations.

3.2 Exclusive Control

Maui maintains exclusive control over the execution of the workload submitted through the local resource management system. Although the *batch* workload is fully controlled, this control cannot address resource availability issues resulting from system failures or processes launched outside of the scope of the local resource management system such as distributed system maintenance tasks, performance monitors, etc. These non-batch loads are generally insignificant and only minimally affect the quality of the resource availability information provided by Maui.

3.3 Event Notification

Maui supports a generalized event management interface that allows it to both provide *event notification* and subscribe to external events which affect its scheduling decisions. This interface supports both solitary and threshold based events. Using this interface, Maui can respond immediately to changes in environment allowing its scheduling to be highly responsive. Event notifi-

cation is provided for all key events regarding changes to jobs and reservations including creation, start, preemption, completion, and various types of failure.

4. REQUESTING RESOURCES

Maui supports a number of functions which allow a peer service to request resources. These requests include a *query and hold* model, a two phase *courtesy reservation*, and a direct resource reservation interface.

4.1 Allocation Offers

As described in the *Available Resource Query Properties* section, Maui supports resource allocation queries. In these queries, full contextual information regarding the request can be provided and Maui will reply with information indicating if and how it can satisfy this request. The response will include *multiple offers* of resource availability covering all potential allocation start times.

4.2 Allocation Cost or Objective Information

Maui interfaces with QBank [Jaca], Gold [Jacb], and other allocation management systems. These systems allow sites to assign costs to resource consumption based on the resource type, time of day, level of service, requesting user and other factors. In responding to a remote resource availability query, Maui responds with availability tuples which include resource availability time, resource quantity, and *cost* information. Further, Maui also supports features enabled in QBank and Gold which allow coordination of cost information across multiple sites and even tracking and enforcement of resource consumption balances between participating sites.

4.3 Advance Reservation

Previous sections provided an overview of Maui's advance reservation capability. This feature is fully mature, highly optimized, and very flexible. It allows peer services complete control over the scheduling of jobs through time. Jobs can be coordinated to start at any chosen time within the time slots reported by the allocation availability query. Only exactly the resources required will be blocked and they will be blocked only for the exact time required.

Silver uses this capability to guarantee local resource availability, enable coallocation based Grid scheduling, and to coordinate compute resources with input data availability.

4.4 Requirement for Providing Maximum Allocation Length in Advance

Maui is a time based scheduler and as such requires some estimate of execution walltime in order to operate. Credential based walltime limits can be configured based on the queue being used, the user submitting the job, or other factors. If no specified or default walltime limit can be obtained, Maui will assume an adequately long allocation length.

If job execution exceeds its wall clock limit, Maui can be configured to always cancel the job, conditionally cancel the job, or allow the job to run indefinitely. If a job exceeds its reserved allocation time, Maui will, subject to site specified policies and reservation priorities, attempt to shuffle reservations as needed to allow extension of the allocation.

4.5 Deallocation Policy

Maui provides a single step resource allocation request. This step creates a resource allocation which will be valid until job completion and requires no further action on the part of the Grid scheduler. Additionally, Maui also supports the concept of a two phase *courtesy* reservation. A courtesy reservation can be requested with a local scheduler or Grid scheduler specified time limit. Once the courtesy reservation is created, a *reservation commit* request must be received from the Grid scheduler within the time limit. If this commit request is not received, the reservation will be automatically removed. The commit request need only be received once to guarantee that the local scheduler maintains the reservation until job completion.

Additional deallocation policies are available for reservation which are created with an ACL which does not map it to a particular job. These reservations are collectively called *user reservations* and can be used by the Grid scheduler to guarantee resource availability for a particular project or user over a given timeframe. These user reservations can be configured to automatically terminate if historical usage does not meet a specified threshold or if the reservation is defined as a *single-use* reservation and the initial job has completed.

4.6 Remote Co-Scheduling

A Grid scheduler is able to stage remote jobs to a local cluster either directly to Maui through the Maui *remote job migration* facility or indirectly through a Grid middleware system such as Globus. In each case, the job is ultimately submitted to the underlying resource manager which is responsible for managing the persistent job queue. The submitting Grid scheduler is given administrative control over jobs and allocations it creates and is able to modify or destroy any such object.

The Silver Grid Scheduler takes advantage of both Globus and the direct Maui job migration facility in conjunction with advance reservations to enable multi-site resource co-allocation.

4.7 Consideration of Job Dependencies

Maui offers basic job dependency support allowing job steps to be blocked until certain prerequisite conditions are met. These conditions may include events such as step execution, step completion, or step failure for specified prerequisite job steps.

5. MANIPULATING THE ALLOCATION EXECUTION

Maui terms any action which stops job execution prior to completion as preemption. Based on the capabilities of the underlying resource manager, Maui supports *suspend/resume*, *checkpoint/restart*, and *requeue/restart* based preemption. As far as these capabilities are available, Maui will utilize them to support QoS based service guarantees, handle resource utilization limit violations, and/or improve overall system utilization. Maui can be configured to preempt existing low priority workload if additional resources are needed to support specific Grid jobs.

Further, if desired, Grid jobs can also be marked such that they are guaranteed to run to completion without being preempted for optimization purposes. These Grid jobs can also be marked such that they are immune to preemption based on resource utilization violations such as using excessive memory or walltime.

5.1 Preemption

Job suspend operations are supported as far as the capability is available within the underlying resource manager. Most major resource managers including PBS, LoadLeveler, and LSF support this feature. Jobs which are not marked *preemptible* will not be suspended.

5.2 Checkpointing

Job *checkpoint and terminate* and *checkpoint and continue* operations are supported as far as the capability is supported within the underlying resource manager. Most major resource managers including PBS, LoadLeveler, and LSF support this feature. Jobs which are not marked *restartable* and *preemptible* will not be checkpointed. Maui supports the concept of both migratable and non-migratable checkpoint files.

5.3 Migration

Maui supports the concept of intra-domain *job migration* but does not automatically utilize this capability for QoS services, load balancing, or other optimization purposes. While scheduling clusters with resource managers which support this concept, job migration can be utilized by administrators or peer services such as a Grid scheduler. Jobs that are not marked *restartable* will not be migrated.

Maui also supports the ability to import jobs arriving via inter-domain job migration, allowing systems such as Grid schedulers to directly stage remote jobs or portions of remote jobs to the local cluster through Maui. Efforts are currently underway to allow Maui to manage the exporting of local jobs to extra-domain clusters but this functionality is not available as of this writing.

5.4 Restart

Job restart (or requeue) is supported and utilized by Maui for quality of service and optimization purposes. Checkpoint files will be utilized if available. Otherwise, the job will be restarted from the beginning. Jobs which are not marked *restartable* will not be restarted.

6. CONCLUSION

The Maui Scheduler provides a powerful suite of scheduling tools, many of which provide unique capabilities not found elsewhere. The Silver Grid Scheduler has been built on top of this feature set and utilizes it to enable effective enterprise level Grid load-balancing, co-allocation, and general scheduling.

This chapter has introduced some of the key Grid scheduling features currently available. With hundreds of sites now using and contributing to this open project, Maui/Silver continues to evolve and improve faster than ever. To learn about the latest developments and to obtain more detailed information about the capabilities described above, see the Maui homepage at `http://www.supercluster.org/maui` or the Silver homepage at `http://www.supercluster.org/silver`.

Chapter 12

SCHEDULING ATTRIBUTES AND PLATFORM LSF

Ian Lumb and Chris Smith
Platform Computing Inc.

Abstract Scheduling is highly complex in the context of Grid Computing. To draw out this complexity, it makes sense to isolate and investigate key areas of the problem. Here we report on communication attributes between higher- and lower-level scheduling instances. Using Platform LSF as the lower-level scheduling instance, we report on overall agreement and a few points of departure relative to the *de facto* reference on scheduling attributes detailed in Chapter 4. The key concerns involve access to tentative schedules and control exclusivity. While understandable, we show how impractical such ideals prove in the case of production Enterprise deployments; we also challenge the necessity of the schedule-access attribute based on experiences with Platform MultiCluster. Furthermore, experience with the Globus Toolkit™allows us to expose a lowest-common-denominator tendency in scheduling attributes. We encourage re-assessment of communication attributes subject to these findings and broader comparisons. We also urge for integration of isolated scheduling activities under the framework provided by the Open Grid Services Architecture (OGSA).

1. INTRODUCTION

The *de facto* reference for scheduling attributes, detailed in Chapter 4, provides a tangible context for discussing attributes for communication between scheduling instances - but not the mechanisms. Here we consider Platform LSF® as a lower-level scheduling instance in the context of those attributes.

Section 2 presents an attribute-by-attribute, objective assessment of Platform LSF with respect to the set of attributes. Four major attribute areas receive consideration. This assessment reveals misalignment between a few scheduling attributes and Platform LSF; two examples receive consideration in Section 3. Platform LSF integrates with higher-level Grid scheduling instances. This allows us to provide insight on the attributes described in Chapter 4 with the hindsight of practical experience. Thus Section 4 presents cases involving

Platform MultiCluster and the Globus Toolkit. We close (see Section 5) by drawing conclusions based on the objective assessment and practical experience, and add recommendations for future progress.

2. OBJECTIVE ASSESSMENT OF SCHEDULING ATTRIBUTES

Chapter 4 defines four areas of attributes for communication between scheduling instances, grouped into four categories: available-information attributes, resource-requesting attributes, allocation-property attributes, and manipulating allocation execution attributes.

In this chapter we provide an objective assessment of version 5.1 of Platform LSF relative to these attributes, with the section number for Chapter 4 in parenthesis for each attribute. Unless stated otherwise, the primary Platform LSF reference for this Section is [LSF].

2.1 Available-Information Attributes

Section 3 of Chapter 4 identifies three available-information attributes. Here we consider each of these.

Tentative-Schedule Access (3.1)

No *practical* provision for tentative-schedule access exists in Platform LSF. We discuss this attribute in detail in Section 3.

Exclusive Control (3.2)

We appreciate the value of exclusive control to the higher-level scheduling instance. However, *exclusive control* is a loaded statement that we expand on in Section 3.

Event Notification (3.3)

Higher-level scheduling instances have the option to subscribe to an event-notification service. In this case, an event program notifies an identified event receiver regarding a number of predefined events. Although the existing list of events conveys information on Platform LSF as a service, our customers have extended the notification to incorporate other types of events. Platform LSF also maintains a log of events in a designated file, `lsb.events`. An agent can be crafted to monitor and notify regarding events trapped through this mechanism.

2.2 Resource-Requesting Attributes

Section 4 of Chapter 4 identifies seven resource-requesting attributes. Here we consider each of these.

Offers (4.1)

With the notable exception of advance reservation, Platform LSF does not expose potential resource allocations. We further discuss this attribute in Section 3.

Allocation Cost or Objective Information (4.2)

Allocation choice based on competing costs is an unsupported attribute in Platform LSF at this time. Of course, developing a customized scheduling policy provides one solution. Based on experience with Enterprise Grids over more than five years, customer requirements are twofold. First, customers seek allocation entitlements commensurate with acquisition contributions. In other words, if two groups share the capital cost at a 60:40 ratio, their allocation entitlements are to the same proportion. Platform LSF provides several approaches (e.g., fairshare policy, host partitions, etc.) for partitioning. Second, customers seek allocation entitlements opposite an allocation bank [Qba]. This is also available in Platform LSF. These requirements are not necessarily mutually exclusive. Although the first may loose meaning in the broader Grid context, the second remains relevant. As service or utility Grids evolve, the need for economic scheduling increases.

Advance Reservation (4.3)

Platform LSF provides both a built-in and MAUI-integrated advance reservation capability. Both approaches need to align with emerging standards [RS02] in this area.

Allocation Lengths in Advance (4.4)

Platform places this attribute in high regard, and practical experience underlines its value. Failure to specify such lengths renders scheduling policies like backfill completely ineffective.

De-allocation Policy (4.5)

Platform LSF does not require explicit de-allocation; this is done automatically.

Remote Co-scheduling (4.6)

Platform LSF supports co-scheduling by a higher-order scheduling instance. For example, MPI jobs are co-scheduled via the resource leasing capability of Platform MultiCluster. Platform MultiCluster receives attention in Section 4.

Job Dependencies (4.7)

Platform LSF has a built-in support for job dependencies. These dependencies are logical expressions based on 15 dependency conditions. Logical operators permit the combination of conditions into more-complex dependency statements. Platform JobScheduler significantly enhances and extends the dependencies present in Platform LSF. Targeting job flow automation, client-side functionality includes GUIs and APIs for flow editing and management, calendar management, plus a command-line interface. A Java API facilitates client-server interaction via SOAP/XML. The server-side functionality includes a job-scheduler service and tight integration with Platform LSF.

2.3 Allocation-Property Attributes

Section 5 of Chapter 4 identifies six allocation-property attributes. Here we consider each of these.

Revocation (5.1)

Tentative allocations undergo revocation due to various situations including executing a job with a higher priority, withdrawing resources from Grid use, etc. By default in Platform LSF, allocation revocation is not a result of such situations. In fact, allocations remain tentative. It is possible to force revocation. Pending-reason information permits identification of revocation situations.

Guaranteed Completion Time (5.2)

The advance reservation capability of Platform LSF guarantees resource allocations by a given deadline. The same holds for the integration of the Maui scheduler [Mau] with Platform LSF.

Guaranteed Number of Attempts to Complete a Job (5.3)

Platform LSF distinguishes between completion attempts as an execution pre-condition and execution condition. User and queue-level pre-conditioning is facilitated via pre-execution scripts. Users and administrators have complete flexibility in establishing the conditions, number of retries, etc. The queue-level `REQUEUE_EXIT_VALUES` attribute applies retries to jobs that fail during execution. This allows jobs to bypass transient effects, for example, a full

process table on a certain execution host. Re-queued jobs can be associated with specific exit codes, and acted on accordingly.

Run-to-Completion (5.4)

By default, Platform LSF allows allocations on given resources to remain active until the job completes. There are several implicit assumptions. First, allocations do not exceed process resource limits, including wall-clock time, memory limits, etc., or resource limits are not in use. Second, it is possible to eliminate competing scheduling policies. In the case of advance reservation, allocations remain active until the end of the requested time window. Time windows of un-expired advance reservations are extensible.

Exclusive (5.5)

Job dispatch, to hosts on which no other Platform LSF job is running, is a supported attribute. Users specify this requirement on job submission. Additionally, the target queue requires the queue-level EXCLUSIVE attribute set to Y for yes. This is effective when Platform LSF has complete control over an entire host. *Complete control* is a loaded statement; we expand on this in Section 3.

Malleable and Moldable (5.6)

Platform LSF applies a high-watermark approach for resource-usage allocations - i.e., the basis is *expected* consumption requirements. A built-in mechanism allows allocations to decay consumption over time on a per-resource basis, and job-accounting data makes visible actual usage.

2.4 Manipulating the Allocation-Execution Attributes

Section 6 of Chapter 4 identifies four allocation-execution attributes for manipulation. Below we consider each of these.

Preemption (6.1)

Since March 1995, Platform LSF has provided a preemptive-scheduling policy. This allows resource re-allocation to higher-priority workloads at the expense of lower-priority workloads. In general, preempted workloads retain resources (e.g., swap space, software licenses, etc.) even though they are in an inactive state. Customer feedback suggests that hoarding resources, whilst in a dormant state, is undesirable. A poignant case relates to software licenses. In some industries, software licenses are the scarce commodity, with each instance bearing a significant cost. To respond to this requirement, Platform

generalized its approach towards preemption. Recent versions of Platform LSF allow preemption of any slot-based resource allocation.

Checkpointing (6.2)

Assuming the application can be checkpointed, Platform LSF provides an interface to support checkpointing through one-or-more means, e.g., from kernel to application level.

Migration (6.3)

Platform LSF provides a mechanism for migration that can be addressed by a higher-level scheduling instance. This interaction has been demonstrated in the case where Platform MultiCluster 4 is the higher-level scheduling instance.

Restart (6.4)

Platform LSF provides an interface to support restart through one-or-more means. Depending on the mechanism used (e.g., kernel-level checkpointing), restart may be limited to the same host or a host of the same architecture.

3. PRACTICAL CONSIDERATIONS FOR SCHEDULING ATTRIBUTES

In Section 2 we provided an objective analysis of Platform LSF relative to the scheduling attributes identified in Chapter 4. On balance, Platform LSF aligns well with the identified attributes. Here we focus attention on two significant points of departure.

3.1 Tentative-Schedule Access

An available-information attribute identifies the need for tentative-schedule access. We will not argue the merit of such an attribute. However, experience suggests that such access is often impractical in real-world situations. Furthermore, experience with Platform MultiCluster illustrates that higher-level scheduling instances do not always require such schedule access.

Scheduling in Platform LSF is event driven in real time. Every scheduling cycle involves the arbitration of allocation requests opposite available resources subject to one-or-more policies plus constraints. Some policies (e.g., fairshare) are highly dynamic in nature, with required calculations on each scheduling cycle. In practice, scheduling cycles occur at 1-minute intervals, by default. Customers use Platform LSF to manage workloads of half-a-million jobs from thousands of users across thousands of servers. With scalability re-

quirements like these, tentative-schedule access is of questionable value - even though Platform LSF can provide this information.

Platform LSF is supportive of co-scheduling via the resource-leasing capability in Platform MultiCluster. Resource leasing proves that co-scheduling does not require schedule access. The same applies to the job-forwarding model of Platform MultiCluster. We discuss both resource leasing and job forwarding in Section 4.

3.2 Exclusive Control

Logicians use the phrase *necessary and sufficient* in expressing conditions for propositions. It is convenient to use this phrase for the proposition *exclusive control* in the context of resource management. Platform LSF provides necessary *but not* sufficient measures for exclusive control.

Platform LSF provides a level control over all resources it manages, that is, it can limit the number of jobs it dispatches to a host. However, Platform LSF executes in user space as a privileged user (Figure 12.1). This means its control is *not* exclusive - only the kernel of the relevant operating system holds the guarantee of exclusivity. In this sense, Platform LSF provides necessary but not sufficient measures for exclusive control. The same applies to any other lower-level scheduling instance that executes in user space, including Altair PBSPro (detailed in Chapter 13), Condor (detailed in Chapter 9), IBM LoadLeveler [IBM01], and Sun Grid Engine [SGE]. This user-versus-kernel-space distinction is explored in detail in [Lum01].

Directly engaging the kernel offers the potential of necessary and sufficient measures for exclusive control. Here we identify two use-case scenarios: slot based and threshold-based.

3.2.1 Slot-Based Scenarios

Many vendors offer support for slot-based scenarios (Figure 12.1 and Table 12.1 with Type=Slot). This applies to any slot-based resource, for example CPUs. Thus hard (static) and soft (dynamic) partitioning technologies exist. In all cases, this technology allows resource allocations to be bound to a set of slot-based resources.

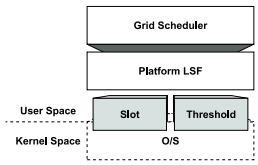


Figure 12.1. Scheduling-instance hierarchy.

3.2.2 Threshold-Based Scenarios

Many vendors offer support for threshold-based scenarios (Figure 12.1 and Table 12.1 with Type=Threshold). This applies to many threshold-based resources, including CPUs, RAM, etc. In all cases, this technology allows resource allocations to be bound to a percentage allotment of threshold-based resources.

Platform LSF integrates with the technologies identified in Table 12.1; the final column of this table identifies specific instances.

Table 12.1. Slot- and threshold-based use case scenarios by vendor.

Vendor	Operating Environment	Product	Type	Nature	Binds
Aurema	Various	ARMTech[a]	Threshold	Dynamic	CPU
HP	HP-UX 11.x	vPars[b]	Slot	Dynamic	CPUs
	HP-UX	PRM, WLM	Threshold	Dynamic	CPU, RAM, process count
	Tru64	ARMTech	Threshold	Dynamic	CPU
IBM	AIX	WLM	Threshold	Dynamic	CPU, RAM, I/O
Quadrics	QsNet	RMS[c]	Slot	Quasi-static	Clustered, low-CPU-count SMPs
Scyld	Linux	Beowulf[d]	Slot	Dynamic	Clustered, commodity systems
SGI	IRIX 6.5.x	Partitions	Slot	Static	System boards
	Linux[e]	Cpumemsets	Slot	Dynamic	CPUs, RAM
	IRIX 6.5.x[f]	Cpusets	Slot	Dynamic	CPUs, RAM
Sun	Solaris[g]	Dynamic System Domains	Slot	Quasi-static	System boards
	Solaris	Psrset[h]	Slot	Dynamic	CPUs
	Solaris	SRM[a,i]	Threshold	Dynamic	CPU, Virtual memory, process count
VMWare	Linux, Windows	VMWare	Slot	Quasi-static	O/S instances

[a] Evolved from ShareII.
[b] Requires a seed CPU per virtual partition on boot-up.
[c] Integration in Platform HPC for HP AlphaServer SC.
[d] Applicable to MPI applications; integrated with Platform LSF [LSF].
[e] As provided on the Altix platform.
[f] Supports attach/detach; Works with topology-aware scheduling; Integration in Platform HPC for SGI.
[g] Sun Enterprise xxK only.
[h] Integrated with Platform LSF [LSF].
[i] Integration described in [DL03].

4. REAL-WORLD EXPERIENCES WITH SCHEDULING ATTRIBUTES

Platform LSF is used in conjunction with higher-order scheduling instances. This provides an opportunity to reflect on Chapter 4 with the hindsight of practical experience. We present two examples. In the first example, Platform MultiCluster serves as the higher-level scheduling instance for Enterprise Grids. The second example we base on experience with the Globus Toolkit® for Partner Grids.

4.1 Platform MultiCluster

Platform MultiCluster provides the first example of a higher-level scheduling instance that interoperates with Platform LSF. Platform MultiCluster [LSF] allows virtualization of clusters based on Platform LSF into an Enterprise Grid. Enterprise Grids typically involve a single organization that spans multiple geographic locations, with the organization's firewall providing the primary means of security. In place since late 1996, customers make use of this solution in their production deployments across a variety of industries. Platform LSF inventories all resources, e.g., desktops, servers, supercomputers, software licenses, etc., on a per-cluster basis. Because the discovery aspect is well-bounded, Grid-wide scheduling is a core competence of Platform MultiCluster. There are two models supported (1) job forwarding and (2) resource leasing.

4.1.1 Job Forwarding

Send/receive queues allow workload exchange (forwarding) between cooperating clusters. Queues serve as cluster-wide entities for managing workload against a rich array of scheduling policies. In this model, sites retain a very high level of local autonomy - i.e., sites selectively identify resources available for Grid use. Typically used to ensure maximal utilization of all compute resources across an entire enterprise, job forwarding applies to mixed workloads, e.g., serial applications, parametric processing, plus shared and distributed memory parallel applications. Supported scheduling policies include advance reservation, backfill, fairshare, memory and/or processor reservation, preemption, etc.

4.1.2 Resource Leasing

Sites can earmark resources for on-demand use by other sites participating in an Enterprise Grid. While under use, the demand-initiating site manages these *leased* resources. With an affinity for Grid Computing, this mode of aggregation can be used to support complex resource requirements, for exam-

ple, co-scheduling an MPI Interface distributed memory parallel application across multiple sites in an Enterprise Grid. As in the case of job forwarding, resource leasing can apply other policies, and make use of the extensible scheduler framework.

Version 4.x of Platform MultiCluster uses a Resource Reservation Protocol (RRP) [Xu01]. With Enterprise Grids, modularity, and extensibility all in mind, Platform re-architected Platform LSF. This formed the basis of version 5.x of Platform LSF. By refactoring the internals, resource leasing became possible in addition to job forwarding. Whereas job forwarding passes resource-allocation requests to remote clusters until is finds a match, a modified version of the RRP applies to resource leasing. All of this is possible without access to a tentative schedule. Not only is it possible, this approach has met the production scalability requirements of Enterprise customers in several industries.

On balance, this Platform MultiCluster - Platform LSF combination provides a working example of Chapter 4's interaction between lower and higher-level scheduling instances. Again, the primary departures relate to tentative-schedule access and exclusive control.

4.2 The Globus Toolkit

Multiple, virtual organizations cause a disruptive transition from Enterprise to Partner Grids. In the Partner Grid case, the Enterprise firewall becomes meaningless, and resource discovery emerges as a key challenge. The Globus Toolkit® addresses these extra-Enterprise tensions in security and discovery. Our focus here is on attributes for communication between scheduling instances. The toolkit *facilitates* interaction between scheduling instances through its Globus Resource Allocation Manager (GRAM) [CFK+98b] component; it does not, however, provide lower- or higher-level scheduling instances. Based on experiences with Platform Globus, we share experiences with GRAM in the context of scheduling attributes.

Platform Globus is a commercially supported distribution of version 2.2.4 of the Globus Toolkit from the Globus Project. Platform adds value through enhancements - improved packaging and installation, improved interoperability with Platform LSF, etc. - technical support, documentation, and the availability of professional services for Grid planning, deployment and ongoing management. Our primary observation is of a tendency towards the Lowest Common Denominator (LCD). We observe this effect in three areas: the RSL Resource Specification Language, job state information, and information services.

4.2.1 Resource Specification Language (RSL)

Out of necessity, the toolkit's RSL supports only a subset of options available in Platform LSF [PG]. In the case of job submission, RSL supports queue

and project names, CPU-time and memory limits, I/O redirection, processor count, plus executable and argument specification. In version 2.0 of the toolkit, all other specifications are out of scope and ignored. Subsequent versions improve on this. Although the RSL-to-native conversions are extensible, the tendency is always towards the LCD - and hence an issue of completeness.

4.2.2 Job State

Table 12.2 presents non-uniqueness and incompleteness in the case of job state [PG]. The first column provides the state identifier used by the Globus Toolkit while the second that used by Platform LSF. The Platform LSF state terms are detailed in [LSF]. Again, a clear tendency towards the LCD.

Table 12.2. Job-state mapping between the Globus Toolkit and Platform LSF.

Globus Toolkit	*Platform LSF*
Active	RUN
Pending	PEND
Suspended	USUSP or PSUSP or SSUSP
Done	DONE
Failed	EXIT or UNKWN or ZOMBI
–	WAIT

4.2.3 Information Services

The toolkit includes Monitoring and Discovery Service (MDS2) schemas for lower-level scheduling instances like Platform LSF. Our experience was that the default schema ignored valuable information routinely provided by Platform LSF. In Platform Globus, we extended the MDS schema for Platform LSF [PG], and made it available - thus avoiding the tendency towards the LCD. This is also an issue of completeness.

The attributes in Chapter 4 acknowledges the issue of completeness, but does not address uniqueness.

5. SUMMARY

We are supportive of Chapter 4's intended purpose - i.e., defining the attributes of a lower-level scheduling instance that can be exploited by a higher-level scheduling instance. Major areas of concern remain:

Access to a tentative schedule

Platform LSF can provide visibility into its tentative schedule. However, practical considerations based on Enterprise deployments illustrate the imprac-

ticality of such an attribute. Furthermore, Platform MultiCluster (higher-level) - Platform LSF (lower-level) interaction proceeds *without* tentative-schedule access. This applies to scheduling within a cluster (the job-forwarding model of Platform MultiCluster) and to co-scheduling between clusters (the resource-leasing model of Platform MultiCluster).

Exclusive control

We considered the exclusive-control attribute in some detail. In general, such a notion is not realistic without deeper interactions. This amplifies the presence of a hierarchy of scheduling instances through real-world examples (e.g., Chapter 4 and [DL03]).

The lowest common denominator tendency

Based on experience with Platform Globus, we gave three examples of this tendency. All three examples relate to completeness and one additionally to uniqueness. Perpetually extending working documents (e.g., Chapter 4) and implementations underlines this as an impediment to progress. Reconciliation services [FG03] show promise as an alternative.

These conclusions suggest modest and major calls to action. The modest call to action is to revisit Chapter 4 with the above conclusions in mind. As others apply Chapter 4 against their own higher-level and lower-level scheduling instances, as detailed in Chapters 13 and 11, such conclusions gain broader exposure. Identification of missing attributes will also be possible. Grid Computing demands such scrutiny for real progress. Recent efforts on a Service Negotiation and Access Protocol (SNAP) [CFK$^+$02] are also of benefit in this iterative process. Chapter 4 is based on a significant deliverable of the Scheduling Attributes Working Group, in the Scheduling and Resource Management Area of the Global Grid Forum (GGF). There are many other GGF groups working on scheduling and resource management - some formed recently, and some yet to emerge. Even though this is a GGF focus area, the groups operate independently. The Open Grid Services Architecture (OGSA) [OGSa] provides the context for integrating these independent activities, and this is the major call to action.

Chapter 13

PBS PRO: GRID COMPUTING AND SCHEDULING ATTRIBUTES

Bill Nitzberg,[1] Jennifer M. Schopf,[2] and James Patton Jones[1]
[1] *Altair Grid Technologies*
[2] *Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract The PBS Pro software is a full-featured workload management and job scheduling system with capabilities that cover the entire Grid computing space: security, information, compute, and data. The security infrastructure includes user authentication, access control lists, X.509 certificate support, and cross-site user mapping facilities. Detailed status and usage information is maintained and available both programmatically and via a graphical interface. Compute Grids can be built to support advance reservations, harvest idle desktop compute cycles, and peer schedule work (automatically moving jobs across the room or across the globe). Data management in PBS Pro is handled via automatic stage-in and stage-out of files. The PBS Pro system has numerous site-tunable parameters and can provide access to available scheduling information, information about requesting resources, allocation properties, and information about how an allocation execution can be manipulated.

1. INTRODUCTION

The Portable Batch System, Professional Edition (PBS Pro), is a flexible workload management and batch job scheduling system originally developed to manage aerospace computing resources at NASA. PBS Pro addresses issues of resource utilization in computing-intense industries and forms the basis of many Grid computing projects.

The PBS Pro software includes capabilities that cover the entire Grid computing space: security, information, compute, and data. We look at the Grid capabilities of PBS Pro 5.3 circa March 2003, as well as how they relate to the scheduling attributes detailed in Chapter 4.

2. HISTORY

PBS has been used in the areas of workload management and Grid computing over the past decade. In the early 1990s NASA needed to replace its outdated NQS batch system but found nothing suitable on the market. Hence NASA led an international effort to generate a list of requirements for a next-generation resource management system. The requirements and functional specification were soon adopted as an IEEE POSIX standard [IEE94]. Next, NASA funded (via the R&D contractor MRJ/Veridian) the design and development of a new resource management system compliant with the standard. Thus, in 1993 the Portable Batch System was born [Hen95]. PBS began to be used on distributed parallel systems and replaced NQS on traditional supercomputers and server systems. Eventually the industry evolved toward distributed parallel systems, taking the form of both special-purpose and commodity clusters. The PBS story continued when Veridian released the professional edition of PBS (PBS Pro), an enterprise-quality workload management solution. Most recently, in January 2003, the PBS technology and associated engineering team were acquired by Altair Engineering, Inc., and set up as a separate, subsidiary company (Altair Grid Technologies) focused on continued development of the PBS Pro product line and Grid computing.

In the mid-1990s PBS was selected as the enabling software for Grid computing (then called *metacomputing*). Examples such as the NASA Metacenter (1996-1997 [Jon96, Jon97a]), the Department of Defense Meta-queueing Project (1997-1998 [Jon97b, Jon98]), and NASA's Information Power Grid (1998-2003+ [JGN99]) demonstrate this capability. As a founding participant of the Global Grid Forum (GGF, see also [GGF]), and co-director of the GGF Scheduling Area, the PBS Pro team has committed to furthering Grid computing technologies.

3. GRID CAPABILITIES

Workload management software such as PBS Pro is a key component of Grid computing. It is middleware technology that sits between compute-intensive or data-intensive applications and the network, hardware, and operating system. The software aggregates all the computing and data resources into a single virtual pool. It schedules and distributes all types of application runs (serial, parallel, distributed memory, etc.) on all types of hardware (desktops, clusters, and supercomputers and even across sites) with selectable levels of security. An overview of the basic Grid capabilities (security, information, compute, and data) is provided in this section. Details of features can be found in the PBS Pro documentation [Jon03a, Jon03b].

3.1 Security

Perhaps the most fundamental capabilities of Grid infrastructure are secure authentication (proving one's identity) and authorization (granting permission). The security capabilities of PBS Pro cover both user and host authentication as well as authorization.

Internally, authentication and authorization are user name based (UNIX or Windows login). Authentication uses standard UNIX and Windows security (with additional access checks based on stringent hostname-IP address rules). Authorization is handled by complex access control lists (ACLs), which permit access restriction (or permission) via user, group, system, and network.

X.509 certificates, the *de facto* Grid standard for identification and authentication, are also supported. PBS Pro can pick up the user's certificate at job submission and automatically create a proxy on the execution nodes assigned to that user's job. The distinguished name (DN) from the certificate is carried with the job throughout its lifetime and is written to the PBS accounting logs as part of the job accounting record. If a user's certificate expires, PBS Pro will place the job on hold and notify the user to renew the certificate.

Furthermore, user identity mapping between sites is handled by a mapping function (and can be set up similarly to the gridmap file used as part of the Globus Toolkit [FK97, GLO]).

3.2 Information

If security is the first fundamental capability of Grid infrastructure, then information management is a close second. Access to the state of the infrastructure itself (e.g., available systems, queue lengths, software license locations), is required to support automatic aggregation of Grid components as well as optimizing assignment of resources to Grid activities. PBS Pro monitors both resource state and common workload management information.

The PBS Pro system monitor and job executor daemon processes (MOMs) collect real-time data on the state of systems and executing jobs. This data, combined with less dynamic information on queued jobs, accounting logs, and static configuration information, gives a complete view of resources being managed. PBS protects this information with ACLs, which allow the PBS manager to ensure that, for example, only the owner of a job can view its current status. The node-utilization data collected by the PBS MOMs can be viewed graphically by using the `xpbsmon` command. Specifically, current node availability and status, node CPU and memory utilization, and assigned jobs are displayed by default. Other indices may be selected by the user.

This data can easily be integrated with larger Grid infrastructure databases (such as the information services within the Globus Toolkit). For example, NASA's Information Power Grid [JGN99] both pushes and pulls

data from PBS Pro into the Globus Toolkit Monitoring and Discovery Service (MDS2) [CFFK01, MDS].

3.3 Compute

In addition to traditional workload management capabilities, specific features of PBS Pro address the compute aspects of Grids. These include *advance reservation* support, *cycle harvesting*, and *peer scheduling*.

An *advance reservation* is a set of resources with availability limited to a specific user (or group of users), a specific start time, and a specified duration. Advance reservations can be used to support co-scheduling, especially among diverse types of Grid resources, for example, one can reserve all resources necessary for tomorrow's vehicle crash test experiment: computer cycles, network bandwidth, crash test database access, visualization systems, and the crash test facility itself.

Advance reservations are implemented in PBS Pro by a user (or a higher-level Grid scheduler) submitting a reservation with the `pbs_rsub` command (or API function). PBS Pro then checks to see whether the reservation conflicts with currently running jobs, other confirmed reservations, and dedicated time. A reservation request that fails this check is denied by the scheduler. Once the scheduler has confirmed the reservation, a queue is created to represent the reservation. The queue has a user-level access control list set to the user who submitted the reservation (or as specified by the higher-level scheduler) and any other users the owner specified. The queue then accepts jobs in the same manner as normal queues. When the reservation start time is reached, the queue is started. Once the reservation is complete, any jobs remaining in the queue or still running are deleted, and the reservation is removed from the server.

Cycle harvesting of idle workstations is a method of expanding the available computing resources by automatically including unused workstations that otherwise would be idle. This is particularly useful for sites that have a significant number of workstations that are unused during nights and weekends (or even during lunch). With this feature, when the *owner* of the workstation isn't using it, the machine can be configured to run PBS Pro jobs. If a system is configured for cycle harvesting, it becomes available for batch usage by PBS Pro if its keyboard and mouse remain unused or idle for a certain period of time, or if the system load drops below a site-configurable threshold (i.e., the workstation is shown to be in state *free* when the status of the node is queried). If the keyboard or mouse is used, the workstation becomes unavailable for batch work; PBS Pro suspends any running jobs on that workstation and does not attempt to schedule any additional work on it until the state changes.

Peer scheduling is a PBS Pro feature that enables a site (or multiple sites) to have different PBS Pro installations automatically run jobs from each other's queues. This provides the ability to dynamically load-balance across multiple, separate PBS Pro installations. These cooperating PBS Pro installations are referred to as *Peers*, and the environment is a peer-to-peer computational Grid environment. When peer scheduling is enabled and resources are available, PBS Pro can *pull* jobs from one or more (remote) *peer servers* and run them locally. No job will be moved if it cannot run immediately. When the scheduler determines that a remote job can run locally, it will move the job to the specified queue on the local server and then run the job. Since the scheduler maps the remote jobs to a local queue, any moved jobs are subject to the policies of the queue they are moved into. If remote jobs are to be treated differently from local jobs, this can be done on the queue level. A queue can be created exclusively for remote jobs, and this will allow queue-level policy to be set for remote jobs. For example, one can set a priority value on one's queues and enable sorting by priority to ensure that remotely queued jobs are always lower (or higher) priority than locally queued jobs.

3.4 Data

PBS Pro has long supported the most basic capability for implementing a data Grid: file staging. Users of PBS Pro can specify any number of input and output files needed by their application at job submission time. The PBS Pro system automatically handles copying files onto execution nodes (stage-in) prior to running the job, and copying files off execution nodes (state-out) after the job completes. PBS Pro will not run a job until all the files requested to be staged-in have successfully been copied. Multiple transport mechanisms are offered, including rcp, scp, and GridFTP [ABB+02a].

The file staging feature of PBS Pro also supports the Globus Toolkit's Global Access to Secondary Storage (GASS) software. Given a complete stage-in directive, PBS Pro will take care of copying the specified input file over to the executing Globus Toolkit machine. The same process is used for a stage-out directive. Globus mechanisms are used for transferring files to hosts that run Globus; otherwise, the normal PBS Pro file transport mechanism is used.

3.5 Other Grid-Related Capabilities

Other Grid-related capabilities of PBS Pro include interfaces to Grid computing environments such as the Globus Toolkit [FK97, GLO] and UNICORE [UNIa].

For example, PBS Pro can serve as a *front end* to Globus, permitting the user to submit jobs requesting Globus resources using the normal PBS Pro commands. When such a job is received, PBS Pro will translate the requests

into a Globus job, and then submit it to the requested site. In addition, PBS Pro can serve as a *back-end* to Globus, receiving jobs from Globus and running them according to local policy.

PBS Pro also acts as a *back end* system for the UNICORE Grid environment. Thus computational Grids built on UNICORE (such as the European DataGrid project [EUR]) can (and do) use PBS Pro as the underlying batch system.

4. ATTRIBUTE BY ATTRIBUTE ASSESSMENT

To help compare scheduling systems, in this section we detail PBS's approach using the attributes defined in Chapter 4. These attributes were defined to aid in characterizing the features of a local resource management system that can be exploited by a Grid environment. They are grouped into four categories: access to available scheduling information, described in Section 4.1; information about requesting resources, described in Section 4.2; allocation properties, discussed in Section 4.3; and information about how an allocation execution can be manipulated, described in Section 4.4.

4.1 Access to Available Scheduling Information

The most basic information a local resource management system can express is the status of the current resource usage, and the upcoming schedule of the jobs.

In PBS Pro, any user can access basic information about the queues and their status using the `qstat` command. This provides a straightforward interface to basic data about the progress a job is making in the queue, and users can extrapolate possible starting times from this data. In addition, administrators of a PBS Pro installation have access to more detailed information about the order in which jobs will be run according to the site-defined scheduling policy.

When jobs are submitted to PBS Pro, an email address is specified for event notification. The user may specify that email be sent to this address when the job starts, ends, or aborts (the default if not specified). Alternatively, the user may request no email notification be performed at all.

4.2 Requesting Resources

Local scheduling systems differ not only in terms of the functionality they provide but also in the type of data used to request resources. The attributes in this category include data about allocation offers, cost information, advance reservation data, de-allocation information, co-scheduling data, and job dependencies allowed.

PBS generates a single resource solution to a *run my job* request, whether it is a standard batch job request or a request for an advance reservation (see

Section 4.3). Once a job is submitted, it will run unless a user cancels it, the system goes down, or it is preempted (see Section 4.4).

Whether a request to run a job includes an estimated completion time from the requestor is configurable. If this data is not included, however, getting a high utilization of the resources is extremely difficult. For example, the backfilling feature needs this data. this functionality is not a requirement of the PBS Pro infrastructure, and it can be configured differently for different queues.

PBS Pro allows the consideration of job dependencies as a part of the job submission process in a variety of flavors. It is simple for a user to specify a number of options, including: Run job Y after job X completes; If job X succeeds, run Job Y, otherwise run job Z; Run Y after X whether X completes or not; Run job Y only after a certain time; Run job X anytime after a specified time.

PBS Pro allows co-scheduling by simply configuring the queues of the system. Hence, a site can have the added benefit of co-scheduling not as a special case but as the norm, so extensive debugging of a rarely used allocation process isn't needed.

4.3 Allocation Properties

Different local resource management systems allow different flexibility with respect to how an allocation is handled. This includes whether an allocation can be revoked, what guarantees are made with respect to completion times, start attempts, and finish times, and whether allocations can change during a run.

In PBS Pro, the user (i.e., the allocation requestor) or an administrator can revoke any allocation (using the `qdel` and `pbs_rdel` commands), both while the job is queued and while the job is running. Jobs can also be preempted by the scheduler, as discussed in the next section, as a configurable option. Depending on the configuration, a preempted job can be suspended, checkpointed, requeued to start over, or terminated. In all cases, preemption implies at least a temporary revocation of the allocation.

In terms of a guaranteed completion time for an allocation, if a request is made for two hours of resource and the job starts at 1 pm, it will finish by 3 pm. The flip side to this situation (e.g., can a user specify that a two-hour job finishes by 3 pm?) can currently be done by using an advance reservation. Work is under development to allow this capability under normal job submissions.

At setup time, one can configure how many job completion attempts should be allowed. This information is needed because some tasks, such as data transfers, may not succeed on the first try. One can also configure whether an allocation is exclusive or not, meaning whether the resource is space-shared or time-shared. Advance reservations are allowed only on space-shared resources.

PBS Pro currently does not support a malleable allocation, that is, an allocation that allows the addition or removal of resources during run time. When this feature is supported by common MPI-2 implementations, it will be added to PBS Pro.

4.4 Manipulating the Allocation Execution

It can be beneficial for a local resource management system to modify a running allocation in order to better coordinate the execution of a set of jobs. Such modifications can be done by using preemption, checkpointing, migration, and restart.

PBS Pro provides a number of options for manipulating allocation executions. Any job can be requeued or restarted. Preemption is a configurable option for any resource, and a site can use a variety of algorithms to specify which jobs should get preempted, how often, and for how long. When preempted, a job can be checkpointed if the underlying operating system allows this, for example SGI Irix and Cray UNICOS. If jobs are checkpointed by a user, they can be requeued to start at the stage in that checkpoint file. Migration usually can be done on-the-fly, but not for MPI jobs as this feature is currently not supported within MPI implementations.

5. SUMMARY AND FUTURE

The Grid computing field is quite young, and only the most basic promise of Grid computing is available today. Although PBS Pro features cover the Grid computing space (security, information, compute, and data), capabilities are constantly being added and refined. PBS Pro development plans include extending the support for X.509 certificates, expanding data Grid support to actively manage network bandwidth, and continuing to drive Grid standards via the Global Grid Forum. In particular, the PBS Pro team is actively involved defining numerous Grid standards: DRMMA [DRM], OGSA [OGSa], GRAAP [GRAa], and UR [UR].

Acknowledgments

We thank the many people who have been involved with the PBS software throughout the years, especially Bob Henderson, who has led the core development team from the very beginning. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, Office of Science, under contract W-31-109-Eng-38.

IV

PREDICTION AND MATCHING FOR GRID RESOURCE MANAGEMENT

Chapter 14

PERFORMANCE INFORMATION SERVICES FOR COMPUTATIONAL GRIDS

Rich Wolski, Lawrence J. Miller, Graziano Obertelli, and Martin Swany
Department of Computer Science, University of California, Santa Barbara

Abstract

Grid schedulers or resource allocators (whether they be human or automatic scheduling programs) must choose the right combination of resources from the available resource pool while the performance and availability characteristics of the individual resources within the pool change from moment to moment. Moreover, the scheduling decision for each application component must be made before the component is executed making scheduling a predictive activity. A Grid scheduler, therefore, must be able to predict what the deliverable resource performance will be for the time period in which a particular application component will eventually use the resource.

In this chapter, we describe techniques for dynamically characterizing resources according to their predicted performance response to enable Grid scheduling and resource allocation. These techniques rely on three fundamental capabilities: extensible and non-intrusive performance monitoring, fast prediction models, and a flexible and high-performance reporting interface. We discuss these challenges in the context of the Network Weather Service (NWS) – an online performance monitoring and forecasting service developed for Grid environments. The NWS uses adaptive monitoring techniques to control intrusiveness, and non-parametric forecasting methods that are lightweight enough to generate forecasts in real-time. In addition, the service infrastructure used by the NWS is portable among all currently available Grid resources and is compatible with extant Grid middleware such as Globus, Legion, and Condor.

1. INTRODUCTION

The problem of scheduling and resource allocation is central to Grid performance. Applications are typically composed of concurrently executing and communicating components resulting in the traditional tension between the performance benefits of parallelism and the communication overhead it intro-

duces. At the same time, Grid resources (the computers, networks, and storage systems that make up a Grid) differ widely in the performance they can deliver to any given application, and this deliverable performance fluctuates dynamically due to contention, resource failure, etc. Thus an application scheduler or resource allocator must choose the right combination of resources from the available resource pool while the performance and availability characteristics of the individual resources within the pool change from moment to moment.

To assign application components to resources so that application performance is maximized requires some form of resource valuation or characterization. A scheduler must be able to determine the relative worth of one resource versus another to the application and choose the ones that are most valuable in terms of the performance they deliver. If the scheduling decision is to be made by an automatic scheduling or resource allocation program (e.g. AppLeS [BWF+96, SW98] or GrADSoft [BCC+01, PBD+01, RIF01a]) this valuation must be in terms of quantifiable metrics that can be composed into a measure of application performance. Moreover, the scheduling decision for each application component must be made before the component is executed making scheduling a *predictive* activity. A Grid scheduler, therefore, must be able to predict what the deliverable resource performance *will be* for the time period in which a particular application component will eventually use the resource.

The performance characteristics associated with a resource can be roughly categorized as either *static* characteristics or *dynamic* characteristics according to the speed with which they change. While the delineation can be rather arbitrary, static characteristics are ones that change slowly with respect to program execution lifetimes. For example, the clock-speed associated with a CPU is a relatively static (and quantifiable) performance metric. It is not completely invariant, however, as a given CPU may be replaced by one that is faster without changing other measurable characteristics. From the perspective of a Grid user, a CPU that has been upgraded to a faster clock-speed may look identical in terms of its other characteristics (memory size, operating system, etc.) before and after the upgrade.

Conversely, dynamic performance characteristics change relatively quickly. CPU loads and network throughput, for example, fluctuate with frequencies measured in minutes or seconds. It is such dynamic fluctuations that make Grid scheduling complex and difficult. Moreover, many studies have shown that the statistical properties associated with these performance fluctuations are difficult to model in a way that generates accurate predictions [HBD96, CB97, GMR+98, HB99]. The chief difficulties are either that the distribution of performance measurements can be modeled most effectively by a power law (i.e. the distribution is said to be long-tailed) or that a time series of measurements for a given resource characteristic displays a slowly-decaying autocorrelation

structure (e.g. the series is self-similar). Despite these statistical properties, however, users routinely make predictions of future performance levels based on observed history. For example, students in a university computer laboratory often use the advertised Unix load average as an indication of what the load will be for some time into the future.

1.1 Grid Resource Performance Prediction

Reconciling the theoretical difficulty associated with dynamic performance prediction with the practical observation that some kind of prediction is necessary in any scheduling context requires a careful formulation of the prediction problem. For Grid scheduling and resource allocation, two important characteristics can be exploited by a scheduler to overcome the complexities introduced by the dynamics of Grid performance response.

- **Observable Forecast Accuracy** — Predictions of future performance measurements can be evaluated dynamically by recording the prediction accuracy once the predicted measurements are actually gathered.
- **Near-term Forecasting Epochs** — Grid schedulers can make their decisions dynamically, just before execution begins. Since forecast accuracy is likely to degrade as a function of time into the future for the epoch being forecast, making decisions at the last possible moment enhances prediction accuracy.

If performance measurements are being gathered from which scheduling decisions are to be made, predictions of future measurements can also be generated. By comparing these predictions to the measurements they predict when those measurements are eventually gathered, the scheduler can consider the quantitative accuracy of any given forecast as part of its scheduling decisions. Applications with performance response that is sensitive to inaccuracy can be scheduled using more stable resources by observing the degree to which those resources have been predictable in the past.

Grid schedulers can often make decisions just before application execution begins, and rescheduling decisions while an application is executing (either to support migration or for still-uncomputed work). Thus the time frame for which a prediction is necessary begins almost immediately after the scheduler finishes its decision-making process.

To exploit these characteristics, the forecasting infrastructure must, itself, be a high-performance, robust, long-running application. When scheduling decisions are made at run time, the time required to make the necessary forecasts and deliver them will be incurred as run time overhead. Hence, the forecasting system must be fast with respect to the application and scheduler execution times. If forecast information (even if it is only performance monitor data) that

is not available because the system serving it has failed, "blind" and potentially performance-retarding decisions must be made. Finally, if forecast accuracy is to be considered as part of the scheduling process, the forecasting system must be constantly gathering historical measurement data and generating predictions from it.

In this chapter, we describe the functionality that is necessary to support resource allocation based on performance measurements taken from Grid resources. We detail our experiences with implementing this functionality as part of the Network Weather Service (NWS) [WSH99a] — a distributed monitoring and forecasting service that is compatible with many different Grid infrastructures. We discuss the design and implementation decisions that are at the core of the system and outline its monitoring and forecasting capabilities. We conclude with a discussion of future research and development challenges that must be overcome to make dynamic resource allocation more accessible to Grid programmers.

2. GRID MONITORING AND FORECASTING INFRASTRUCTURE

Any system that is designed to support resource allocation based on performance data should provide three fundamental functionalities.

- **Monitoring**: Data from a distributed set of performance monitors must be gathered and managed so that it can be served.
- **Forecasting**: Resource allocators require forecasts of future performance. It is the forecast data (and not the monitor data) that ultimately must be served.
- **Reporting**: The information served by the system must be available in a wide-range of formats so that different scheduling and allocation implementations may be supported.

In Grid settings, these functionalities present some unique challenges.

Grid performance systems must be robust with respect to resource failure and/or restart, and must be carefully implemented so that their intrusiveness is minimized. Resource monitors, particularly those that probe resources by loading them, must be able to withstand frequent resource failure. Often, it is the faulty resources in a Grid that are of the most performance concern. If a monitor process requires manual intervention at restart, for example, there may be long periods of time for which no data is available precisely from the resources that must be most carefully monitored.

Moreover, if implemented as middleware, system administrators may not view performance monitoring and analysis systems as having the same level

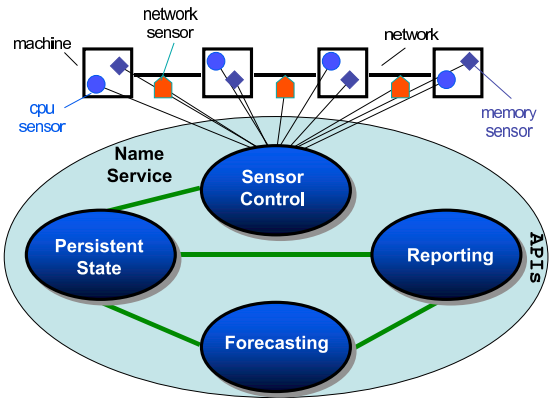


Figure 14.1. The logical architecture of the NWS.

of importance as native operating system functionality. If and when a performance problem is reported by a user, the most frequent early response by many system administrators is to terminate any non-operating system monitoring processes for fear that they are cause of the observed difficulties. It is rare, however, for a monitor that has been prophylactically killed to be restarted once the true source of the performance problem is located. Therefore, middleware-based performance monitors must be self-reinitializing and they must be able to store the data that they produce in persistent storage that survives local intervention or system failure.

Another problem that Grid performance monitors must face stems from their use of the resources that they are monitoring. The intrusiveness of performance monitors, in terms of their induced CPU load, memory footprint, and storage footprint, constitutes system overhead and, thus, must be tightly controlled.

2.1 The Network Weather Service

The Network Weather Service (NWS) is a Grid monitoring and forecasting tool that has been designed to support dynamic resource allocation and scheduling. Figure 14.1 depicts its logical architecture in terms of independent subsystems. Sensors (typically independent processes) generate time-stamp, performance measurement pairs. For robustness and to limit intrusiveness, the system supports a sensor control subsystem that is distributed and replicated. Sensor processes can put control of their measurement cycle, sensor restart, etc. under the control of the NWS by adopting either a socket-based API, or an internal library API.

The NWS also assumes that performance sensors will be stateless, both to improve robustness and as a way of minimizing memory and storage footprints. To capture and preserve measurement data, the Persistent State sub-

system exports a simple socket-based API that allows sensors to store their measurements remotely in time-series order. The number of Persistent State repositories, as well as the location and storage footprint of each are specifiable as installation parameters. In addition, new repositories can be added to the running system without reconfiguration.

Forecasts of future performance levels draw the historical data they require from the Persistent State system (and not the sensors). Thus, any process that can exercise the storage API exported by the Persistent State system, can inject measurements into the system for forecasting.

The forecasting subsystem is extensible, allowing the inclusion of new forecasting models into a forecaster library through a configuration-time API. To allow applications a way of trading off system complexity for performance, the NWS forecasting library can either be compiled into a Forecaster process and accessed remotely (thereby saving the local CPU and memory overhead) or loaded directly with the application.

To allow compatibility with a variety of Grid computing infrastructures, the NWS supports multiple reporting interfaces. These interfcaes communicate with the other subsystems via socket-based remote APIs as well, improving both flexibility and performance. New reporting formats can be added by providing a process or library that converts the NWS-internal API to the desired format.

In addition, this organization provides a convenient methodology for implementing replication and caching. Performance information (both measurement data and statistical forecasts) flow from the sensors, through the persistent state repositories and the forecasters to the reporting APIs, but not in the reverse direction. As such, reporting caches can be located near where the reports are consumed and can be replicated. Moreover by interrogating an internal Name Service (see below) the reporting caches can determine the frequency with which individual sensors are updating the various persistent state repositories. By doing so, each cache can refresh itself only when new data is expected from each sensor. When a scheduler or resource allocator queries a local cache, it receives up-to-date information without having to directly query the individual Persistent State repositories where the desired information is stored.

All components within an NWS installation register with an internal Name Service. The Name Service keeps track of the type, location (IP address and port number), and configuration parameters associated with each NWS process. In addition, all registrations are time limited and must be refreshed by their various components. Overall system status is determined by the active registrations that are contained within a given Name Service instantiation.

Under the current architecture, each instance of the Name Service defines a self-contained NWS installation. By using the name space to isolate separate NWS instantiations, multiple installations can overlay the same set of re-

sources. Debugging or experimentation with alternative configurations (while a production version continues to run) is made easier by this design choice. At the same time, all of the components, including the sensors that are part of the distributed NWS release, run without privileged access. Thus, separate users can run individual instantiations of the NWS, each with its own Name Service.

2.2 The NWS Implementation

The engineering of a Grid performance system, particularly one designed to support resource allocation and scheduling, presents a unique challenge. In addition to the performance goals (response time and scalability) which are largely architectural issues, the implementation itself must be ubiquitous, robust, and non-intrusive. Ubiquity stems from two critical requirements: portability and the need to run with minimal privilege. Robustness and non-intrusiveness come, in part, from careful implementation techniques and extensive testing.

Any performance monitoring and forecasting system must be able to execute on *all* platforms available to the user. If a scheduler cannot "see" a system because no performance information is available, the system is for all intents and purposes not part of the Grid. This need is especially critical when a Grid is to be used to couple cheap, commodity resources with a unique instrument or machine. If the Grid infrastructure cannot execute on or monitor the unique instrument, the instrument cannot become part of a Grid.

To meet this need for ubiquity, the NWS is written primarily in C. At the time of this writing, it is the experience of the NWS implementation team that C is the most portable programming language. Most rare or unusual architectures support a C compiler and a subset of the Unix system calls. The NWS (with the exception of some of the sensor code) has been carefully coded to use only the most basic system services and generic ANSI C functionality. As a result, the core services have been quick to port to new systems as they become available. It is worth noting that the choice of C is not motivated, in this case, by performance but rather portability. The Java language environment is intended to provide the kind of portability the NWS requires. Many of the systems that users wish to access via a Grid, however, are large-scale machines with unique configurations. To date, the availability of a portable Java environment to machines of this class lags far behind the availability of C, if such a Java environment becomes available at all. At the same time, systems that do support a robust and standardized Java environment also support the baseline C functionality that is required by the NWS. Figure 14.2 depicts the software organization of the system.

The internal subsystems, the NWS-supplied sensors, the C and Unix command-line interface code are written in C. The HTML interface uses a combi-

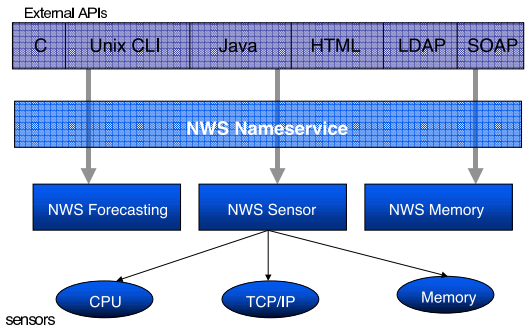


Figure 14.2. The software organization of the NWS implementation.

nation of CGI and GNU tools (not distributed with the system) and the LDAP and SOAP interfaces are derived from open source software for implementing each protocol.

A second design decision alluded to earlier is that all NWS components must be able to run without privileged access. If an individual site wishes to configure a sensor that runs "as root," the extensibility of the system will permit it. Often, due to the security concerns associated with middleware, the need for privileged access tends to delay the deployment of a particular middleware component. Because the forecasting functionality is critical to resource allocation and scheduling, the NWS is coded to run with only minimal access privilege (e.g. a standard user login).

3. PERFORMANCE MONITORS

There are roughly two categories of performance monitor types: *passive* and *active*. A passive monitor is one which reads a measurement gathered through some other means (e.g. the local operating system). The best example of a passive monitor that most Grid systems report is the Unix Load Average metric. Almost all Unix and Linux systems (and their derivatives) record some measure of the number of jobs in the run queues of each processor on the machine. The frequency with which the queue length is sampled is operating system and operating system version specific. On most systems, however, a 1 minute, 5 minute, and 15 minute average of the run queue length are available although the way in which the average is calculated (arithmetic, geometric, exponentially smoothed, etc.) is again operating-system specific. This smoothed average of the run queue length defines the Load Average metric.

Systems such as the Globus Meta Directory Service [CFFK01] report Unix Load Average by periodically querying the load average value and posting the result. Thus, the Globus load sensor passively reads and reports a performance metric (Unix Load Average) that is gathered and maintained by the native operating system.

3.1 Intrusiveness versus Accuracy

The main advantage of passive sensing is that it is non-intrusive. The Unix Load Average is a measure that is already being generated. The sensor need only format and transmit the measured values appropriately. The difficulty with quantities such as Unix Load Average, however, is that they are sometimes complex to understand from a resource allocation perspective. For example, using load average as a measure of machine "busy-ness" allows machines of equivalent processing power to be ranked in terms of their expected execution speeds. The assertion that most Grid resource schedulers make is that in a pool of identical machines, the one with the smallest load average value is the one that will execute a sequential piece of code the fastest.

Using Unix Load Average to rank execution speeds implies that the presence of other jobs in each run queue will affect the performance of the scheduled application in the same way. Unix and Linux use an exponential aging algorithm to determine execution priority. Furthermore, the aging factor on some systems grows larger with occupancy time. The goal of this algorithm is to permit jobs that have recently completed an I/O operation to get the CPU immediately as an aid to response time. Consider interactive text editors as an example. After each key stroke, the editor is scheduled at a very high priority so that it can echo the character and then reblock waiting for the next key stroke. However, the priority aging algorithm rapidly lowers a processes priority to its set level if it does not immediately re-sleep after an I/O. Consider a system with a load average value of 2.0 where the two jobs are rapidly sleeping and waking. A CPU-bound Grid job sharing this system will get a different fraction of the CPU than on a system in which both jobs in the run queue are, themselves, CPU bound. In this latter case, the typical Unix scheduling algorithm degenerates into a round robin scheme. Thus, the load average implies a performance impact on a scheduled job that depends on the qualities of the other jobs that are running. This information, even it were published on a job-by-job basis, is difficult to interpret because it is the way in which jobs of different priorities interact that ultimately defines how load affects scheduling.

As an alternative method, a Grid performance monitor can periodically load the resource it is monitoring and record the observed performance response. This active approach has the advantage of disambiguating the relationship between a monitored quantity and performance impact. Returning to the load

average example, if a CPU monitor were to simply run a CPU bound process periodically, it could record the utilization that process enjoyed during each run. The fraction of wall-clock time that the process occupied the CPU can be used as the inverse of the slowdown caused by competing jobs on that system (e.g. a process getting 20% utilization can be thought of as 5 times slower than if it had received 100% utilization). The obvious difficulty with this approach is that the monitor must completely load the resource in order to measure it thereby leaving less resource available for actual computation.

There is an inherent tension between monitor accuracy and monitor intrusiveness that must be considered when designing a Grid performance sensor. The accuracy that active sensing makes possible must be balanced against the amount of resource it consumes. If good passive sensing techniques are available, it is sometimes possible to combine the two methods through some form of automatic regression technique.

As part of the NWS Grid monitoring infrastructure, we have implemented a CPU sensor that combines Unix Load Average with active CPU probing. The sensor reads the 1 minute Load Average value periodically, according to a parameter set when the sensor is initialized. It also initiates a register-only CPU bound process (called the CPU probe) with a much lower periodicity and records the utilization that it experiences. The duration of the CPU probes execution is also a parameter. Anecdotally, we have found that a probe duration of 1.5 seconds is typically enough to yield accurate results.

Next, the sensor converts Unix Load Average to a utilization estimate. It assumes that the run queue will be serviced round-robin and that all jobs are CPU bound hence an equal fraction of time will be given to each. The sensor combines both the probe utilization and the Load Average reading by automatically calculating a bias value. If, for example, the utilization predicted by Load Average is 10% less than observed, the bias is computed as $+10$. Should the Load Average over-estimate utilization, the bias is negative.

The sensor reports as a measurement a utilization estimate that is generated by biasing the load average with the last bias recorded. Since load average is sampled much more frequently than the probe is run, the intrusiveness is less than if only the probe were used. At the same time, the probe captures some of the interaction between itself and other contending jobs in the run queue.

Finally, the NWS CPU sensor controls the periodicity with which the probe is executed based on the changing size of the bias. If the bias value is fluctuating the sensor assumes that the load is highly fluctuating and the CPU should be probed again in a relatively short period of time. If the bias is relatively stable, the probe frequency is decreased. Both the maximum and minimum frequencies as well as the stability threshold are configuration parameters to the sensor.

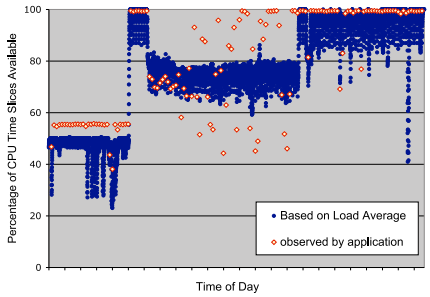


Figure 14.3. A comparison of available CPU cycles as measured with Unix load average to actual observed occupancy percentage.

Figures 14.3 and 14.4 depict the effects of this sensing technique using a workstation as an example. In Figure 14.3, the solid circles show the percentage of available CPU time slices (over a 10 second period) that are measured by Unix Load Average. The $y - axis$ values are measurements, and the $x - axis$ values show time of day. One measurement occurs at every 10 second interval, and the total trace covers a 24-hour period. This particular workstation was being used by a graduate student at the time to finish her Ph.D. thesis, making the load variation (however non-synthetic) potentially atypical.

To convert a load average measurement to an available occupancy percentage, the NWS passive sensor uses the formula

$$load\_avg\_available\_cpu = 100.0/(load\_average + 1.0) \tag{14.1}$$

where the load average covers 1 minute. Again, based on the assumption that all processes in the run queue have equal priority, the available fraction of CPU time for a new process is 1 divided by the number of currently runnable processes plus an additional process. Multiplying by 100 simply converts this number into a percentage.

Diamond shapes (drawn in outline with a light-colored fill) show the occupancy observed by a test program that occurs at less frequent intervals (every 10 minutes) in the trace. When executed, the test program spins in a tight loop for 30 seconds, measured in wall-clock time, and records the user and system occupancy time during the execution. The ratio of actual occupancy time to wall-clock time is the observed availability fraction. Both the 10 minute interval, and the 30 second execution duration allow the smoothed load average value to "recover" from the load introduced by the test program. During the measurement period, the test program and the load average sensor were coordinated so that a load average measurement was taken immediately before each test program run, and both were assigned the same time stamp. Thus the vertical distance between each light colored diamond and corresponding solid

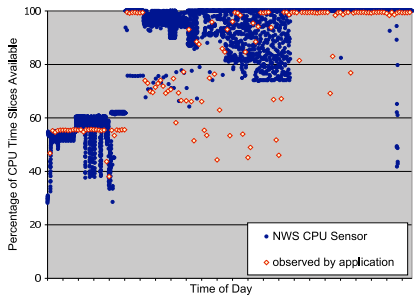


Figure 14.4. A comparison of available CPU cycles as measured with NWS CPU sensor to actual observed occupancy percentage.

circle in the figure shows graphically the measurement error associated with each measurement.

Figure 14.4 shows the same accuracy comparison for the NWS CPU sensor. In it, each solid circle represents the biased NWS sensor value and, as in the previous figure, each light-colored diamond shows the occupancy observed by the test program. By learning and then applying a bias value, the NWS sensor is better able to measure the true availability experienced by the test application with little added intrusiveness.

More generally, however, this example illustrates the need for Grid resource monitoring systems to capture measurement error. Many such systems report the metrics that are available to users (e.g. Unix Load Average) but few provide estimates of how those measurements translate into observable application performance. For resource allocation and scheduling purposes, the measurement error associated with passive measurements is a useful and often overlooked quantity.

3.2 Intrusiveness versus Scalability

Another important design point concerns the trade-off between intrusiveness and scalability. Consider the problem of gathering periodic end-to-end network probe information. The naive implementation furnishes each sensor with a list of other sensors to contact within a Grid, and a periodicity. Each sensor operates on its own clock and with the specified periodicity probes all of the other sensors.

In Figure 14.5 we show a network performance time series of the TCP/IP performance observed between a pair of Unix hosts connected via 10 megabit-per-second Ethernet. Each bandwidth reading is generated by timing a 64 kilobyte transfer using a TCP/IP socket with 32 kilobyte socket buffers. During the first half of the trace (the left side of the figure) only one pair of hosts — a sender and a receiver — was probing the network. Midway through the trace,

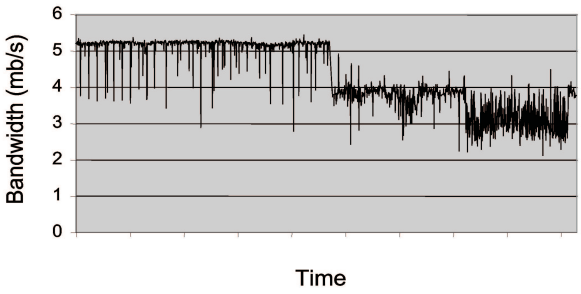


Figure 14.5. TCP/IP sensor contention.

a second host pair began to probe the network simultaneously. The loss of available bandwidth, which is visually apparent from the trace, results from the interaction of colliding network probes.

To produce a complete end-to-end picture of network performance between N hosts, $2 * (N^2 - N)$ such measurements would be required (i.e. one in each direction and hosts do not probe themselves). If each host uses its own local clock to determine when to probe the network, the likelihood of probe contention goes up at least quadratically as the Grid scales.

To prevent probe contention, the NWS end-to-end network sensor uses a token-passing protocol to implement mutual exclusion between "cliques" of hosts. Hosts within a specified clique pass the entire clique list in a token. The NWS clique protocol implements a simplified leader election scheme that manages token loss/recovery, and network partitioning. If a token is lost because the host hold it fails, the other hosts in the clique will time out and attempt to elect themselves leader by regenerating and sending out a new token. Time stamps on the token are used to resolve the possibility of multiple simultaneous time outs. When a host encounters two different tokens (from two different leaders) it will "kill" the older one. This scheme also manages network partitioning. If the network partitions, the hosts that are separated from the current leader will elect a new leader on their side of the partition. When the partition is resolved, the two tokens will once again circulate across the entire host list, and one of them (the older one) will be annihilated. This form of active replication makes the clique protocol robust to both host and network failure.

To permit extensibility and scalability, sensors can participate in multiple cliques at the same time, and each clique can contain any number of hosts greater than or equal to 2. Thus, the clique organization can capture a variety of non-clique monitoring topologies if probe contention is not a concern. For example, one common topology that many sites wish to monitor is a "star" topology: one distinguished host connected to a set of satellite hosts, without connectivity between the satellites. If probe contention is not an issue, one

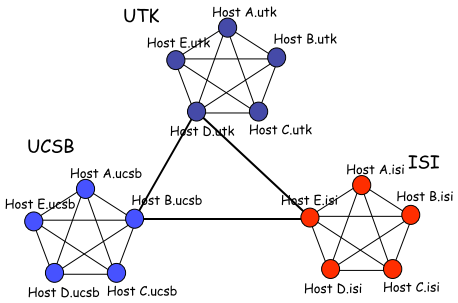


Figure 14.6. Example NWS clique hierarchy.

clique consisting of a satellite node and the central node can be created for each satellite node. Since the central node participates in multiple cliques simultaneously, this organization implements the desired measurement topology. This, the NWS clique abstraction can be used to implement other monitoring topologies according to the needs of each individual installation.

To gain scalability, cliques can be organized into a hierarchy. At the bottom level of the hierarchy are cliques of hosts. Each clique "promotes" a distinguished representative to participate in a higher-level clique, forming a tree. Consider the example shown in Figure 14.6. In it, five hosts (labeled A, B, C, D, and E) are configured into "base" cliques at each of three sites: UCSB, ISI, and UTK. One distinguished host from each site participates in a higher-level clique that captures inter-site connectivity.

Notice that this organization can capture the full N^2 matrix of connectivity if the inter-site connectivity performance is similar for all nodes communicating between sites. For example, if UCSB is the University of California in Santa Barbara, and UTK is the University of Tennessee, in Knoxville, any host in the UCSB clique communicating with any host in the UTK clique will likely observe the same network performance since much of the network between the two will be shared. That is, since virtually all UCSB-to-UTK network traffic will traverse common network elements, a single UCSB-UTK pair can measure the inter-site connectivity. By using the inter-site measurements of the distinguished pair in the higher-level clique in place of the missing measurements, the NWS can construct a full N^2 picture without conducting N^2 measurements. At the same time, measurements within each clique will not contend.

4. FORECASTING

The problem of determining a resource allocation or schedule that maximizes some objective function is inherently a predictive one. When a decision is made about the resources to allocate, some assumption about the future behavior of the resources or application is either implicitly or explicitly included. For example, if a large MPI program is to be assign to a parallel machine because of the processor speeds, memory capacity, and interconnect speed, the scheduler or resource allocator making that decision is making an assumption of what the processor speeds, memory availability, and interconnect speed *will be* when the MPI program runs. Users of space-shared parallel machines often assume that these predictions are easy to make statically. Interconnect speed, however, may be influenced by parallel jobs running in other partitions, so even in traditional parallel computing settings, predictions of dynamically changing behavior may be required.

In Grid settings, however, where resources are federated and interconnected by shared networks, the available resource performance can fluctuate dramatically. The same scheduling decisions based on predictions of future resource performance are necessary if applications are to obtain the performance levels desired by their users. Therefore, some methodology is required to make forecasts of future performance levels that upon which scheduling decisions can be based. Note that even though we have outlined the need for forecasting explicitly, all Grid users go through this activity either explicitly or implicitly. When a user chooses a particular data repository, for example, because it is connected to a "faster" network, he or she is making the prediction that the network *will be* faster when it is used to access the user's data. Most often this forecast is based on past experience with the resource. The NWS includes statistical methods that attempt to mechanize and automate the forecasting process for the user based on similar historical experience.

4.1 The NWS Non-Parametric Forecasting Method

The principle behind the NWS forecasting technique is that the best forecasting technique amongst a number of available options can be determined from past accuracy. Each forecasting method is configured into the system with its own parameters. It must be able to generate, on demand, a prediction based on a previous history of measurements and forecasts. That is, for each forecasting method f at measurement time t,

$$prediction_f(t) \quad = \quad METHOD_f(history_f(t)) \tag{14.2}$$

where

$prediction_f(t)$ = the predicted value made by method f for the measurement value at $t+1$,

$history_f(t)$ = a finite history of measurements, forecasts, and forecast errors generated previously to time t using method f, and

$METHOD_f$ = forecasting method f.

Each method is presented with a history of previous measurements (represented as a time series) and maintains its own history of previous predictions and accuracy information. In particular,

$$err_f(t) = value(t) - prediction_f(t-1) \tag{14.3}$$

is the error residual associated with a measurement $value(t)$ taken at time t and a prediction of that measurement generated by method f generated at time $t-1$.

The *primary forecasters* are able to produce a forecast based on time-series data and some set of parameters. The forecaster interface is general enough to accept a variety of forecasting techniques. Because the autocorrelation structure of many performance series is complex, and because series stationarity is unlikely, a large set of fast, simple predictors that can be constantly re-evaluated is the most effective configuration.

The primary techniques to produce forecasts include mean-based and median-based methods for producing completely non-parametric forecasts. Based on the observation that more recent data is often more indicative of current conditions, the primary forecasters make use of varying amounts of history using "sliding window" techniques. In this same spirit exponential smoothingtechniques, parameterized by the amount of *gain*, are used to produce forecasts as well. Each of these forecasting modules accepts data as a time-series and computes a forecast from that data and any parameters that the module accepts.

4.2 Secondary Forecasters: Dynamic Predictor Selection

The NWS operates all of the primary forecasters (and their various parameterizations) simultaneously at the time a forecast is requested through the API. It then uses the error measure calculated in Equation 14.3 to produce an overall fitness metric for each method. The method exhibiting the lowest cumulative

error at time t is used to generate a forecast for the measurement at time $t+1$ and that forecast is recorded as the "winning" primary forecast. The NWS conducts two such error tournaments for each forecast: one based on the mean square errorand one based on the mean absolute error.

$$MSE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}(err_f(i))^2 \tag{14.4}$$

and the mean absolute prediction error

$$MPE_f(t) = \frac{1}{t+1}\sum_{i=0}^{t}|(err_f(i)| \tag{14.5}$$

We then define

$$MIN\_MSE(t) = predictor_f(t) \quad \text{if } MSE_f(t) \text{ is the minimum over all methods at time } t \tag{14.6}$$

and

$$MIN\_MAE(t) = predictor_f(t) \quad \text{if } MAE_f(t) \text{ is the minimum over all methods at time } t. \tag{14.7}$$

That is, at time *t*, the method yielding the lowest mean square prediction error is recorded as a forecast of the next measurement by MIN_MSE. Similarly, the forecasting method at time t yielding the lowest overall mean absolute prediction error becomes the MIN_MAE forecast of the next measurement.

Both of these error metrics use the cumulative error spanning the entire history available from the series. In an attempt to address the possibility that the series is non-stationary, the system also maintains error-minimum predictors where the error is recorded over a limited previous history. The goal of this approach is to prevent error recordings that span a change point in the series from polluting the more recent error performance of each predictor. More formally

$$MIN\_MSE_W(t,w) = predictor_f(t) \quad \text{if } MSE_f(t) \text{ is the minimum over all methods at time } t \text{ for the most recent w measurements} \tag{14.8}$$

and

$$MIN\_MAE_W(t,w) = predictor_f(t) \quad \text{if } MAE_f(t) \text{ is the minimum over all methods at time } t \text{ for the most recent w measurements} \tag{14.9}$$

where w is a fixed window of previous measurements.

4.3 The NWS Forecast

Finally, the NWS forecast that is generated on demand is the one that "wins" an error tournament at time t for the set of primary and secondary forecasters described in this section. That is

$$NWS\_MSE(t) = predictor_f(t) \quad \text{if } MSE_f(t) \text{ is the minimum over all primary and secondary methods at time } t \tag{14.10}$$

and

$$NWS\_MAE_W(t) = predictor_f(t) \quad \text{if } MAE_f(t) \text{ is the minimum over all primary and secondary methods at time } t. \tag{14.11}$$

Table 14.1 summarizes the primary and secondary methods that are the NWS uses for analysis. The NWS combines these methods to produce an NWS_MSE and NWS_MAE forecast for each value in each series presented to the forecasting subsystem. Because many of the primary forecasters can be implemented using computationally efficient algorithms, the overall execution cost of computing the final NWS forecasts is low. For example, on a 750 megahertz Pentium III laptop, each NWS_MSE and NWS_MAE requires 161 microseconds to compute.

Table 14.1 summarizes the primary and secondary methods that are the NWS uses for analysis.

5. CONCLUSIONS, CURRENT STATUS, FUTURE WORK

The heterogeneous and dynamic nature of Grid resource performance makes effective resource allocation and scheduling critical to application performance. The basis for these critical scheduling functionalities is is a predictive capability that captures future expected resource behavior. Typically, Grid users and schedulers will use the immediate performance history (the last observed value or a running average) to make an implicit prediction of future performance. However, there are several important ways in which such an *ad hoc* methodology can be improved.

To be truly effective, the performance gathering system must be robust, portable, and non-intrusive. Simply relying on available resource performance measurements, or building naive probing mechanisms can result in additional resource contention and a substantial loss of application performance. Moreover, by carefully considering measurement error, it is possible to automatically and adaptively balance the accuracy of explicit resource probing with the non-intrusiveness of passive measurement. Similarly, overhead introduced by

Table 14.1. Summary of forecasting methods.

Predictor	**Description**	**Parameters**
LAST	last measurement	
RUN_AVG	running average	
EXP_{SM}	exponential smoothing	$g = 0.05$
EXP_{SM}	exponential smoothing	$g = 0.10$
EXP_{SM}	exponential smoothing	$g = 0.15$
EXP_{SM}	exponential smoothing	$g = 0.20$
EXP_{SM}	exponential smoothing	$g = 0.30$
EXP_{SM}	exponential smoothing	$g = 0.40$
EXP_{SM}	exponential smoothing	$g = 0.50$
EXP_{SM}	exponential smoothing	$g = 0.75$
EXP_{SM}	exponential smoothing	$g = 0.90$
EXP_{SMT}	exponential smooth+trend	$g = 0.05, tg = 0.001$
EXP_{SMT}	exponential smooth+trend	$g = 0.10, tg = 0.001$
EXP_{SMT}	exponential smooth+trend	$g = 0.15, tg = 0.001$
EXP_{SMT}	exponential smooth+trend	$g = 0.20, tg = 0.001$
MEDIAN	median filter	$K = 31$
MEDIAN	median filter	$K = 5$
SW_AVG	sliding window avg.	$K = 31$
SW_AVG	sliding window avg.	$K = 5$
TRIM_MEAN	α-trimmed mean	$K = 31, \alpha = 0.3$
TRIM_MEAN	α-trimmed mean	$K = 51, \alpha = 0.3$
ADAPT_MED	adaptive window median	$max = 21, min = 5$
ADAPT_MED	adaptive window median	$max = 51, min = 21$
MIN_MSE	adaptive minimum MSE	
MIN_MAE	adaptive minimum MAE	
MIN_MSE_W	windowed adaptive minimum MSE	$w = 1$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 1$
MIN_MSE_W	windowed adaptive minimum MSE	$w = 5$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 5$
MIN_MSE_W	windowed adaptive minimum MSE	$w = 10$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 10$
MIN_MSE_W	windowed adaptive minimum MSE	$w = 30$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 30$
MIN_MSE_W	windowed adaptive minimum MSE	$w = 50$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 50$
MIN_MSE_W	windowed adaptive minimum MSE	$w = 100$
MIN_MAE_W	windowed adaptive minimum MAE	$w = 100$

the performance gathering system must be explicitly controlled, particularly if probes can contend for resources. The ability to implement this control in a way that scales with the number of resources requires an effective system architecture for the performance monitoring system.

By using fast, robust time-series techniques, and running tabulations of forecast error, it is possible to improve the accuracy of performance predictions with minimal computational complexity. In addition to point-valued predictions, these same adaptive techniques can generate empirical confidence intervals and automatic resource classifications thereby improving scheduler design and scalability.

Thus, effective support for dynamic resource allocation and scheduling requires an architecture, a set of analysis techniques, and an implementation strategy that combine to meet the demands of the Grid paradigm. The NWS has been developed with these realizations in mind. It is a robust, portable, and adaptive distributed system for gathering historical performance data, making on-line forecasts from the data it gathers, and disseminating the values it collects.

5.1 Status

Currently, the NWS is available as a released and supported Grid middleware system from the National partnership for Advanced Computational Infrastructure (NPACI) and from the National Science Foundation's Middleware Initiative (NMI) public distribution. These distributions include portable CPU, TCP/IP socket sensors, a non-paged memory sensor for Linux systems, and support for C, Unix, HTML, and LDAP interfaces (the latter via a caching proxy). In addition, the NWS team distributes a non-supported version with additional prototype functionalities that have yet to make it into public release. At the time of this writing, the working prototypes include an NFS probing file system sensor, a portable non-paged memory sensor, an I/O sensor, and a senor that monitors system availability. There is also a prototype Open Grid Systems Architecture [FKNT02] interface and a Microsoft .NET/C# implementation as well.

5.2 Future Work

NWS development will expand the system's utility in three ways. First, we are investigating new statistical techniques that enable more accurate predictions than the system currently generates. While the NWS forecasts are able to generate useful predictions from difficult series, the optimal postcast measures indicate that more accurate forecasts are still possible. We are also studying ways to predict performance characteristics that do not conform well to the time series model. Periodic measurement is difficult to ensure in all settings

and a key feature of the NWS is its ability to cover the space of Grid forecasting needs. Secondly, we are exploring new reporting and data management strategies such as caching OGSA proxies and relational data archives. These new data delivery systems are needed to support an expanding user community, some of whom wish to use the NWS for statistical modeling as well as resource allocation. Finally, we are considering new architectural features that will enable the system to serve work in peer-to-peer settings where resource availability and dynamism are even more prevalent than in a Grid context.

For Grid resource allocation and scheduling, however, the NWS implements the functionalities that are necessary to achieve tenets of the Grid computing paradigm [FK99b] with the efficiency that application users demand. While we have outlined the requirements for Grid performance data management in terms of the NWS design, we believe that these requirements are fundamental to the Grid itself. As such, any truly effective resource allocation and scheduling system will need the functionality that we have described herein, independent of the way in which the NWS implements this functionality. Thus, for the Grid to be a success, effective forecasts of resource performance upon which scheduling and allocation decisions will be made, are critical.

Acknowledgments

This work was supported, in part, by a grant from the National Science Foundation's NGS program (EIA-9975020) and NMI program (ANI-0123911) and by the NASA IPG project.

Chapter 15

USING PREDICTED VARIANCE FOR CONSERVATIVE SCHEDULING ON SHARED RESOURCES

Jennifer M. Schopf[1] and Lingyun Yang[2]

[1]*Mathematics and Computer Science Division, Argonne National Laboratory*
[2]*Computer Science Department, University of Chicago*

Abstract In heterogeneous and dynamic environments, efficient execution of parallel computations can require mappings of tasks to processors with performance that is both irregular and time varying. We propose a *conservative scheduling policy* that uses information about expected future variance in resource capabilities to produce more efficient data mapping decisions.

We first present two techniques to estimate future load and variance, one based on normal distributions and another using tendency-based prediction methodologies. We then present a family of stochastic scheduling algorithms that exploit such predictions when making data mapping decisions. We describe experiments in which we apply our techniques to an astrophysics application. The results of these experiments demonstrate that conservative scheduling can produce execution times that are significantly faster and less variable than other techniques.

1. INTRODUCTION

Clusters of PCs or workstations have become a common platform for parallel computing. Applications on these platforms must coordinate the execution of concurrent tasks on nodes whose performance is both irregular and time varying because of the presence of other applications sharing the resources. To achieve good performance, application developers use performance models to predict the behavior of possible task and data allocations and to assist selecting a performance-efficient application execution strategy. Such models need

to accurately represent the dynamic performance variation of the application on the underlying resources in a manner that allows the scheduler to adapt application execution to the current system state, which means adapting to both the irregular (heterogeneous) nature of the resources and their time-varying behaviors.

We present a conservative scheduling technique that uses the predicted mean and variance of CPU capacity to make data mapping decisions. The basic idea is straightforward: We seek to allocate more work to systems that we expect to deliver the most computation, where this is defined from the viewpoint of the application. We often see that a resource with a larger capacity will also show a higher variance in performance and therefore will more strongly influence the execution time of an application than will a machine with less variance. Also, we keep in mind that a cluster may be homogeneous in machine type but quite heterogeneous in performance because of different underlying loads on the various resources.

Our conservative scheduling technique uses a conservative load prediction, equal to a prediction of the resource capacity over the future time interval of the application added to the predicted variance of the machine, in order to determine the proper data mapping, as opposed to just using a prediction of capacity as do many other approaches. This technique addresses both the dynamic and heterogeneous nature of shared resources.

We proceed in two steps. First, we define two techniques to predict future load and variance over a time interval, one based on using a normal distribution, the other using a tendency-based prediction technique defined in [YFS03]. Then, we use stochastic scheduling algorithms [SB99] that are parameterized by these predicted means and variances to make data distribution decisions. The result is an approach that exploits predicted variance in performance information to define a time-balancing scheduling strategy that improves application execution time.

We evaluate the effectiveness of this conservative scheduling technique by applying it to a particular class of applications, namely, loosely synchronous, iterative, data-parallel computations. Such applications are characterized by a single set of operations that is repeated many times, with a loose synchronization step between iterations [FJL+88, FWM94]. We present experiments conducted using Cactus [ABH+99, AAF+01], a loosely synchronous iterative computational astrophysics application. Our results demonstrate that we can achieve significant improvements in both mean execution time and the variance of those execution times over multiple runs in heterogeneous, dynamic environments.

2. RELATED WORK

Many researchers [Dai01, FB96, KDB02, WZ98] have explored the use of time balancing or load balancing models to reduce application execution time in heterogeneous environments. However, their work has typically assumed that resource performance is constant or slowly changing and thus does not take later variance into account. For example, Dail [Dai01] and Liu et al. [LYFA02] use the 10-second-ahead predicted CPU information provided by the Network Weather Service (NWS)([Wol98, WSH99a], also described in Chapter 14) to guide scheduling decisions. While this one-step-ahead prediction at a time point is often a good estimate for the next 10 seconds, it is less effective in predicting the available CPU the application will encounter during a longer execution. Dinda et al. built a Running Time Advisor (RTA) [Din02] that predicts the running time of applications 1 to 30 seconds into the future based on a multistep-ahead CPU load prediction.

Dome [ABL+95]i and Mars [GR96] support dynamic workload balancing through migration and make the application adaptive to the dynamic environment at runtime. But the implementation of such adaptive strategies can be complex and is not feasible for all applications.

In other work [SB99] we define the basic concept of stochastic values and their use in making scheduling decisions. This chapter extends that work to address the use of additional prediction techniques that originally predicted only one step ahead using a tendency-based approach [YFS03]. We define a time-balancing scheduling strategy based on a prediction of the next interval of time and a prediction of the variance (standard deviation) to counteract the problems seen with a one-step-ahead approach. Our technique achieves faster and less variable application execution time.

3. PROBLEM STATEMENT

Efficient execution in a distributed system can require, in the general case, mechanisms for the discovery of available resources, the selection of an application-appropriate subset of those resources, and the mapping of data or tasks onto selected resources. In this chapter we assume that the target set of resources is fixed, and we focus on the data mapping problem for data parallel applications.

We do not assume that the resources in this resource set have identical or even fixed capabilities in that they have identical underlying CPU loads. Within this context, our goal is to achieve data assignments that balance load between processors so that each processor finishes executing at roughly the same time, thereby minimizing execution time. This form of load balancing is also known as time balancing.

Time balancing is generally accomplished by solving a set of equations, such as the following, to determine the data assignments:

$$\begin{array}{l} E_i(D_i) = E_j(D_j) \ \ \forall\, i, j \\ \sum D_i = D_{Total}, \end{array} \tag{15.1}$$

where

- D_i is the amount of data assigned to processor i;
- D_{Total} is the total amount of data for the application;
- $E_i(D_i)$ is the execution time of task on processor i and is generally parameterized by the amount of data on that processor, D_i. It can be calculated by using a performance model of the application. For example, a simple application might have the following performance model:

$$\begin{array}{rl} E_i(D_i) = & Comm(D_i) * (futureNWCapacity) + \\ & Comp(D_i) * (futureCPUCapacity). \end{array} \tag{15.2}$$

Note that the performance of an application can be affected by the future capacity of both the network bandwidth behavior and the CPU availability.

In order to proceed, we need mechanisms for: (a) obtaining some measure of future capability and (b) translating this measure into an effective resource capability that is then used to guide data mapping. As we discuss below, two measures of future resource capability are important: the expected value and the expected variance in that value. One approach to obtaining these two measures is to negotiate a service level agreement (SLA) with the resource owner under which the owner would contract to provide the specified capability [CFK+02]. Or, we can use observed historical data to generate a prediction for future behavior [Din02, SB99, SFT98, VS02, WSH99b, YFS03]. We focus in this chapter on the latter approach and present two techniques for predicting the future capability: using normal distributions and using a predicted aggregation. However, we emphasize that our results on topic (b) above are also applicable in the SLA-negotiation case.

4. PREDICTING LOAD AND VARIANCE

The Network Weather Service (NWS) [Wol98] provides predicted CPU information one measurement (generally about 10 seconds) ahead based on a time series of earlier CPU load information. Some previous scheduling work [Dai01, LYFA02] uses this one-step-ahead predicted CPU information as the future CPU capability in the performance model. For better data distribution and scheduling, however, what is really needed is an estimate of the average CPU load an application will experience during execution, rather than

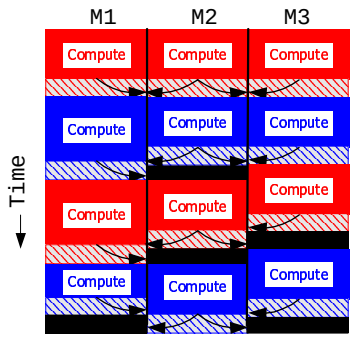


Figure 15.1. The interrelated influence among tasks of a synchronous iterative application.

the CPU information at a single future point in time. One measurement is simply not enough data for most applications.

In loosely synchronous iterative applications, tasks communicate between iterations, and the next iteration on a given resource cannot begin until the communication phase to that resource has been finished, as shown in Figure 15.1. Thus, a slower machine not only will take longer to run its own task but will also increase the execution time of the other tasks with which it communicates–and ultimately the execution time of the entire job. In Figure 15.1, the data was evenly divided among the resources, but M1 has a large variance in execution time. If M1 were running in isolation, it would complete the overall work in the same amount of time as M2 or M3. Because of its large variation, however,however, it is slow to communicate to M2 at the end of the second iteration, in turn delaying the task on M2 at the third computation step (in black), and hence delaying the task on M3 at the fourth computation step. Thus, the total job is delayed. It is this wave of delayed behavior caused by variance in the resource capability that we seek to avoid with our scheduling approach.

In the next subsections, we address two ways to more accurately predict longer-range load behavior: using a normal distribution and extending a one-step-ahead load prediction developed in previous work [YFS03]

4.1 Normal Distribution Predictions

Performance models are often parameterized by values that represent system or application characteristics. In dedicated, or single-user, settings it is often sufficient to represent these characteristics by a single value, or *point value*. For example, we may represent bandwidth as 7 Mbits/second. However, point values are often inaccurate or insufficient representations for characteristics that change over time. For example, rather than a constant valuation of

7 Mbits/second, bandwidth may actually vary from 5 to 9 Mbits/second. One way to represent this variable behavior is to use a *stochastic value*, or distribution.

By parameterizing models with stochastic information, the resulting prediction is also a stochastic value. Stochastic-valued predictions provide valuable additional information that can be supplied to a scheduler and used to improve the overall performance of distributed parallel applications. Stochastic values can be represented in a variety of way—as distributions [SB98], as intervals [SB99], and as histograms [Sch99]. In this chapter we assume that we can adequately represent stochastic values using normal distributions. Normal distributions, also called Gaussian distributions, are representative of large collections of random variables. As such, many real phenomena in computer systems generate distributions that are close to normal distributions [Adv93, AV93].

A normal distribution can be defined by the formula

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \qquad -\infty < x < \infty \tag{15.3}$$

for parameters μ, the *mean*, which gives the *center* of the range of the distribution, and σ, the *standard deviation*, which describes the variability in the distribution and gives a range around the mean. Normal distributions are symmetric and bell shaped and have the property that the range defined by the mean plus and minus two standard deviations captures approximately 95% of the values of the distribution.

Figure 15.2 shows a histogram of runtimes for an SOR benchmark on a single workstation with no other users present, and the normal distribution based on the data mean, m, and standard deviation, sd. Distributions can be represented graphically in two common ways: by the *probability density function* (PDF), as shown on the left in Figure 15.2, which graphs values against their probabilities, similar to a histogram, and by the *cumulative distribution function* (CDF), as shown on the right in Figure 15.2, which illustrates the probability that a point in the range is less than or equal to a particular value.

In the following subsections we describe the necessary compositional arithmetic to use normal distributions in predictive models; in Sections 4.1.2 and 4.1.3 we discuss alternatives to consider when the assumption of a normal distribution is too far from the actual distribution of the stochastic value.

4.1.1 Arithmetic Operations over Normal Distributions

In order for prediction models to use stochastic values, we need to provide a way to combine stochastic values arithmetically. In this subsection we define common arithmetic interaction operators for stochastic values represented by normal distributions by taking advantage of the fact that normal distributions are closed under linear combinations [LM86].

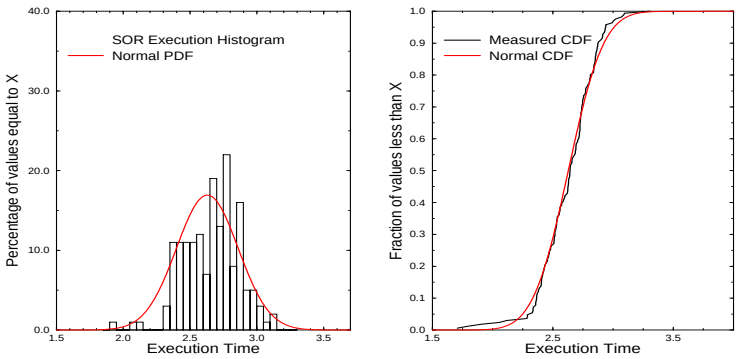


Figure 15.2. Graphs showing the PDF and CDF of SOR benchmark with normal distribution based on data mean and standard deviation.

For each arithmetic operation, we define a rule for combining stochastic values based on standard statistical error propagation methods [Bar78]. In the following, we assume that point values are represented by P and all stochastic values are of the form (m_i, sd_i) and represent normal distributions, where m_i is the mean and sd_i is the standard deviation.

When combining two stochastic values, two cases must be considered: correlated and uncorrelated distributions. Two distributions are *correlated* when there is an association between them, that is, they jointly vary in a similar manner [DP96a]. More formally, correlation is the degree to which two or more attributes, or measurements, on the same group of elements show a tendency to vary together. For example, when network traffic is heavy, available bandwidth tends to be low, and latency tends to be high. When network traffic is light, available bandwidth tends to be high, and latency tends to be low. We say that the distributions of latency and bandwidth are correlated in this case.

When two stochastic values are *uncorrelated*, they do not jointly vary in a similar manner. This case may occur when the time between measurements of a single quantity is large or when the two stochastic values represent distinct characteristics. For example, available CPU on two machines not running any applications in common may be uncorrelated.

Table 15.1 summarizes the arithmetic operations between a stochastic value and a point value, two stochastic values from correlated distributions, and two stochastic values from uncorrelated distributions.

Note that the product of stochastic values with normal distributions does not itself have a normal distribution. Rather, it is long-tailed. In many circumstances, we can approximate the long-tailed distribution with a normal distribution and ignore the tail, as discussed below in Section 4.1.2.

Table 15.1. Arithmetic combinations of a stochastic value with a point value and with other stochastic values [Bar78].

	Addition	Multiplication
Point Value and Stochastic Value	$(m_i, sd_i) + P = ((m_i + P), sd_i)$	$P(m_i, sd_i) = (Pm_i, Psd_i)$
Stochastic Values with Correlated Distributions	$\sum_{i=1}^{n} (m_i, sd_i) = \left(\sum_{i=1}^{n} m_i, \quad \sum_{i=1}^{n} \lvert sd_i \rvert\right)$	$(m_i, sd_i)(m_j, sd_j) = (m_i m_j, (sd_i m_j + sd_j m_i + sd_i sd_j))$
Stochastic Values with Uncorrelated Distributions	$\sum_{i=1}^{n} (m_i, sd_i) \approx \left(\sum_{i=1}^{n} m_i, \sqrt{\sum_{i=1}^{n} sd_i^2}\right)$	$(m_i, sd_i)(m_j, sd_j) \approx \left(m_i m_j, \left(m_i m_j \sqrt{\left(\frac{sd_i}{m_i}\right)^2 + \left(\frac{sd_j}{m_j}\right)^2}\right)\right)$

4.1.2 Using Normal Distributions to Represent Nonnormal Stochastic Model Parameters

In this section, we provide examples of stochastic parameters that are not normal but can often be adequately represented by normal distributions.

Not all system characteristics can be accurately represented by normal distributions. Figure 15.3 shows the PDF and CDF for bandwidth data between two workstations over 10 Mbit Ethernet. This is a typical graph of a *long-tailed distribution*; that is, the data has a threshold value and varies monotonically from that point, generally with the median value several points below (or above) the threshold. A similarly shaped distribution, shown in Figure 15.4 on the left, may be found in data resulting from dedicated runs of a nondeterministic distributed genetic algorithm code.

Neither of these distributions is normal; however, it may be adequate to approximate them by using normal distributions. Normal distributions are a good substitution for long-tailed model parameters only when inaccuracy in the predictions generated by the structural model can be tolerated by the scheduler, performance model, or other mechanism that uses the data.

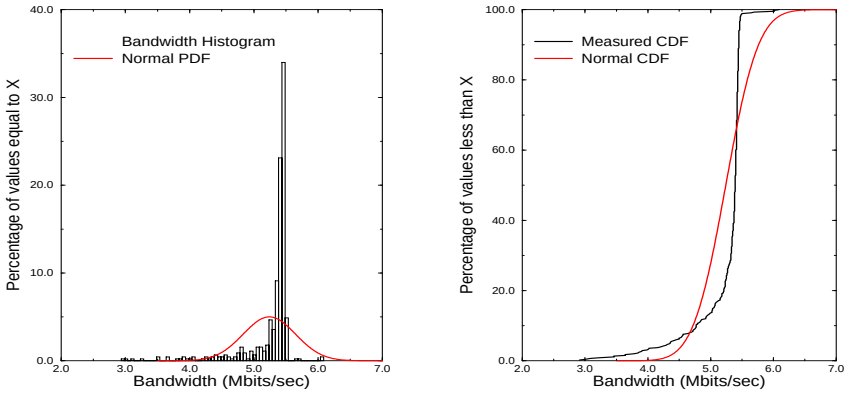


Figure 15.3. Graphs showing the PDF and CDF for bandwidth between two workstations over 10 Mbit Ethernet with long-tailed distribution and corresponding normal distribution.

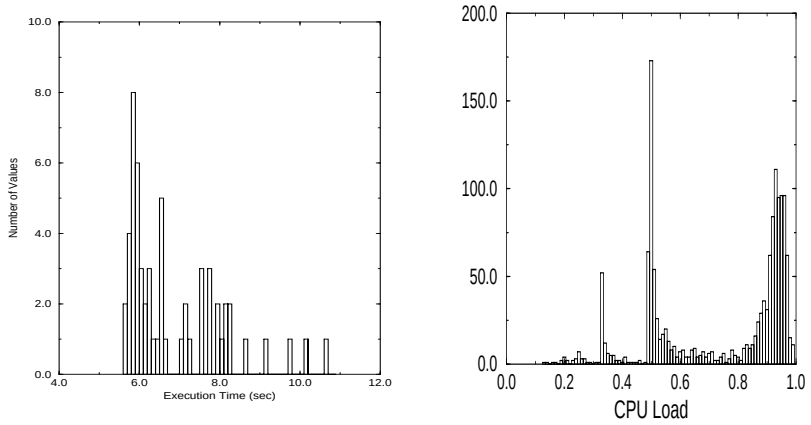


Figure 15.4. Two examples of nonnormal distribution behavior: a histogram of a nondeterministic application on the left; Available CPU on a production workstation on the right.

Alternatively, some model parameters are best represented by multimodal distributions. One characterization of the general shape of a distribution is the number of peaks, or *modes*. A distribution is said to be *unimodal* if it has a single peak, *bimodal* if it has two peaks, and *multimodal* if it has more than two peaks. Figure 15.4 on the left shows a histogram of available CPU data for an Ultra Sparc workstation running Solaris taken over 12 hours using `vmstat`. The Unix tool `vmstat` reports the exact CPU activity at a given time, in terms of the processes in the run queue, the blocked processes, and the swapped processes as a snapshot of the system every n seconds (where for our

trace, n = 5). For this distribution, the majority of the data lies in three modes: a mode centered at 0.94, a mode centered at 0.49, and a mode centered at 0.33.

For this data, the modes are most likely an artifact of the scheduling algorithm of the operating system. Most Unix-based operating systems use a round-robin algorithm to schedule CPU bound processes: When a single process is running, it receives all of the CPU; when two processes are running, each uses approximately half of the CPU; when there are three, each gets a third; and so forth. This is the phenomenon exhibited in Figure 15.4.

To represent a modal parameter using a normal distribution, we need to know whether values represented by the parameter remain within a single mode during the timeframe of interest. If the values of the parameter remain within a single mode (i.e. if they exhibit *temporal locality*), we can approximate the available CPU as a normal distribution based on the data mean and standard deviation of the appropriate mode without excessive loss of information. An example of this (from a 24-hour trace of CPU loads) is shown as a time series in Figure 15.5 on the left.

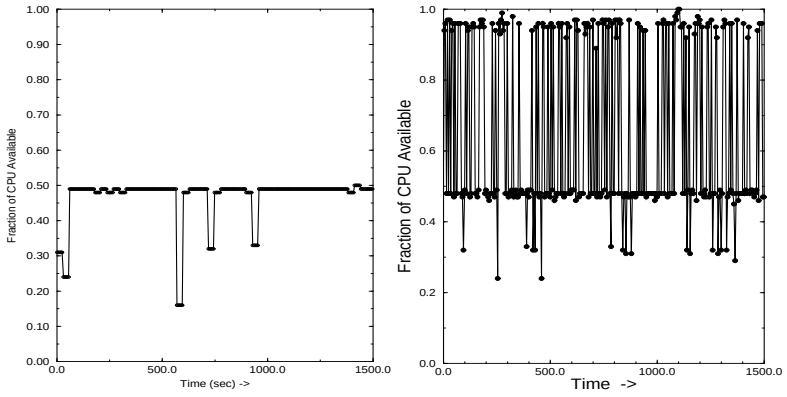


Figure 15.5. Two time series showing temporal locality (on the left) and nontemporal locality (on the right) for CPU data.

If the values of the parameter change modes frequently or unpredictably, we say that that the data exhibits *temporal nonlocality*. An example of this, taken from the same 24-hour CPU trace as before, is shown as a time series in Figure 15.5 on the right. In this case, some way of deriving a prediction must be devised that takes into account the fluctuation of the parameter data between multiple modes.

A brute-force approach to representing multimodal data would be to simply ignore the multimodality of the data and represent the stochastic value as a normal distribution based on the mean and standard deviation of the data as a whole. This approximation is, however, unlikely to capture the relevant behavior characteristics of the data with any accuracy. Because of the multi-

modal behavior, a mode with a small variance in actuality may end up being represented by a normal distribution with a large variance (and a large standard deviation). If this brute-force method were used for the data in Figure 15.5, the mean would be 0.66 and the standard deviation would be 0.24.

An alternative approach is to calculate an *aggregate mean* and *aggregate standard deviation* for the value based on the mean and standard deviation for each mode. Let (m_i, sd_i) represent the mean and the standard deviation for the data in mode i. We define the *aggregate mean (AM)* and the *aggregate standard deviation (ASD)* of a multimodal distribution by

$$AM = \sum p_i(m_i) \tag{15.4}$$

$$ASD = \sum p_i(sd_i), \tag{15.5}$$

where p_i is the percentage of data in mode i. Since we represent each individual mode in term of a normal distribution, (AM, ASD) will also have a normal distribution. For the data in Figure 15.5, AM = 0.68 and ASD = 0.031.

Note that using the aggregate mean and the aggregate standard deviation is an attempt to define a normal distribution that is somehow close to the multimodal distribution. Determining whether two distributions are close is itself an interesting problem that we discuss briefly in the subsection below.

4.1.3 When Is a Distribution *Close* to Normal?

In the preceding subsections, we made a key assumption that the values in the distribution were *close* to (could be adequately represented by) normal distributions. To define "close," we can consider several methods for determining the similarity between a given data set and the normal distribution represented by its data mean and its data standard deviation.

One common measurement of goodness of fit is the chi-squared (χ^2) technique [DP96b]. This is a quantitative measure of the extent to which observed counts differ from the expected counts over a given range, called a cell. The value for χ^2 is the sum of a goodness of fit for all quantities

$$\chi^2 = \sum_{all\ cells} \frac{(\text{observed cell count - expected cell count})^2}{\text{expected cell count}} \tag{15.6}$$

for the observed data and the expected data resulting from the normal distribution. The value of the χ^2 statistic reflects the magnitude of the discrepancies between observed and expected cell counts: a larger value indicates larger discrepancies.

Another metric of closeness in the literature is called 1-distance between PDF's [MLH95], where

$$\|f_n - f_d\|_1 = \int_{-\infty}^{\infty} |f_n(x) - f_p(x)| dx \tag{15.7}$$

for function f_d for the data and f_n for the normal distribution based on the data mean and standard deviation. This corresponds to a maximal error between the functions.

For both of these metrics, a user or scheduler would need to determine a threshold for closeness acceptable for their purposes.

If we approximate nonnormal data using a normal distribution, there may be several effects. When the distribution of a stochastic value is represented by a normal distribution but is not actually normal, arithmetic operations might exclude values that they should not. By performing arithmetic on the mean and standard deviation, we are able to use optimistic formulas for uncorrelated values in order to narrow the range that is considered in the final prediction. If the distributions of stochastic values were actually long-tailed, for example, this might cut off values from the tail in an unacceptable way.

Normal distributions are closed under linear combinations [LM86], but general distributions are not. If we use arithmetic rules defined for normal distributions on nonnormal data, we have no information about the distribution of the result. Further, it may not be possible to ascertain the distribution of a stochastic value, or the distribution may not be sufficiently close to normal. In such cases, other representations must be used. In the next section, we explore an alternative for representing stochastic values using an aggregated prediction technique.

4.2 Aggregate Predictions

In this section we describe how a time series predictor can be extended to obtain three types of predicted CPU load information: the next step predicted CPU load at a future time point (Section 4.2.1); the average interval CPU load for some future time interval (Section 4.2.2); and the variation of CPU load over some future time interval (Section 4.2.3).

4.2.1 One-Step-Ahead CPU Load Prediction

The tendency-based time series predictor developed in our previous work can provide one-step-ahead CPU load prediction based on history CPU load

```
// Determine Tendency
      if ((V_(T-1) - V_T )<0)
 Tendency="Increase";
    else if ((V_T - V_(T-1)))<0)
           Tendency="Decrease";
      if (Tendency="Increase") then
             PT+1 = V_T + IncrementConstant;
             IncrementConstant adaptation process
      else if (Tendency="Decrease") then
             PT+1 = V_T - V_T*DecrementFactor;
             DecrementFactor adaptation process
```

Figure 15.6. Psuedo-code for Tendency algorithm.

information [YFS03]. This predictor has been demonstrated to be more accurate than other predictors for CPU load data. It achieves prediction errors that are between 2% and 55% less (36% less on average) than those incurred by the predictors used within the NWS on a set of 38 machines load traces. The algorithm predicts the next value according to the tendency of the time series change assuming that if the current value increases, the next value will also increase and that if the current value decreases, the next value will also decrease.

Given the preceding history data measured at a constant-width time interval, our mixed tendency-based time series predictor uses the algorithm in Figure 15.6, where V_T is the measured value at the Tth measurement and $PT + 1$ is the predicted value for measurement value V_{T+1}.

We find that a mixed-variation (that is, different behavior for the increment from that of the decrement) experimentally performed best. The IncrementConstant is set initially to 0.1, and the DecrementFactor is set to 0.01. At each time step, we measure the real data (V_{T+1}) and calculate the difference between the current measured value and the last measured value (V_T) to determine the real increment (decrement) we should have used in the last prediction in order to get the actual value. We adapt the value of the increment (decrement) value accordingly and use the adapted IncrementConstant (or DecrementFactor) to predict the next data point.

Using this time series predictor to predict the CPU load in the next step, we treat the measured preceding CPU load time series as the input to the predictor. The predictor's output is the predicted CPU load at the next step, P_{n+1}. So if the time series $C = c_1, c_2 \ldots c_n$ is the CPU load time series measured at constant-width time interval and is used as input to the predictor, the result is the predicted value P_{n+1} for the measurement value c_{n+1}.

4.2.2 Interval Load Prediction

Instead of predicting one step ahead, we want to be able to predict the CPU load over the time interval during which an application will run. Since the CPU load time series exhibits a high degree of self-similarity [Din99], averaging values over successively larger time scales will not produce time series that are dramatically smoother. Thus, to calculate the predicted average CPU load an application will encounter during its execution, we need to first aggregate the original CPU load time series into an interval CPU load time series, then run predictors on this new interval time series to estimate its future value.

Aggregation, as defined here, consists of converting the original CPU load time series into an interval CPU load time series by combining successive data over a nonoverlapping larger time scale. The aggregation degree, M, is the number of original data points used to calculate the average value over the time interval. This value is determined by the resolution of the original time series and the execution time of the applications, and need be only approximate.

For example, the resolution of the original time series is 0.1 Hz, or measured every 10 seconds, and if the estimated application execution time is about 100 seconds, the aggregation degree M can be calculated by

$$\begin{aligned} M &= ExecTimeOfApplication * FreqOfOriginalTimeSeries \\ &= 100 * 0.1 \\ &= 10 \end{aligned} \tag{15.8}$$

Hence, the aggregation degree is 10. In other words, 10 data points from the original time series are needed to calculate one aggregated value over 100 seconds. The process of aggregation consists of translating the incoming time series, ($C = c_1, c_2, \ldots c_n$), into the aggregated time series, ($A = a_1, a_2, \ldots a_k$), such that

$$a_i = \frac{\sum_{j=1..M} C_{n-(k-i+1)*M+j}}{M} \tag{15.9}$$

for $i = 1 \ldots k$i for $k = \lceil \frac{n}{M} \rceil$. Each value in the interval CPU load time series a_i is the average CPU load over the time interval that is approximately equal to the application execution time.

After the aggregated time series is created, the second step of our interval load prediction involves using the one-step-ahead predictor on the aggregated time series to predict the mean interval CPU load. So the aggregated time series A_i is fed into the one-step-ahead predictor, resulting in pa_{K+1}, the predicted value of a_{k+1}, which is approximately equal to the average CPU load the application will encounter during execution.

4.2.3 Load Variance Prediction

To predict the variation of CPU load, for which we use standard deviation, during the execution of an application, we need to calculate the standard deviation time series using the original CPU load time series C and the interval CPU load time series A (defined in the preceding section).

Assuming the original CPU load time series is $C = c_1, c_2, \ldots c_n$, the interval load time series is $A = a_1, a_2, \ldots a_k$, and an aggregation degree of M, we can calculate the standard deviation CPU load time series $S = s_1, s_2, \ldots s_k$:

$$S_i = \sqrt{\sum_{j=1\ldots M} \frac{\left(C_{n-(k-i+1)*M+j-A_i}\right)^2}{M}} \tag{15.10}$$

for $i = 1 \ldots k$.

Each value in standard deviation time series s_i is the average difference between the CPU load and the mean CPU load over the interval.

To predict the standard deviation of the CPU load, we use the one-step-ahead predictor on the standard deviation time series. The output ps_{k+1} will be the predicted value of s_{k+1}, or the predicted CPU load variation for the next time interval.

5. APPLICATION SCHEDULING

Our goal is to improve data mapping in order to reduce total application execution time despite resource contention. To this end, we use the time-balancing scheduling algorithm described in Section 3, parameterized with an estimate of future resource capability.

5.1 Cactus Application

We apply our scheduling algorithms in the context of Cactus, a simulation of a 3D scalar field produced by two orbiting astrophysical sources. The solution is found by finite differencing a hyperbolic partial differential equation for the scalar field. This application decomposes the 3D scalar field over processors and places an overlap region on each processor. For each time step, each processor updates its local grid point and then synchronizes the boundary values. It is an iterative, loosely synchronous application, as described in Section 4. We use a one-dimensional decomposition to partition the workload in our experiments. The full performance model for Cactus is described in [LYFA02], but in summary it is

$$\begin{aligned} E_i(D_i) = \quad & startUpTime + (D_i * Comp_i + Comm_i) \\ & * slowdown(\textit{effective CPU load}) \end{aligned} \tag{15.11}$$

$Comp_i$ and $Comm_i$, the computation time of per data point and communication time of the Cactus application in the absence of contention, can be calculated by formulas described in [RIF01b]. We incur a startup time when initiating computation on multiple processors in a workstation cluster that was experimentally measured and fixed. The function *slowdown(effective CPU load)*, which represents the contention effect on the execution time of the application, can be calculated by using the formula described in [LYFA02].

The performance of the application is greatly influenced by the actual CPU performance achieved in the presence of contention from other competing applications. The communication time is less significant when running on a local area network, but for wide-area network experiments this factor would also be parameterized by a capacity measure.

Thus, our problem is to determine the value of CPU load to be used to evaluate the slowdown caused by contention. We call this value the effective CPU load and equate it to the average CPU load the application will experience during its execution.

5.2 Scheduling Approaches

As shown in Figure 1, variations in CPU load during task execution can also influence the execution time of the job because of interrelationships among tasks. We define a conservative scheduling technique that always allocates less work to highly varying machines. For the purpose of comparison, we define the effective CPU load in a variety of ways, each giving us a slightly different scheduling policy. We define five policies to compare in the experimental section:

- One-step scheduling (OSS): Use the one-step-ahead prediction of the CPU load, as described in Sections 4.2.1, for the effective CPU load.
- Predicted mean interval scheduling (PMIS): Use the interval load prediction, described in Section 4.2.2, for the effective CPU load.
- Conservative scheduling (CS): Use the conservative load prediction, equal to the interval load prediction (defined in Section 4.2.2) added to a measure of the predicted variance (defined in Section 4.2.3) for the effective CPU load. That is, effective CPU load= $pa_{k+1} + ps_{k+1}$.
- History mean scheduling (HMS): Use the mean of the history CPU load for the 5 minutes preceding the application start time for the value for effective CPU load. This approximates the estimates used in several common scheduling approaches [TSC00, WZ98].
- History conservative scheduling (HCS): Use the conservative estimate CPU load defined by using the normal distribution stochastic value de-

fined in Section 4.1. In practice, this works out to adding the mean and variance of the history CPU load collected for 5 minutes preceding the application run as the effective CPU load.

6. EXPERIMENTS

To validate our work, we conducted experiments on workstation clusters at the University of Illinois at Champaign-Urbana (UIUC) and the University of California, San Diego (UCSD), which are part of the GrADS testbed [BCC+01]

6.1 Experimental Methodology

We compared the execution times of the Cactus application with the five scheduling policies described in Section 5: one-step scheduling (OSS), predicted mean interval scheduling (PMIS), conservative scheduling (CS), history mean scheduling (HMS), and history conservative scheduling (HCS).

At UIUC, we used a cluster of four Linux machines, each with a 450 MHz CPU; at UCSD, we used a cluster of six Linux machines, four machines with a 1733 MHz CPU, one with a 700 MHz CPU, and one with a 705 MHz CPU. All machines were dedicated during experiments.

To evaluate the different scheduling polices under identical workloads, we used a load trace playback tool [DO00] to generate a background workload from a trace of the CPU load that results in realistic and repeatable CPU contention behavior. We chose nine load time series available from [Yan03]. These are all traces of actual machines, which we characterize by their mean and standard deviation. We used 100 minutes of each trace, at a granularity of 0.1 Hz. The statistic properties of these CPU load traces are shown in Table 15.2. Note that even though some machines have the same speed, the performance that they deliver to the application varied that they each experienced different background loads.

Table 15.2. The mean and standard deviation of 9 CPU load traces.

CPU Load Trace Name	Machine Name	Mean	SD
LL1	abyss	0.12 (L)	0.16 (L)
LL2	axp7	0.02 (L)	0.06 (L)
LH1	vatos	0.22 (L)	0.31 (H)
LH2	axp1	0.14 (L)	0.29 (H)
HL1	mystere	1.85 (H)	0.14 (L)
HL2	pitcairn	1.19 (H)	0.12 (L)
HH	axp0	1.07 (H)	0.48 (H)
HH2	axp10	1.18 (H)	0.31 (H)

6.2 Experimental Results

Results from four representative experiments are shown in Figures 15.7–15.10. A summary of the testbeds and the CPU load traces used for the experiments is given in Table 15.3.

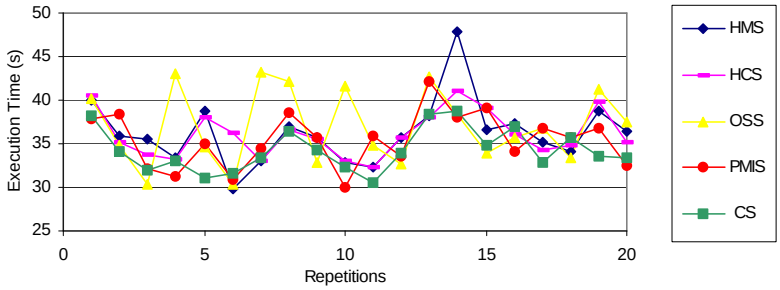


Figure 15.7. Comparison of the history mean, history conservative, one-step, predicted mean interval and conservative scheduling policies on the UIUC cluster with two low-variance machines (one with a low mean and the other with a high mean) and two high-variance machines (one with a low mean, the other with a high mean).

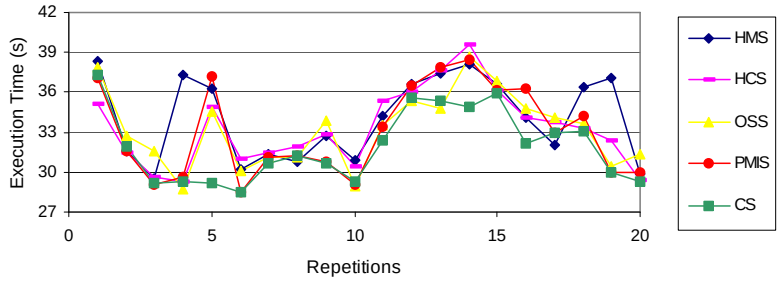


Figure 15.8. Comparison of the history mean, history conservative, one-step, predicted mean interval and conservative scheduling policies on the UIUC cluster with two low-variance machines and two high-variance machines (all with a low mean).

Table 15.3. CPU load traces used for each experiment.

Experiments	Testbed	CPU Load Traces
Fig. 15.7	UIUC	LL1, LH1, HL1, HH1
Fig. 15.8	UIUC	LL1, LL2, LH1, LH2
Fig. 15.9	UCSD	LL1, LL2, LH1, LH2, HL1, HL2
Fig. 15.10	UCSD	LH1, LH2, HL1, HL2, HH1, HH2

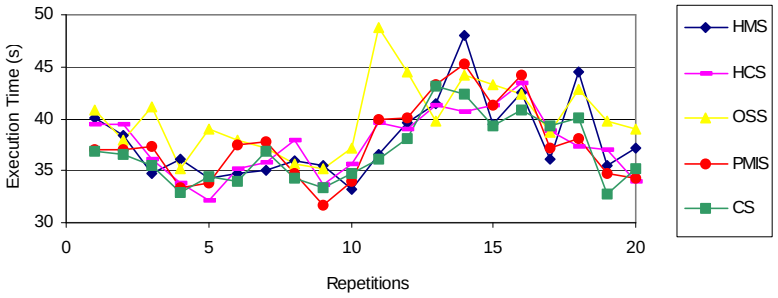


Figure 15.9. Comparison of the history mean, history conservative, one-step, predicted mean interval and conservative scheduling policies on the UCSD cluster with four low-variance machines (one with a low means and two with a high means) and two high-variance machines (with low means).

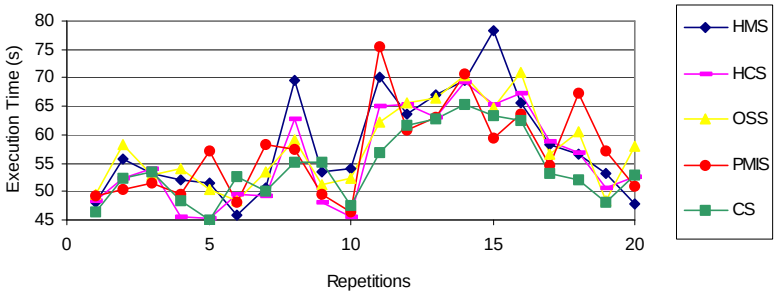


Figure 15.10. Comparison of the history mean, history conservative, one-step, predicted mean interval and conservative scheduling policies on the UCSD cluster with two low-variance machines (all with high means) and four high-variance machines (two with a low mean, two with a high mean).

To compare these policies, we used two metrics: an absolute comparison of run times and a relative measure of achievement. The first metric involves an average mean and an average standard deviation for the set of runtimes of each scheduling policy as a whole, as shown in Table 15.4. This metric gives a rough valuation on the performance of each scheduling policy over a given interval of time. Over the entire run, the conservative scheduling policy exhibited 2%–7% less overall execution time than history mean and history conservative scheduling policies, by using better information prediction, and 1.2%–7% less overall execution time than did the one-step and predicted mean interval scheduling policies. We also see that taking variation information into account in the scheduling policy results in more *predictable* application behavior: The history conservative scheduling policy exhibited 9%–29% less standard deviation of execution time than did the history mean. The conservative scheduling

Table 15.4. Average mean and average standard deviation for entire set of runs for each scheduling policy.

Exp.	HMS		HCS		OSS		PMIS		CS	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
Fig. 15.7	36.2	3.7	36.1	2. 6	37.0	4.2	35.4	3.2	34.3	2.4
Fig. 15.8	34.1	3.1	33.3	2.8	33.2	2.7	33.0	3.4	31.9	2.7
Fig. 15.9	38.0	3.8	37.6	3.0	37.8	3.5	37.6	3.8	36.8	3.1
Fig. 15.10	58.2	9.1	55.7	8.1	57.7	7.2	57.0	8.0	54.2	6.1

policy exhibited 1.5%–41% less standard deviation in execution time than the one-step scheduling policy and 20%–41% less standard deviation of execution time than the predicted mean interval scheduling policy.

The second metric we used, *Compare*, is a relative metric that evaluates how often each run achieves a minimal execution time. We consider a scheduling policy to be better than others if it exhibits a lower execution time than another policy on a given run. Five possibilities exist: *best* (best execution time among the five policies), *good* (better than three policies but worse than one), *average* (better than two policies and worse than two), *poor* (better than one policy but worse than three), and *worst* (worst execution time of all five policies).

These results are given in Table 15.5, with the largest value in each case shown in boldface. The results indicate that conservative scheduling using predicted mean and variation information is more likely to have a *best* or *good* execution time than the other approached on both clusters. This fact indicates that taking account of the average and variation CPU information during the period of application running in the scheduling policy can significantly improve the application's performance.

To summarize our results: independent of the loads and CPU capabilities considered on our testbed, the conservative scheduling policy based on our tendency-based prediction strategy with mixed variation achieved better results than the other policies considered. It was both the best policy in more situations under all load conditions on both clusters, and the policy that resulted in the shortest execution time and the smallest variation in execution time.

7. CONCLUSIONS AND FUTURE WORK

We have proposed a conservative scheduling policy able to achieve efficient execution of data-parallel applications even in heterogeneous and dynamic environments. This policy uses information about the expected mean and variance of future resource capabilities to define data mappings appropriate for dynamic resources. Intuitively, the use of variance information is appealing

Table 15.5. Summary statistics using Compare to evaluate five scheduling policies.

Experiment	Policy	Best	Good	Avg	Poor	Worst
Fig. 15.7	HMS	2	2	7	3	6
	HCS	1	4	6	6	3
	OSS	5	5	0	2	8
	PMIS	6	2	3	7	2
	CS	6	7	4	2	1
Fig. 15.8	HMS	2	2	5	5	6
	HCS	2	3	4	6	5
	OSS	4	2	5	3	6
	PMIS	1	8	3	5	3
	CS	11	5	3	1	0
Fig. 15.9	HMS	4	3	5	4	4
	HCS	4	3	7	4	2
	OSS	1	1	4	4	10
	PMIS	1	10	0	6	3
	CS	10	3	4	2	1
Fig. 15.10	HMS	2	2	5	7	4
	HCS	4	3	6	5	2
	OSS	0	3	6	5	6
	PMIS	4	8	1	1	6
	CS	10	4	2	2	2

because it provides a measure of resource reliability. Our results suggest that this intuition is valid.

Our work comprises two distinct components. First, we show how to obtain predictions of expected mean and variance information. Then we show how information about expected future mean and variance (as obtained, for example, from our predictions) can be used to guide data mapping decisions. In brief, we assign less work to less reliable (higher variance) resources, thus protecting ourselves against the larger contending load spikes that we can expect on those systems. We apply our prediction techniques and scheduling policy to a substantial astrophysics application. Our results demonstrate that our techniques can obtain better execution times and more predictable application behavior than previous methods that focused on predicted means alone. While the performance improvements obtained are modest, they are obtained consistently and with no modifications to the application beyond those required to support nonuniform data distributions.

We are interested in extending this work to other dynamic system information, such as network status. Another direction for further study is a more sophisticated scheduling policy that may better suit other particular environments and applications.

Acknowledgments

We are grateful to Peter Dinda for permitting us to use his load trace play tool, and to our colleagues within the GrADS project for providing access to testbed resources. This work was supported in part by the Grid Application Development Software (GrADS) project of the NSF Next Generation Software program, under Grant No. 9975020, and in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under contract W-31-109-Eng-38.

Chapter 16

IMPROVING RESOURCE SELECTION AND SCHEDULING USING PREDICTIONS

Warren Smith
Computer Sciences Corporation
NASA Ames Research Center

Abstract

The introduction of computational Grids has resulted in several new problems in the area of scheduling that can be addressed using predictions. The first problem is selecting where to run an application on the many resources available in a Grid. Our approach to help address this problem is to provide predictions of when an application would start to execute if submitted to specific scheduled computer systems. The second problem is gaining simultaneous access to multiple computer systems so that distributed applications can be executed. We help address this problem by investigating how to support advance reservations in local scheduling systems. Our approaches to both of these problems are based on predictions for the execution time of applications on space-shared parallel computers. As a side effect of this work, we also discuss how predictions of application run times can be used to improve scheduling performance.

1. INTRODUCTION

The existence of computational Grids allows users to execute their applications on a variety of different computer systems. An obvious problem that arises is how to select a computer system to run an application. Many factors go into making this decision: The computer systems that the user has access to, the user's remaining allocations on these systems, the cost of using different systems, the location of data sets for the experiment, how long the application will execute on different computers, when the application will start executing, and so on. We wish to help users make this decision, so in Section 3, we discuss approaches to predicting when a scheduling system for a space shared parallel computer will start executing an application.

The large number of resources available in computational Grids leads users to want to execute applications that are distributed across multiple resources, such as running a large simulation on 2 or more supercomputers. The difficulty with this is that different resources may have different scheduling systems without any mechanisms to guarantee that an application obtains simultaneous access to the different resources. To address this problem, Section 5 describes ways to incorporate advance reservations into scheduling systems and analyzes their performance. Users can use advance reservations to ask for resources from multiple schedulers at the same time in the future and obtain simultaneous access. As a side effect of this work, in Section 4 we discuss how to improve the performance of scheduling systems even when no reservations are needed.

We base much of the work above on techniques to predict the execution time of applications on space shared parallel computers. We begin this chapter by describing two techniques to calculate predictions of application run times and discuss their performance.

2. RUN TIME PREDICTIONS

There have been many efforts to predict the execution time of serial and parallel applications. We can roughly classify prediction techniques by whether the resources that are used are shared or dedicated, the type and detail of application and resource models that are used, and the type of prediction technique used.

There have been many efforts to predict the execution time of applications on shared computer systems [DI89, Din01, WSH99b, SB98] and dedicated computer systems [SFT98, SW02, Dow97, Gib97, IOP99, KFB99]. Almost all of the techniques referenced above use very simple models of an application such as the information provided to the scheduler or this information plus a few application-specific parameters [KFB99]. An exception is the work by Schopf [SB98, Sch97] where high-level structural models of applications were created with the assistance of the creators of the applications.

The type of prediction technique that is used can generally be separated into statistical, which uses statistical analysis of applications that have completed, and analytical, which constructs equations describing application execution time. Statistical approaches use time series analysis [WSH99b, Din01], categorization [SFT98, Dow97, Gib97], and instance-based learning or locally weighted learning [KFB99, IOP99, SW02]. Analytical approaches develop models by hand [SB98] or use automatic code analysis or instrumentation [TWG+01].

We present and analyze the performance of two techniques that we have developed. Both techniques are statistical and only use the information provided

when an application is submitted to a scheduler to make predictions. The first technique uses categories to form predictions and the second technique uses instance-based learning.

2.1 Categorization Prediction Technique

Our first approach to predicting application run times is to derive run time predictions from historical information of previous similar runs. This approach is based on the observation [SFT98, Dow97, FN95, Gib97] that similar applications are more likely to have similar run times than applications that have nothing in common.

One difficulty in developing prediction techniques based on similarity is that two applications can be compared in many ways. For example, we can compare applications on the basis of application name, submitting user name, executable arguments, submission time, and number of nodes requested. When predicting application run times in this work, we restrict ourselves to those values recorded in traces obtained from various supercomputer centers. The workload traces that we consider originate from 3 months of data from an IBM SP at Argonne National Laboratory (ANL), 11 months of data from an IBM SP at the Cornell Theory Center (CTC), and 2 years of data from an Intel Paragon at the San Diego Supercomputer Center (SDSC). The characteristics of jobs in each workload vary but consist of a relatively small set of characteristics such as *user name,* application name, *queue name,* run time, and so on.

Our general approach to defining similarity is to use job characteristics to define templates that identify a set of categories to which applications can be assigned. For example, the template (queue,user) specifies that applications are to be partitioned by queue and user. On the SDSC Paragon, this template generates categories such as (q16m, wsmith), (q64l, wsmith), and (q16m, foster).

We find that categorizing discrete characteristics (such as user name) in the manner just described works reasonably well. On the other hand, the number of nodes is an essentially continuous parameter, and so we prefer to introduce an additional parameter into our templates, namely, a node range size that defines what ranges of requested number of nodes are used to decide whether applications are similar. For example, the template (u, n=4) specifies a node range size of 4 and generates categories (wsmith, 1-4 nodes) and (wsmith, 5-8 nodes).

In addition to the characteristics of jobs contained in the workloads, a *maximum history*, *type of data* to store, and *prediction type* are also defined for each run time prediction template. The maximum history indicates the maximum number of data points to store in each category generated from a template. The type of data is either an actual run time or a relative run time. A relative run

time incorporates information about user-supplied run time estimates by storing the ratio of the actual run time to the user-supplied estimate (the amount of time the nodes are requested for). The prediction type determines how a run time prediction is made from the data in each category generated from a template. We considered four prediction types in our previous work: a mean, a linear regression, an inverse regression, and a logarithmic regression [DS81]. We found that the mean is the single best predictor [Smi99], so this work uses only means to form predictions. We also take into account running time, how long an application has been running when a prediction is made, when making predictions by ignoring data points from a category that have a run time less than this running time.

Once a set of templates has been defined (using a search process described later), we simulate a workload of application predictions and insertions. For each insertion, an application is added to the categories that contain similar applications. For each prediction, an execution time and a confidence interval is calculated. A prediction is formed from each similar category of applications and the prediction with the smallest confidence interval is selected to be the prediction for the application.

We use a genetic algorithm search to identify good templates for a particular workload. While the number of characteristics included in our searches is relatively small, the fact that effective template sets may contain many templates means that an exhaustive search is impractical. Genetic algorithms are a probabilistic technique for exploring large search spaces, in which the concept of cross-over from biology is used to improve efficiency relative to purely random search [Gol89]. A genetic algorithm evolves individuals over a series of generations. Our individuals represent template sets. Each template set consists of between 1 and 10 templates, and we encode the previously described information in each template. The process for each generation consists of evaluating the fitness of each individual in the population, selecting which individuals will be mated to produce the next generation, mating the individuals, and mutating the resulting individuals to produce the next generation. The process then repeats until a stopping condition is met. The stopping condition we use is that a fixed number of generations are processed. Further details of our searches are available in [Smi99].

The accuracy of the prediction parameters found by our searches are shown in the second column of Table 16.1. The prediction error is 39 percent on average and ranges from 28 percent for the SDSC96 workload to 54 percent for the CTC workload. We compare these results to the run time prediction performance technique proposed by Gibbons [Gib97] and the run time prediction technique used by Downey [Dow97] to predict how long jobs will wait at the

Table 16.1. Performance of our categorization technique versus those of Gibbons and Downey.

Workload	Mean Error (minutes)			Mean Run
	Ours	Gibbons	Downey	Time (minutes)
ANL	34.52	75.26	97.01	97.08
CTC	98.28	124.06	179.46	182.49
SDSC95	43.20	74.05	82.44	108.16
SDSC96	47.47	122.55	102.04	166.85

head of a scheduling queue before beginning to execute. Both of these techniques categorize applications that have completed executing, find categories that are similar to an application whose run time is to be predicted, and then form a prediction from these categories. The categories and techniques used to calculate predictions differ between the two techniques and differ from our technique. Table 16.1 shows that our predictions have between 21 and 61 percent lower mean error than the Gibbons' approach and 45 to 64 percent lower mean error than the better of Downey's two techniques.

2.2 Instance-Based Learning Approach

In our second approach, we predict the execution time of applications using *instance-based learning* (IBL) techniques that are also called *locally weighted learning techniques* [AMS97, SM00]. In this type of technique, a database of experiences, called an experience base, is maintained and used to make predictions. Each experience consists of input and output features. Input features describe the conditions under which an experience was observed and the output features describe what happened under those conditions. Each feature consists of a name and a value where the value is of a simple type such as integer, floating-point number, or string. In this work, the input features are the same ones as used in our first approach: The user who submitted the job, the application that was executed, the number of CPUs requested, and so on. The execution time of the job is the only output feature of the experience.

When a prediction is to be performed, a query point consisting of input features is presented to the experience base. The data points in the experience base are examined to determine how relevant they are to the query where relevance is determined using the distance between an experience and the query. There are a variety of distance functions that can be used [WM97] and we choose to use the Heterogeneous Euclidean Overlap Metric [WM97]. This distance function can be used on features that are linear (numbers) or nominal (strings). We require support for nominal values because important features such as the names of executables, users, and queues are nominal. As a further refinement, we perform feature scaling to stretch the experience space and increase the importance that certain features are similar.

Once we know the distance between experiences and a query point, the next question to be addressed is how we calculate estimates for the output features of the query point. For linear output features, such as execution time, our approach is to use a distance-weighted average of the output features of the experiences to form an estimate. We choose to use a Gaussian function to form this distance-weighted average. Further, we include a factor, called the kernel width, so that we can compact or stretch the Gaussian function to give lower or higher weights to experiences that are farther away.

To perform an estimate, we must select values for the parameters discussed above along with the maximum experience base size and the number of nearest neighbors (experiences) to use. Our approach to determine the best values for these parameters is to once again perform a genetic algorithm search [Gol89] to select values that minimize the prediction error.

Table 16.2 shows a comparison of our categorization technique and our instance-based learning technique using the same trace data that we used to evaluate our first technique. The table shows that at the current time, our instance-based learning technique has an error which is 44 percent of mean run times and 59 percent lower than the user estimates available in the ANL and CTC workloads. The user estimates for the ANL and CTC workloads are provided along with each job. Our IBL technique has an 89 percent lower error than the run time estimates that can be derived from the SDSC workloads. This system has many queues with different resource usage limits. We derive the run time limits for each queue by examining the workloads and finding the longest running job submitted to each queue.

Table 16.2 also shows that the error of our categorization technique is currently 10 percent lower than our IBL technique. There are several possible reasons for this result. First, we performed more extensive searches to find the best parameters used in the categorization technique. Second, the categorization technique essentially uses multiple distance functions and selects the best results obtained after using each of these functions instead of the single distance function used by our IBL approach. In future work, we will evaluate how well our IBL approach performs when using multiple distance functions.

3. QUEUE WAIT TIME PREDICTIONS

On many high-performance computers, a request to execute an application is not serviced immediately but is placed in a queue and serviced only when the necessary resources are released by running applications. On such systems, predictions of how long queued requests will wait before being serviced are useful for a variety of tasks. For example, predictions of queue wait times can guide a user in selecting an appropriate queue or, in a computational Grid, an appropriate computer [FK99b]. Wait time predictions are also useful in a

Table 16.2. A comparison of our two execution time prediction techniques.

Workload	IBL Mean Error (minutes)	Categorization Mean Error (minutes)	Mean Error of User Estimate (minutes)	Mean Run Time (minutes)
ANL	36.93	34.52	104.35	97.08
CTC	103.75	98.28	222.71	182.49
SDSC95	51.61	43.20	466.49	108.16
SDSC96	52.79	47.47	494.25	166.85

Grid computing environment when trying to submit multiple requests so that the requests all receive resources simultaneously [CFK+98b]. A third use of wait time predictions is to plan other activities in conventional supercomputing environments.

We examine two different techniques for predicting how long applications wait until they receive resources in this environment. Our first technique for predicting wait times in scheduling systems is to predict the execution time for each application in the system (using the categorization technique presented in Section 2) and then to use those predicted execution times to drive a simulation of the scheduling algorithm. This allows us to determine the start time of every job in the scheduling system. The advantage of this technique is that, for certain scheduling algorithms and accurate run time predictions, it can potentially provide very accurate wait time predictions. A disadvantage is that if the scheduling algorithm is such that the start times of applications in the queues depend on applications that have not yet been submitted to the queues, the wait time predictions will not be very accurate. A second disadvantage of this technique is that it requires detailed knowledge of the scheduling algorithm used by the scheduling system.

Our second wait time prediction technique predicts the wait time of an application by using the wait times of applications in the past that were in a similar scheduler state. For example, if an application is in a queue with four applications ahead of it and three behind it, how long did applications in this same state wait in the past? This approach uses the same mechanisms as our approach to predicting application execution times with different characteristics used to describe the conditions we are predicting.

3.1 Scheduling Algorithms

We use three basic scheduling algorithms in this work: first-come first-served (FCFS), least work first (LWF), and conservative backfill [Lif96, FW98] with FCFS queue ordering. In the FCFS algorithm, applications are given resources in the order in which they arrive. The application at the head of the

Table 16.3. Wait time prediction performance using actual and maximum run times.

Workload	Scheduling Algorithm	Wait Time Prediction Error: Actual Run Times (minutes)	Wait Time Prediction Error: Maximum Run Times (minutes)	Mean Wait Time (minutes)
ANL	FCFS	0.00	996.67	535.84
ANL	LWF	37.14	97.12	86.71
ANL	Backfill	5.84	429.05	177.29
CTC	FCFS	0.00	125.36	97.94
CTC	LWF	4.05	9.86	10.49
CTC	Backfill	2.62	51.16	26.93
SDSC95	FCFS	0.00	162.72	55.16
SDSC95	LWF	5.83	28.56	14.95
SDSC95	Backfill	1.12	93.81	28.17
SDSC96	FCFS	0.00	47.83	16.61
SDSC96	LWF	3.32	14.19	7.88
SDSC96	Backfill	0.30	39.66	11.33

queue runs whenever enough nodes become free. The LWF algorithm also tries to execute applications in order, but the applications are ordered in increasing order using estimates of the amount of work (number of nodes multiplied by estimated wall clock execution time) the application will perform.

The conservative backfill algorithm is a variant of the FCFS algorithm. The difference is that the backfill algorithm allows an application to run before it would in FCFS order if it will not delay the execution of applications ahead of it in the queue (those that arrived before it). When the backfill algorithm tries to schedule applications, it examines every application in the queue, in order of arrival time. If an application can run (there are enough free nodes and running the application will not delay the starting times of applications ahead of it in the queue), it is started. If an application cannot run, nodes are reserved for it at the earliest possible time. This reservation is only a placeholder to make sure that applications behind it in the queue do not delay it; the application may actually start before this time.

3.2 Predicting Queue Wait Times: Technique 1

Our first wait time prediction technique simulates the actions performed by a scheduling system using predictions of the execution times of the running and waiting applications. We simulate the FCFS, LWF, and backfill scheduling algorithms and predict the wait time of each application when it is submitted to the scheduler.

Table 16.3 shows the wait time prediction performance when actual or maximum run times are used during prediction. The actual run times are the exact

running times of the applications, which are not known ahead of time in practice. The maximum run time is the amount of time a job requests the nodes for and is when an application should be terminated if it hasn't already completed. This data provides upper and lower bounds on wait time prediction accuracy and can be used to evaluate our prediction approach. When using actual run times, there is no error for the FCFS algorithm because later arriving jobs do not affect the start times of the jobs that are currently in the queue. For the LWF and backfill scheduling algorithms, wait time prediction error does occur because jobs that have not been queued can affect when the jobs currently in the queue can run. This effect is larger for the LWF results where later-arriving jobs that wish to perform smaller amounts of work move to the head of the queue. When predicting wait times using actual run times, the wait time prediction error for the LWF algorithm is between is between 34 and 43 percent. There is a very high built-in error when predicting queue wait times of the LWF algorithm with this technique because there is a higher probability that applications that have not yet been submitted will affect when already submitted applications will start. There is also a small error (3 - 4%) when predicting the wait times for the backfill algorithm.

The table also shows that the wait time prediction error of the LWF algorithm when using actual run times as run time predictors is 59 to 80 percent better than the wait time prediction error when using maximum run times as the run time predictor. For the backfill algorithm, using maximum run times results in between 96 and 99 percent worse performance than using actual run times. Maximum run times are provided in the ANL and CTC workload and are implied in the SDSC workloads because each of the queues in the two SDSC workloads have maximum limits on resource usage.

The third column of Table 16.4 shows that our run time prediction technique results in run time prediction errors that are from 33 to 86 percent of mean application run times and the fourth column shows that the wait time prediction errors that are from 30 to 59 percent of mean wait times. The results also show that using our run time predictor result in mean wait time prediction errors that are 58 percent worse than when using actual run times for the backfill and LWF algorithms but 74 percent better than when using maximum run times.

3.3 Predicting Queue Wait Times: Technique 2

Our second wait time prediction technique uses historical information on scheduler state to predict how long applications will wait until they receive resources. This is an instance of the same categorization prediction approach that we use to predict application run times in Section 2. We selected scheduler state characteristics that describe the parallel computer being scheduled, the application whose wait time is being predicted, the time the prediction is being

Table 16.4. Wait time prediction performance of our two techniques.

Workload	Scheduling Algorithm	Technique 1 Run Time Prediction Error (minutes)	Technique 1 Wait Time Prediction Error (minutes)	Technique 2 Wait Time Prediction Error (minutes)
ANL	FCFS	38.26	161.49	260.36
ANL	LWF	54.11	44.75	76.78
ANL	Backfill	46.16	75.55	130.35
CTC	FCFS	125.69	30.84	76.18
CTC	LWF	145.28	5.74	9.80
CTC	Backfill	145.54	11.37	22.95
SDSC95	FCFS	53.14	20.34	39.79
SDSC95	LWF	58.98	8.72	13.67
SDSC95	Backfill	57.87	12.49	25.50
SDSC96	FCFS	55.92	9.74	10.55
SDSC96	LWF	54.27	4.66	6.83
SDSC96	Backfill	54.82	5.03	9.26

made, the applications that are waiting in the queue ahead of the application being predicted, and the applications that are running to use when making predictions. The characteristics of scheduler state are continuous parameters, similar to the number of nodes specified by an application. Therefore, a range size is used for all of these characteristics.

The fifth column of Table 16.4 shows the performance of this wait time prediction technique. The data shows that the wait time prediction error is 42 percent worse on average than our first wait time prediction technique. One trend to notice is that the predictions for the FCFS scheduling algorithm are the most accurate for all of the workloads, the predictions for the backfill algorithm are all the second most accurate, and the predictions for the LWF algorithm are the least accurate. This is the same pattern when the first wait time prediction technique is used with actual run times. This indicates that our second technique is also affected by not knowing what applications will be submitted in the near future.

4. SCHEDULING USING RUN TIME PREDICTIONS

Many scheduling algorithms use predictions of application execution times when making scheduling decisions [Lif96, FW98, Fei95]. Our goal in this section is to improve the performance of the LWF and backfill scheduling algorithms that use run time predictions to make scheduling decisions. We measure the performance of a scheduling algorithm using *utilization*, the average

Table 16.5. Scheduling performance using actual and maximum run times.

Workload	Scheduling Algorithm	Actual Run Times		Maximum Run Times	
		Utilization (percent)	Mean Wait Time (minutes)	Utilization (percent)	Mean Wait Time (minutes)
ANL	LWF	70.34	61.20	70.70	83.81
ANL	Backfill	71.04	142.45	71.04	177.14
CTC	LWF	55.18	11.15	55.18	36.95
CTC	Backfill	55.18	27.11	55.18	123.91
SDSC95	LWF	41.14	14.48	41.14	14.95
SDSC95	Backfill	41.14	21.98	41.14	28.20
SDSC96	LWF	46.79	6.80	46.79	7.88
SDSC96	Backfill	46.79	10.42	46.79	11.34

percent of the machine that is used by applications, and *mean wait time*, the average amount of time that applications wait before receiving resources.

Table 16.5 shows the performance of the scheduling algorithms when the actual run times are used as run time predictors and when maximum run times are used as run time predictors. These numbers give us upper and lower bounds on the scheduling performance we can expect. The data shows that while maximum run times are not very accurate predictors, this has very little effect on the utilization of the simulated parallel computers. Predicting run times with actual run times when scheduling results in an average of 30 percent lower mean wait times.

To evaluate how well our run time predictions can improve scheduling performance, the first thing we need to determine is what template sets to use to predict application run times. We initially tried to minimize the run time prediction error for workloads generated by running the scheduling algorithms using maximum run times as predictors and recording all predictions that were made. We were not satisfied with the scheduling performance obtained using the parameters obtained by searching over these workloads. So, instead of attempting to minimize run time prediction error, we perform scheduling simulations using different run time prediction parameters and attempt to directly minimize wait times. We do not attempt to maximize utilization because utilization only changes very slightly when different template sets are used or even when a different scheduling algorithm is used.

Table 16.6 shows the performance of the results of these searches. As expected, our run time prediction has minimal impact on utilization. Using our run time predictions does decrease mean wait time by an average of 25 percent over using maximum wait times. These mean wait times are 5 percent more than the wait times achieved when using actual run times.

Table 16.6. Scheduling performance using our run time prediction technique.

Workload	Scheduling Algorithm	Mean Run Time Prediction Error (minutes)	Scheduling Utilization (percent)	Scheduling Mean Wait Time (minutes)
ANL	LWF	72.88	70.75	69.31
ANL	Backfill	116.52	71.04	174.14
CTC	LWF	182.96	55.18	10.58
CTC	Backfill	61.73	55.18	28.39
SDSC95	LWF	142.01	41.14	14.82
SDSC95	Backfill	38.37	41.14	22.21
SDSC96	LWF	112.13	46.79	7.43
SDSC96	Backfill	47.06	46.79	10.56

5. SCHEDULING WITH ADVANCE RESERVATIONS

Some applications have very large resource requirements and would like to use resources from multiple parallel computers to execute. In this section, we describe one solution to this *co-allocation* problem: Advance reservation of resources. Reservations allow a user to request resources from multiple scheduling systems at a specific time and thus gain simultaneous access to sufficient resources for their application.

We investigate several different ways to add support for reservations into scheduling systems and evaluate their performance. We evaluate scheduling performance using utilization and mean wait time, as in the previous section, and also *mean offset from requested reservation time*; the average difference between when the users initially want to reserve resources for each application and when they actually obtain reservations. The mean offset from requested reservation time measures how well the scheduler performs at satisfying reservation requests.

In this section, we use these metrics to evaluate a variety of techniques for combining scheduling from queues with reservation. There are several assumptions and choices to be made when doing this. The first is whether applications are restartable. Most scheduling systems currently assume that applications are not restartable (a notable exception is the Condor system [LL90] and Chapter 9). We evaluate scheduling techniques when applications both can and cannot be restarted. We assume that when an application is terminated, intermediate results are not saved and applications must restart execution from the beginning. We also assume that a running application that was reserved cannot be terminated to start another application. Further, we assume that once the scheduler agrees to a reservation time, the application will start at that time.

If we assume that applications are not restartable and the scheduler must fulfill it once it is made, then we must use maximum run times when predicting application execution times to ensure that nodes are available. The resulting scheduling algorithms essentially perform backfilling. This approach is discussed in Section 5.2

If applications are restartable, there are more options for the scheduling algorithm and this allows us to improve scheduling performance. First, the scheduler can use run time predictions other than maximum run times. Second, there are many different ways to select which running applications from the queue to terminate to start a reserved application. Details of these options and their performance are presented in Section 5.3

Due to space limitations, we only summarize the performance data we collected for the techniques presented in this section. Please see [Smi99] for a more comprehensive presentation of performance data.

5.1 Reservation Model

In our model, a reservation request consists of the number of nodes desired, the maximum amount of time the nodes will be used, the desired start time, and the application to run on those resources. We assume that the following procedure occurs when a user wishes to submit a reservation request:

1 The user requests that an application run at time T_r on N nodes for at most M amount of time.

2 The scheduler makes the reservation at time T_r if it can. In this case, the reservation time, T, equals the requested reservation time, T_r.

3 If the scheduler cannot make the reservation at time T_r, it replies with a list of times it could make the reservation and the user picks the available time T which is closest in time to T_r.

The last part of the model is what occurs when an application is terminated. First, only applications that came from a queue can be terminated. Second, when an application is terminated, it is placed back in the queue from which it came in its correct position.

5.2 Nonrestartable Applications

In this section, we assume that applications cannot be terminated and restarted at a later time and that once a scheduler agrees to a reservation, it must be fulfilled. A scheduler with these assumptions must not start an application from a queue unless it is sure that starting that application will not cause a reservation to be unfulfilled. Further, the scheduler must make sure that reserved applications do not execute longer than expected and prevent other re-

served applications from starting. This means that only maximum run times can be used when making scheduling decisions.

A scheduler decides when an application from a queue can be started using an approach similar to the backfill algorithm: The scheduler creates a timeline of when it believes the nodes of the system will be used in the future. First, the scheduler adds the currently running applications and the reserved applications to the timeline using their maximum run times. Then, the scheduler attempts to start applications from the queue using the timeline and the number of nodes and maximum run time requested by the application to make sure that there are no conflicts for node use. If backfilling is not being performed, the timeline is still used when starting an application from the head of the queue to make sure that the application does not use any nodes that will be needed by reservations.

To make a reservation, the scheduler first performs a scheduling simulation of applications currently in the system and produces a timeline of when nodes will be used in the future. This timeline is then used to determine when a reservation for an application can be made.

One parameter that is used when reserving resources is the relative priorities of queued and reserved applications. For example, if queued applications have higher priority, then an incoming reservation cannot delay any of the applications in the queues from starting. If reserved applications have higher priority, then an incoming reservation can delay any of the applications in the queue. The parameter we use is the percentage of queued applications can be delayed by a reservation request and this percentage of applications in the queue is simulated when producing the timeline that defines when reservations can be made.

5.2.1 Effect of Reservations on Scheduling

We begin by evaluating the impact on the mean wait times of queued applications when reservations are added to our workloads. We assume the best case for queued applications: When reservations arrive, they cannot be scheduled so that they delay any currently queued applications.

We add reservations to our existing workloads by randomly converting either ten or twenty percent of the applications to be reservations with requested reservation times randomly selected between zero and two hours in the future. We find that adding reservations increases the wait times of queued applications in almost all cases. For all of the workloads, queue wait times increase an average of 13 percent when 10 percent of the applications are reservations and 62 percent when 20 percent of the applications are reservations. Our data also shows that if we perform backfilling, the mean wait times increase by only 9 percent when 10 percent of the applications are reservations and 37 percent when 20 percent of the applications are reservations. This is a little over half of the increase in mean wait time when backfilling is not used. Further, there is

a slightly larger increase in queue wait times for the LWF queue ordering than for the FCFS queue ordering.

5.2.2 Offset from Requested Reservations

Next, we examine the difference between the requested reservation times of the applications in our workload and the times they receive their reservations. We again assume that reservations cannot be made at a time that would delay the startup of any applications in the queue at the time the reservation is made.

The performance is what is expected in general: The offset is larger when there are more reservations. For 10 percent reservations, the mean difference from requested reservation time is 211 minutes. For 20 percent reservations, the mean difference is 278 minutes.

Our data also shows that the difference between requested reservation times and actual reservation times is 49 percent larger when FCFS queue ordering is used. The reason for this may be that LWF queue ordering will execute the applications currently in the queue faster than FCFS. Therefore, reservations can be scheduled at earlier times.

We also observe that if backfilling is used, the mean difference from requested reservation times increases by 32 percent over when backfilling is not used. This is at odds with the previous observation that LWF queue ordering results in smaller offsets from requested reservation times. Backfilling also executes the applications in the queue faster than when there is no backfilling. Therefore, you would expect a smaller offsets from requested reservation times. An explanation for this behavior could be that backfilling is packing applications from the queue tightly onto the nodes and is not leaving many gaps free to satisfy reservations before the majority of the applications in the queue have started.

5.2.3 Effect of Application Priority

Next, we examine the effects on mean wait time and the mean difference between reservation time and requested reservation time when queued applications are not given priority over all reserved applications. We accomplish this by giving zero, fifty, or one hundred percent of queued applications priority over a reserved application when a reservation request is being made.

As expected, if more queued applications can be delayed when a reservation request arrives, then the wait times are generally longer and the offsets are smaller. On average, for the ANL workload, decreasing the percent of queued applications with priority from 100 to 50 percent increases mean wait time by 7 percent and decreases mean offset from requested reservation times by 39 percent. Decreasing the percent of queued application with priority from 100 to 0 percent increases mean wait time by 22 percent and decreases mean offset

by 89 percent. These results for the change in the offset from requested reservation time are representative of the results from the other three workloads: as fewer queued applications have priority, the reservations are closer to their requested reservations.

5.3 Restartable Applications

If we assume that applications can be terminated and restarted, then we can improve scheduling performance by using run time predictions other than maximum run times when making scheduling decisions. We use our categorization run time prediction technique that we described in Section 2.

The main choice to be made in this approach is how to select which running applications to terminate when CPUs are needed to satisfy a reservation. There are many possible ways to select which running applications that came from a queue should be terminated to allow a reservation to be satisfied. We choose a rather simple technique where the scheduler orders running applications from queues in a list based on termination cost and moves down the list stopping applications until enough CPUs are available. Termination cost is calculated using how much work (number of CPUs multiplied by wall clock run time) the application has performed and how much work the application is predicted to still perform. Our data shows that while the appropriate weights for work performed and work to do vary from workload to workload, in general, the amount of work performed is the more important factor.

We performed scheduling simulations with restartable applications using the ANL workload. When we compare this performance to the scheduling performance when applications are not restartable, we find that the mean wait time decreases by 7 percent and the mean difference from requested reservation time decreases by 55 percent. There is no significant effect on utilization. This shows that there is a performance benefit if we assume that applications are restartable, particularly in the mean difference from requested reservation time.

6. SUMMARY

The availability of computational Grids and the variety of resources available through computational Grids introduce several problems that we seek to address through predictions. The first problem is selecting where to run an application in a Grid that we address by providing predictions of how long applications will wait in scheduling queues before they begin to execute. The second problem is gaining simultaneous access to multiple resources so that a distributed application can be executed which we address by evaluating techniques for adding support for advance reservations into scheduling systems.

Our approaches to both of these problems are based on predictions of the execution time of applications. We propose and evaluate two techniques for

predicting these execution times. Our first technique categorizes applications that have executed in the past and forms a prediction for an application using categories of similar applications. Our second technique uses historical information and an instance-based learning approach. We find that our categorization approach is currently the most accurate and is more accurate than the estimates provided by users or the techniques presented by two other researchers.

We address the problem of selecting where to run an application in a computational Grid by proposing two approaches to predicting scheduling queue wait times. The first technique uses run time predictions and performs scheduling simulations. The second technique characterizes the state of a scheduler and the application whose wait time is being predicted and uses historical information of wait times in these similar states to produce a wait time prediction. We find that our first technique has a lower prediction error of between 30 and 59 percent of the mean wait times.

We address our second major problem of gaining simultaneous access to distributed resources by describing several ways to modify local scheduling systems to provide advance reservations. We find that if we cannot restart applications, we are forced to use maximum run times as predictions when scheduling. If we can restart applications, then we can use our run time predictions when scheduling. We find that supporting advance reservations does increase the mean wait times of applications in the scheduling queues but this increase is smaller if we are able to restart applications. As a side effect of this work, we find that even when there are no reservations, we can improve the performance of scheduling algorithms by using more accurate run time predictions.

Acknowledgments

We wish to thank Ian Foster and Valerie Taylor who investigated many of the problems discussed here with the author. This work has been supported by Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research of the U.S. Department of Energy, the NASA Information Technology program and the NASA Computing, Information and Communications Technology program.

Chapter 17

THE CLASSADS LANGUAGE

Rajesh Raman, Marvin Solomon, Miron Livny, and Alain Roy
Department of Computer Science, University of Wisconsin-Madison

Abstract The *Classified Advertisements* (ClassAds) language facilitates the representation and participation of heterogeneous resources and customers in the resource discovery and scheduling frameworks of highly dynamic distributed environments. Although developed in the context of the Condor system, the ClassAds language is an independent technology that has many applications, especially in systems that exhibit the uncertainty and dynamism inherent in large distributed systems. In this chapter, we present a detailed description of the structure and semantics of the ClassAds language.

1. INTRODUCTION AND MOTIVATION

This chapter provides a detailed description of the ClassAds (*Classified Advertisements*) language. The ClassAds language was designed in context of the Condor distributed high throughput computing system ([LLM88], [LL90]), but is an independent technology that facilitates the construction of general-purpose resource discovery and scheduling frameworks in dynamic and heterogeneous environments.

Resource discovery and scheduling are two key functionalities of distributed environments, but are difficult to provide due to the uncertainty and dynamism inherent in such large distributed complexes. Examples of difficulties include:

1 *Distributed Ownership.* Resources are not owned by a single principal, and therefore may span several administrative domains that each impose distinct and idiosyncratic allocation policies.

2 *Heterogeneity.* Vastly different kinds of computational resources (workstations, storage servers, application licenses) complicate resource representation and allocation semantics.

3 *Resource Failure.* Resources may fail unpredictably and disappear from the environment, invalidating resource rosters and allocation schedules.

4 *Evolution.* Evolving hardware and software configurations and capabilities invalidate the implicit system model used by allocators to determine allocation schedules.

We have previously argued that a Matchmaking paradigm can be used to opportunistically schedule such non-deterministic environments in a flexible, natural and robust manner ([RLS98]). In the Matchmaking scheme, providers and customers of services advertise their capabilities as well as constraints and preferences on the entities they would like to be matched with. A Matchmaker pairs compatible advertisements and notifies the matched entities, who then activate a separate claiming protocol to mutually confirm the match and complete the allocation. All policy considerations are contained within the advertisements themselves: the Matchmaker merely provides the mechanism to interpret the specified policies in context. Advertisements are short-lived and periodically refreshed, resulting in matches made using the up-to-date information of entities that currently participate in the environment; the claiming protocol is used to resolve the few remaining inconsistencies.

The goal of this chapter is to provide a complete description of the syntax and semantics of the ClassAds language. Although we include some ClassAd examples from the Matchmaking context, a full description of the details and capabilities of the Matchmaking approach is beyond the scope of this chapter. Interested readers are referred to other documents that fully address this topic ([RLS98], [Ram00]).

2. OVERVIEW

Stated in database terms, a ClassAd encapsulates data, schema (metadata) and queries (predicates) in a simple, self-defining structure. In effect, a ClassAd is an extension of a property list, where the properties may be arbitrary expressions (Figure 17.1).

```
[
  room       = "kitchen";
  length     = 17;
  breadth    = 8.54;
  floor_area = length * breadth;
  tile_size  = 1.5;
  num_tiles  = ceiling(length/tile_size) *
               ceiling(breadth/tile_size);
  scrap      = (num_tiles * tile_size * tile_size) +
               - floor_area
]
```

Figure 17.1. An example ClassAd expression.

The ClassAd expression (also called a *record expression*) consists of a set of *attribute definitions*, each of which is a binding between an *attribute name* and an *expression*. As seen from the example, the expressions may consist of simple constants or arbitrary collections of constants and variables combined with arithmetic and logic operators and built-in functions. The language also has a list constructor, and operators to subscript from lists and select from records. Lists and records may be arbitrarily nested.

The ClassAds language is functional (expression-oriented) and free of side-effects: Expression evaluation has no other effect than generating the required value. Expressions may be simplified into forms that are equivalent in context. For example, the expressions bound to *num_tiles* and *scrap* in the previous example may be replaced by `72` and `16.82` respectively.

Expressions that may not be simplified further are said to be in *normal form*, and are called *constants*. The language has been carefully designed so that evaluation is efficient. In particular, it is possible for an implementation to evaluate an expression in time proportional to the expression size.

ClassAd expressions are strongly but dynamically typed. Types supported by the language include integer and floating point numbers, Boolean values (true and false), character strings, timestamps, time intervals, and special *error* and *undefined* values. The error value is generated when invalid operations are attempted, such as `"foo"3.14`, `17/0`, or `{"foo", "bar"}`. The undefined value is generated when attribute references (i.e., attribute names mentioned in expressions) cannot be successfully resolved due to missing or circular attribute definitions. Attribute resolution during evaluation is *block structured*: An attribute reference is resolved by searching all record expressions containing the reference, from innermost outward, for a case-insensitive matching definition. For example, the expression

```
[ a = 1; b = c;
  d = [ f = g; i = a; j = c; k = l; ];
  L = d.k; C = 3; ]
```

evaluates to

```
[ a = 1; b = 3;
  d = [ f = undefined; i = 1; j = 3; k = undefined; ];
  L = undefined; C = 3;
]
```

ClassAds have several representations. The canonical (internal) representation is an *abstract syntax tree*, consisting of a tree of operators and operands. Concrete syntaxes are external representations, consisting of sequences of characters. The above examples were rendered in the *native syntax*, although other concrete syntaxes, including an XML syntax also exist. New concrete syntaxes may be defined in the future.

3. MATCHMAKING

The ClassAds language was designed to address the problem of heterogeneous resource representation, discovery and matching in distributed environments. The ClassAds language plays a central role in the Condor distributed high throughput computing system, in which the characteristics and policies of resources (such as workstations and storage servers) and customers (serial and parallel jobs) are represented using the ClassAds language. Figures 17.2 and 17.3 illustrate example advertisements of workstations and jobs respectively.

```
[
  Type = "Machine";
  Activity = "Idle";
  KeybrdIdle = RelTime("00:23:12");
  Disk = 323.4M;
  Memory = 64M;
  State = "Unclaimed";
  LoadAvg = 0.045;
  Mips = 104;
  Arch = "INTEL";
  OpSys = "SOLARIS7";
  KFlops = 21893;
  Name = "ws10.cs.wisc.edu";
  Rank = other.Owner == "solomon" ? 1 : 0;
  Requirements = other.Type == "Job"
                 && LoadAvg < 0.3
                 && KeybrdIdle > RelTime("00:15")
                 && Memory - other.ImageSize >= 15M
]
```

Figure 17.2. An example workstation advertisement.

In order for an advertisement to be meaningfully included in the matchmaking process, it must define top-level attribute definitions named `Requirements` and `Rank`, which denote the constraints and preferences of the advertising entity. The `Requirements` expression must specifically evaluate to true (not undefined or any other value) for the constraints to be considered satisfied. The `Rank` expression may be considered to be a goodness metric to measure the desirability of a candidate match, with higher values denoting more desirable matches.

When determining the compatibility of two advertisements, the Matchmaker places both advertisements in a single evaluation environment in which each advertisement may use the attribute name `other` to refer to to the candidate match ClassAd. Thus, a job may refer to the memory of a machine using

the expression `other.memory`, while the machine may simultaneously refer to the job's owner using the expression `other.owner`. Interestingly, the evaluation environment used for ClassAds is itself a ClassAd expression.

```
[
  Type = "Job";
  QueueDate = AbsTime("2002-10-03T10:53:31-6:00");
  CompletionDate = undefined;
  Owner = "solomon";
  Cmd = "run_sim";
  WantRemoteSyscalls = true
  WantCheckpoint = true
  Iwd = "/usr/raman/sim2";
  Args = "-Q 17 3200 10";
  Memory = 31M;
  Rank = other.Kflops/1e3 + other.Memory/32;
  Requirements = other.Type == "Machine"
                 && other.Arch == "INTEL"
                 && other.OpSys == "SOLARIS7"
]
```

Figure 17.3. An example job advertisement.

4. STRUCTURE AND SYNTAX

The ClassAds language defines a single abstract syntax, as well as several concrete syntaxes. In this section, we describe the abstract syntax, as well as the native concrete syntax. An XML concrete syntax has also been developed, and future concrete syntaxes may be defined.

4.1 Abstract Syntax

The abstract syntax serves as the internal canonical representation of ClassAd expressions. The abstract syntax tree is a rooted, ordered tree consisting of a collection of nodes and edges. Each node is either an *internal node* that includes a sequence of *child nodes*, or a child-less *leaf* node. Every node (except the unique *root* node) also defines the unique *parent node* of which it is a child. The leaves of an expression consist only of *scalar constants* and *references*.

4.1.1 Constants

Constants may either be *scalar* or *aggregate*. A scalar constant may be *integer*, *real*, *string*, *boolean*, *absolute time*, *relative time*, *error* or *undefined*. Aggregate constants may be either *records* or *lists*. A record constant is one in

which every attribute name is bound to a constant, and a list constant is a finite sequence of constants.

Each constant has a value that belongs to the domain defined by the type of the constant, as shown in Table 17.1. There is a unique value of type Error; error values may be annotated with an ASCII string for human consumption, but all values of type Error are equivalent. Similar remarks apply to Undefined.

Table 17.1. Scalar constants.

Type	Value or annotation
Integer	32-bit signed, two's complement integer
Real	64-bit IEEE 754 double-precision floating point number
String	Zero or more non-null ASCII characters
Boolean	The value true or false
AbsTime	Time offset to the base UNIX epoch time
RelTime	Interval between two absolute times
Error	Zero or more non-null ASCII characters for human consumption
Undefined	Zero or more non-null ASCII characters for human consumption

4.1.2 Attribute Names

A leaf node may be a reference, which contains an attribute name—a sequence of one or more non-null ASCII characters. Two attribute names match if the character sequences are of the same length and the corresponding characters are the same except for differences in case. The reserved words `error`, `false`, `is`, `isnt`, `true`, and `undefined` may not be used as attribute names. Reserved words are defined independent of case, so `false`, `FALSE` and `False` all refer to the same reserved word.

4.1.3 Operators

Much of the richness of the ClassAds language derives from the variety of operators that the language offers. Operator nodes always reside as internal nodes in the abstract syntax tree, and have one or more *operands* that are represented by the child nodes of the operator node. Operators are *unary*, *binary*, *ternary* or *varargs* according to whether they require one, two, three or a variable number of operands. Operator *precedence* is used to disambiguate expressions where multiple operator interpretations are possible, with larger values denoting higher precedence. Thus, the expression $5 + 15/3$ is interpreted as $5 + (15/3)$ because the DIV operator has higher precedence than the PLUS operator. Table 17.2 lists the various operators along with their symbols in the native concrete syntax and the corresponding precedence value. The LIST, RECORD and FUNCTION_CALL operators are varargs, the UPLUS,

Table 17.2. Table of operators.

Name (Symbol)	Precedence
UPLUS (+), UMINUS (-), BITCOMPLEMENT (~), NOT (!)	12
TIMES (*), DIV (/), MOD (%)	11
PLUS (+), MINUS (-)	10
LEFT_SHIFT (<<), RIGHT_SHIFT (>>), URIGHT_SHIFT (>>>)	9
LESS (<), GREATER (>), LESS_EQ (<=), GREATER_EQ (>=)	8
EQUAL (==), NOT_EQUAL (!=), SAME (is), DIFFERENT (isnt)	7
BITAND (&)	6
BITXOR (^)	5
BITOR (|)	4
AND (&&)	3
OR (||)	2
COND (? :)	1
SUBSCRIPT ([])	-
SELECT (.)	-
LIST ({ ... })	-
RECORD ([...])	-
FUNCTION_CALL (name(...))	-

UMINUS, BITCOMPLEMENT and NOT operators are unary, and the COND operator is ternary. All other operators are binary.

Almost all operators accept arbitrary expressions for any of their operands. Since ClassAd expressions are dynamically typed, even illegal expressions like `"foo"/34.3` are valid: such expressions merely evaluate to error. The three operators that place restrictions on their operands are the SELECT, RECORD and FUNCTION_CALL operators. The second operand of the SELECT operator, which is called the *selector* must be an attribute name. The first operand of the varargs FUNCTION_CALL operator, the *function name*, must be a non-empty sequence of characters and must be chosen from the fixed set of built-in function names, as listed in Section 5.3.8: the remaining operands, the *actual parameters*, may be arbitrary expressions. The RECORD operator has zero or more operands, each of which is an *attribute definition*. An attribute definition is an ordered pair consisting of an attribute name and an arbitrary expression.

4.2 Native Syntax

The native syntax of the ClassAds language is similar to C, C++ or Java. The character representation of the expression consists of a sequence of tokens that are separated by white space and comments. As in C++ and Java, there are two kinds of comments: line comments are introduced by the sequence // and extend up to the next newline character, whereas non-nestable block comments are introduced by the sequence /* and extend up to the first */ sequence.

4.2.1 Tokens

The tokens that comprise the native syntax include *integer*, *floating-point* and *string* literals, as well as character sequences to define *attribute names* and *operators*, including miscellaneous delimiting characters. The native syntax character sequences that specify operators are listed in Table 17.2. The sequences that define the other token types are described below.

Integers. Integer literals may be specified in decimal, octal or hexadecimal notations. E.g., `17`, `077`, `0xdeadbeef`.

Reals. Real literals may be specified using conventional notations such as `3.1415` or `1.3e5`. In addition, integer or real literals followed by the suffixes B, K, M, G and T result in an a real valued literal whose value is the original value times 1, 1024, 1048576, 1073741824 or 1099511627776 respectively. (As can be guessed, these suffixes stand for byte, kilobyte, megabyte, gigabyte and terabyte respectively.)

Strings. A string literal is specified by enclosing zero or more characters between double quotes. E.g., `"foo"`, `"bar"`. In addition, escaped character sequences such as `"hello,\\n\\tworld"` may also be specified.

Attribute Names. An attribute name may be specified either as an unquoted name or as a quoted name. Unquoted attribute names are simple sequences of alphanumeric characters and underscores in which the first character is not a digit. All attribute names in the example expressions have been unquoted attribute names. Quoted attribute names are delimited by single quotes, and can contain arbitrary characters in the attribute name, including escape sequences. Examples of quotes attribute names include `'a long attr name'` and `'9 times 9\nequals \n\t81'`.

4.2.2 Grammar

The native syntax is defined by the informal context-free grammar illustrated in Figure 17.4. The grammar has been simplified for ease of readability: since Figure 17.2 specifies both operator precedence and concrete syntactic representations of operators, these details have been omitted from the grammar.

4.2.3 Parsing and Unparsing

The process of parsing associates a node in the abstract syntax tree with every activation of the *expression*, *binary_expression*, *prefix_expression* or *suffix_expression* productions. For example, a *binary_expression* of the form $expr_1$

```
expression        ::= binary_expression
                    | binary_expression '?' expression
                      ':' expression
binary_expression ::= binary_expression binary_operator
                      prefix_expression
                    | prefix_expression
prefix_expression ::= suffix_expression
                    | unary_operator suffix_expression
suffix_expression ::= atom
                    | suffix_expression '.' attribute_name
                    | suffix_expression '[' expression ']'
atom              ::= attribute_name
                    | 'error' | 'false'
                    | 'undefined' | 'true'
                    | integer_literal
                    | floating_point_literal
                    | string_literal
                    | list_expression
                    | record_expression
                    | function_call
                    | '(' expression ')'
list_expression   ::= '{' (expression (','
                        expression)* ','?)? '}'
record_expression ::= '[' (attr_def (';' attr_def)* ';'?)?  ']'
attr_def          ::= attribute_name '=' expression
function_call     ::= unquoted_name '(' (expression
                                         (',' expression)*)? ')'
```

Figure 17.4. Grammar of the concrete native syntax.

$+expr_2$ parses to an internal PLUS expression with operands corresponding to the parsed output of $expr_1$ and $expr_2$. Due to the presence of whitespace, comments and optional parentheses, it is possible for multiple distinct strings to parse to the same internal form. To support certain features like loss-less transport and the built-in `string` function, we also define a *canonical unparsing* of an expression which, when parsed, yields an equivalent expression.

The canonical unparsing of an expression is completely parenthesized, and has no comments or whitespace outside string literals or attribute names. For example, the strings `"b * b - 4 * a * c"` and `"(b*b)-(4*(a*c))"` both parse to the same internal form: the second string is the canonical unparsing.

The canonical unparsing of an unquoted attribute name is simply the name itself. Otherwise, a quoted name syntax is used where the name is delimited by apostrophes and normal characters are left untouched, but backslashes are doubled, apostrophes and double quotes are escaped and non-printable characters

are converted to escaped sequences. The canonical unparsing of string literals follows the same rules, except that string literals are delimited with double quotes.

The canonical unparsing of a record or list omits the optional trailing delimiter, so the empty record and list unparse to `[]` and `{}` respectively. The true, false, error and undefined values respectively unparse to the literal character sequences `true`, `false`, `error` and `undefined` of the native syntax.

The canonical unparsing of an integer is always performed in decimal notation. In order to ensure loss-less representation of floating point numbers, the canonical unparsing of a floating point number is defined to be the string representation of a call to a built-in function. Specifically, the canonical unparsing is `ieee754("XXXXXXXXXXXXXXXX")`, where the sequence of X's represent the bit pattern (in lower-case hexadecimal) of the 64-bit quantity that specifies the floating point number.

Similarly, the canonical unparsing of time values have the form of the built-in functions `reltime` or `abstime` applied to a string containing the ISO 8601 representation of the date. Examples are `reltime("5:15:34")` and `abstime("2002-03-02T08:21:31")`.

5. EVALUATION AND SEMANTICS

The semantics of expression evaluation are defined assuming bottom-up evaluation of the internal expression tree, along with call-by-value semantics. In other words, each internal node is evaluated by evaluating all its children and then applying the relevant operator to the resulting sequence of values. Other evaluation strategies, such as a "lazy" evaluation scheme that only evaluates subexpressions whose values are required, are permissible provided that the same results are produced.

5.1 Types, Undefined and Error

Every expression evaluates to a constant, which is an expression that contains no free attribute references, and no operators other than LIST and RECORD. Each constant has a *type*, which is one of Integer, Real, String, Boolean, AbsTime, RelTime, Undefined, Error, List or Record. Integer and Real are collectively called numeric types. AbsTime and RelTime are collectively called timestamp types. Each operator imposes constraints on the types of its operands. If these constraints are not met, the result of the evaluation is error. In addition, attribute references evaluate to undefined if the reference could not be successfully resolved either due to absent or circular attribute definitions. For example, all attributes of the expression `[ a = b ; b = a ; c = x ]` evaluate to undefined.

Most operators are strict with respect to undefined and error That is, if any operand is undefined or error, the operator correspondingly evaluates to undefined or error (error, if both are present). The only exceptions are the Boolean operators described in Section 5.3.5, the `is` and `isnt` operators described in Section 5.3.2, and the LIST and RECORD constructors described in Section 5.3.7. Strict evaluation obeys the following ordered sequence of rules.

1 If the operands do not obey the type restrictions imposed by the operator, the result is error. The following sections list all combinations of types accepted by each operator.

2 Otherwise, if any operand of a strict operator is undefined, the result is undefined.

3 Otherwise, the result is computed from the operands as described in the following sections.

5.2 Leaves

A leaf node that is a constant evaluates to itself. A leaf node that is a reference R with attribute name N is evaluated by finding the closest ancestor record node R' containing a definition $N' = expr$ such that N' matches N, ignoring differences in case. If no such ancestor exists, the resulting value is undefined. Otherwise, the value of R is the value of $expr$. If the evaluation of $expr$ directly or indirectly leads to a re-evaluation of R (i.e., a circular definition), the value of R is undefined.

5.3 Internal Nodes

5.3.1 Boolean Operators

The binary Boolean operators && and ||, and the ternary operator _?_:_ are non-strict: they are evaluated "left-to-right" so that

true || $x \Rightarrow$ true
false && $x \Rightarrow$ false
true ? val : $x \Rightarrow val$
false ? x : $val \Rightarrow val$

even if x evaluates to error. The Boolean operators treat Boolean values as a three-element lattice with false $<$ undefined $<$ true. With respect to this lattice, AND returns the minimum of its operands, OR returns the maximum, and NOT interchanges true and false.

5.3.2 The SAME and DIFFERENT Operators

The SAME and DIFFERENT operators have non-strict semantics, and check if the two comparands are identical or different, where different is defined as the negation of the identical relation.

Specifically, val_1 SAME val_2 evaluates to true if and only if both val_1 and val_2 are identical, and false otherwise. The *identical-to* relation is defined as follows:

- Values of different types are never identical. So, 3 and 3.0 are not identical.
- Two values both of type Integer, Real, AbsTime and RelTime are identical if and only if the values are numerically equal.
- Two values of type Boolean are identical if and only if they are both true or both false.
- All undefined values are identical, as are all error values.
- Two values of type String are identical if and only if they are identical character by character, with case being significant. So the expressions, `("One" == "one")` and `("One" isnt "one")` both evaluate to true.
- Two values of type List or Record are identical if and only if they are both created by the same expression. So, in the expression `[a = { 1 }; b = { 1 } ; c = a is b ; d = a is a]` the values of `c` and `d` are false and true respectively.

Note that the SAME and DIFFERENT operators always evaluate to true or false, never undefined or error.

5.3.3 Comparison Operators

The comparison operators EQUAL, NOT_EQUAL, LESS, LESS_EQ, GREATER and GREATER_EQ evaluate to true only if the two comparands are both numeric (i.e., Integer or Real), AbsTime, RelTime or String. The comparison evaluates to error if the values being compared are not of the above combinations. For numeric and timestamp values, the comparison proceeds as if the underlying numbers are being compared. String values are compared lexicographically, ignoring case.

5.3.4 Arithmetic Operators

The arithmetic operators operate in the natural way when operating on numeric types — they are defined to deliver the same results as the corresponding

operators in Java. In particular if both operands of a binary arithmetic operator are Integers, the result is an Integer; if one operand is an Integer and the other is a Real, the Integer value is promoted to a Real and the result is computed using double precision floating point arithmetic.

The DIV operator's action on integral argument results in a integral quotient that has been truncated towards zero. Likewise, the MOD operator generally returns a remainder that has the same sign as the dividend. The action of both these operators on a zero divisor yields the error value.

5.3.5 Bitwise Boolean and Shift Operators

The ClassAds language supports the bitwise operators BITAND, BITOR, BITXOR, BITNOT and the shift operators LEFT_SHIFT, RIGHT_SHIFT and URIGHT_SHIFT. Their semantics is exactly like those of their Java counterparts. All apply to Integer operands and all but the shifts apply to Boolean operands.

5.3.6 Select and Subscript

The SELECT operator requires two operands, the *base* and the *selector*, which must be an attribute name. The expression val SELECT $attr$ (which is written in the native syntax as $val.attr$) evaluates to undefined if val is undefined and error if val is not a Record or a List. If val is the list LIST(e_0, ..., e_n), the value of val SELECT $attr$ is LIST(e_0 SELECT $attr$, ... , e_n SELECT $attr$). If val is a RECORD, the value of val SELECT $attr$ is identical to making the reference $attr$ in the Record value val. In other words, the Record values enclosing val are searched successively from innermost to outermost for a definition of the form $attr = value$, where the attribute names are matched ignoring case. If a match is found, the resulting value is $value$. Otherwise, the resulting value is undefined.

The SUBSCRIPT operator also requires two operands, the *base* and a *subscript*. The types of these operands must be either List and Integer, or Record and String; any other combination of types results in the generation of undefined or error values as appropriate. If the base is a List with n components and the subscript is an Integer with value k, the result is the k^{th} component of the list, where $0 \leq k < n$. If k is outside the numeric bounds above, the result of the expression is error.

When the base is a Record and the subscript is a String, the expression $base$ SUBSCRIPT $subscript$ behaves exactly like the corresponding SELECT operator. Thus the value of `b` is 1 in both the following expressions:

```
[ a = [ One = 1 ; Two = 2 ]; b = a.One ]
[ a = [ One = 1 ; Two = 2 ]; b = a["one"] ]
```

5.3.7 List and Record Constructors

The LIST and RECORD operators are vararg operators due to their variable arity. The LIST operator takes a sequence of arbitrary values and creates a List value by composing the values in order. Similarly a RECORD operator takes a sequence of $(name, value)$ pairs and creates a Record value with the appropriate entries.

Note that although the call-by-value semantics of these operators defines complete evaluation of all constituent expressions, other more efficient evaluation schemes (such as "lazy evaluation" or call-by-name) may be used if the semantics are not altered. Thus, the expression `{...}[3]` may be efficiently evaluated by evaluating only the third component of the list. Similarly, `[...].aa` may directly evaluate to undefined without evaluating any of the definitions of the record if `aa` is not defined in the record. If a match is found, only the matched expression and the sub-expressions used by the matched definition require evaluation.

5.3.8 Function Calls

The FUNCTION_CALL operator takes a function name and a variable number of actual parameters, which may be arbitrary values. The function name must match the name of one of the built-in functions, but case is not significant. Thus `int(2.3)`, `Int(2.3)` and `INT(2.3)` all call the same function.

All the functions listed in Figure 17.3 are strict on all arguments with respect to undefined and error, so if undefined or error is supplied for any argument, the function correspondingly returns undefined or error. (Future versions of this specification may introduce non-strict functions). Functions also return error if the number or type of arguments do not match the function's prototype.

6. APPLICATIONS

The primary application of the ClassAds language is the representation and participation of resources and customers in Condor's matchmaking-based resource discovery and allocation framework. As described in Section 3, the language allows resources and customers to represent their characteristics and policies in a simple, yet expressive scheme. In a similar fashion, ClassAds have been used to distribute jobs across widely distributed Grid sites by the European Data Grid's Resource Broker ([GPS+02]). However, the ClassAds language has applicability beyond this role.

The Hawkeye distributed system administration application represents problem symptoms as ClassAds, and leverages the matchmaking process to identify problems in distributed environments ([Haw]). Within the Condor system itself, ClassAds have also been used in self-describing network protocols,

Table 17.3. List of built-in functions.

Function	Description
IsUndefined(V)	True iff V is the undefined value.
IsError(V)	True iff V is the error value.
IsString(V)	True iff V is a string value.
IsList(V)	True iff V is a list value.
IsClassad(V)	True iff V is a classad value.
IsBoolean(V)	True iff V is a boolean value.
IsAbsTime(V)	True iff V is an absolute time value.
IsRelTime(V)	True iff V is a relative time value.
Member(V,L)	True iff scalar value V is a member of the list L
IsMember(V,L)	Like Member, but uses is for comparison instead of ==.
CurrentTime()	Get current time
TimeZoneOffset()	Get time zone offset as a relative time
DayTime()	Get current time as relative time since midnight.
MakeDate(M,D,Y)	Create an absolute time value of midnight for the given day. M can be either numeric or string (e.g., "jan").
MakeAbsTime(N)	Convert numeric value N into an absolute time
MakeRelTime(N)	Convert numeric value N into a relative time
GetYear(AbsTime)	Extract year component from timestamp
GetMonth(AbeTime)	0=jan, ..., 11=dec
GetDayOfYear(AbsTime)	0 ... 365 (for leap year)
GetDayOfMonth(AbsTime)	1 ... 31
GetDayOfWeek(AbsTime)	0 ... 6
GetHours(AbsTime)	0 ... 23
GetMinutes(AbsTime)	0 ... 59
GetSeconds(AbsTime)	0 ... 61 (for leap seconds)
GetDays(RelTime)	Get days component in the interval
GetHours(RelTime)	0 ... 23
GetMinutes(RelTime)	0 ... 59
GetSeconds(RelTime)	0 ... 59
InDays(Time)	Convert time value into number of days
InHours(Time)	Convert time value into number of hours
InMinutes(Time)	Convert time value into number of minutes
InSeconds(Time)	Convert time value into number of seconds
StrCat(V1, ... , Vn)	Concatenates string representations of each value.
ToUpper(String)	Convert to uppercase
ToLower(String)	Convert to lowercase
SubStr(S,offset [,len])	Returns substring of S.
RegExp(P,S)	Checks if S matches pattern P
Int(V)	Converts V to an integer.
Real(V)	Converts V to a real.
String(V)	Converts V to its string representation
Bool(V)	Converts V to a boolean value.
AbsTime(V)	Converts V to an absolute time.
RelTime(V)	Converts V to an relative time.
Floor(N)	Floor of numeric value N
Ceil(N)	Ceiling of numeric value N
Round(N)	Rounded value of numeric value N

semi-structured log records that can be filtered using ClassAd predicates and configuration information for programs.

Other more complex kinds of matchmaking may also be developed using the ClassAds language as the substrate. For example, *Gangmatching* uses the ClassAds language to implement a multilateral extension to Condor's strict machine-job bilateral matchmaking scheme ([RLS03], [Ram00]). The ClassAds language has also been extended slightly to allow matches against sets of resources, where the size of the set is not known in advance ([LYFA02]). Such set matching requires relatively few changes to the ClassAds language, but very different matchmaking algorithms.

7. CONCLUSIONS

This chapter has provided an overview of the ClassAds language. The ClassAds language has been used extensively in the Condor system to support resource representation, discovery and allocation. The language's dynamically typed, semi-structured data model naturally accommodates the heterogeneity and dynamism observed in distributed Grid environments. ClassAd expressions are expressive enough to capture the policy expectations of resources and customers that participate in high throughput and opportunistic Grid computing. At the same time, the simplicity of the language affords efficient and robust implementation. These properties have collectively enabled ClassAds to be used in other domains independent of the Condor system.

Chapter 18

MULTICRITERIA ASPECTS OF GRID RESOURCE MANAGEMENT

Krzysztof Kurowski,[1] Jarek Nabrzyski,[1] Ariel Oleksiak,[1] and Jan Węglarz[1,2]
[1] *Poznań Supercomputing and Networking Center,*
[2] *Institute of Computing Science, Poznań University of Technology*

Abstract Grid resource management systems should take into consideration the application requirements and user preferences on the one hand and virtual organizations' polices on the other hand. In order to satisfy both users and resource owners, many metrics, criteria, and constraints should be introduced to formulate multicriteria strategies for Grid resource management problems.

In this chapter we argue that Grid resource management involves multiple criteria and as such requires multicriteria decision support. We discuss available multicriteria optimization techniques and methods for user preference modeling. The influence of the Grid nature on the resource management techniques used is also emphasized, including issues such as dynamic behavior, uncertainty, and incomplete information. We present three aspects of the resource management process: (i) providing the resource management system with all the necessary information concerning accessible resources, application requirements, and user preferences, (ii) making decisions that map tasks to resources in the best possible way, and (iii) controlling the applications and adapting to changing conditions of the Grid environment.

1. INTRODUCTION

This chapter has two main objectives: to prove a need for a multicriteria approach to Grid resource management, and to present ways to achieve this approach. To accomplish these objectives, we identify participants in a resource management process. We present methods of modeling their preferences and making the final decision concerning the best compromise for mapping tasks to resources. These methods are illustrated by an example. Furthermore, we define additional techniques to cope with the dynamic behavior of tasks and resources as well as uncertain and incomplete information.

The rest of this chapter is organized as follows. Section 2 contains basic notions and definitions related to a multicriteria approach. In Section 3 we justify the multicriteria approach in Grid resource management. The reader can find there an answer to the question why Grid resource management involves multiple criteria. With this end in view, this section aims at brief clarification and comparison of the two main resource management strategies in the Grid: application-level scheduling and job scheduling. We view these strategies as two contradictory approaches from the perspective of performance and high-throughput as well as that of Grid resource management. However, we conclude that it is possible to combine both strategies and take advantages of a multicriteria approach and AI support techniques. Additionally, the main participants of the Grid resource management process are defined in this section.

In Section 4 we show how the multicriteria concept can be used in the Grid resource management process. The steps that have to be performed, as well as appropriate methods, are presented. This section includes an analysis of possible criteria that can be used by various groups of stakeholders of the Grid resource management process , presents methods of obtaining and modeling their preferences, and shows how preference models can be exploited in order to assign tasks to resources in the most appropriate way. In Section 5 ideas and methods presented in the previous section are illustrated by an example. This example depicts how multicriteria Grid resource management may look in the presence of multiple constraints, criteria, and participants of this process. In Section 6 we show how specific features of the Grid influence the Grid resource management. We also present techniques and approaches that can cope with these features. In Section 7 we draw some conclusions and present the directions of our future research.

2. MULTICRITERIA APPROACH

In this section, we present the basis of our multicriteria approach, along with the basic concepts and definitions needed for the rest of the chapter. We start from a formulation of the multicriteria decision problem. Then we present basic definitions and describe different ways of modeling decision maker's preferences.

2.1 Multicriteria Decision Problem

In general, a decision problem occurs if there exist several different ways to achieve a certain goal. In addition, the choice of the best method for the achievement of this goal cannot be trivial. These different ways are *decision actions*, also called *compromise solutions* or *schedules* in this chapter. An evaluation of decision actions is often based on multiple criteria. Generally, the main goal of a multicriteria decision making process is to build a global

model of the decision maker's preferences and exploit them in order to find the solution that is accepted by a decision maker with respect to values on all criteria. In order to accomplish this, preferential information concerning the importance of particular criteria must be obtained from a decision maker. If these preferences are not available, then the only objective information is the dominance relation in the set of decision actions. The basic concepts are briefly presented in the next section.

2.2 Basic Definitions

The concepts of the *non-dominated* and *Pareto-optimal set* are crucial to the choice of the compromise solution. Two spaces must be distinguished: *decision variable space* and *criteria space*. The former contains possible solutions along with values of their attributes. The set of solutions in this space will be denoted by *D* in the chapter. The set D can be easily mapped into its image in criteria space, which creates the set of *points* denoted by *C(D)*. The most important definitions concerning these sets are presented below. The functions $f_i(x)$ represents values of criteria, and their number is denoted by *k*.

Definition 1 (Pareto Dominance)

Point **z** $\in$ *C(D) dominates* **z'** $\in$ *C(D)*, denoted as **z** $\succ$ **z'**, if and only if $\forall j \in 1, ..., k, z_j \geq z'_j \wedge \exists i \in 1, ..., k : z_i > z'_i$ (**z'** is partially less than **z**).

Thus, one point dominates another if it is not worse with respect to all criteria and it is better on at least one of them.

Definition 2 (Pareto Optimality)

Point **z'** $\in$ *C(D)* is said to be *non-dominated* with respect to the set *C(D)* if there is no **z** $\in$ *C(D)* that dominates **z'**. A solution **x** is *Pareto optimal(efficient)* if its image in the criteria space is non-dominated.

Definition 3 (Pareto-optimal Set)

The set P^* of all *Pareto-optimal* solutions is called the *Pareto-optimal set*.

Thus, it is defined as: $P^* = \{\mathbf{x} \in D | \neg \exists \mathbf{x'} \in D$: **z'** $\succ$ **z** $\}$, where z' and $z \in C(D)$

Definition 4 (Pareto Front)

For a given Pareto optimal set P^*, *Pareto front* (PF^*) is defined as

$PF^* = \{z = \{f_1 = z_1, ..., f_k(\mathbf{x}) = z_k\} | \mathbf{x} \in P^*\}$

A Pareto front is also called a *non-dominated set* because it contains all non-dominated points.

2.3 Preference Models

As mentioned in Section 2.15, using the decision maker's preferential information concerning the importance of particular criteria, one can build a *preference model* that defines a preference structure in the set of decision actions. Preference models can aggregate evaluation on criteria into

a) functions (e.g., weighted sum)

b) relations (e.g., binary relation: action *a* is at least as good as action *b*)

c) logical statements (e.g., decision rules: "if conditions on criteria, then decision," where "if conditions on criteria" is a conditional part of the rule and "then decision" is a decision part of a rule).

Preference models and the relations between them are investigated, for example, in [SGM02].

3. MOTIVATIONS FOR MULTIPLE CRITERIA

In this section we briefly describe the main motivations behind multicriteria strategies in Grid resource management. Multicriteria approaches focus on a compromise solution (in our case a compromise schedule). In this way, we can increase the level of satisfaction of many stakeholders of the decision making process and try to combine various points of view, rather than provide solutions that are very good from only one specific perspective as is currently common in other Grid resource management approaches. Such specific perspectives result in different, often contradictory, criteria (along with preferences) and make the process of mapping jobs to resources difficult or even impossible. Consequently, all the different perspectives and preferences must be somehow aggregated to please all the participants of the Grid resource management process.

3.1 Various Stakeholders and Their Preferences

Grid scheduling and resource management potentially involves the interaction of many human players (although possibly indirectly). These players can be divided into three classes: (i) *end users* making use of Grid applications and portals, (ii) *resource administrators* and *owners*, and (iii) *virtual organization (VO) administrators* and *VO policy makers* (compare [FK99b]).

One goal of Grid resource management systems is to automate the scheduling and resource management processes in order to minimize stakeholders' participation in the entire process. That is why the final decision is often delegated to such systems. Thus, in this chapter, by the *decision maker* notion we mean a *scheduler*, and all human players listed above are *stakeholders* of a decision making process. This model assumes a single (artificial) decision maker in a VO. In real Grids, however, multiple stakeholders (agents) may of-

ten act according to certain strategies, and the decision may be made by means of negotiations between these agents. Such an approach requires distributed decision making models with multiagent interactions [San99], which will not be discussed in this chapter.

We refer to the assignment of resources to tasks as a *solution* or a *schedule*. Since we consider many contradictory criteria and objective functions, finding the optimal solution is not possible. The solution that is satisfactory from all the stakeholders' points of view and that takes into consideration all criteria is called the *compromise solution* (see definitions in Section 2.1).

The need for formulation of Grid resource management as a multicriteria problem results from the characteristics of the Grid environment itself. Such an environment has to meet requirements of different groups of stakeholders listed at the beginning of this section. Thus, various points of view and policies must be taken into consideration, and results need to address different criteria for the evaluation of schedules. Different stakeholders have different, often contradictory, preferences that must be somehow aggregated to please all stakeholders. For instance, administrators require robust and reliable work of the set resources controlled by them, while end users often want to run their applications on every available resource. Additionally, site administrators often want to maintain a stable load of their machines and thus load-balancing criteria become important. Resource owners want to achieve maximal throughput (to maximize work done in a certain amount of time), while end users expect good performance of their application or, sometimes, good throughput of their set of tasks. Finally, VO administrators and policy makers are interested in maximizing the whole performance of the VOs in the way that satisfy both end users and administrators.

Different stakeholders are not the only reason for multiple criteria. Users may evaluate schedules simultaneously with respect to multiple criteria. For example, one set of end users may want their applications to complete as soon as possible, whereas another one try to pay the minimum cost for the resources to be used. Furthermore, the stakeholders may have different preferences even inside one group. For example, a user may consider time of execution more important than the cost (which does not mean that the cost is ignored).

3.2 Job Scheduling

Job scheduling involves using one central scheduler that is responsible for assigning the Grid resources to applications across multiple administrative domains. As presented in Figure 18.1, there are three main levels: a set of applications, a central scheduler, and the Grid resources. In principle, this approach is in common use and can be found today in many commercial and public-domain job-scheduling systems, such as Condor [CON] (also Chap-

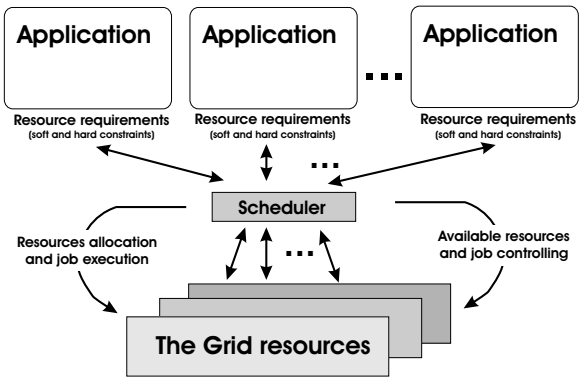


Figure 18.1. Major blocks and flow of information processes in the job scheduling approach.

ter 17, PBS [PBS] (also Chapter 13), Maui [Mau] (also Chapter 11), and LSF [LSF] (also Chapter 12). A detailed analysis and their comparative study are presented in [KTN+03].

The first block in Figure 18.1 describes applications and their requirements. The second block represents a scheduler and its role in processing resource allocation, control, and job execution. Note that the dynamic behavior seen in Grid environments is most often the result of competing jobs executing on the resources. By having a control over all the applications, as happens in the job-scheduling approach, a central scheduler is able to focus on factors that cause variable, dynamic behavior, whereas application-level schedulers have to cope with these effects.

3.3 Application-Level Scheduling

With application-level scheduling, applications make scheduling decisions themselves, adapting to the availability of resources by using additional mechanisms and optimizing their own performance. That is, applications try to control their behavior on the Grid resources independently. The general schema of this approach is presented in Figure 18.2. Note that all resource requirements are integrated within applications.

Note that such an assumption causes many ambiguities in terms of the classic scheduling definitions. When we consider classic scheduling problems, the main goal is to effectively assign all the available task requirements to available resources so that they are satisfied in terms of particular objectives. In general, this definition fits more the job scheduling approach, where job scheduling systems take care of all the available applications and map them to resources suitably. In both cases,

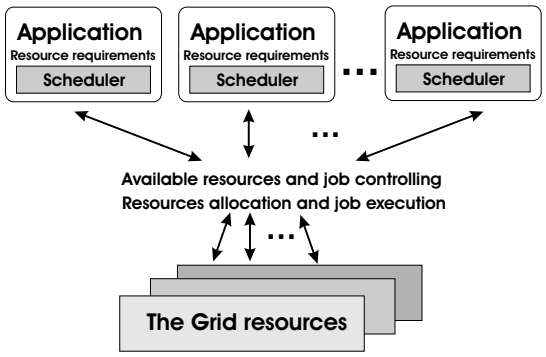


Figure 18.2. Major blocks and flow of information processes in the application level scheduling approach.

Application-level scheduling differs from the job-scheduling approach in the way applications compete for the resources. In application-level scheduling, an application schedule no knowledge of the other applications, and in particular of any scheduling decisions made by another application. Hence, if the number of applications increases (which is a common situation in large and widely distributed systems like a Grid), we observe worse and worse scheduling decisions from the system perspective, especially in terms of its overall throughput. Moreover, since each application must have its own scheduling mechanisms, information processes may be unnecessarily duplicated (because of the lack of knowledge about other applications' scheduling mechanisms).

Hence, the natural question is: Why do we need the application level scheduling approach? In practice, job scheduling systems do not know about many specific internal application mechanisms, properties, and behaviors. Obviously, many of them can be represented in the form of aforementioned hard constraints, as applications requirements, but still it is often impossible to specify all of them in advance. More and more, end users are interested in new Grid-aware application scenarios [AAG+02] (also detailed in Chapter 3. New scenarios assume that dynamic changes in application behavior can happen during both launch-time and run-time, as described in Chapter 6. Consequently, Grid-aware applications adapt on-the-fly to the dynamically changing Grid environment if a purely job scheduling approach is used. The adaptation techniques, which in fact vary with classes of applications, include different application performance models, and the ability to checkpoint and migrate.

Consequently, job scheduling-systems are efficient from high-throughput perspective *(throughput based criteria)* whereas application-level schedulers miss this important criterion, even though internal scheduling mechanisms can significantly improve particular application performance *(application performance based criteria)*.

3.4 Hard and Soft Constraints

In this section we show that by extending present job-scheduling strategies (especially to the way application requirements are expressed), we are able to deal with both throughput and application-centric needs.

As noted in the preceding section, a central scheduler receives many requests from different end-users to execute and control their applications. Requests are represented in the form of *hard constraints*, a set of application and resources requirements that must be fulfilled in order to run an application or begin any other action on particular resources. These constraints are usually specified by a *resource description language* such as RSL in the Globus Toolkit [FK98a], ClassAds in Condor (as described in Chapter 17), or JDL in DataGrid [EDGa]. Simply stated, a resource description language allows the definition of various hard constraints, such as the operating system type, environment variables, memory and disk size, and number and performance of processors, and generally varies with classes of applications on the Grid. Based on our experiences with large testbeds and respecting the idea of Grid computing (as explained in Chapter 1), we are of the opinion that many extensions to the resource description languages are needed to represent *soft constraints*, the criteria for resource utilization, deadlines, response time, and so forth needed when many stakeholders are involved. Unfortunately, soft constraints are rarely expressed in the resource description language semantics, and consequently they are omitted in many resource management strategies. We note that some work has been done in the GridLab project (XRSL description language) to address these issues [GL].

The main difference between hard and soft constraints is that hard constraints must be obeyed, whereas, soft constraints they should be taken into consideration in order to *satisfy* all the stakeholders as far as possible. In other words, *we are able to consider soft constraints if and only if all hard constraints are met*. Yet, soft constraints are actually criteria in the light of decision making process, and they are tightly related to the preferences that the stakeholders of the process want to consider.

We have indicated that resource management in a Grid environment requires a multicriteria approach. However, proposition of a method coping with multiple criteria during the selection of the compromise schedule does not solve the resource management problem entirely. Therefore, apart from presenting our view of the multicriteria Grid resource management process illustrated by an example, we point out possible difficulties and problems as well as techniques that can be used to cope with them in order to perform successfully resource management in the Grid.

4. MULTICRITERIA APPROACH FOR GRID RESOURCE MANAGEMENT

In this section we propose a multicriteria approach to resource management that combines both the performance of an individual job (application-level scheduling), and the performance of the Grid environment (job-scheduling approach) and combines two conflicting sets of criteria introduced as soft constraints. This approach leads to a more intelligent resource management strategy that can fit the needs of the global Grid community as well as their Grid applications.

We start by analyzing possible criteria that can be used in the Grid resource management by various groups of stakeholders. We then propose several possible methods of obtaining and modeling decision makers' preferences. Finally, we show how preference models can be exploited using both decision rules and multicriteria optimization in order to select the most appropriate schedule.

4.1 Criteria

Adequate criteria have to be chosen in order to represent all stakeholders involved in the resource management process. In addition, all aspects of the system and application efficiency must be considered. Hence, our approach must consider criteria that measure application performance as well as the throughput of the overall set of resources. We distinguish two general groups of criteria: that, related to particular stakeholders (such as end users or resource owners) and, that, related to the behavior of the whole system (which can be represented by a VO policy maker or administrator).

Criteria can be also categorized into time criteria (e.g. mean response time, makespan, mean weighted lateness), cost criteria (e.g. weighted resource consumption, cost of the computation) and resource utilization criteria (e.g. load balancing, minimal machine idleness).

The first group applies primarily to end users. This group is also interesting for VO administrator, however, in the context of the whole VO's performance. Cost criteria are important to end users, resource owners and VO administrator. Finally, resource owners and administrators focus on the last group of criteria. In addition, VO administrator's criteria may vary depending on the objectives and polices adopted in a VO.

Within time criteria there is a subgroup of criteria that address lateness criteria including deadlines and due dates. This subgroup is often treated as a constraint when optimizing other criterion (i.e., makespan), although sometimes exceeding the due date does not result in a failure of the whole job or even the single task. Therefore, the difference between a task completion time and its due date can be treated as a criterion that should be minimized.

The important issue is to estimate precisely values of criteria. For instance, calculation of all data transfers needed to provide a task with all required files must to be done. Furthermore, all overheads related to a preparation of a task for execution must be considered, including, for example, installation of required software, compilation of source code, etc.

Examples of definitions of criteria for representatives of different groups of stakeholders that participate in a multicriteria resource management process in the Grid are described in Section 5.

4.2 Modeling the Preferences

As explained in Section 2, decision maker's preferences must be provided to the scheduling algorithm in order to direct the search for compromise schedules. Modeling preferences in a precise way allows participants of the decision process to find satisfactory solutions. As described in Section 2.3, the preferences of a decision maker can be expressed in the form of decision rules, utility functions, or relations. Relational models are adequate for relatively small sets of decision actions.

Consequently, we take advantage of the first two approaches. In the multicriteria grid resource management system, preferences can be represented as decision rules and weights in scalarizing functions. Rules express static preferences regarding local resource management policy. Additionally, a rules induction algorithm can take advantage of previous decisions in order to model decision makers' preferences automatically.

On the other hand, the weights that define the importance of particular criteria can be obtained on the basis of priorities of particular tasks, priorities of end users, or information obtained from stakeholders using the method presented in [KNP01]. In this method, preferences are included in MC-RSL resource specification language, which extends the standard RSL language by adding the possibility of formulating a multicriteria approach for Grid resource management.

An example of expressing preferences of stakeholders in the form of both decision rules and scalarizing functions is presented in Section 5.

4.3 Selection Method

In our opinion, a Grid resource management system should consist of many different scheduling algorithms because of the diversity of applications and dynamics of the Grid environment. An application can be an independent task or a large set of tasks containing precedence constraints, and it can run from several seconds to days. Additionally, the rate of incoming tasks can vary over time. Furthermore, resource owners or system administrators will have

preferences for their general policies. A rule-based system is able to meet such requirements because all policies can be taken into consideration.

Therefore, preferences of stakeholders can be expressed by using decision rules on a high level and functions on a lower level within particular scheduling algorithms. These issues concerning both rule-based systems and scalarizing functions used in multicriteria optimization are considered in the next two subsections.

4.3.1 Rule-Based System

On the basis of the considerations presented above we can identify the following set of requirements that a rule-based system should meet:

a) *Expression of policies.* The system should allow VO administrator and policy maker to define rules with different levels of priorities to determine required behavior for specific end users, tasks, and resources. A rule-based system has to enable stakeholders to define all these preferences (in fact, hard constraints) dynamically, that is, during work of the resource management system.

b) *Execution of different scheduling procedures on the basis of a job type.* Tasks submitted to the Grid may have very differentiated characteristics. In general, for longer and more complex jobs, more advanced scheduling procedures are needed. In addition, different scheduling algorithms should be used for independent and dependent tasks or for interactive or batch applications.

c) *Adaptation to the environment.* Different scheduling strategies can be adapted to a changing Grid environment in order to make the most of the accessible resources. Some decisions can be based on certain factors, such as frequency of incoming tasks, a load of resources, and expressed in the form of rules.

d) *Selection of the best solutions from the Pareto-optimal set.* If we perform a multicriteria choice without any information concerning the importance of a particular criterion, the result is the set of Pareto-optimal solutions. Thus, further processing of the obtained solutions is required. In order to select one compromise schedule without interference of stakeholders, the set of appropriate rules should be used. Rules can be defined by the VO's administrators and end users or determined on the basis of their previous decisions (see [GMS01] to get more information about methods for induction of decision rules).

The conditional part of a rule (see Section 2.3) should contain conditions such as job type, particular resource requirements, and user identification. Most of the conditions can be predefined or obtained in the process of clustering and classification. Furthermore, since it is often difficult to express conditions by using precise numbers, fuzzy conditions (see Section 7) should be allowed as well. For instance, the rule: *'if the job is long, then run optimization using rescheduling'* could be more appropriate than the rule *'if job is expected*

to execute more than one hour, then run optimization using metaheuristic'. In the case of the latter rule, optimization will not be run for a job that usually finishes after fifty-nine minutes. The MC-Broker presented in [KNP01] is an example of a system that takes advantage of decision rules in order to discover resources, evaluate them, and select the best compromise solution from the Pareto-optimal set. Examples of rules used in a multicriteria resource management are given in Section 5.

4.3.2 Multicriteria Optimization

When a number of decision actions (solutions) is large or even infinite, it is impossible to check all of them. In that case, methods searching the space of possible solutions must be applied. We call this process a *multicriteria optimization*.

The majority of scheduling problems are computationally hard even in the single-criterion case. Thus, since exact algorithms are not appropriate for such problems, enumerative or heuristic methods should be used. *Dynamic programming* and *branch and bound* [Węg99] belong to the former approaches, whereas *minimum completion time* or more general *hill-climbing* heuristic can serve as the example of the latter methods. Enumerative methods are, in general, practical only for relatively small instance sizes, because of their computational complexity. On the other hand, the heuristics mentioned above often take a single specific metric into consideration and make decisions on the basis of local knowledge. Consequently, all these methods may turn out to be inefficient, especially in the multicriteria case and for large sets of solutions.

Recently *metaheuristics* [Glo86] have been successfully used to solve various combinatorial problems. They include, among others, *Tabu Search* [Glo89], [Glo90], *Simulated Annealing* [KGJV83] and [Čer85] and *Evolutionary Algorithms* [Hol75], [Gol89] and [Mic92]. The main idea of these methods is general, so they can be used for single- and multi-criteria problems with various metrics as the optimization criteria. The general approach to the single-criterion Grid scheduling problem using metaheuristics has been presented in Chapter 19. These metaheuristics, however, can be extended to multicriteria problems.

In [Nab00] a multicriteria approach based on tabu search has been proposed. The use of this methodology for Grid environments has also been studied and presented in this thesis.

Many researchers emphasize that evolutionary algorithms (EAs) are especially suitable for multicriteria optimization, because of their ability to process many solutions simultaneously in the population, whereas other techniques usually process one solution. This is a good feature in multicriteria optimization, when the goal is to find a set or a subset of Pareto optimal solutions. The main difference between single and multiobjective EAs lies in the selection

phase. On the basis of values on particular criteria, a fitness value is assigned to each solution. There are several techniques of multicriteria selection. Generally, methods of evaluating solution quality may be classified into two groups: using scalarizing (utility) functions, and the so-called Pareto ranking of solutions. It is difficult to judge which of these two methods is more effective because both have advantages and disadvantages. The former approach has been investigated in [IMT96], [Jas98] and [IM98], where it has been successfully used in a multiobjective genetic local search algorithm for workflow scheduling. The latter has been introduced by [Gol89] and studied in [KC00a]. In addition, a comparison of two algorithms using these two different approaches is described in details in [KC00b].

We use scalarizing (utility) functions instead of finding a set or a subset of Pareto-optimal solutions because a resource management system needs to select one schedule rather than the list of possible good solutions. The examples of scalarizing functions, their use, and also certain drawbacks are presented in the next section.

5. EXAMPLE OF MULTICRITERIA ANALYSIS

In this section we present an example of a multicriteria matching of tasks to resources in a single VO. We show how a compromise schedule is selected in the presence of multiple criteria and various participants of the decision making process. Although this example is significantly simplified, it shows the need for a multicriteria approach and explains how notions and definitions given in the preceding sections can be used in practice. We assume that there is one *scheduler* controlling R = 7 *resources* and then $N^{EU} = 3$ *end-users (eu1, eu2, eu3)* submit simultaneously their *tasks* $t_{11}, t_{21}, t_{22}, t_{31}$ using this scheduler. For simplicity, tasks of a particular end-user belong to the single job (which cannot be assumed in general case). In addition, resources are managed and administrated by $N^{RO} = 3$ *resource owners (ro1, ro2, ro3),* and there is also one global *VO administrator* (see the first column in the in Figure 18.3).

Theoretically, the number of possible solutions for this example is 210, and in general the number of solutions increases exponentially with the problem instance size. Note that due to many hard constraints appearing in practice (specified by means of resource description language, policy rules in a scheduler, etc.), the number of feasible solutions can be decreased significantly. For instance, due to specific requirements of tasks (t_{11}, t_{21}, t_{22}, and t_{31}) the set of available resources is limited to r_1, r_2, r_3 and r_4. In addition, the manager's task t_{11} is assigned automatically to the dedicated resource r_3 (because invoking appropriate administration rules) so only r_1, r_2, and r_4 are considered (see the second column in Figure 18.1). Once all hard constraints are met, we can examine the soft constraints and multicriteria analysis.

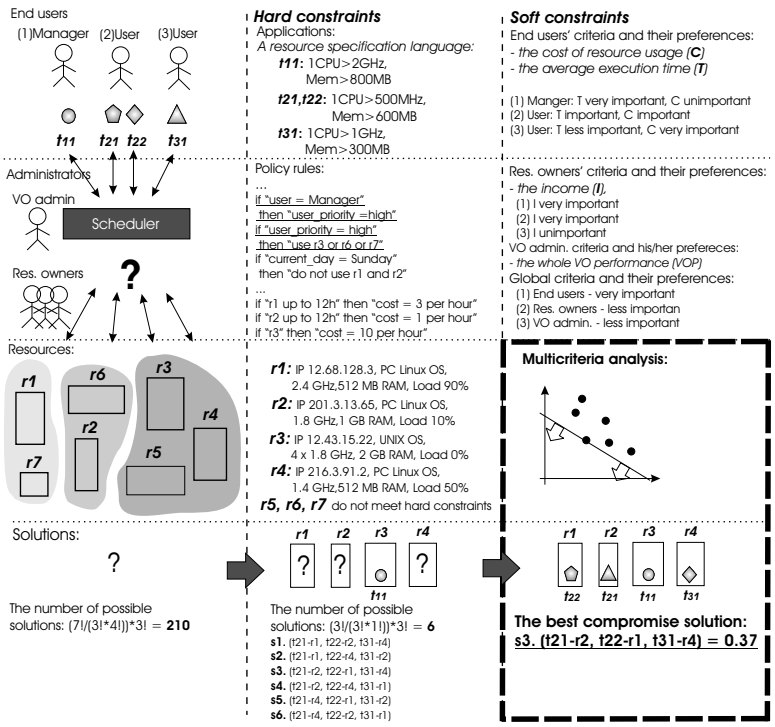


Figure 18.3. Example of a multicriteria approach to Grid resource management.

On Table 18.1 we present the complete notation of parameters using for multicriteria analysis in the example.

Groups of stakeholders (end users, resource owners, administrators, etc.) represent their own preferences. Furthermore, each of these stakeholders may want to evaluate solutions using multiple criteria. The novelty of our approach is that all different points of view of various stakeholders (in the form of soft constraints) are taken into consideration in the Grid resource management process. Besides preferences, the performance of the whole VO is taken into account as well. Thus, we consider *two levels* of criteria: criteria of particular stakeholders (end users and resource owners) along with their preferences concerning these criteria, and aggregated criteria of stakeholders and criteria important from the viewpoint of the VO policy (established by the VO administrator).

To make this example clear, we confine ourselves to $K^{EU}_{eu2,eu3} = 2$ end-users' criteria: T - the average execution time (see Equation (1)) and C - the cost of resource usage (see Equation (2)). Note, that we consider only two end users because eu1's task has been assigned $t_{11} \Rightarrow r_3$. There is also a single criterion $K^{RO}_{ro1,ro2,ro3} = 1$ for resource owners: I - income (see Equation (3))

Table 18.1. Notation of parameters used in the example.

Symbol	Definition
R	number of resources
N^{EU}	number of end-users
N^{RO}	number of resource owners
K_i^{EU}	number of criteria for ith end-user
K_i^{RO}	number of criteria for ith resource owner
K^{VO}	number of criteria for a VO administrator
w_{ij}^{EU}	weight of jth criterion of a ith end-user
w_{ij}^{RO}	weight of jth criterion of a ith resource owner
w_j^{VO}	weight of jth criterion of a VO administrator
p_i^{EU}	priority of ith end-user
p^{RO}	priority of ith resource owner
T_i	number of task of ith user
$tijk$	execution time of jth task of ith user on a kth resource
c_{ijk}	cost of the usage of kth resource by jth task of a ith user
x	solution containing numbers of resources allocated by particular tasks
$\overline{f}_i^{EU}(x)$	scalarizing function for ith end-user
$\overline{f}_i^{RO}(x)$	scalarizing function for ith resource owner
$\overline{f}^{VO}(x)$	scalarizing function for a VO administrator
$f_{ij}^{EU}(x)$	value of jth criterion for ith end-user
$f_{ij}^{RO}(x)$	value of jth criterion for ith resource owner
$f_j^{VO}(x)$	value of jth criterion for a VO administrator

and a single criterion $K^{VO} = 1$ for the VO administrator: VOP - VO's overall performance (see formula (7)).

The goal is to minimize T, C, and VOP (and thus maximize the performance of the whole VO) criteria and maximize the I criterion in order to satisfy all groups of stakeholders. Below, the criteria are presented along with their short descriptions:

- The average execution time (T). This is an average execution time of all tasks submitted by the ith end-user. Optionally weights can be added in order to express task priorities. This criterion can be expressed as

$$T_i(x) = \frac{1}{T_i} * \sum_{j=1}^{T_i} t_{ijx_{ij}}, \tag{18.1}$$

where *i* denotes the number of the end user.

- The cost of resource usage (C). This is the sum of the costs of the resources used by the end user's tasks. This criterion can be expressed as

Table 18.2. Example vales of the total completion time(T) and cost(C) matrix for tasks t_{21}, t_{22}, t_{31} and resources r_1, r_2, r_4

	T				**C**		
	r_1	r_2	r_4		r_1	r_2	r_4
t_{21}	11h	13h	15h	t_{21}	33	13	10
t_{22}	7h	9h	11h	t_{22}	21	9	10
t_{31}	136h	160h	178h	t_{31}	200	100	10

$$C_i(x) = \sum_{j=1}^{T_i} c_{ijx_{ij}}, \tag{18.2}$$

where *i* denotes the number of the end user.

- The income (I). This is the total income for the ith resource owner. This criterion can be expressed as:

$$I_i(x) = \sum_{i,j:x_{ij}=k} c_{ijk}i, \tag{18.3}$$

where *k* denotes the number of the resource, *i* denotes the number of the end user and $j = 1..T_j$

First, the values of particular criteria for end users have to be computed. Therefore, we need basic information regarding the values of parameters needed for T and C criteria, see Table 18.2:

The values of criteria have to be standardized in view of their aggregation using an arithmetic average. This standardization can be done by taking advantage of the knowledge of the extreme values of the particular criteria. These values can be obtained on the basis of various techniques (more details are presented in Section 6), directly from an end user or during a negotiation process, as described in Chapter 8. In our example these ranges are set on the basis of minimum and maximum values. In addition, we assume that the following preferences *unimportant*, *less important*, *important*, and *very important*, denote values of weights: 0, 1, 2, and 3, respectively (see the third column in Figure 18.3).

Scaled values on criteria (T, C) for the second and third end user (Table 18.3) as well as the visualization of end-users' preferences are presented in Figure 18.4.

The Pareto-optimal set for the second end-user includes s3, s6 solutions. Note that for the third end-user all the solutions form the Pareto set. Because of the preferences of the end users concerning the C and T criteria, the best

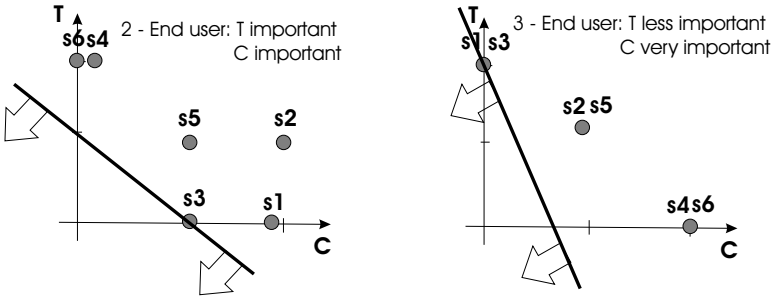


Figure 18.4. Scaled values of solutions in the criteria space (C, T) for second and third end user.

compromise solutions are: *s3* for eu2, and *s1 or s3* for eu3 (see Section 2.2). Let us now begin the second level of multicriteria analysis.

Aggregation of an end user's criteria can be done by using a scalarizing function, for instance

$$\overline{f}_i^{EU}(x) = \frac{1}{\sum_{j=1}^{K_i^{EU}} w_{ij}^{EU}} * \sum_{j=1}^{K_j^{EU}} (w_{ij}^{EU} * f_{ij}^{EU}(x)), \tag{18.4}$$

where *i* denotes the number of the end user.

We need to aggregate the criteria of the end-users and administrators in order to represent their points of view in a global aggregation with VO-specific criteria. In the example we assume that the priority of the second end user (eu2) is adjusted to $p_{eu2}^{EU} = 2$, and the third to (eu3) $p_{eu3}^{EU} = 1$ and all the resource owners have the same priority p = 1. We propose the following (minimization) criteria:

- End users' satisfaction (EUS). This criterion measures to what extent an end-users' preferences have been met. It can be defined as a weighted average of the end users' scalarizing functions values:

Table 18.3. Scaled values on criteria C and T.

Solutions	T_{eu2}	C_{ue2}	T_{eu3}	T_{eu3}
s1 $(t_{21} \Rightarrow r_1, t_{22} \Rightarrow r_2, t_{31} \Rightarrow r_4)$	0.00	0.96	1.00	0.00
s2 $(t_{21} \Rightarrow r_1, t_{22} \Rightarrow r_4, t_{31} \Rightarrow r_2)$	0.50	1.00	0.57	0.47
s3 $(t_{21} \Rightarrow r_2, t_{22} \Rightarrow r_1, t_{31} \Rightarrow r_4)$	0.00	0.63	1.00	0.00
s4 $(t_{21} \Rightarrow r_2, t_{22} \Rightarrow r_4, t_{31} \Rightarrow r_1)$	1.00	0.17	0.00	1.00
s5 $(t_{21} \Rightarrow r_4, t_{22} \Rightarrow r_1, t_{31} \Rightarrow r_2)$	0.50	0.50	0.57	0.47
s6 $(t_{21} \Rightarrow r_4, t_{22} \Rightarrow r_2, t_{31} \Rightarrow r_1)$	1	0	0	1

$$EUS(x) = \frac{1}{\sum_{i=1}^{N^{EU}} p_i^{EU}} * \sum_{i=1}^{N^{EU}} (p_i^{EU} * \overline{f}_i^{EU}(x))$$

- Resource owners satisfaction (ROS). This criterion measures to what extent a resource owners preferences have been met. It can be defined as a weighted average of resource owners' scalarizing functions values:

$$ROS(x) = \frac{1}{\sum_{i=1}^{N^{RO}} p_i^{RO}} * \sum_{i=1}^{N^{RO}} (p_i^{RO} * \overline{f}_i^{RO}(x))$$

- VO's overall performance (VOP). This criterion can be defined using various methods, e.g. measuring throughput or average completion time of all tasks submitted in a VO. We present formula for the average completion time below:

$$VOP(x) = \frac{1}{\sum_i T_i} * \sum_{ij} t_{ij x_{ij}}, \text{ where } i = 1, ..., N^{EU}, j = 1, ..., T_i$$

In Figure 18.5, all the solutions are presented in the global criteria space. Note, that in the final multicriteria analysis there are three Pareto-optimal solutions: s3, s5, s6. In this case, *the best compromise solution* is s3 (the minimum overall value equals 0.37 according to overall calculations including aggregated criteria and global preferences).

The weights (priorities) were set up by the global VO administrator in order to satisfy the end-users (weight 3 for EUS) more than resource owners and the whole performance of VO (weights equal 1 for ROS and VOP). By changing these weights a VO administrator can introduce different administration policies into practice in the form of soft constraints, another big advantage of our approach.

We emphasize again that this example is simplified for easier understanding of the multicriteria approach to Grid resource management. For instance, it assumes that tasks start their execution immediately after being submitted to a resource. Thus, the task execution time is equal to the task completion time. In addition, the simplest aggregation operators have been selected, such as a weighted average. This operator has many drawbacks. One of them is a

Table 18.4. Scaled values on the global criteria space.

Weights	3	1	1	
Solutions	EUS	ROS	VOP	Overall
s1 $(t_{21} \Rightarrow r_1, t_{22} \Rightarrow r_2, t_{31} \Rightarrow r_4)$	0.40	0.03	1.00	0.45
s2 $(t_{21} \Rightarrow r_1, t_{22} \Rightarrow r_4, t_{31} \Rightarrow r_2)$	0.67	0.53	0.58	0.62
s3 $(t_{21} \Rightarrow r_2, t_{22} \Rightarrow r_1, t_{31} \Rightarrow r_4)$	*0.29*	*0.02*	*1.00*	*0.37*$\star$
s4 $(t_{21} \Rightarrow r_2, t_{22} \Rightarrow r_4, t_{31} \Rightarrow r_1)$	0.64	0.52	0.00	0.49
s5 $(t_{21} \Rightarrow r_4, t_{22} \Rightarrow r_1, t_{31} \Rightarrow r_2)$	0.50	0.50	0.58	0.52
s6 $(t_{21} \Rightarrow r_4, t_{22} \Rightarrow r_2, t_{31} \Rightarrow r_1)$	0.58	0.50	0.00	0.45

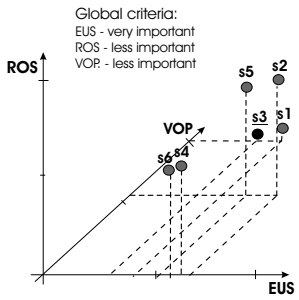


Figure 18.5. Solutions in the global criteria space (EUS, ROS and VOP).

compensation of criteria values. Consequently, the decision is very sensitive to the scaling of the attributes. Moreover, some values of aggregation function are close to each other so it is difficult to conclude which of them is the best. The use of more suitable operators will be considered in future work.

Additionally, we assumed in the example that exact values of all parameters are known, an assumption that may be not true in a practice. Thus, the use of prediction techniques and methods for representing imprecise information should be investigated, and we point out to these issues in the next section. Finally, since state of resources and tasks may change dynamically, a Grid resource management system should be able to adapt its decisions to this variable environment.

6. GRID SPECIFICITY AND AI SUPPORT

In the preceding sections we have presented motivations and methods of the multicriteria approach for Grid resource management, as illustrated by the example in Section 5. While designing such methods, however, we cannot forget about the peculiarity of the Grid environment.

In the Grid environment we have to deal with more rapidly changing and often unpredictable input data, causing uncertainty and incompleteness of information. The availability of resources as well as the status of jobs may change rapidly. Additionally, in practice, in terms of scalability and efficiency, one central job scheduler is not able to control all the resources in the VO. On the other hand, even advanced adaptive techniques for applications do not guarantee efficient scheduling because of the lack of knowledge of the global state of resources and jobs.

Thus, a multicriteria decision making process must fit the Grid nature and be supported by additional techniques in order to meet the requirements of Grid environments.

In particular, two issues are of great importance for efficient resource management in the Grid: dealing with imprecision of information and adapting to

the changing environment. We consider these in the next two subsections. In view of the main goal of the chapter, which presents the multicriteria approach for a resource management in the Grid, we confine ourselves to indicating possible methods that can be used to cope with the problems given at the beginning of this paragraph.

6.1 Dealing with Imprecision of Information

A Grid environment contains a large number of heterogeneous resources being used simultaneously by many users running their various applications. In such an environment the resource management system is not likely to have access to all necessary up-to-date information regarding the current state of the resources and submitted jobs.

We see two approaches that could help to cope with imprecise information: appropriate *representation of information* and *prediction techniques*.

6.1.1 Representation of Information

The imprecision of information is a very common phenomenon both in the natural environment and in complex systems created by people. We have to deal with various forms of imprecision, including inaccuracy, uncertainty, incompleteness, random variability, or ambiguity of information. As a consequence, many methods of modeling imprecise information have been introduced. We present briefly three approaches along with possibilities of their use in multicriteria Grid resource management.

Calculus of probability and mathematical statistics. The calculus of probability and mathematical statistics provide well-known and widely used measures and tools to take the random variability of parameters into consideration. Basic measures such as an average and variance enable a scheduling algorithm to use the knowledge concerning the uncertainty of input information. Furthermore, tools such as regression allow a more advanced analysis, for example, expressing a job execution time as a function of an input data size. The use of regression in the prediction of job execution time is presented in [VS03].

Fuzzy set theory [Zad65]. Given a classical set, we have to determine whether each object belongs to the set. In the case of a fuzzy set, objects may belong to the set to a certain extent, for example, 0.6. This extent is expressed by the *membership function*. Values of this function range from 0 to 1. As far as Grid resource management is concerned, fuzzy sets can be constructed in two ways: through the analysis of frequencies of some events included in historical data or the definition of the so-called linguistic variables. For example, fuzzy sets can express an execution time of jobs because exact, single values are usually hard to determine. Of course, such a representation of information requires special methods in order to exploit this knowledge. Thus, fuzzy

operators have to be defined, and scheduling algorithms should be adapted to operate on such data. The example of a fuzzy logic utilization in a load balancing system is investigated in [Che01].

Rough set theory [Paw82]. Rough set theory is a mathematical tool for the analysis of inconsistency or ambiguity that results from a granulation of information. Objects (jobs) having the same description are indiscernible (similar) with respect to the available information. Thus the universe is partitioned into blocks of indiscernible objects, called elementary sets. Every object is associated with a certain amount of information, expressed by means of a set of attributes such as the size of the input data or a list of parameters. A subset of jobs may be characterized using *upper* and *lower approximations*. The lower approximation of the subset is composed of objects that certainly belong to this subset, while the upper approximation consists of objects that may belong to this subset. On the basis of such sets, certain and possible decision rules can be found. As a consequence, one can distinguish between certain and possible information. In the case of jobs in a Grid environment, the scheduling algorithm is then able to take advantage of the knowledge of, for example, certain and possible job execution times. More details concerning the use of rough sets in multicriteria decision making can be found in [GMS01].

6.1.2 Prediction Techniques

Sometimes input information required by a scheduler is not available, so the necessary parameters must be estimated on the basis of historical data. Generally speaking, these parameters can be predicted by using statistical or AI methods.

Statistical techniques. Statistical methods are commonly used for the analysis of data in various disciplines. A variety of statistical tools are available. In particular, *correlation* and *regression* can be used in order to find dependencies between particular attributes of jobs, for example, to what extent a job execution time depends on the number of processors available or a size of input data. Statistical methods can also be used in an analysis of *time series* [WSH99a].

AI techniques (knowledge discovery and machine learning). AI techniques have become increasingly popular in various domains. These methods take advantage of automatic learning in order to find dependencies in data or predict future behavior of a system or an environment. Several techniques from this field that can be used in Grid resource management are briefly described in the next paragraphs.

- *Classification.* The classification analysis is the organization of data into predefined classes. Decision trees [Qui86], naive Bayes classifiers [LIT92], and neural networks [RHW86] are the best-known methods for the classification problem. In the case of Grid resource management, we can classify jobs according to, for example, memory usage. Then, analyzing the classified

jobs, we can find characteristics of these jobs that require particular amounts of memory. Furthermore, classified jobs may be used for prediction. It is possible, on the basis of job parameters, to predict the amount of memory that should be reserved for this job. The resource management system can take advantage of this knowledge, using it directly during the optimization process.

- *Clustering.* Clustering is a method similar to classification. The main difference is that it does not require any initial definition of classes. Using this method, one can divide jobs into groups and find descriptions of these groups. For example, groups of jobs having similar resource requirements can be found. The k-means algorithm [DM90] is an example of clustering.

- *Sequential patterns.* This method takes a sequence of operations into consideration. Search dependencies are not restricted to single records of data (e.g., jobs). The order of events is important, too. The most popular algorithm is based on an associative rules generation method called Apriori [AS94]. Used in a Grid environment, it can give information about frequent sequences of jobs run by particular users. Appropriate use of this knowledge should improve significantly efficient resource use because the scheduler would be able to predict the order of submitted jobs.

In view of an imprecision or even lack of necessary input information to the Grid, an efficient use of the methods listed above for multicriteria resource management will be investigated in our future work.

6.2 Adapting to Changing Environment

It is often assumed that the role of the scheduling algorithm ends when a job is submitted to resources. However, this is not true for Grid environments where the state of the resources and tasks changes rapidly, various faults occur or hardware configurations evolve during an application runtime. Hence, various methods have to be developed to address dynamic application adaptation. These approaches can be divided into three groups: *monitoring, rescheduling* and *task migration.*

The goal of monitoring is to gather information about various faults, application performance, and the state of the resources. A rescheduling component has to calculate a new schedule for the application. This must be lightweight and fast because rescheduling is done while the application is running. Consequently, unsophisticated heuristics or the so-called incremental methods that take advantage of the first schedule (perhaps modifying it slightly) are needed. Task migration techniques (including checkpointing) enable an application to stop the execution, migrate to another resource (perhaps with the required data), and start from the same point where it was previously stopped.

Generally speaking, decisions concerning the adaptation of an application can be made in the following three situations: (i) in the case of faults or changes in a hardware configuration that results in an application failure, (ii) if it is im-

possible to meet earlier defined requirements such as deadlines (e.g., in the case of dramatic changes in the availability of resources), or (iii) in order to improve execution of an application (e.g., shorten the execution time or decrease the cost of a resource consumption).

Techniques that enable applications to dynamically adapt to a changing state of resources and jobs are being developed in the scope of the GridLab project [GL].

7. CONCLUSIONS

The approaches described in this chapter cover some of the methods that can be used to improve the multicriteria Grid resource management process. Such methods are crucial if we want to fully exploit the possibilities of the Grid. In Section 3 we emphasized the need for compromise between two leading approaches to scheduling and their conflicting goals. These goals result in the existence of two groups of conflicting criteria, high-performance and high-throughput, that together reflect the wider perspective of decision processes and the need for new strategies. We present possible methods for aggregation of such criteria in Section 4. In Section 5 we illustrated these methods using an example and gave more practical view of how a multicriteria approach can be used in the context of Grid resource management. However, we also indicate that our approach requires additional support from AI-based techniques mainly during information gathering and application execution control. Some of the issues are described briefly in Section 6.

Our general analysis is motivated by the main objective of this chapter to define a general framework for the Grid resource management processes from the multicriteria decision making perspective. Consequently, the identification of problems, possible techniques, and research directions are considered in the scope of this chapter, rather than a detailed analysis of specific solutions. We also identified challenges that need to be investigated in future research, including methods of aggregating criteria in the presence of many decision makers, representing imprecise information concerning a state of jobs and resources, using prediction techniques, and adapting applications to a changing environment.

Acknowledgments

We are pleased to acknowledge support from the EU GridLab project (IST-2001-32133).

Chapter 19

A METAHEURISTIC APPROACH TO SCHEDULING WORKFLOW JOBS ON A GRID

Marek Mika,[1] Grzegorz Waligóra,[1] and Jan Węglarz[1,2]
[1] *Institute of Computing Science, Poznań University of Technology*
[2] *Poznań Supercomputing and Networking Center*

Abstract In this chapter we consider the problem of scheduling workflow jobs on a Grid. This problem consists in assigning Grid resources to tasks of a workflow job across multiple administrative domains in such a way that minimizes the execution time of a particular set of tasks. The considered problem is formulated as a multi-mode resource-constrained project scheduling problem with schedule-dependent setup times, which is an extension of the classical resource-constrained project scheduling problem to minimize the makespan (RCPSP). We present a binary linear programming (0-1 LP) formulation of the problem, and propose a local search metaheuristic approach to solve the considered problem.

1. INTRODUCTION

We consider the problem of scheduling workflow applications on a Grid. The objective of our research is to formulate the problem as an extension of the resource-constrained project scheduling problem, to build its mathematical model, and to propose a local search metaheuristic approach to solve it. Although the approach we develop is general and allows the scheduling of any set of tasks on any set of resources, we consider so-called workflow applications because of their particular practical importance. Workflow applications can be viewed as complex sets of precedence-related various transformations (tasks) performed on some data. They are mostly scientific, data intensive applications that require large amounts of computing power to be executed efficiently. Although different criteria may be considered when evaluating schedules, we focus on a time criterion, more precisely, we try to minimize the time to execute a given workflow job.

The problem of scheduling workflow applications across many sites on a Grid is very complex, especially when the network capacity varies between

the sites. In addition, we often do not possess complete information about the jobs. The process of obtaining a performance model of a job is not trivial. In particular, the processing times of all the tasks on different computer systems (Grid resources) are not easy to evaluate. Also other parameters, for example bandwidth, resource availability, etc., may change quite rapidly in Grid environments. Thus, generally, we deal with scheduling under uncertainty. In this chapter we will show under what assumptions we can bring the considered problem to a purely deterministic scheduling problem. More precisely, under these assumptions we will formulate our problem as an extension of the resource-constrained project scheduling problem (RCPSP). In this problem, activities (tasks) are scheduled in such a way that the precedence as well as resource constraints are satisfied, and the project duration (the completion time of a given set of activities, also known as the *makespan*) is minimized.

It is easy to see the similarity between a workflow job and a project. In both cases we have a set of precedence-related tasks (activities) that are to be scheduled on a given set of resources. Thus, it is justified to describe the problem of scheduling workflow jobs in terms of project scheduling. There are many extensions of the RCPSP. Since in Grid environments tasks can be processed on different types of resources, i.e. they can be executed in several different ways (modes), we consider project scheduling in its multi-mode version (MRCPSP). Moreover, we base our approach on an extension of the MRCPSP with so-called setup times, where a setup time is a time necessary to prepare the required resource for processing a task. Setup times in our problem are transmission times of files between tasks. These times depend on which resources the tasks are scheduled. As a result, we obtain a problem that is called the multi-mode resource-constrained project scheduling problem with schedule-dependent setup times (MRCPSP-SDST).

The RCPSP is known to be NP-hard, and therefore the MRCPSP-SDST is also NP-hard since it is a more general problem. Later in this chapter we will formulate the considered problem as a 0-1 LP problem (binary linear programming problem). Although this problem can be solved optimally by using specialized binary linear solvers, the complexity of any exact method for solving it is exponential. Therefore, in this research we propose another approach to solve the MRCPSP-SDST, namely to use local search metaheuristics that have proved to be very efficient strategies for the classical MRCPSP.

This chapter is organized as follows. In Section 2 we describe the problem of scheduling workflow applications on a Grid in more detail. We define the information the superscheduler needs to build a schedule, distinguishing between parameters characterizing tasks, resources and networks. Section 3 is devoted to the mathematical formulation of the considered problem. In this section we introduce the assumptions and notation, and we formulate our problem as the MRCPSP-SDST. Finally, we present a 0-1 LP problem that is to be solved in

order to construct an optimal schedule. In Section 4 we present a metaheuristic approach to the considered problem. In this section we briefly describe the idea of local search, then we detail the most commonly used metaheuristics: simulated annealing, tabu search, and genetic algorithms. Next we propose a solution representation, a method of choosing a starting solution, a definition of the objective function, and a neighborhood generation mechanism, as elements of a local search algorithm approach for our problem. We also describe the method of transforming a feasible solution into a schedule (a so-called decoding rule), as well as the process of mapping, defined as assigning actual resources existing on a Grid to tasks. The last section contains conclusions and some directions for further research, including various extensions of the model and the approach proposed.

2. PROBLEM DESCRIPTION

In this section we describe the problem of scheduling workflow jobs on a Grid. We define the information (about applications as well as about the Grid environment itself) that the superscheduler needs in order to schedule tasks of a workflow application on Grid resources. Moreover, we distinguish between information provided by the user and information that can be obtained by some Grid service. We formulate all the assumptions needed to approach the considered problem from the resource-constrained project scheduling point of view.

2.1 Problem Overview

Let us start with a brief description of a workflow application. In many scientific areas, such as high-energy physics, bioinformatics, astronomy, and others, we encounter applications involving numerous simpler components that process large data sets, execute scientific simulations, and share both data and computing resources. Such data intensive applications consist of multiple components (tasks) that may communicate and interact with each other over the course of the application. The tasks are often precedence-related, and the precedence constraints usually follow from data flow between them. Data files generated by one task are needed to start another task. In this way, an output of one task becomes an input for the next task. Although this is the most common situation, the precedence constraints may follow from other reasons as well, and may be arbitrarily defined by the user. Such complex applications, consisting of various precedence-related transformations (tasks) performed on some data with data files transmitted between them often, are called *workflow applications*. For example, in astronomy, workflows with thousands of tasks need to be executed during the identification of galaxy clusters within the Sloan Digital Sky Survey [SKT+00]. Because of large amounts of computations and

data involved, such workflows require high computing power to be executed efficiently. This can be delivered by a Grid.

Since workflow applications are usually very time consuming (even if single tasks can be quite short), and input/output data files for tasks can be quite large, the problem of scheduling such applications on a Grid is very challenging, and has great practical importance these days. Below we address this problem in more detail.

In general, the superscheduling problem has been defined as assigning Grid resources to tasks across multiple administrative domains. Users submit their jobs (in this case, workflow applications) to a Grid. As it has been mentioned, *jobs* consist of multiple tasks. A *task* can be anything that needs a resource — a bandwidth request, a schedulable computation, a data access, or an access to any remote resource, such as remote instruments, databases, humans-in-the-loop, etc. A *resource* is anything that can be scheduled, for example a machine, disk space, a QoS network, a person, etc. Each resource is described by a set of attributes. These attributes are, in general, static data concerning the CPU, memory, disks, operating system, and network interfaces.

A superscheduler has to assign available Grid resources to tasks taking into account an assumed performance measure. We try to minimize the time of execution of a given workflow job although various other measures may be considered for the scheduling criterion, for example cost, reliability, resource leveling, etc., and a multi-objective approach that combines two or even more measures (see [KNP01] and Chapter 18) may be justified.

In order to build a formal model of the scheduling problem considered we need to define precisely the assumptions, and decide what features of the real problem should be taken into account. If we consider too few attributes, a flaw in having a too general approach, the model may be unrealistic, but on the other hand, too many attributes can result in a model that is too complex to be solved by any existing or newly developed method. Therefore it is crucial to identify the parameters that allow us to build both a realistic and a solvable model of the problem.

2.2 Problem Parameters

A workflow job consists of multiple tasks. Thus, first of all the order of execution of tasks must be defined, i.e. the precedence constraints between them. This information is usually represented by a directed acyclic graph (DAG), where vertexes correspond to tasks while arcs represent the precedence constraints. A DAG specifies the sequence of execution of tasks, and no task can be started before all its predecessors are finished. The precedence constraints usually follow from data flow between tasks, thus, they are given in advance and known a priori.

Each task can be processed on different resources. The information determining the required types of resources (CPU, minimum memory size, minimum available disk space, operating system, etc.) on which a particular task can be performed is also needed by the superscheduler. This information defines the list of admissible Grid nodes (resource units) for each task. Once the set of resources is known, an estimated processing time of each task, for each node on which the task may be performed is needed in order to evaluate the makespan. In general, this may be done using a function to bind the duration of a task with some characteristics of nodes as well as with input data of this task, but any method must be able to approximate the processing time of the task on each node. This information can be obtained from a prediction system, for example as described in Chapter 16 or [KNP00, SFT98], or we can use a technique to determine the expected execution time of a process, for example an analysis of the data obtained from previous executions. The latter approach is especially applicable considering the fact that workflow applications are often launched many times on different sets of data. In any case, we assume that this information is available for the superscheduler. In Table 19.1 we present an example task vs. node matrix, where value p_{jk} is the processing time of task j on node, k, and X denotes that this task cannot be executed on the relevant node.

Table 19.1. Matrix of processing times of tasks.

	node 1	*node 2*	...	*node k*	...	*node N*
task 1	p_{11}	X	...	p_{1k}	...	p_{1N}
task 2	p_{21}	p_{22}	...	X	...	p_{2N}
⋮	⋮	⋮		⋮		⋮
task j	p_{j1}	p_{j2}	...	p_{jk}	...	p_{jN}
⋮	⋮	⋮		⋮		⋮
task n	p_{n1}	p_{n2}	...	p_{nk}	...	X

As mentioned before, tasks mostly communicate between themselves in such a way that an output file of one task becomes an input file for another task. Usually the sizes of these files are very large, and therefore their transmission times must be considered. Thus, the next necessary parameter to define is the transmission time of the input/output files between all pairs of communicating tasks. This value cannot be defined without knowing which nodes will execute a task, but this is unknown in advance. However, in practice, the transmission time can be calculated using only the sizes of transmitted files as well as a few parameters (e.g. average bandwidth, latency, etc.) of the physical link between the two nodes executing the tasks. Of course, the results of these calculations

depend on the prediction methods used, as well as on the sizes of the transmitted files [VS02].

Thus, the user should provide the maximum expected size of the input and output files for each task, as well as a communication graph describing what files are to be transferred from one task to another. We assume that after the execution of a task, an output file it generates is saved on the node that ran the task. In the case of tasks that use as an input file already existing on the Grid (not generated by other tasks), we assume that the sizes and locations of these files are known in advance. Of course, a particular file can be replicated and distributed across multiple sites, and therefore it may exist in many different locations, as detailed in Chapter 22 and elsewhere [CDF+02, SSA+02].

We do not give consideration to the time to transfer the executables of the tasks themselves. The size of these files are insignificant in comparison with the size of data files of a workflow job, and therefore their transmission times may be neglected. Still, this assumption is not critical for the approach, and these times can be easily considered when it is needed.

Continuing, the system must supply up-to-date information about the quality of the physical link between nodes of a Grid. We assume that the transmission of files between different pairs of tasks may be executed in parallel. In other words, we assume that the bandwidth of the network interface of each node is much greater than the bandwidth of the physical link between these nodes. Thus, it is possible to transmit several files from one task to several other tasks in parallel [ABB+02a, ABB+02b]. We also assume that if more than one file is transmitted from a given task to another one, then these files are transmitted sequentially. In Table 19.2 we present a sample bandwidth matrix between nodes of a Grid. Value b_{kl} denotes the bandwidth between nodes k and l, and it is assumed to be independent from the data transfer direction i.e. b_{kl}=b_{lk}.

Table 19.2. Matrix of bandwidth between nodes.

	node 1	*node 2*	...	*node k*	...	*node N*
node 1	X	b_{12}	...	b_{1k}	...	b_{1N}
node 2	b_{21}	X	...	b_{jk}	...	b_{2N}
⋮	⋮	⋮		⋮		⋮
node k	b_{k1}	b_{k2}	...	X	...	b_{kN}
⋮	⋮	⋮		⋮		⋮
node N	b_{N1}	b_{N2}	...	b_{Nk}	...	X

Summarizing, we assume that data to characterize the following is available:

- Workflow job properties:
  - A precedence directed acyclic graph (DAG) that describes the control flow between tasks of the job.
  - The processing time of each task on each type of resource. This may be a function defined on the types of resources on which this task can be performed and on its input data.
  - A communication graph that lists the files to be transmitted between tasks, as well as the sizes of those files.
  - Any deadlines and/or ready times of the tasks, if they are imposed.
- Grid environment properties:
  - The point-to-point characteristics (bandwidth, latency, etc.) of the network connections between the nodes of a Grid.
  - The characteristics of resources on which particular tasks can be executed.

Many of these parameters are temporal, average, or uncertain. Some of them change slowly, and some can change in minutes or seconds. For example, the processing time of a task may be determined by the user, according to his expectations. However, this processing time may change for some unpredictable reasons. This means that some tasks may be completed earlier, but others may take longer to run than expected. In such cases, the job may be canceled by the node on which it was running, and therefore may need to be executed again (perhaps on another node), or else the entire job may be canceled. In addition, the network characteristics can change very rapidly, which may result in a miscalculation of the transmission times of some files, and consequently affect the schedule. Such changes in network performance may be able to be predicted, using some specialized services, for example the Network Weather Service, detailed in Chapter 14.

Nevertheless, despite these difficulties, it is still reasonable to build a model of the considered problem under the above assumptions. The time benefit following from proper scheduling tasks of a workflow application on a Grid can be large, even when realizing some uncertainty or changes of the problem data.

3. MATHEMATICAL MODEL

In this section we present a mathematical model of the considered problem, as the multi-mode resource constrained project scheduling problem with schedule-dependent setup times (MRCPSP-SDST). The MRCPSP-SDST is an extension of the *resource-constrained project scheduling problem* (RCPSP)

that can be used to model various discrete optimization problems including cutting stock problems, high school timetabling, audit staff scheduling, or machine scheduling (single-machine scheduling problems, problems with identical or uniform parallel machines, and shop scheduling problems). In this problem activities (tasks) are to be scheduled in such a way that the precedence as well as resource constraints are satisfied, and the project duration (the makespan) is minimized.

3.1 Resource-Constrained Project Scheduling Problems

Let us start with the definition of the RCPSP. We assume we have a set of n non-preemptable activities that need to be executed using and consuming some resources. In general, resources can be renewable, non-renewable, doubly constrained, or partially renewable. *Renewable resources* are available at any time in limited numbers of units, i.e. an available amount of such a resource is renewed from period to period. For *non-renewable resources*, total consumption of the resource units is limited for the entire project. For *doubly constrained resources*, both total and per period availabilities are limited. Availability of *partially renewable resources* is defined for a specific time interval (a subset of periods). Each activity uses and/or consumes several units of some renewable, non-renewable, doubly constrained, or partially renewable resources.

Precedence constraints are defined between activities, as given by relations of the type: $i \rightarrow j$, where $i \rightarrow j$ means that activity j must not start before activity i is completed. A project is generally represented by a directed acyclic graph $G(V, E)$. There are two possible representations of graph G. The first one is called *activity-on-node* (AoN) network, where a set of vertexes V for graph G corresponds to the set of project activities, and a set of arcs $E = \{(i, j) : i, j \in V; i \rightarrow j\}$ represent the precedence constraints between activities of set V. The second representation is called *activity-on-arc* (AoA) network, where set E corresponds to activities of the project and set V consists of time events, representing, for example, completion of several activities.

It is possible to execute each activity in one of several alternative *modes* that represent a relation between the used and consumed resources, and the duration of the activity. In such a case, the resulting problem is called the *multi-mode resource-constrained project scheduling problem* (MRCPSP). The objective is to find an assignment of modes to activities, as well as precedence- and resource-feasible starting times of all activities, such that we minimize the makespan of the project.

The MRCPSP problem is strongly NP-hard, since it is a generalization of RCPSP. The RCPSP is strongly NP-hard as a generalization of the well-known job shop problem [BLRK83]. Moreover, for more than one non-renewable resource, the problem of finding a feasible solution of the MRCPSP is NP-

complete [Kol95]. For a comprehensive survey on project scheduling see [BDM+99], and a review of recent models, algorithms, and applications can be found in [Węg99].

In several papers, extensions of the MRCPSP with setup times have been considered, where, generally, a *setup time* is the time necessary to prepare the required resource for processing an activity. There are two types of setup times considered in the literature: *sequence-independent setup times* [Kol95] and *sequence-dependent setup times* [NSZ02]. In the first case, setup times depend only on the resource that processes the respective activity. In the second case, setup times depend not only on the resource but also on the sequence of activities processed on this resource. For example, if there are three activities i, j, k such that $i \rightarrow k$ and $j \rightarrow k$, and they are to be processed on the same resource, the setup time needed for processing activity k may vary, depending on whether activity k is executed immediately after activity i, i.e. the order is (j, i, k) or immediately after activity j, i.e. the order is (j, k, i).

In the problem of scheduling workflow jobs on a Grid, setup times follow from transmissions of files between tasks. A node is prepared for processing a task only when all input files required for the task are stored on the local disks of this node. However, it is easy to see that the transmission times of the files depend on which nodes the files are transferred between. In other words, the transmission times (setup times) depend not only on the node to which the files are transferred, but also on the nodes where they are originally located. Because of this, setup times in our problem do not depend only on the sequences of tasks on particular resource units, but, more generally, on the assignment of nodes to tasks over time. Thus, they are not just sequence-dependent, but they are *schedule-dependent*. We call the resulting problem the *multi-mode resource-constrained project scheduling problem with schedule-dependent setup times* (MRCPSP-SDST). In the next subsection we show how we model our problem as the MRCPSP-SDST.

3.2 MRCPSP-SDST Model of the Problem

A workflow job and its tasks correspond to a project and its activities, respectively. Precedence constraints of the type $i \rightarrow j$ usually follow from data flow between tasks i and j, that is, data files generated by task i are required to start task j. Thus, the structure of the workflow job can by represented by an AoN graph $G(V, E)$, where set V is the set of tasks, and set E represents the precedence constraints between tasks. No task may be started before all its predecessors are finished. Vertexes in graph G are numerically numbered, and a task has always a higher number than all its predecessors.

Graph G may differ from the original DAG submitted by the user for two general reasons. First, we assume that graph G submitted to the superscheduler

is presented in its minimal form. For example, a job submitted by the user may contain tasks previously submitted by the same or another user, and therefore some data files required as an input to other tasks may already exist somewhere on a Grid. Thus, there is no need to execute these tasks once more, and they can be removed from the graph. Of course, the action of reducing the original DAG will not be performed by the superscheduler, rather it should be processed by a specialized application. Second, it is usually assumed that a project should have exactly one starting and exactly one finishing activity. If graph G, after the reduction phase, has more than one starting and/or finishing task, it may be necessary to modify the graph. This can be done by adding (if necessary) one additional dummy starting task that should be processed before all other tasks of the project, and/or one additional dummy finishing task that must not start before the completion of all other tasks. The dummy tasks do not require any resources, and their processing times are equal to 0.

We assume that there are two types of tasks represented by vertexes of graph G. The first type are *computational tasks* that may need input data and can produce output data files. The second type are *schedule-independent data transfers*, or transmissions of files that are not output files of other tasks of a given workflow job. For example, if it is necessary to transfer a file that already exists from a given physical location to a task, then this data transfer is treated as a task. Otherwise, if a data transfer concerns a file that does not exist yet, but will be generated as an output file of another task, it is not treated as a task but as setup time. Thus, we distinguish two types of data transfers: schedule-independent data transfers treated as tasks, and schedule-dependent data transfers (setups) that, in contrast, are not treated as tasks.

Before defining the second type of data transfers, we need to introduce the concepts of execution modes, resource types, and resource groups. As described in Section 2, each node existing on a Grid can be described by a set of attributes that contain the information such as CPU, RAM, disks, network interfaces, operating system, etc. It is possible to divide the set of all Grid computational resource units into disjoint subsets according to these attributes in such a way that computational resource units with the same attributes belong to the same subset, and computational resource units with different attributes belong to different subsets. These subsets will be called *resource types*. As a consequence, resource units having identical characteristics (described by an identical set of attributes) are considered as resources of the same type.

Of course, there are also resources other than computational ones. For the problem of scheduling workflow jobs on a Grid, network resources are also important. We assume that the bandwidth of network interfaces is high enough to exclude them from further consideration. This means that the bandwidth of each network interface is much greater than the bandwidth of the physical link between these interfaces, therefore the only limitation is the bandwidth of the

physical link. It is possible to gather basic information about the network resources (average or temporary bandwidth of the physical link, average latency, etc.) between every two nodes of a Grid, as we have shown in Table 19.2.

Next, we can divide the set of all computational resource units into disjoint subsets according to this information, similar to the way in which the resource units were divided into resource types. These subsets are called *resource locations*, and contain nodes physically placed in the same location. Of course, some characteristics may differ slightly among nodes physically located in the same place, nevertheless, if the differences are negligible, then these nodes should belong to the same subset. It may be necessary to define the range of acceptable deviations for each of the network characteristics.

Next, we define a *resource group* as an intersection of a resource type and a resource location. In other words, a group represents resource units of the same characteristics placed in the same location, i.e. computers with the same parameters connected to the same LAN. Let us stress that when taking into account the types of resources existing in Grid environment, it is justified to consider all the resources as renewable ones.

Continuing, each task will be executed on one node of a Grid. Each node belongs to a particular resource type. The processing time of a task depends on the type of resource used to execute this task. In other words, the task may be executed in one of several modes. Modes represent various combinations of resource requirements and processing times.

Now, it is possible to define a *schedule-dependent data transfer* as a transmission of a data file between the output of task i and the input of task j, where $i \rightarrow j$. We call this type of data transfer schedule-dependent because the time of such a transmission depends on which nodes (unknown in advance, and therefore dependent on the schedule) tasks i and j will be processed.

Let G_{kp} be the group defined as the intersection of resource type R_k and resource location L_p. If task i is executed on a node from group G_{kp} and task j is executed on a node from group G_{lr}, then the transmission time s_{pr} of file F_{ij}, the output of task i and the input of task j, is calculated using the following formula:

$$s_{pr} = \left\{ \begin{array}{ll} |F_{ij}| \cdot B_{pr} + \Lambda_{pr} & if\ p \neq r \\ |F_{ij}| \cdot B_{pr} & if\ p = r \\ 0 & if\ i \otimes j \end{array} \right\} \tag{19.1}$$

where $|F_{ij}|$ is the size of file F_{ij}, B_{pr} is the bandwidth between locations L_p and L_r, Λ_{pr} is the sum of constant values (latency, transfer negotiation, etc.) that do not depend on the size of the file but because of its size cannot be neglected, and $i \otimes j$ denotes that task j is processed on the same node as task i.

In Equation 19.1 we assume that if the transfer is carried out between nodes belonging to the same location ($p = r$), then $\Lambda_{pr} \approx 0$, and if both task i and j are executed on the same node, then there is no need to transfer the file. Let C_i be the completion time of task i and S_j the starting time of task j, then the restriction $C_i + s_{pr} \leq S_j$ must be satisfied. It is also usually assumed that transmission time s_{pr} satisfies the triangle inequality $s_{pq} + s_{qr} \geq s_{pr}$ for every triple $p, q, r \in \{1, 2, \ldots, L\}$, where L is the number of resource locations.

Schedule-independent data transfers are usually represented by DAG vertexes without predecessors. This means that a data file to transfer already exists in one or more locations on a Grid. We model the possibility of transferring the file from several locations using the concept of modes. Each mode of such a task corresponds to a resource group from which the data file can be transferred. For each such mode, the duration of the task executed in this mode is the minimum transmission time (s_{pr}) value over all locations where replicas of this data file can be found. In other words, we assume that the data file considered is always transferred from the nearest (in terms of transmission time) location of this file.

The goal is to minimize the makespan, that is, to find an assignment of modes to tasks and, simultaneously, precedence- and resource-feasible starting times of tasks such that the completion time of the finishing task of the workflow job is minimized. We take this objective into consideration because it is most accurate for both the owner of the job, who wants to obtain the results as soon as possible, and the owner of the resources, who wants to maximize the utilization of his resources.

3.3 Notation and 0-1 LP Formulation

Summarizing the considerations from the two previous subsections, we introduce the following notation given in Table 19.3.

Using this notation, the problem of scheduling workflow jobs on a Grid can be formulated as the following 0-1 LP problem:

$$minimize\ C_{max} = \sum_{t=EF_n}^{LF_n} t \cdot x_{n1t}$$

$$subject\ to$$

1. $$\sum_{m=1}^{|M_j|} \sum_{t=EF_j}^{LF_j} x_{jmt} = 1; \qquad j = 1, \ldots, n$$

2. $$\sum_{m=1}^{|M_h|} \sum_{t=EF_h}^{LF_h} t \cdot x_{hmt} \leq \sum_{m=1}^{|M_j|} \sum_{t=EF_j}^{LF_j} (t - p_{jm} - s_{pr}) \cdot x_{jmt};$$
$$j = 1, \ldots, n; h \in Pred_j; h \otimes L_p; j \otimes L_r$$

3. $\sum_{j=1}^{n} \sum_{m=1}^{|M_j|} r_{jkm} \sum_{b=t}^{t+p_{jm}-1} x_{jmb} \leq R^k; \qquad k = 1, \ldots, R; t = 1, \ldots, T$
4. $x_{jmt} \in \{0, 1\}; \qquad j = 1, \ldots n; m \in M_j; t = 1, \ldots, T$

Table 19.3. Notation for our approach.

Symbol	*Definition*
W	workflow job;
V	set of tasks of workflow job W;
$n = \|V\|$	number of tasks of workflow job W;
$i \rightarrow j$	precedence constraint between tasks i and j;
E	set of precedence constraints of the type $i \rightarrow j$ between tasks in workflow job W;
$G(V, E)$	directed AoN graph representing the structure of workflow job W;
$Pred_j$	set of direct predecessors of task j;
$Succ_j$	set of direct successors of task j;
M_j	set of modes of task j;
m_j	execution mode of task j;
p_{jm}	processing time of task j in mode m;
S_j	starting time of task j;
C_j	completion time of task j;
EF_j	earliest completion time of task j;
LF_j	latest completion time of task j;
R	set of renewable resources;
$N = \|R\|$	number of all available resource units (nodes);
R	number of resource types;
R_k	resource type k;
R^k	number of available units of resource type R_k: $\sum_{k=1}^{R} R^k = N$;
r_{jkm}	usage of resource type k by task j executed in mode m;
L	number of resource locations;
L_p	resource location p;
G_{kp}	resource group defined by the intersection of resource type R_k and resource location L_p;
$\|F_{ij}\|$	size of file F_{ij} transferred between tasks i and j;
B_{pr}	bandwidth of the physical link between two resource locations L_p and L_r;
Λ_{pr}	constant time factor of the schedule-dependent data transfer between two resource locations L_p and L_r;
s_{pr}	schedule-dependent transfer time of files between two tasks executed on nodes in resource locations L_p and L_r (setup time);
t	time;
T	time horizon;

where x_{jmt} is a decision variable that is equal to 1 if and only if task j is performed in mode m and finished at time t, and $j \otimes L_p$ denotes that task j is executed on a node in location L_p.

The objective function (denoted as C_{max}) defines the execution time of the workflow job that is equal to the completion time of the finishing task n. This dummy task has only one execution mode. Constraint Set 1 assures that each task is assigned exactly one mode and exactly one completion time. Precedence feasibility is maintained by Constraint Set 2. These constraints make sure that if task h, executed in resource location L_p, is a predecessor of task j, executed in location L_r, then task j must not start before task h is completed and required data files are transmitted from location L_p to location L_r. Constraint Set 3 addresses the resource limitations. And finally, Constraint Set 4 defines the binary status of the decision variables.

Of course, as we have mentioned in the Introduction, a binary linear programming problem can be solved optimally by using specialized solvers, but the complexity of any exact method for solving it is exponential. Therefore, in this research we propose another approach to solve the formulated combinatorial optimization problem – to attack it by local search metaheuristics. They have proved to be very efficient strategies for the classical MRCPSP, being able to find optimal solutions in about 90% instances of problems with up to 30 activities. For more details concerning applications of metaheuristics to the MRCPSP see [Har98] – genetic algorithm, [NI01] – tabu search, and [JMR$^+$01] – simulated annealing.

4. METAHEURISTIC APPROACH

In the previous sections we have described the problem of scheduling workflow jobs on a Grid, and we have formulated its mathematical model as the multi-mode resource-constrained project scheduling problem with schedule-dependent setup times. However, as it has been mentioned in Section 3, the RCPSP and all its extensions (including the MRCPSP-SDST) are strongly NP-hard problems, which makes the process of finding their optimal solutions computationally intractable in a general case. Heuristic approaches have to be developed to solve NP-hard problems. In this section we propose a local search metaheuristic approach to our problem.

4.1 Local Search

Generally, in a combinatorial optimization problem we have a finite set of *feasible solutions*, and a so-called *objective function* (or criterion) that determines the quality of each feasible solution. We look for an *optimal solution*, i.e. a solution that optimizes the criterion over the whole set of feasible solutions. Heuristic algorithms can be divided into two classes: *constructive algorithms*

Iterative improvement method

```
Initialize(x ∈ X);
repeat
    Select(x' ∈ N(x));
    if c(x') < c(x) then x := x';
until ( c(x') ≥ c(x) for all x' ∈ N(x) );
```

Figure 19.1. Iterative improvement method psuedo-code. X is the set of feasible solutions, $c : X \to \Re$ is the objective function, and $N(x)$ is the neighborhood of solution x.

and *local search algorithms*. Constructive algorithms build partial solutions, and stop at the moment of constructing a complete solution.

Local search algorithms use the notion of a *neighborhood*. It is a set of solutions possible to generate from the current solution according to a defined *neighborhood generation mechanism*. These algorithms start from a chosen *initial solution*, and move from one solution to another one by searching successive neighborhoods in order to get a solution as close to optimum as possible. The idea of local search (or neighborhood search) has already been known for about 45 years, one of the earliest references to it is by [Cro58] who used the idea to get good solutions of the traveling salesman problem. Nowadays, the local search strategy is widely used for solving hard combinatorial optimization problems. Comprehensive reviews on local search strategies can be found e.g. in [Ree93] or [AL97]. The simplest local search algorithm is the *iterative improvement method* that is given in Figure 19.1.

4.2 Metaheuristics

Iterative improvement methods may not find an optimal solution since they perform improving moves only so the final solution is a local optimum, which may be different from the global one. In addition, these approaches strongly depend on the starting solution chosen. Therefore, most recent local search methods are designed to escape from the local optima traps. Among them, local search metaheuristics have a particularly important position. A *metaheuristic* [Glo86] is an iterative master process that guides and modifies the operations of subordinate heuristics in order to efficiently produce high-quality solutions. It may manipulate a single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, a simple local search, or just a constructive algorithm. The advantage of metaheuristics is a reasonable computational effort. Even if they do not guarantee finding an optimal solution they have proved to be efficient approaches to many combinatorial optimization problems, and have a great number of successful applications.

Simulated annealing

```
Initialize(k, x, x*, Tp_k, MC_k);
repeat
    for l := 1 to MC_k do begin
        Select_Randomly(x' ∈ N(x));
        ? c := c(x') – c(x);
        if ? c = 0 then x := x'
        else if exp(-? c/Tp_k) = Random then x := x' end;
    if c(x) < c(x*) then x* := x;
    k := k + 1;
    Update(Tp_k, MC_k);
until Stop_Criterion;
```

Figure 19.2. Simulated annealing pseudo-code. The value k is the iteration number, Tp_k is the temperature at k-th iteration, and MC_k is so-called the length of the Markov chains and determines the number of transitions for a given value of the control parameter; x^* is the best solution found by the algorithm.

The family of metaheuristics includes, but is not limited to, adaptive memory procedures, ant systems, evolutionary methods, mimetic algorithms, variable neighborhood search, greedy randomized adaptive search, scatter search, neural networks, and their hybrids. For an extensive survey on metaheuristic strategies see [OK96, RS96]. Three strategies have been most popular and successful from among the metaheuristics over the years: simulated annealing, tabu search, and genetic algorithms. Below we briefly describe the principles underlying these algorithms.

4.2.1 Simulated Annealing

Simulated annealing (SA) is a well-known local search metaheuristic that belongs to a class of the threshold algorithms as presented in [AKvL97], and can be viewed as a special case of the *First Fit Strategy*, where the next solution is accepted with a certain probability. SA is based on the Monte Carlo method introduced by [MRR$^+$53]. This idea was originally used to simulate a physical annealing process, and was applied to combinatorial optimization for the first time in the 1980's, independently by [KGJV83] and [Čer85]. Most adaptations of the SA algorithm use its homogeneous version [vLA87].

SA works by searching the set of all feasible solutions, and reducing the chance of getting stuck in a poor local optimum by allowing moves to inferior solutions under the control of a randomized scheme. Specifically, if a move from solution x to a neighboring inferior solution x' results in a change in the objective function value equal to Δc, then the move is still accepted if $\exp(-\Delta c/Tp) \geq Random$, where Tp (temperature) is a control parameter, and $Random \in [0;1]$ is a uniform random number. The parameter Tp is initially high, allowing many inferior moves to be accepted, and is slowly reduced

Tabu search

```
Initialize(k, x, x*, TabuList);
repeat
    Select_Best(x': c(x') = min{c(y): y ∈ N(x) \ TabuList});
    x := x';
    if c(x) < c(x*) then x* := x;
    k := k + 1;
    Update(TabuList);
until Stop_Criterion;
```

Figure 19.3. Tabu search pseudo-code.

to a value where inferior moves are nearly always rejected. The process of decreasing the value of temperature is called the *cooling scheme*. Determining the initial value of the control parameter as well as the method of decreasing its value are important elements of the cooling scheme. A general form of the SA algorithm in its homogeneous version is presented in Figure 19.2.

4.2.2 Tabu Search

Tabu search (TS) is a metastrategy based on neighborhood search with overcoming local optimality. Unlike simulated annealing, it works in a deterministic way, trying to model human memory processes. Memory is implemented by the implicit recording of previously seen solutions, using simple but effective data structures. This approach focuses on the creation of a tabu list of moves that have been performed in the recent past of the search, and that are forbidden for a certain number of iterations, to help to avoid cycling and to promote a diversified search over the set of feasible solutions. Tabu search was originally developed by [Glo86, Glo89, Glo90], and a comprehensive report of the basic concepts and recent developments was given in [GL97]. Tabu lists can be managed in many different ways. Several tabu list management methods have been considered in the literature, choosing one of them is an important decision when applying the TS algorithm to an optimization problem. At each iteration TS moves to the best solution that is not forbidden, regardless of the fact whether this is an improving move or not. As a result, the algorithm is independent of local optima. A general form of tabu search can be presented in Figure 19.3.

4.2.3 Genetic Algorithms

Genetic algorithms (GA) can also be viewed as a form of local search, although their original inspiration comes from population genetics. Unlike SA and TS, GA use a collection (or population) of solutions from which better solutions are produced using selective breeding and recombination strategies. Simple genetic operators, such as crossover and mutation, are used to con-

Genetic algorithm

Initialize(k, P_k);
Evaluate(P_k);
repeat
  Select_Individuals(P_k);
  Recombinate(P_k);
  Evaluate(P_k);
  $k := k + 1$;
until ***Stop_Criterion***;

Figure 19.4. Genetic algorithm psuedo-code. The value P_k is the population of individuals at k-th iteration, and ***Evaluate*** is a function determining the quality of individuals in the population (usually inversely dependent on their objective function values).

struct new solutions from pieces of old one in such a way that the population steadily improves. The population is then maintained according to the "survival of the fittest" principle. GA operate on a population of individuals, each one representing a feasible solution of the considered problem. The random nature of genetic operators, assuming accidental changes of randomly selected individuals from the current population, sometimes results in the difficulty in maintaining the feasibility of individuals during the evolution process. The initial development of GA concepts was done by [Hol75], and an extensive survey on genetic algorithms is given in [Mic92].

GA, like SA and TS, also requires several parameters to tune. But, like the other metaheuristics, can often be fairly robust in its basic version, with simple recombination operators and a selection method based on well-known strategies like roulette wheel, tournament, or ranking selection. A general form of a GA algorithm can be presented in Figure 19.4.

4.3 Problem-Specific Decisions

All the three algorithms presented in the last subsections can be used with a variety of applications in business, engineering, economics, and science, and are able to produce high-quality solutions in reasonable time horizons for many hard combinatorial optimization problems. In order to apply a metaheuristic algorithm to a given problem, several decisions have to be made. They can be divided into two categories: decisions related to the metaheuristic itself (so-called *generic decisions*), and decisions related to the problem being solved (so-called *problem-specific decisions*). Generic decisions are characteristic for a particular approach used, and have to be considered in connection with this approach. However, problem-specific decisions can be considered separately, without respect to any particular metaheuristic applied.

Problem-specific decisions include:

- Solution representation: the form of representing a feasible solution of the problem;
- Objective function: the way of calculating the objective function value for each feasible solution;
- Neighborhood generation mechanism: the method of generating the neighborhood of the current solution; and
- Starting solution: the way of constructing an initial solution for the search process.

In this section we propose definitions of the above elements for the considered problem of scheduling workflow jobs on a Grid.

A *feasible solution* is represented by two n-element lists, a Task List and a Node List. The *Task List* (TL) is a precedence-feasible permutation of task. Each task is placed on the list after all its predecessors but before all its successors. The *Mode List* (ML) is a list of modes in which tasks will be executed. In other words, the value in position j on this list represents the execution mode m_j of task j. This value determines the resource group in which task j will be executed, as well as its duration in this mode.

Figure 19.5 shows a simple graph G in which all tasks may be executed in one of five possible modes and Task 1 and Task 10 are dummies. An example solution for this graph is is presented in Figure 19.6.

A good *starting solution* can be generated by setting all tasks on ML in an ascending order, and assigning the mode with the shortest duration to each

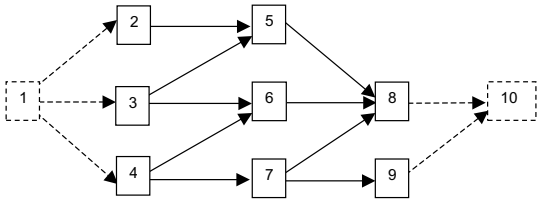


Figure 19.5. An example of graph G representing a workflow job.

1	4	2	3	7	9	6	5	8	10	*Task List*
2	3	4	1	5	5	4	2	2	3	*Mode List*

Figure 19.6. An example of a solution for the job presented in Figure 19.5.

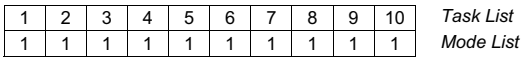


Figure 19.7. A sample starting solution for the job presented in Figure 19.5.

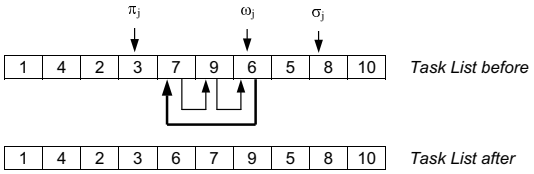


Figure 19.8. An example of a shift operation.

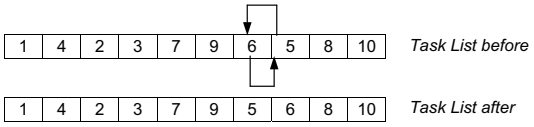


Figure 19.9. An example of a pairwise interchange.

task. Assuming that modes of each task are numbered in an ascending order of duration, the starting solution for the considered example job is presented in Figure 19.7.

A *neighbor* of a current solution can be constructed by a change either on TL or on ML. The simplest change on ML is to choose a random a position on the list and change the corresponding value. The changes on TL can be performed in several ways, including using a shift operator and a pairwise interchange operator among others.

A neighbor generated using the shift operator is constructed as follows. We select ask j in position ω_j on TL. The nearest predecessor π_j of task j from the left hand side, and the nearest successor σ_j of task j from the right hand side of ω_j, are found. Next, task j is moved to certain position ϖ_j between π_j and σ_j, and all tasks between ω_j and ϖ_j are shifted towards position ω_j. An example shift operation is shown in Figure 19.8.

Using the pairwise interchange, a neighbor of the current solution can be obtained by choosing two tasks that are not precedence-related, and interchanging them on TL. An example pairwise interchange operation is presented in Figure 19.9. Of course, also other operators can be used to generate neighboring solutions.

The solution representation described above determines only the execution modes of all tasks and a possible task execution order, but it does not contain any information about the schedule. Therefore, it is necessary to provide another mechanism for calculating starting times of tasks, as well as for mapping resources to tasks. The schedule is constructed using a so-called *decoding rule*. We use the serial *Schedule Generation Scheme* (SGS) [Kel63] in a slightly modified version. It has been shown in [Kol95] that this approach always generates active schedules i.e. schedules where none of the tasks can be started earlier without delaying some other task, and for makespan minimization problems the optimal solution is a member of the set of active schedules, that is, there is always a task list TL for which the serial SGS gives an optimal solution.

The serial SGS for a task list TL can be described as follows. We select the first unscheduled task j from TL, and we schedule it at the earliest possible starting time S_j that does not violate precedence or resource constraints, also considering setup times (times of schedule-dependent data transfers). No task may be started before all its predecessors are completed, and all transmissions of data files from these predecessors are finished. Thus, the starting time S_j of task j executed in location L_r has to fulfill the inequality $C_i + s_{pr} \leq S_j$ for every task $i \in Pred_j$ executed in location L_p.

For every two precedence-related tasks executed in the same resource group with a schedule-dependent data transfer between them, we have to decide whether these tasks should be executed on the same node or not. This decision is vital since if both tasks are performed on the same node, the data transfer time between these tasks is equal to zero, according to Equation 19.1. We propose the following solution of this problem. If task i is executed in group G_{kp}, and its immediate successor j is also executed in G_{kp} and can be executed immediately after the completion of task i, we assume that there is no need to transfer any data between these tasks because they can be executed on the same node. Unfortunately, it is possible that more than one immediate successor of task i will be assigned to the same resource group, and can be executed immediately after the completion of task i. In such a case, the superscheduler must decide which task should be executed on the same node as task i. We propose a simple rule that chooses the task that occurs first on TL from the set of all alternative tasks. The remaining alternative tasks are scheduled in the order in which they occur on TL, either on the same node as task i at the earliest possible starting time S_x (if the transmission of the corresponding file from task i to another node in G_{kp} would be finished after S_x) or otherwise on another node in G_{kp}. In other words, this node from among all nodes in G_{kp} is chosen on that the considered task can be started at the earliest. We represent the information that task j will be executed on a given node or on any other node from G_{kp} by the flag Φ_j, which is set to 0 if the task may be mapped to an

arbitrary node from the considered resource group, and is set to the number of its immediate predecessor if the task has to be executed on the same node as its predecessor. In case there are more than one immediate predecessors for task j executed in the same group G_{kp}, task j will be executed on the same node as task i, such that the value $p_{im} + s_{pp}$ is minimized, where m is the execution mode of task i and $i \rightarrow j$.

Finally, the mapping of resources to tasks is performed according to the rule that a task is scheduled on any node from G_{kp} if $\Phi_j = 0$, or on the same node as task i if $\Phi_j = i$.

The *objective function* is defined as the total execution time of the considered workflow job i.e. the makespan or C_{max}. This value is obtained as a result of the execution of the serial SGS procedure for a given feasible solution.

4.4 Example

An example of a feasible solution, i.e. a pair (TL, ML), and the corresponding schedule obtained by using the described serial SGS for task lists TL is presented in Figure 19.10. The structure of the workflow job used in this example is defined by graph G from Figure 19.5. The feasible solution considered is presented in Figure 19.10 above the schedule. There are two resource types ($R = 2$): R_1 and R_2, and two resource locations ($L = 2$): L_1 and L_2. We assume that resource type R_1 is available in resource location L_1, and resource type R_2 is available in resource location L_2. Resource type R_1 is available in 3 units ($R^1 = 3$), and resource type R_2 in 2 units ($R^2 = 2$). Thus, the total number of available resource units (nodes) $N = 5$. Therefore, there are two resource groups: G_{11} and G_{22}.

The bandwidth within resource location L_1 is equal to 10M/t ($B_{11} = 10$), within resource location L_2 is equal to 5M/t ($B_{22} = 10$), and between these two locations is equal to 2M/t ($B_{12} = B_{21} = 2$), where M is a data unit and t is a time unit. We assume for simplicity that $\Lambda_{12} = \Lambda_{21} = 0$.

All data concerning tasks are presented below in Tables 19.4 and 19.5. In Table 19.4 resource requirements and durations of tasks are given, and in Table 19.5 we present the sizes of files transferred between tasks. In this example we assume for simplicity that the tasks are not parallel computations, i.e. each task j requires in each mode exactly one resource unit: $r_{jkm} = 1$. Table 19.5 is the bandwidth matrix between resource locations L_1 and L_2.

Before presenting the schedule, let us complete the assumptions for the considered example. From our example, Task 2 is a schedule-independent data transfer that is responsible for providing an input data file for Task 5. This file already exists on a Grid, so Task 2 is a data transmission itself and therefore $|F_{25}| = 0$. The minimum time needed to transfer the required file by Task 2 to group G_{11} equals 8, whereas to group G_{22} equals 10 time units. Remember

Table 19.4. Example – tasks.

j	p_{j1}	m_1 r_{j11}	r_{j21}	p_{j2}	m_2 r_{j12}	r_{j22}
1	0	0	0	–	–	–
2	8	0	1	10	1	0
3	10	1	0	15	0	1
4	12	0	1	14	1	0
5	5	1	0	10	0	1
6	4	1	0	9	0	1
7	3	1	0	4	0	1
8	9	0	1	12	1	0
9	6	1	0	8	0	1
10	0	0	0	–	–	–

Table 19.5. Example file transfer times (left) and bandwidth values (right).

i	j	$\|F_{ij}\|$
2	5	0
3	5	16M
3	6	10M
4	6	30M
4	7	25M
5	8	25M
6	8	30M
7	8	60M
7	9	20M

	L_1	L_2
L_1	10M/t	2M/t
L_2	2M/t	5M/t

that the file transmitted by Task 2 is saved on the same node as Task 2 is executed. Tasks 3 and 4 do not need any input files, but they both generate some output files. The remaining tasks both need input files and generate output files, however, files generated by Tasks 8 and 9 are not required by any other tasks, the data is just saved on the nodes that executed these tasks. Using these assumptions for the example, we obtain a schedule presented in Figure 19.10.

5. SUMMARY

In this chapter we address the problem of scheduling workflow jobs on a Grid. The problem consists of assigning Grid resources to tasks of a workflow application across multiple administrative domains in such a way that the time of execution of a particular set of tasks is minimized.

In this work we formulate the problem as an extension of the resource-constrained project scheduling problem (RCPSP) to minimize the makespan.

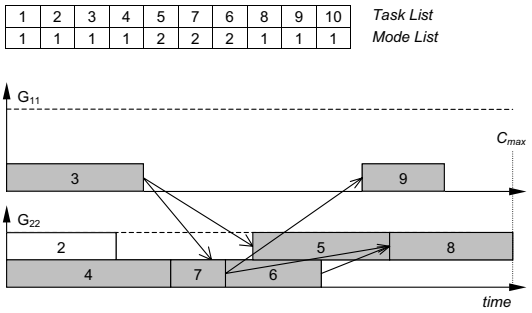


Figure 19.10. A feasible solution and corresponding schedule for the the considered example. Task 2 is not shaded since it is a schedule-independent data transfer, not a computational task. Arrows represent schedule-dependent data transfers (or setups).

The multi-mode version of this problem with schedule-dependent setup times has been used. The problem of scheduling workflow jobs on a Grid has been defined as the MRCPSP-SDST, and formulated as a 0-1 LP problem. We propose using a local search metaheuristic approach to solve this combinatorial optimization problem. We present ways to address the problem specific decisions (solution representation, objective function, neighborhood, starting solution) required for a local search algorithm.

Our model is purely deterministic, and we assume all parameters are fixed and known a priori. This may not be possible in real Grid environments where some information can be temporal or uncertain. Therefore, future directions may include developing other approaches to the problem such as stochastic techniques, using fuzzy or rough set based approaches, using simulation, or involving learning systems. Another approach to this problem would be from the sensitivity and robustness point of view, which would look for schedules that may be not that high-quality in terms of the execution time, but are more robust and breakdown resistant. Also other criteria can be considered, for example cost, reliability, resource leveling, etc., as well as a multi-objective approach. Our current approach is being implemented within the GridLab project [AAG+02]. Examining the approach in a real Grid environment will be the ultimate verification of its efficiency.

Acknowledgments

This research has been supported by the Polish State Committee for Scientific Research (KBN) grant no. KBN 4T11F 001 22. We express our gratitude to Jarek Nabrzyski and Krzysiek Kurowski for their valuable help and fruitful discussions.

V

DATA-CENTRIC APPROACHES FOR GRID RESOURCE MANAGEMENT

Chapter 20

STORAGE RESOURCE MANAGERS

Essential Components for the Grid

Arie Shoshani, Alexander Sim, and Junmin Gu
Lawrence Berkeley National Laboratory

Abstract

Storage Resource Managers (SRMs) are middleware components whose function is to provide dynamic space allocation and file management of shared storage components on the Grid. They complement Compute Resource Managers and Network Resource Managers in providing storage reservation and dynamic information on storage availability for the planning and execution of a Grid job. SRMs manage two types of resources: space and files. When managing space, SRMs negotiate space allocation with the requesting client, and/or assign default space quotas. When managing files, SRMs allocate space for files, invoke file transfer services to move files into the space, pin files for a certain lifetime, release files upon the clients request, and use file replacement policies to optimize the use of the shared space. SRMs can be designed to provide effective sharing of files, by monitoring the activity of shared files, and make dynamic decisions on which files to replace when space is needed. In addition, SRMs perform automatic garbage collection of unused files by removing selected files whose lifetime has expired when space is needed. In this chapter we discuss the design considerations for SRMs, their functionality, and their interfaces. We demonstrate the use of SRMs with several examples of real implementations that are in use today in a routine fashion or in a prototype form.

1. INTRODUCTION

The grand vision of the Grid is to provide middleware services that give a client of the Grid the illusion that all the compute and storage resources needed for their jobs are running on their local system. This implies that the client only logs in and gets authenticated once, and that the middleware software figures out what is the most efficient way to run the job, reserves compute, network, and storage resources, and executes the request. Initially, the Grid

was envisioned as a way to share large compute facilities, sending jobs to be executed at remote computational sites. For this reason, the Grid was referred to as a *Computational Grid*. However, very large jobs are often data intensive, and in such cases it may be necessary to move the job to where the data sites are in order to achieve better efficiency. Thus, the term *Data Grid* was used to emphasize applications that produce and consume large volumes of data. In some applications, the volume of data is so large (in the order of hundreds of gigabytes to terabytes) that partial replication of the data is performed ahead of time to sites where the computation is expected to take place.

In reality, most large jobs that require Grid services, especially in the scientific domain, involve the generation of large datasets, the consumption of large datasets, or both. Whether one refers to the Grid as a Computational Grid or a Data Grid, one needs to deal with the reservation and scheduling of storage resources when large volumes of data are involved, similar to the reservation and scheduling of compute and network resources.

In addition to storage resources, Storage Resource Managers (SRMs) also need to be concerned with the *data resource* (or files that hold the data). A data resource is a chunk of data that can be shared by more than one client. For the sake of simplifying the discussion, we assume that the granularity of the data resource is a file. In some applications there may be hundreds of clients interested in the same subset of files when they perform data analysis. Thus, the management of shared files on a shared storage resource is also an important aspect of SRMs. In particular, when a file has to be stored in the storage resource that an SRM manages, the SRM needs to allocate space for the file, and if necessary remove another file (or files) to make space for it. Thus, the SRM manages both the space and the content of that space. The decision of which files to keep in the storage resource is dependent on the cost of bringing the file from some remote system, the size of the file, and the usage level of that file. The role of the SRM is to manage the space under its control in a way that is most cost beneficial to the community of clients it serves.

In general, an SRM can be defined as a middleware component that manages the dynamic use of a storage resource on the Grid. This means that space can be allocated dynamically to a client, and that the decision of which files to keep in the storage space is controlled dynamically by the SRM.

The initial concepts of SRMs were introduced in [SSG02]. In this chapter we expand on the concepts and functionality of SRMs. As described in [SSG02], the concept of a storage resource is flexible; an SRM could be managing a disk cache (we refer to this as a Disk Resource Manager - DRM), or managing a tape archiving system (Tape Resource Manager - TRM), or a combination of both, called a Hierarchical Resource Manager (HRM). Further, an SRM at a certain site can manage multiple storage resources, thus have the flexibility to be able to store each file at any of several physical stor-

age systems it manages or even to replicate the files in several storage systems at that site. The SRMs do not perform file transfers, but can invoke middleware components that perform file transfers, such as GridFTP [ABB+02b].

All SRMs have the same uniform interface, thus allowing any current or future hardware configurations to be addressed in the same manner. For example, an archive does not have to be a robotic tape system; a disk system could serve as archival storage as well. As far as the client is concerned, getting a file into an SRM-managed space may incur a delay because the file is being retrieved from a tape system or from a remote site over the network, or both.

This chapter is structured as follows: Section 2 overviews the basic functionality of SRMS by walking through several related scenarios. The concepts of permanent, volatile and durable files are defined in Section 3. Section 4 describes how to manage space reservations, and Section 5 details the concepts of negotiations, site and transfer URLs, and the semantics of file pinning. Section 6 describes several projects currently using SRMs in practice. We conclude in Section 7.

2. THE FUNCTIONALITY OF STORAGE RESOURCE MANAGERS

2.1 Example Scenarios

To illustrate the functionality that SRM services expose to a client, let us consider a series of related scenarios. Each will illustrate increasingly more complex functionality required of the SRMs.

2.1.1 Local Data Consumption by a Single Client

Consider a simple scenario where a set of requested files are to be moved to the clients system. Let's assume that the client needs files from several locations on the Grid, that the client knows exactly the site of each file (the machine name and port), and the physical location of the file (i.e. directory and file name). In this case, the client can go directly to each location and get each file. In this case, the client needs to monitor the file transfers to make sure that each is completed successfully, and to recover from failures.

Now, suppose that the client wants to get 500 files of 1 GB each, but the amount of available space on the local disk is only 50 GBs. Suppose further that the client can process each file independently of the other files, and therefore it is possible to bring in files incrementally. The client has the burden of bringing in the files, monitoring the file transfers, recovering from failures, keeping track of which files were brought in, removing each file after it is processed to make space for additional files, and keeping track of available space.

An SRM can be used to alleviate this burden of dealing with a large number of files by providing a service where the client issues a single request for all 500 files. The SRM deals with dynamic space allocation, removal of files, and recovering from transient failures. This is illustrated schematically in Figure 20.1. Note that the local storage system is a disk, and that the two remote systems shown are a disk, and a mass storage system that includes a robotic tape. The solid line from the client to DRM represents a multi-file request. The dashed line from the client to the disk represents file access (open, read, write, close, copy, etc.).

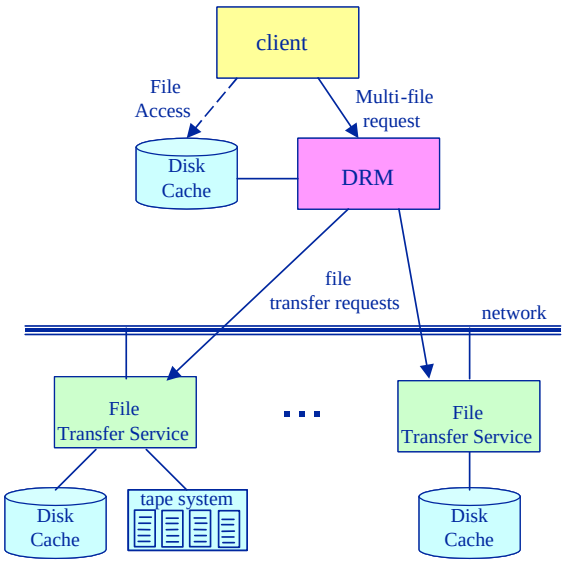


Figure 20.1. A multi-file request service by an SRM.

Each file in the multi-file request can be represented as a Uniform Resource Identifier (URI) given that the protocol to get the file is known (such as GridFTP, ftp [PR85], http [FGM+99], etc.). For example, `gridftp://cs.berkeley.edu:4004/tmp/fileX` represents a file `fileX` in the directory `tmp` of the machine `cs.Berkeley.edu` and the `gridftp` transfer protocol that uses port 4004. In this manner, the multi-file request can be composed of files on different systems.

The functionality provided by the SRM to support such a multi-file request includes: (a) queuing the set of files requested, (b) keeping track of the files in the disk cache, (c) allocating space for each file to be brought in, (d) if the requested file is already in cache because of a previous use, mark it as needed, so it is not removed by the SRM, (e) if the file is not in cache, invoke a file transfer service to get the file.

2.1.2 Shared Local Data Consumption by Multiple Clients

Figure 20.2 shows a similar situation to Figure 20.1, except that multiple clients share the local disk. This situation is not unusual, especially for applications that need to process a large amount of data and it is prohibitive to provide each client with a dedicated large disk. Another advantage of sharing a disk cache is that files accessed by one client can be shared by other clients. In this case, the file does not have to be brought in from the remote site again, saving time to the clients and unnecessary traffic on the network. Of course, this can be done for read-only files or when the master file has not been updated since the last read.

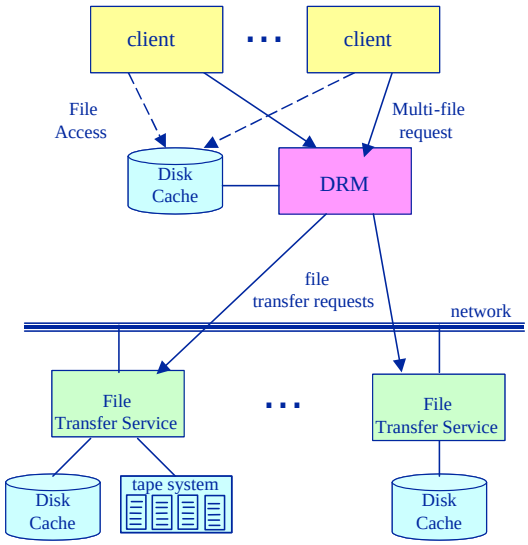


Figure 20.2. Sharing a DRM-managed disk cache.

Even with this simple scenario, one can already notice an important concept emerging: The concept of *file pinning*. When a file is brought into the disk cache, there is no guarantee that it will stay there, because the space may be needed by other clients. Thus, in order to keep the file in cache, it is necessary for the SRM to *pin* the file. It is also necessary to provide the client with a release pin call so that the SRM can unpin the file. Another related concept that needs to be supported is the concept of a *lifetime of a pin*. This is necessary in case that a client neglects to *release* (or *unpin*) the file. This can happen if the clients program crashes, goes into a loop, or simply terminates without releasing the file. The lifetime of a file is a necessary feature for automatic garbage collection of unused files.

The above concepts are meaningful when dealing with shared temporary space, which we refer to as *volatile* space. We use the term volatile rather than temporary because there is a guarantee that files will not be removed so long as they are pinned. As we'll see in the next sub-section, other types of files and spaces need to be supported as well.

There are several implications to the functionality described above. First, in order to support multi-file requests for multiple users it is necessary to introduce the concept of a *quota* per user. The management of the quota depends on the policy of the specific SRM. It could be fixed, or change dynamically based on the number of clients at the time a request is made. Second, a *service policy* must be supported. A good service policy will make sure that all requests are serviced in a fair manner, so that no one request is starved. Third, since files have to be removed when space is needed, a *replacement policy* has to be provided, whose function is to determine which file (or files) to evict in order to make space for a file that needs to be brought in. A good replacement policy would maximize the use of shared files. There are many papers discussing good replacement policies in the domain of web caching [AY97, CI97, DFJ+96, Gue99] but these policies do not take into account the cost of bringing in the file if the SRM needs to get it again. Recent work has shown that different policies need to be used when considering large files over a Grid [OOS02].

2.1.3 Pinning Remote Files

When a file is needed from a remote site, there is no guarantee that the file will be there when the file transfer starts, or whether the file is deleted or modified in the middle of the file transfer. Normally, archived files are not removed and this issue does not come up. However, on the Grid there may be replicas that are stored temporarily in shared disk caches that may be removed at any time. For example, one model of the Grid is to have a tier architecture where the long term archive is a Tier 0 site, a shared disk cache may exist in a Tier 1 regional site (such as Southern Europe), a shared disk may exist at a Tier 2 local site (such as the Lyon area), and a Tier 3 disk cache may exist at some university. The tier architecture is designed so that files are always accessed from the closest tier if possible. However, over time the content of the intermediary shared disk caches change depending on the access patterns. Therefore, it is possible to have a file removed just before transfer is requested or while the transfer is in progress.

To avoid this possibility, and also to make sure that files are not removed prematurely, one can use the same pinning concept for remote files as well. Thus, remote sites that have SRMs managing their storage resources can be requested to pin a file. This is shown in Figure 20.3, where the remote sites have SRMs associated with them.

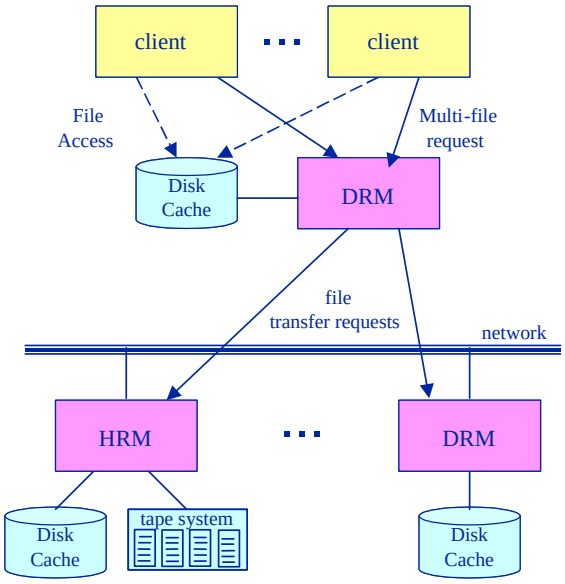


Figure 20.3. Accessing remote SRMs.

For every file that needs to be brought into a local site, the following actions take place: 1) the local SRM allocates space, 2) the local SRM requests that the remote SRM pins the file, 3) the remote SRM acknowledges that the file was pinned, and the physical location of the file is returned; this is referred to as the *transfer URL*, 4) the local SRM invokes the file transfer service, and 5) upon successful completion of the transfer, the local SRM notifies the remote SRM to release the pin.

2.2 Where do SRMs Fit in a Grid Architecture?

Running jobs on the Grid requires the coordination of multiple middleware components. It is convenient to think of these components as layered. One popular view of layering of the Grid services is described in [FKT01]. There are five layers, labeled: fabric, connectivity, resource, collective, and application. Typically, an application at the top layer makes a request for running a job to the *request manager*, a component at the *collective* layer. The request manager may include a component called a *request planner* that figures out the best way to run the job by consulting metadata catalogs, file replica catalogs, monitoring information (such as the Network Weather Service [WSH99a]), etc. The plan, which can be represented as a workflow graph (referred to as *execution DAG* [DAG] by the Condor project [LL90] described in Chapter 9) can

then be handed to a *request executer* that will then contact compute resource managers and storage resource managers to allocate resources for executing the job. Figure 20.4 shows the view where compute and storage resources can be anywhere on the Grid, and the results returned to the clients site.

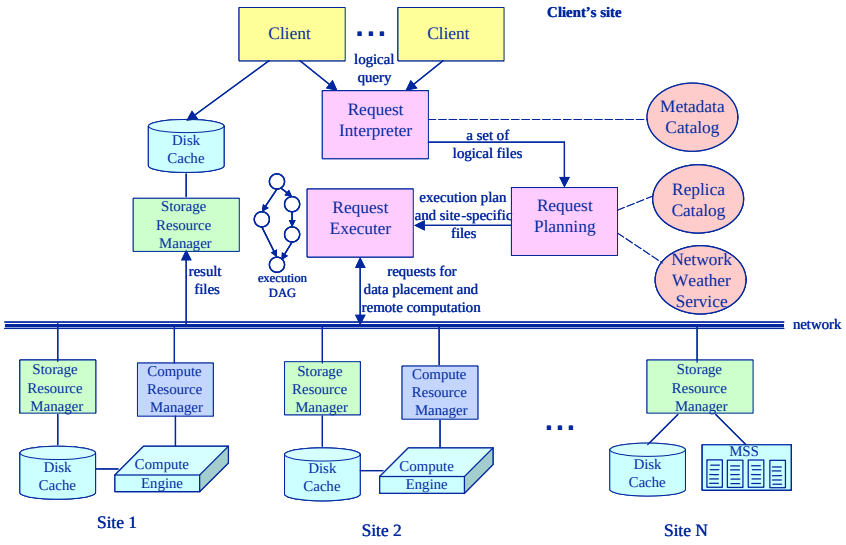


Figure 20.4. Interaction of SRMs with the request executer.

According to this view, SRMs belong in the *resource* layer, in that a component, such as a *request executer* would rely on the SRMs to perform space allocation and file management when requested to do so.

Figure 20.4 introduces another requirement that SRMs need to support. They can be invoked not only by clients or client applications, but also by other middleware software. This implies that SRMs should be able to report information on how busy they are (in case that they have many files to serve on their queue), how much free space they have, and what is the quota policy they support. Similarly, they should be able to provide information on whether they still have specific files in the managed storage resource.

Considering SRMs as belonging to the *storage layer* is not accurate, since they do provide a limited service of *brokering*, a function usually considered at the *collective* layer. Recall that the SRM can be handed a set of files to be accessed, each with a source URL. If the file is found locally, either because it was placed there permanently or because it was brought in by a previous request, the SRM simply returns the *location* of the file. If it is not available locally, the SRM will contact the source location to pin the file, and then invoke a transfer service to get the file. This last action is a brokering action, which is

quite useful to clients. Thus, an SRM as we described it here, can be thought of as being a *local SRM* that provides a *brokering* capability of getting files from remote sites.

The general function of determining the site or sites where to reserve spaces and to move data into the spaces, is referred to as *storage scheduling* and *data placement* services, which are indeed at the collective layer. These should not be confused with SRMs. Rather, their task is to negotiate space reservation, and schedule data movement by requesting such services from multiple SRMs.

3. TYPES OF FILES

The concepts of *permanent* and *temporary* spaces are supported by most shared file systems. A permanent space is a space that a user controls, and only that user can put in and remove files from that space. A temporary space is a shared space that is allocated to a user, but can be reclaimed by the file system. If space is reclaimed, all the files in that space are removed by the file system. The implication is that files in these spaces are also permanent and temporary.

3.1 Permanent and Volatile Files

On the Grid, the same concepts of permanent and temporary can be applied to file types, but temporary files require additional functionality. Temporary files on shared Grid spaces cannot be removed arbitrarily. Some minimal amount of time must be guaranteed by the SRM for the client to rely on. This is the reason for a lifetime of a file. This feature of a lifetime for a file is associated with each user accessing the file. That is, a lifetime is associated with a file for a user when the file is made available to the user. If another user requests the same file later, the file gets a fresh lifetime associated with that user. We refer to a file that is temporary in nature, but has a lifetime guarantee as a *volatile* file. Since volatile files can be shared by multiple users at the discretion of the SRM, volatile files can be thought of as owned by the SRM, and permission to use them by the users is granted by the SRM on a temporary basis enforced by the lifetime. Alternatively, an SRM can choose to replicate files for each user, although this approach is less space efficient. The concept of volatile files is very useful for sharing space, automatic garbage collection, and sharing of files on a temporary basis. Most shared disk caches on the Grid are likely to provide support only for volatile files.

In contrast, a *permanent file* is a file associated with long-term archival storage that may or may not be shared by clients. Similar to file systems, permanent files can only be removed by the owner.

3.2 Durable Files

For Grid applications, one needs another type of a file that has properties of both permanent and volatile files. A *durable* file has the behavior of a volatile file in that it has a lifetime associated with it, but also the behavior of a permanent file in that when the lifetime expires the file is not automatically eligible for removal. Instead, the SRM may advise the client or an administrator that the lifetime expired or take some other action. For example, the SRM may move the file to a permanent space, release the durable space, and notify the client. A durable file type is necessary in order to provide temporary space for files generated by large simulations, which need to be eventually archived. Since the archiving process (for example, into a mass storage system) is usually slower than the data generation process, attempting to move files directly to the archive will slow down and waste computational resources. By storing durable files into available shared disk caches, the simulation can continue efficiently, yet it is guaranteed that the files will not be removed before they are moved to the archive as a secondary task. This is a more reasonable approach for sharing temporary space, while still protecting important files. Similar to volatile files, durable files can be released by the client as soon as the files have been moved to a permanent location.

4. MANAGING SPACE RESERVATIONS

Space reservation is a necessary feature of SRMs since reservations are needed in order to support the request planning and request execution steps of running jobs on the Grid. Space reservation is also needed in order to replicate data to locations where computations take place, or to have space to store the results of computations. However, space reservation brings up many difficult issues. What policies to use when space is requested? Is the space guaranteed? Should the SRM set aside the space requested? What if the space is not used for a long time?

The answer to these questions depends on the cost model used. To support space reservations, there needs to be a way to provide a *ticket* or a *capability* to the user in order to claim the space. There needs to be an authority that manages this capability. There needs to be a mechanism for reporting the space-time consumed. And there needs to be a way of taking away the space if it exceeds the reservation period.

Dynamic space reservation is not a concept generally supported by file systems, but is essential for supporting shared storage resources on the Grid. For example, in the Unix system, space is allocated by the administrator (the root owner) on a long-term basis. A user cannot request additional space dynamically, or release space dynamically. SRMs are designed to provide dynamic space allocation and release.

4.1 Types of Spaces

In order to support space reservations, we found it useful to apply the concepts of permanent, durable, and volatile to spaces as well. The semantics are similar. Permanent space is owned by the user and has an indefinite lifetime. Durable space is also owned by the user, but when a lifetime expires there are two cases to consider. If the durable space does not contain any files, the space is released. If it contains durable files, the system claims all the unused space, and notifies the file owner that the lifetime of the durable files has expired. Volatile space also has a lifetime, but when it expires, all the space and the files in it can be removed by the SRM.

Reserving permanent space is obviously needed for dynamic archival storage reservation. Similarly, reserving volatile space is necessary for shared disk resources that are intended to be used on a temporary basis. However, support for reservation of durable space requires an explanation as to its value.

Recall that durable files are files that have a lifetime associated with them, but the SRM cannot remove them without some explicit action. Durable files are especially useful when a client generates a large number of files that need to be stored temporarily in some shared disk cache before being archived. The purpose of durable space is similar: reserving a guaranteed space on a temporary basis that cannot be removed without an explicit action by the client. Similar to volatile space, durable space can be released by the client. Durable space is particularly useful for request planning.

In contrast to durable space, which is a space that cannot be removed without an explicit action, volatile space can be reclaimed by the SRM if space is needed. For example, suppose that a client asks for 200 GB volatile space when there are very few clients using the SRM. The SRM can initially allocate this space, but when many new clients request space the SRM can reduce the space used by the first client to say, 50 GBs, provided that there are no pinned files in that space.

4.2 Best Effort Space

The concept of *best effort* is well known in association with reservations [BS98]. It stems from the wish to use the resources efficiently, even if there is a way to charge for the space. For example, suppose that an SRM controls some space, and that a client asks and is granted a space reservation of 200 GBs for 5 hours. If this space is reserved and not used by the client, and if as a result some other clients were denied, then the space was wasted, and the client was charged for space that was not used. This can be thought of as fair, but it may not be the clients fault. It could be that another resource that was scheduled (such as a compute resource) was not available. A good request

planner would have released the space, but this cannot be depended on. The result is that the allocation was lost, and the space was wasted.

Best effort is an alternative that requires some flexibility on the part of the client, but makes more efficient use of resources. The reservation is considered *advisory*. The SRM will try to honor the request within the reservation time window. Space is only allocated as it is claimed, i.e. when files move into the space. If competition is high, the client may not get all the space requested, and may need to look for additional space elsewhere dynamically. This is difficult for human beings to manage, but may be perfectly reasonable to expect from a request planner since in general it needs to deal with various failure modes on the Grid (such as a temporary network partition, archive temporarily down, unscheduled maintenance, etc.)

Best effort can be flexible in that some minimum is guaranteed. This minimum can vary with the type of space. For example, a durable space should have priority in honoring the reservation over volatile space. Here again, the policy of best effort reservation is a local policy for each SRM.

The above discussion implies that in order for a request planner to optimize resource usage, it needs to find out the policies provided by various SRMs it might want to use. The general policies can be advertised, but the SRMs need to provide dynamic information as well, such as the quota currently available in the case that a quota policy is variable. These issues are quite complex, and so far there are no standard methods for advertising and requesting policy and quota information.

4.3 Assignment of Files to Spaces

Can a volatile file reside in permanent space? At first glance, one would consider a one-to-one mapping between file types and the space types they are assigned to. However, it is actually useful to support volatile files in permanent space, for example. If a client has the right to get permanent space, it is reasonable for the client to use it for volatile files, so that space can be automatically reclaimed if it is needed. If we rank volatile, durable, and permanent files types from low to high rank, then a file of a particular type should be able to be stored in a higher ranking space type. We note, however, that such a policy of assigning files to spaces is a choice of the local SRM policies. In general, such policies may be used for more efficient space utilization, but it is harder to manage. For this reason, most SRMs will most likely assign file types to spaces of the same type.

There is a condition that needs to be enforced when assigning files to spaces: no file can be pinned for a lifetime longer than the lifetime of the space it is put into. This condition guarantees that when the space lifetime expires, the lifetime of all the files in that space expired too.

5. OTHER IMPORTANT SRM DESIGN CONCEPTS

In this section we discuss additional items that were added to the design of SRMs as a result of experience of their development by several groups.

5.1 Transfer Protocol Negotiation

When making a request to an SRM, the client needs to end up with a protocol that the storage system supports for the transfer of files. In general, systems may be able to support multiple protocols and clients may be able to use multiple protocols depending on the system they are running on. While it is advantageous to select a single standard transfer protocol that each SRM should support (such as GridFTP), this approach is too restrictive. There could be some university researcher without access to GridFTP who wishes to access files through regular FTP, or there could be another FTP service that some community prefers to use. There needs to be a way to match the transfer protocol that a client wishes to use with protocols supported by the SRMs storage system. This is referred to as *protocol negotiation*.

The mechanism to support protocol negotiation is to allow clients to specify an ordered list of protocols in their requests, and let the SRM respond with the protocol that matches the highest possible ranking. For example, suppose that the client wants to use a new FTP service called FastFTP. The client can submit in the request the ordered list: FastFTP, GridFTP, FTP. An SRM whose storage system has a FastFTP service will respond with Fast FTP, otherwise it will respond with GridFTP, or with FTP if it does not have a GridFTP service. In this way all SRM sites that support FastFTP will be able to use it. This mechanism will allow a community to adopt their preferred transfer protocol, and to introduce new protocol services over time without modifying the SRMs.

5.2 Other Negotiations and Behavior Advertising

In general, it should be possible to negotiate any meaningful quantities that may affect the behavior of SRMs. In particular, it should be possible to negotiate the lifetime of spaces and files, and the space-time quantities of reservations. It may also be possible to negotiate how many simultaneous requests the same user may have. In all such cases, it is up to each SRM what policies to use, and how to respond to negotiable requests. An SRM may simply choose to always respond with the same fixed lifetime period, for example.

SRMs can also choose what type of spaces they support. For example, SRMs that manage shared disk caches may not be designed to support permanent spaces. Other SRMs may not support durable space, only volatile space. SRMs may also choose to give one type of service to certain user groups, but not to others. Furthermore, SRMs should be able to change these choices dy-

namically. To accommodate this dynamic behavior, it is necessary to have a way of advertising the SRMs capabilities, policies, and dynamics loads. Another approach is the passive approach, where SRMs respond to inquiries about their capabilities, dynamic resource capacity available, and dynamic load of the SRM (i.e. how much work they still have in their queue). The passive approach is preferable since the SRMs do not need to find out and coordinate where to advertise except when they are first added to the Grid.

5.3 Source URLs, Transfer URLs, and Site URLs

In previous sections we represented to the location of a file on the Grid as a URL. The example used for such a URL was `gridftp://cs.berkeley.edu:4004/tmp/fileX`. In this example, we refer to `gridftp` as the protocol, and `cs.berkeley.edu:4004/tmp/fileX` as the *Physical File Name* (PFN). Since on the Grid there may be many replicas of a file, each will have a different PFN. This necessitates a unique file name that is location and machine independent. This is referred to as a *Logical File Name*(LFN). A middleware component, such as the Globus Toolkit *Replica Catalog* [SSA+02] or the *Replication Location Service* (RLS) [CDF+02] can provide a mapping of the LFNs to one or more PFNs. SRMs are not concerned with LFNs, since it is the task of the client or the Request Planner to consult with the RLS ahead of time and to make the choice of the desired PFN.

Because SRMs support protocol negotiation, it is not necessary to specify a specific protocol as part of the source URL. Thus, a file that is controlled by an SRM would have `srm` instead of a specific protocol. For the example above, the source URL will be `srm://cs.berkeley.edu:4004/tmp/fileX`. Assume that the preferred transfer protocol in the request is GridFTP, and that it is supported at `cs.berkeley.edu`, then the transfer URL the SRM will return will be: `gridftp://cs.berkeley.edu:4004/tmp/fileX`.

The transfer URL returned to the client does not have to have the same PFN that was provided to the SRM. This is the case when the client wants the SRM to get a file from a remote site. For example, suppose that the source URL `srm://cs.berkeley.edu:4004/tmp/fileX` was provided to an SRM with the host address `dm.fnal.gov:4001`. That SRM will act as a broker, requesting the file from the SRM at `cs.berkeley.edu:4004`. It may choose to place the file in a local directory `/home/cs/newfiles/`, keep the same file name `fileX`, and select the protocol `gridftp`. In this case, after the file is transferred to the SRM at `dm.fnal.gov:4001`, the transfer URL that will be returned will be `gridftp://dm.fnal.gov:4001/home/cs/newfiles/fileX`.

Another important case where the PFN in the transfer URL may be different from the PFN in the source URL is that an SRM at some site may be controlling multiple physical resources, and may want to move the files dynamically between these resources. Such an SRM maintains a mapping between the external PFN exposed to the outside world and the internal PFN of the file depending on which storage component it is stored on. Furthermore, the SRM may choose to keep the same file in multiple storage components, each having its own internal PFN. Note, that the external PFN is not really a physical file name, but a file name maintained by the site SRM. We refer to the URL that contains this external PFN as the *site URL*.

5.4 On the Semantics of the Pinning Concept

In database systems the concept of *locking* is commonly used to coordinate the content of objects in the database (records, disk blocks, etc.) when they are subject to updates. Furthermore, the concept of locking is tightly coupled with the concept of a transaction in that locking is used as a means of ensuring that the entire transaction that may involve multiple objects completes correctly. This led to the well-known theory of serializability and concurrency control [BHG87], and to the widely used two-phase locking algorithm [GR94]. How does the concept of pinning differ from locking? Below, we will limit our discussion to files, without loss of generality. The same concepts apply to any granularity of data, but on the Grid most applications deal with files.

The concept of pinning is orthogonal to locking. While locking is associated with the *content* of a file to coordinate reading and writing, pinning is associated with the *location* of the file. Pinning a file is a way of keeping the file in place, not locking its content. Both locks and pins have to be released. Releasing a lock implies that its content can now be read. Releasing a pin means that the file can now be removed.

The concept of a lifetime is important to both locks and pins. A lifetime on a lock is a way of ensuring that the lock is not unreasonably long, making that file unavailable to others. A lifetime on a pin is a way of ensuring that the file does not occupy space beyond the time necessary for it to be accessed. Pinning is used mainly to manage space allocated to the pinned files. It is necessary for garbage collection of un-released files, cleaning up, or releasing space for reuse.

The length of a lifetime is dependent on the application. Locking lifetimes are usually short (measured in seconds), since we do not want to make files unavailable for a long time. Pinning lifetimes can be made long (measured in minutes) since pinned read-only files can be shared. The only penalty for a long pin lifetime is that the space of the pinned file may not be available for that lifetime period. Releasing a pin as soon as possible ensures the efficient

use of resources, and should be used by any middleware or applications using SRMs.

In many scientific applications, the order that the files are retrieved for processing may not be important. As an example, suppose that client A on site X needs 60 files (1 GB each) from site Y, and a client B on site Y needs 60 files (1 GB each) from site X. Suppose that each site has 100 GBs altogether. Now, for each request the 60 files have to be pinned first. If the pins happened simultaneously, each system has only 40 GBs available, no additional space can be released. Thus, the two processes will wait for each other forever. This is a *pin-lock*.

There are many elaborate schemes to deal with deadlocks, ranging from coordination between processes to avoid deadlock, to optimistic solutions that assume that most of the time deadlocks do not occur and that it is only necessary to detect a deadlock and preempt one of the processes involved. For SRMs, the lifetime on a file should be a natural mechanism for preemption. However, since we also have durable and permanent file types, it is worth incorporating some mechanism to prevent pin-locks. Assuming that pin-locks are rare, a mechanism similar to two-phase locking is sufficient, which we may call *two-phase pinning*. It simply states that all spaces and pins must be acquired first before any transfers take place. Otherwise, all pins and spaces must be relinquished, and the process can be tried again. It is unlikely, but possible that this will lead to thrashing. If this is the case, then there must be additional coordination between processes.

6. SOME EXAMPLES OF SRM IMPLEMENTATION AND USE

In this section we describe several efforts that have already shown the value of SRMs.

6.1 Using SRMs to Access Mass Storage Systems

SRMs have been developed at four laboratories as components that facilitate Grid access to several different Mass Storage Systems (MSSs). At the Thomas Jefferson National Accelerator Facility (TJNAF) an SRM was developed to provide Grid access to the JASMine MSS [BHK01]. At Fermi National Accelerator Laboratory (Fermilab) an SRM was developed as part of the Enstore MSS [BBH+99]. At Lawrence Berkeley National Laboratory (LBNL), an SRM was build to function as an independent component in front of HPSS [HPS]. At CERN, a prototype SRM was recently developed for the CASTOR system [CAS]. All these systems use the same interface using the WSDL and SOAP web service technology. The interoperability of these SRMs

was demonstrated to allow files to be accessed through the same interface over the Grid.

The above experience showed the flexibility of the SRM concepts. In the case of JASMine and Enstore, it was possible to develop SRMs as part of the MSSs because the developers are either the developers of the MSSs as was the case at TJNAF, or the developers had access to the source code, as was the case at Fermilab. The reason that the SRM-HPSS was developed as an independent component in front of HPSS was that HPSS is a product supported by a commercial vendor, IBM. It would have been impossible to develop an SRM as part of HPSS without negotiating with the commercial vendor. Instead, the SRM was build as an independent component that controls its own disk.

6.2 Robust File Replication Service Using SRMs

File replication of thousands of files is an extremely important task in data intensive scientific applications. For example, large Climate Modeling simulations [CGD] may be computed in one facility, but the results need to be stored in an archive in another facility. This mundane, seemingly simple task is extremely time consuming and prone to mistakes. Moving the files by writing scripts requires the scientist to monitor for failures, to have procedures for checking that the files arrived at their destination correctly, and to recover from transient failures, such as a refusal by an MSS to stage a file.

This task is particularly problematic when the files have to be replicated from one MSS to another. For each file, three conditions have to occur properly: staging of the file in the source MSS, transferring the file over the network, and archiving the file at the target MSS. When staging files, it is not possible to ask for all files to be staged at once. If too many staging requests are made, the MSS will refuse most of the requests. The scientist's script have to monitor for refusals and any other error messages that may be issued by the MSS. Also, if the MSS is temporarily out of order, the scripts have to be restarted. Similar problems can occur on the target MSS in the process of archiving files. In addition, since we are transferring hundreds of large files (order of GBs each), it is possible that network failures occur while transfers are taking place. There is a need to recover from these as well. Replicating hundreds of large files is a process that can take many hours. Providing a service that can support massive file replication in a robust fashion is a challenge.

We realized that SRMs are perfectly suited to perform massive file replication automatically. The SRMs queue the multi-file request, allocate space, monitor the staging, transfer, and archiving of files, and recover from transient failures. Only a single command is necessary to request the multi-file transfer. Figure 20.5 shows the setup of having SRMs at NCAR and LBNL to achieve

continuous file replication of hundreds of files in a single request. We note that in this setup the source MSS is the NCAR-MSS, and the target MSS is HPSS.

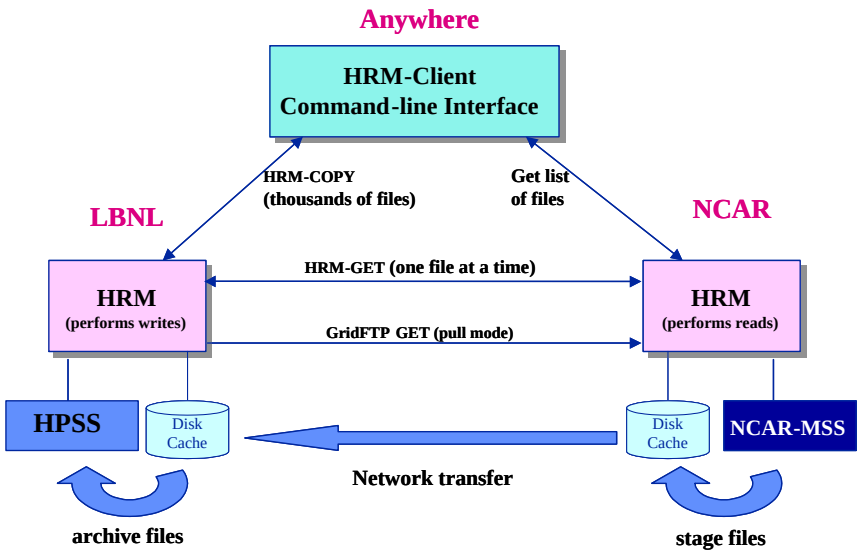


Figure 20.5. The use of SRMs for file replication between two different mass storage systems.

In Figure 20.5 we refer to the SRMs as Hierarchical (storage) Resource Managers (HRMs) because each HRM has its own disk and it accesses an MSS that has a robotic tape. As can be seen in the figure the request for a multi-file replication can be initiated anywhere. The request can be for an entire directory to be replicated. The SRM-Command-Line-Interface connects first to the source HRM to get the list of files to be moved, and then a single request to replicate these files is submitted to the target HRM.

The great value of using HRMs is that they perform all the operations necessary for multi-file replication as part of their functionality. This setup of file replication is in routine use between an HPSS system at Brookhaven National Laboratory (BNL) and Lawrence Berkeley National Laboratory (LBNL). Each multi-file transfer involves hundreds of files and 10-100s of gigabytes. A typical multi-file replication can take many hours. Another important feature of this setup is that multiple files can be staged at the source concurrently, multiple files can be transferred over the network concurrently, and multiple files can be archived concurrently. Consequently, we achieve a greater efficiency in the end-to-end transfer rate.

This setup can also replicate files from multiple sites. Since URLs are used for the source files, the multi-file transfer request can be from any SRM, including SRMs that manage only disk caches (DRMs). This generality of the SRM

design can support a data production scenario that involves multiple sites. For example, a simulation can be run at one or more sites, and the files dumped into several disk caches. A single request to the target SRM where the files have to be archived can get the files from all the temporary disks, and release the space when the files are safely in the archival storage.

6.3 Providing GridFTP to a Storage System Through an SRM

Some storage systems may not be accessible using GridFTP. It is not uncommon to need access to a mass storage system that has not been retrofitted to support GridFTP since this is a non-trivial task and GridFTP would need to be implemented as part of the code base of that system. Using SRMs it is possible to alleviate this problem by having the GridFTP server daemon and the SRM run on a Grid-enabled machine external to the MSS hardware.

A prototype of this nature was developed for HPSS at LBNL. It works as follows. When a GridFTP request for a file is made, the GridFTP code was modified to send a request to HRM. HRM issues a request to HPSS to stage a file. When this completes, the HRM returns the location of the file to GridFTP, which proceeds to transfer the file to the requester. When this completes, GridFTP issues a release to the HRM, so that the space on the HRM disk can be reused. The opposite occurs when a file archiving is requested.

The implementation of this setup did not require any changes to the HRM. The HRM functionality was sufficient to support all the interaction with the GridFTP server daemon. We note that this can be applied with any HRM, not just HRM-HPSS without any modification to the HRM using the same modified GridFTP server code. Also, this can be applied to any system that supports a DRM, and therefore can be applied to any file system not connected to the Grid to make it accessible by GridFTP.

7. CONCLUSIONS

In this chapter we introduced the concepts necessary to support storage resources on the Grid. Similar to compute and network resources, storage resources need to be dynamically reserved and managed in order to support Grid jobs. In addition to managing space, Storage Resource Managers (SRMs) manage the content (or files) in the spaces used. Managing the content permits file sharing between clients in order to make better use of the storage resources. We have shown that the concepts of file pinning, file releasing (or unpinning) and lifetime of files and spaces are necessary and useful to support the reliable and efficient use of SRMs. We have also shown that the same standard SRM functionality and interfaces can be used for all types of storage resources, including disk systems, robotic tape systems, mass storage systems, and mul-

tiple storage resources managed by a single SRM at a site. SRMs have already been implemented on various systems, including specialized mass storage systems, and their interoperability demonstrated. They are also routinely used to provide massive robust multi-file replication of 100-1000s of files in a single request.

Acknowledgments

While the initial ideas of SRMs were first developed by people from the Lawrence Berkeley National Laboratory (LBNL), many of the ideas and concepts described in this chapter were developed over time and suggested by various people involved in joint meetings and discussions, including from the European Data Grid: Jean-Philippe Baud, Jens Jensen, Emil Knezo, Peter Kunszt, Erwin Laure, Stefano Occhetti, Heinz Stockinger, Kurt Stockinger, Owen Synge, from US DOE laboratories: Bryan Hess, Andy Kowalski, Chip Watson (TJNAF), Don Petravick, Rich Wellner, Timur Perelmutov (FNAL), Brian Tierney, Ekow Otoo, Alex Romosan (LBNL). We apologize for any people we might have overlooked.

This work was supported by the Office of Energy Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

Chapter 21

NEST: A GRID ENABLED STORAGE APPLIANCE

John Bent,[1] Venkateshwaran Venkataramani,[2] Nick LeRoy,[1] Alain Roy,[1] Joseph Stanley,[1] Andrea C. Arpaci-Dusseau,[1] Remzi H. Arpaci-Dusseau,[1] and Miron Livny[1]

[1]*Department of Computer Science, University of Wisconsin-Madison*
[2]*Oracle*

Abstract We describe NeST, a flexible software-only storage appliance designed to meet the storage needs of the Grid. NeST has three key features that make it well-suited for deployment in a Grid environment. First, NeST provides a generic data transfer architecture that supports multiple data transfer protocols (including GridFTP and NFS), and allows for the easy addition of new protocols. Second, NeST is dynamic, adapting itself on-the-fly so that it runs effectively on a wide range of hardware and software platforms. Third, NeST is Grid-aware, implying that features that are necessary for integration into the Grid, such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, are a part of the NeST infrastructure. We include a practical discussion about building Grid tools using the NeST software.

1. INTRODUCTION

Data storage and movement are of increasing importance to the Grid. Over time, scientific applications have evolved to process larger volumes of data, and thus their overall throughput is inextricably tied to the timely delivery of data. As the usage of the Grid evolves to include commercial applications [Loh02], data management will likely become even more central than it is today.

Data management has many aspects. While performance has long been the focus of storage systems research, recent trends indicate that other factors, including reliability, availability, and manageability, may now be more relevant. In particular, many would argue that manageability has become the dominant criterion in evaluating storage solutions, as the cost of storage management

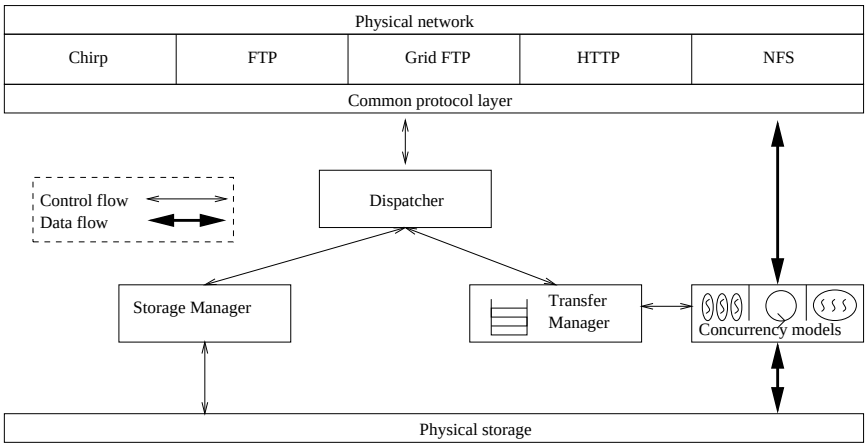


Figure 21.1. NeST Software Design. The diagram depicts NeST and its four major components: the protocol layer, the storage manager, the transfer manager, and the dispatcher. Both control and data flow paths are depicted.

outweighs the cost of the storage devices themselves by a factor of three to eight [Pat02].

One potential solution to the storage management problem is the use of specialized storage devices known as *appliances*. Pioneering products such as the filers of Network Appliance [HLM94] reduce the burden of management through *specialization*; specifically, their storage appliances are designed solely to serve files to clients, just as a toaster is designed solely to toast. The results are convincing: in field testing, Network Appliance filers have been shown to be easier to manage than traditional systems, reducing both operator error and increasing system uptime considerably [LR01].

Thus, storage appliances seem to be a natural match for the storage needs of the Grid, since they are easy to manage and provide high performance. However, there are a number of obstacles that prevent direct application of these commercial filers to the Grid environment. First, commercial storage appliances are inflexible in the protocols they support, usually defaulting to those common in local area Unix and Windows environments (*e.g.*, NFS [WLS+85] and CIFS [Sha99]). Therefore, filers do not readily mix into a world-wide shared distributed computing infrastructure, where non-standard or specialized Grid protocols may be used for data transfer. Second, commercial filers are expensive, increasing the cost over the raw cost of the disks by a factor of ten or greater. Third, storage appliances may be missing features that are crucial for integration into the Grid environment, such as the ability to interact with larger-scale global scheduling and resource management tools.

To overcome these problems and bring appliance technology to the Grid, we introduce NeST, an open-source, user-level, software-only storage appliance. As compared to current commercial storage appliances, NeST has three primary advantages: flexibility, cost, and Grid-aware functionality. We briefly discuss each of these advantages in more detail.

First, NeST is more flexible than commercial storage appliances. NeST provides a generic data transfer architecture that concurrently supports multiple data transfer protocols (including GridFTP [ABB+02b] and NFS). The NeST framework also allows new protocols to be added as the Grid evolves.

Second, because NeST is an open-source software-only appliance, it provides a low-cost alternative to commercial storage appliances; the only expenses incurred are the raw hardware costs for a PC with a few disks. However, because NeST is a software-based appliance, it introduces new problems that traditional appliances do not encounter: NeST must often run on hardware that it was not tailored for or tested upon. Therefore, NeST contains the ability to adapt to the characteristics of the underlying hardware and operating system, allowing NeST to deliver high performance while retaining the ease of management benefits of storage appliances.

Finally, NeST is Grid-aware. Key features, such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, are a fundamental part of the NeST infrastructure. This functionality enables NeST to be integrated smoothly into higher-level job schedulers and distributed computing systems, such as those detailed in Chapters 23, 22, 10, 20, and 8.

The rest of this chapter is organized as follows. Section 2 describes the overall design of NeST. The protocol layer which mediates interaction with clients is described in Section 3. Section 4 describes the transfer manager which is responsible for monitoring and scheduling concurrency and quality of service, and Section 5 describes the storage layer which manages the actual physical storage of the system. An example usage of NeST is traced within Section 6, Section 7 describes the user interface, comparisons to related work are in Section 8, and conclusions are drawn in Section 9.

2. DESIGN OVERVIEW

As a Grid storage appliance, NeST provides mechanisms both for file and directory operations as well as for resource management. The implementation to provide these mechanisms is heavily dependent upon NeST's modular design, shown in Figure 21.1. The four major components of NeST are its *protocol layer, storage manager, transfer manager* and *dispatcher*. We first briefly

examine each of these components separately; then we show how they work together by tracing an example client interaction.

2.1 Component Descriptions

The *protocol layer* in NeST provides connectivity to the network and all client interactions are mediated through it. Clients are able to use NeST to communicate with any of the supported file transfer protocols, including HTTP [FGM+99], NFS [PR85], FTP [PR85], GridFTP [ABB+02b], and Chirp, the native protocol of NeST. The role of the protocol layer is to transform the specific protocol used by the client to and from a common request interface understood by the other components in NeST. We refer to this as a *virtual protocol connection* and describe it and the motivation for multiple protocol support in Section 3.

The *dispatcher* is the main scheduler and macro-request router in the system and is responsible for controlling the flow of information between the other components. It examines each client request received by the protocol layer and routes each appropriately to either the storage or the transfer manager. Data movement requests are sent to the transfer manager; all other requests such as resource management and directory operation requests are handled by the storage manager. The dispatcher also periodically consolidates information about resource and data availability in the NeST and can publish this information as a ClassAd (see Chapter 23) into a global scheduling system [TBAD+01].

The *storage manager* has four main responsibilities: virtualizing and controlling the physical storage of the machine (*e.g.*, the underlying local filesystem, raw disk, physical memory, or another storage system), directly executing non-transfer requests, implementing and enforcing access control, and managing guaranteed storage space in the form of *lots*. Lots are discussed in more detail below.

Because these storage operations execute quickly (in the order of milliseconds), we have chosen to simplify the design of the storage manager and have these requests execute synchronously. It is the responsibility of the dispatcher to ensure that storage requests are serialized and executed at the storage manager in a thread-safe schedule.

The *transfer manager* controls data flow within NeST; specifically, it transfers data between different protocol connections (allowing transparent three- and four-party transfers). All file data transfer operations are managed asynchronously by the transfer manager after they have been synchronously approved by the storage manager. The transfer manager contains three different concurrency models, threads, processes and events, and schedules each transfer using one of these models. Scheduling policies, such as preferential schedul-

ing, and scheduling optimizations are the responsibility of the transfer manager and are discussed in Section 4.

2.2 An Example Client Interaction

We now examine how these four components function together by tracing the sequence of events when interacting with a client. In this example, we consider the case when a client first creates a new directory (*i.e.*, a non-transfer request) and then inserts a file into that directory (*i.e.*, a transfer request).

When the client initially connects to NeST with the request to create the directory, the dispatcher wakes and asks the protocol layer to receive the connection. Depending upon the connecting port, the protocol layer invokes the handler for the appropriate protocol. The handler then authenticates the client, parses the incoming request into the common request format, and returns a virtual protocol connection to the dispatcher.

The dispatcher then asks the storage manager to create the directory. After checking for access permissions, the storage manager synchronously creates the directory and sends acknowledgment back to the client through the dispatcher and the virtual protocol connection.

At this point, the dispatcher assumes responsibility for the client and listens for further requests on its channel. After the client sees that the directory is created successfully, it requests permission to send a file. The dispatcher invokes its virtual protocol connection to receive this request and again queries the storage manager. The storage manager allows the transfer and returns a virtual protocol connection into which the transfer can be written. The dispatcher passes both connections to the transfer manager, stops listening on the client channel, and sleeps, waiting for the next client request.

The transfer manager is then free to either schedule or queue the request; once the request is scheduled, the transfer manager uses past information to predict which concurrency model will provide the best service and passes the connection to the selected model. The transfer continues as the chosen concurrency model transfers data from the client connection to the storage connection, performing an acknowledgment to the client if desired. Finally, the transfer status is returned to the transfer manager and then up to the dispatcher.

In the following sections, we describe the most important aspects of NeST. First, we motivate the importance of supporting multiple communication protocols within a virtual protocol layer. Second, we describe how the transfer manager adapts to the client workload and underlying system to pick the concurrency model with the best performance. Third, we show how the transfer manager can apply scheduling policies among different connections. Fourth, we explain the role of storage guarantees in NeST, and explain how the storage manager implements this functionality.

3. PROTOCOL LAYER

Supporting multiple protocols is a fundamental requirement of storage appliances used in the Grid. Though there has been some standardization toward a few common protocols within the Global Grid Forum [OGSc], diversity is likely to reign in a community as widespread and fast-moving as the Grid. For example, even if all wide-area transfers are conducted via GridFTP, local-area file access will still likely be dominated by NFS, AFS, and CIFS protocols.

Multiple protocols are supported in NeST with a virtual protocol layer. The design and implementation of our virtual protocol layer not only allows clients to communicate with NeST using their preferred file transfer protocol, but also shields the other components of NeST from the detail of each protocol, allowing the bulk of NeST code to be shared among many protocols. Thus, the virtual protocol layer in NeST is much like the the virtual file system (VFS) layer in many operating systems [Kle86].

An alternative approach to having a single NeST server with a virtual protocol layer is to implement separate servers that understand each individual protocol and run them simultaneously; we refer to this latter approach as "Just a Bunch Of Servers" or "JBOS". The relative advantage of JBOS is that servers can be added or upgraded easily and immediately once any implementation of that protocol is available; with NeST, incorporating a new or upgraded protocol may take more effort, as the protocol operations must be mapped onto the NeST common framework.

However, we believe the advantages of a single server outweigh this implementation penalty for a number of reasons. First, a single server enables complete control over the underlying system; for example, the server can give preferential service to requests from different protocols or even to different users across protocols. Second, with a single interface, the tasks of administering and configuring the NeST are simplified, in line with the storage appliance philosophy. Third, with a single server, optimizations in one part of the system (*e.g.*, the transfer manager or concurrency model) are applied to all protocols. Fourth, with a single server, the memory footprint may be considerably smaller. Finally, the implementation penalty may be reduced when the protocol implementation within NeST can leverage existing implementations; for example, to implement GridFTP, we use the server-side libraries provided in the Globus Toolkit and we use the Sun RPC package to implement the RPC communication in NFS.

At this point, we have implemented five different file transfer protocols in NeST: HTTP, NFS, FTP, GridFTP, and the NeST native protocol, Chirp. In our experience, most request types across protocols are very similar (*e.g.*, all have directory operations such as `create`, `remove`, and `read`, as well as file operations such as `read`, `write`, `get`, `put`, `remove`, and `query`) and fit

easily into our virtual protocol abstraction. However, there are interesting exceptions; for instance, Chirp is the only protocol that supports lot management (lots will be discussed further in Section 5) and NFS is the only protocol with a lookup and mount request. Note that mount, not technically part of NFS, is actually a protocol in its own right; however, within NeST, mount is handled by the NFS handler.

We plan to include other Grid-relevant protocols in NeST, including data movement protocols such as IBP [PBB+01] and resource reservation protocols, such as those being developed as part of the Global Grid Forum [OGSc]. We expect that as new protocols are added, most implementation effort will be focused on mapping the specifics of the protocol to the common request object format, but that some protocols may require additions to the common internal interface.

Since the authentication mechanism is protocol specific, each protocol handler performs its own authentication of clients. The drawback of this approach is that a devious protocol handler can falsify whether a client was authenticated. Currently, we allow only Grid Security Infrastructure (GSI) authentication [FKTT98], which is used by Chirp and GridFTP; connections through the other protocols are allowed only anonymous access.

4. TRANSFER MANAGER

At the heart of data flow within NeST is the transfer manager. The transfer manager is responsible for moving data between disk and network for a given request. The transfer manager is *protocol agnostic*: thus, all of the machinery developed within the manager is generic and moves data for all of the protocols, highlighting one of the advantages of the NeST design.

4.1 Multiple Concurrency Models

Inclusion in a Grid environment mandates the support for multiple on-going requests. Thus, NeST must provide a means for supporting concurrent transfers. Unfortunately, there is no single standard for concurrency across operating systems: on some platforms, the best choice is to use threads, on others, processes, and in other cases, events. Making the decision more difficult is the fact that the choice may vary depending on workload, as requests that hit in the cache may perform best with events, and those that that are I/O bound perform best with threads or processes [PDZ99].

To avoid leaving such a decision to an administrator, and to avoid choosing a single alternative that may perform poorly under certain workloads, NeST implements a flexible *concurrency architecture*. NeST currently supports three models of concurrency (threads, processes, and events), but in the future we plan to investigate more advanced concurrency architectures (*e.g.*,

SEDA [WCB01] and Crovella's experimental server [CFHB99]). To deliver high performance, NeST dynamically chooses among these architectures; the choice is enabled by distributing requests among the architectures equally at first, monitoring their progress, and then slowly biasing requests toward the most effective choice.

4.2 Scheduling

Because there are likely to be multiple outstanding requests within a NeST, NeST is able to selectively reorder requests to implement different scheduling policies. When scheduling multiple concurrent transfers, a server must decide how much of its available resources to dedicate to each request. The most basic strategy is to service requests in a first-come, first-served (FCFS) manner, which NeST can be configured to employ. However, because the transfer manager has control over all on-going requests, many other scheduling policies are possible. Currently, NeST supports both *proportional share* and *cache-aware* scheduling in addition to FCFS.

4.2.1 Quality of Service

Proportional-share scheduling [WW95] is a deterministic algorithm that allows fine-grained proportional resource allocation and has been used previously for CPU scheduling and in network routers [KMC+00]. Within the current implementation of NeST, it is used to allow the administrator to specify proportional preferences per protocol class (*e.g.*, NFS requests should be given twice as much bandwidth as GridFTP requests); in the future, we plan to extend this to provide preferences on a per-user basis.

Using byte-based strides, this scheduling policy accounts for the fact that different requests transfer different amounts of data. For example, an NFS client who reads a large file in its entirety issues multiple requests while an HTTP client reading the same file issues only one. Therefore, to give equal bandwidth to NFS requests and HTTP requests, the transfer manager schedules NFS requests N times more frequently, where N is the ratio between the average file size and the NFS block size.

NeST proportional share scheduling is similar to the Bandwidth and Request Throttling module [How] available for Apache. However, proportional share scheduling in NeST offers more flexibility because it can schedule across multiple protocols, whereas Apache request-throttling only applies to the HTTP requests the Apache server processes, and thus cannot be applied to other traffic streams in a JBOS environment.

The overhead and achieved fairness of proportional share scheduling in NeST is shown in Figure 21.2. The first set of bars shows our base case in

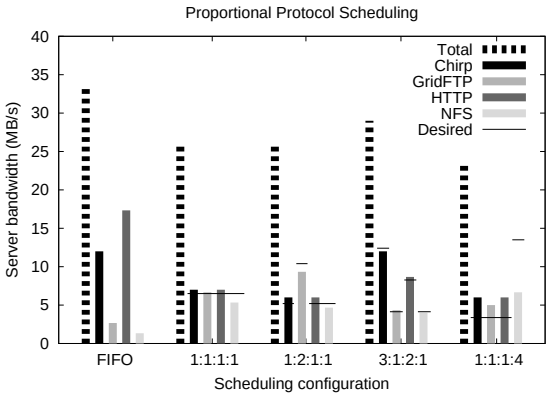


Figure 21.2. Proportional Protocol Scheduling. This graph measures the fairness and overhead of quality of service scheduling in a NeST running a synthetic workload. Within each set of bars, the first bar represents the total delivered bandwidth across all protocols; the remaining bars show the bandwidth per protocol. The labels for the sets of bars show the specified proportional ratios; the desired lines show what the ideal proportions would be. Note that NeST is able to achieve very close to the desired ratios in each case except the right-most.

which the NeST transfer manager uses the simple FIFO scheduler. The other sets of bars adjust the desired ratio of bandwidth for each protocol.

We can make two conclusions from this graph. First, the proportional share scheduler imposes a slight performance penalty over FIFO scheduling, delivering a total of approximately 24-28 MB/s instead of 33 MB/s. Second, the proportional-share scheduler achieves very close to the desired ratios in almost all cases. Specifically, using Jain's metric [CJ89] of fairness in which a value of 1 represents an ideal allocation, we achieve values of greater than 0.98 for the 1:1:1:1, the 1:2:1:1, and the 3:1:2:1 ratios.

The only exception is that allocating additional bandwidth to NFS (*e.g.*, 1:1:1:4 for Chirp:GridFTP:HTTP:NFS) is extremely difficult; the Jain's fairness value in this case drops to 0.87. The challenge is that due to the smaller block size used by NFS there are not a sufficient number of NFS requests for the transfer manager to schedule them at the appropriate interval; in the case where there is no available NFS request, our current implementation is work-conserving and schedules a competing request, rather than allow the server to be idle. We are currently implementing a non-work-conserving policy in which the idle server waits some period of time before scheduling a competitor [ID01]; such a policy might pay a slight penalty in average response time for improved allocation control.

4.2.2 Cache-Aware Scheduling

Cache-aware scheduling is utilized in NeST to improve both average client perceived response time as well as server throughput. By modeling the kernel buffer cache using gray-box techniques [ADAD01], NeST is able to predict which requested files are likely to be cache resident and can schedule them before requests for files which will need to be fetched from secondary storage. In addition to improving client response time by approximating shortest-job first scheduling, this scheduling policy improves server throughput by reducing the contention for secondary storage.

In earlier work [BBADAD02], we examined cache-aware scheduling with a focus toward web workloads; however, given the independence between the transfer manager and the virtual protocol layer, it is clear that this policy works across all protocols. This illustrates a major advantage that NeST has over JBOS in that optimizations in the transfer code are immediately realized across all protocols and need not be reimplemented in multiple servers.

5. STORAGE MANAGER

Much as the protocol layer allows multiple different types of network connections to be channeled into a single flow, the storage manager has been designed to virtualize different types of physical storage and to provide enhanced functionality to properly integrate into a Grid environment. The three specific roles fulfilled by the storage manager are to implement access control, virtualize the storage namespace, and to provide mechanisms for guaranteeing storage space.

Access control is provided within NeST via a generic framework built on top of collections of ClassAds, as described in Chapter 23. AFS-style access control lists determine read, write, modify, insert, and other privileges, and the typical notions of users and groups are maintained. NeST support for access control is generic, as these policies are enforced across any and all protocols that NeST supports; clients need only be able to communicate via the native Chirp protocol (or any supported protocol with access control semantics) to set them.

NeST also virtualizes the physical namespace of underlying storage, thus enabling NeST to run upon a wide variety of storage elements. However, in our current implementation, we currently use only the local filesystem as the underlying storage layer for NeST; we plan to consider other physical storage layers, such as raw disk, in the near future.

When running in a remote location in the Grid, higher-level scheduling systems, individual users and Grid middleware, such as SRM descibredin Chapter 20, all must be assured that there exists sufficient storage space to save the data produced by their computation, or to stage input data for subsequent ac-

cess. To address this problem, NeST provides an interface to guarantee storage space, called a *lot*, and allows requests to be made for space allocations (similar to reservations for network bandwidth [ZDE+93]).

Each lot is defined by four characteristics: owner, capacity, duration, and files. The *owner* is the client entity allowed to use that lot; only individual owners are currently allowed but group lots will be included in the next release. The *capacity* indicates the total amount of data that can be stored in the lot. The *duration* indicates the amount of time for which this lot is guaranteed to exist. Finally, each lot contains a set of *files*; the number of files in a lot is not bounded and a file may span multiple lots if it cannot fit within a single one.

When the duration of a lot expires, the files contained in that lot are not immediately deleted. Rather, they are allowed to remain indefinitely until their space needs to be reclaimed to allow the creation of another new lot. We refer to this behavior as *best-effort* lots and are currently investigating different selection policies for reclaiming this space.

To create files on a NeST, a user must first have access to a lot; however, most file transfer protocols do not contain support for creating lots. In our environment, a lot can be obtained in two different ways. First, when system administrators grant access to a NeST, they can simultaneously make a set of default lots for users. Second, a client can directly use the Chirp protocol to create a lot before accessing the server with an alternative data-transfer protocol. Section 7 demonstrates via example how to use the Chirp protocol for lot managament.

To provide maximize flexibility and user-customization, NeST currently supports two different implementations of lots: kernel enforced and solely NeST-managed. Kernel enforced lots rely on the quota mechanism of the underlying filesystem, which allows NeST to limit the total amount of disk space allocated to each user. Utilizing the quota system affords a number of benefits: direct access to the file system (perhaps not through NeST) must also observe the quota restrictions, thus allowing clients to utilize NeST to make the space guarantee and then to bypass NeST and transfer data directly into a local file system. However, one limitation in this approach is that all of a user's lots must be consolidated into a single quota limit. This consolidation makes it therefore possible for a user to overfill one of his or her lots and then be unable to fill another lot to capacity. Another limitation of the kernel enforced lots is that the NeST must be run as super-user. The NeST-managed lot implementation does not suffer from these limitations but it does require that all access to a NeST machine be through one of the supported protocols. We leave it to the individual administrator to select which implementation is more appropriate for their users.

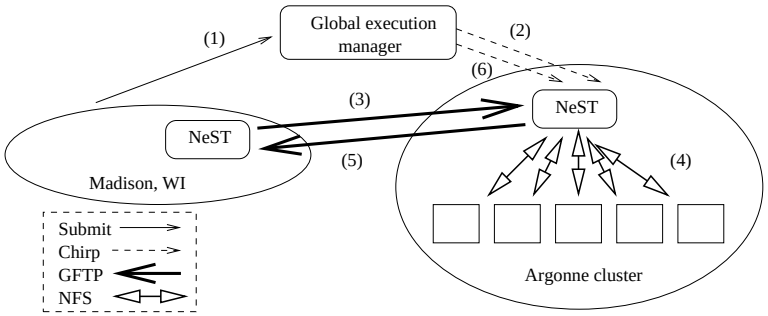


Figure 21.3. NeST and the Grid. The diagram illustrates information flow in a scenario in which multiple NeST servers are utilized on the Grid.

6. NeST AND THE GRID

With a basic understanding of NeST in place, we now illustrate how multiple NeST servers might be used in a global Grid environment. Figure 21.3 depicts such a scenario; all major events are labeled with the sequence numbers as defined in the following description.

In the figure, a user has their input data permanently stored at their *home site*, in this case at a NeST in Madison, Wisconsin. In step 1, the user submits a number of jobs for remote execution to a global execution manager. This manager is aware that a remote cluster, labeled the *Argonne cluster*, has a large number of cycles available. The NeST "gateway" appliance in the Argonne cluster has previously published both its resource and data availability into a global Grid discovery system [TBAD+01]. The manager is therefore also aware that the Argonne NeST has a sufficient amount of available storage.

The manager decides to run the user's jobs at the Argonne site, but only after staging the user's input data there. Thus, in step 2, the manager uses Chirp to create a lot for the user's files at Argonne, thus guaranteeing sufficient space for input and output files. For step 3, the manager orchestrates a GridFTP third-party transfer between the Madison NeST and the NeST at the Argonne cluster. Other data movement protocols such as Kangaroo could also be utilized to move data from site to site [TBSL01].

In step 4, the manager begins the execution of the jobs at Argonne, and those jobs access the user's input files on the NeST via a local file system protocol, in this case NFS. As the jobs execute, any output files they generate are also stored upon the NeST. Note that the ability to give preference to some users or protocols could be harnessed here, either by local administrators who wish to ensure preference for their jobs, or by the global manager to ensure timely data movement.

Finally, for step 5, the jobs begin to complete, at which point the manager moves the output data back to Madison, again utilizing GridFTP for the wide area movement. The manager is then free to use Chirp to terminate the lot in step 6, and inform the user that the output files are now available on the local NeST.

Note that many of the steps of guaranteeing space, moving input data, executing jobs, moving output data, and terminating reservations, can be encapsulated within a request execution manager such as the Condor Directed-Acyclic-Graph Manager (DAGMan) [DAG]. Also, higher-level storage resource managers such as SRM could use NeST services to synchronize access between globally-shared storage resources, as detailed in Chapter 20.

7. USING NeST SOFTWARE

The NeST software is available for download [NeS]. Currently, NeST builds on both Linux and Solaris platforms. However, the kernel enforced lot implementation relies on Linux kernel utilities and is therefore not available with the Solaris version. Generally, using both the NeST client and server is comparable to using other file transfer software such as FTP. Therefore, most of this discussion will focus on using lots as they are a unique feature of NeST.

7.1 The NeST Server

Using the NeST server is straightforward. Since NeST is user-level software that doesn't require any modified kernel patches, it can be installed and run simply. Please refer to the installation instructions on the webpage [NeS] for information about enabling the kernel quota system and downloading and installing necessary libraries such as the Globus Grid API bundle [FK97, GLO]. We recommend initially running NeST with the NeST-managed lot implementation instead of the kernel enforced version because it can be run without super-user privilege.

7.2 The NeST Interactive Client and Client Libraries

Because NeST supports many different protocols, clients can interact with the NeST server using any of the standard client programs such as GridFTP, FTP, HTTP and NFS. However, only the native Chirp protocol has lot management support. Included with the NeST software is a client library for putting Chirp protocol requests directly into user software and a thin interactive client program built from this library. In addition to all of the standard directory and data transfer operations, these client programs have support for user and group management as well as lot management. Figures 21.4 and 21.5 show how to use both the interactive client and the client library for lot management. Note that the Chirp protocol is used in both of these examples for both the lot man-

```
chirp:db16:~johnbent/> lot user 10
    Lot Handle - 1
chirp:db16:~johnbent/> lot stat 1
    Lot Handle  : 1
    Lot Type    : Guaranteed
    OwnerUser   : johnbent
    Lot Size    : 10.000000 MB
    Lot Used    : 0.000000 MB (  0.00%)
    Start time  : 1/12/2003 12:34:00
    Duration    : 1440.00 Minutes
    Time left   : 1440.00 Minutes
chirp:db16:~johnbent/> lot update 1 0 10
chirp:db16:~johnbent/> put /usr/share/dict/linux.words
    409286 /     409286 (2.56 MB/s)
chirp:db16:~johnbent/> lot stat 1
    Lot Handle  : 1
    Lot Type    : Guaranteed
    OwnerUser   : johnbent
    Lot Size    : 10.000000 MB
    Lot Used    : 0.390326 MB (  3.90\%)
    Start time  : 1/12/2003 12:34:00
    Duration    : 1450.00 Minutes
    Time left   : 1448.72 Minutes
```

Figure 21.4. Interactive NeST-client. This shows one example server-client session using the provided interactive nest-client program. In this session, a user first creates a lot, then increases its duration by 10 minutes, writes a file into the lot and then queries the status of the lot.

agement as well as the data transfer operations. However, in practice users may prefer to use Chirp for lot management and then a different protocol such as GridFTP for the data transfers.

8. RELATED WORK

As a storage appliance, NeST relates most closely to the filers of Network Appliance [HLM94] and the Enterprise Storage Platforms of EMC [EMC]. NeST does not attempt to compete with these commercial offerings in terms of raw performance as it is primarily intended for a different target domain. As such, NeST offers a low-cost, software-only alternative that offers more protocol flexibility and Grid-aware features that are needed to enable scientific computations in the Grid.

Within the Grid community, there are a number of projects that are related to NeST. GARA, described in Chapter 23, is an architecture that provides advance reservations across a variety of resources, including computers, networks, and storage devices. Like NeST, GARA provides reservations (similar to NeST's

```
#include "nest_client.h"

void main() {
    NestReplyStatus stat;
    NestConnection  server;
    NestLot         lot;
    NestLotHandle   lot_handle;

    stat = NestOpenConnection( &server, "nest.cs.wisc.edu" );
    stat = NestRequestUserLot( &lot_handle, server, 10, 1440 );
    stat = NestUpdateLot( server, &lot_handle, 0, 10 );
    stat = NestSendFile( "/usr/share/dict/linux.words",
                         "words", server );
    stat = NestGetLotStat( &lot, server, &lot_handle );
}
```

Figure 21.5. NeST-client library. This code sample, (which ignores error-handling for the sake of brevity), demonstrates the same functionality as shown using the interactive client in Figure 21.4. This code first connects to a nest server, then creates and updates a lot, writes a file to it and then queries its status.

lots), but allows users to make them in advance. However, GARA does not provide the best-effort lots or the sophisticated user management that NeST provides.

The Disk Resource Managers in SRM, described in Chapter 20, the storage depots in IBP [PBB+01] and the LegionFS servers, described in Chapter 10 also provide Grid storage services. However, each of these projects is designed to provide both local storage management and global scheduling middleware. Conversely, NeST is a local storage management solution and is designed to integrate into any number of global scheduling systems. This distinction may account for one key difference between NeST and the storage servers in each of these systems: as they are all designed to work primarily with their own self-contained middleware, none of these other projects have protocol independence in their servers. Another unique feature of NeST is its dynamic concurrency adaptation; we note however that this is not intrinsic to the design of NeST and could be incorporated in these other systems.

SRM and IBP provide space guarantees in manners similar to NeST lots. One difference however in SRM is that SRM guarantees space allocations for multiple related files by using two-phased pinning; lots in NeST provide the same functionality with more client flexibility and control and less implementation complexity.

In comparing NeST lots with IBP space guarantees, one difference is that IBP reservations are allocations for byte arrays. This makes it extremely diffi-

cult for multiple files to be contained within one allocation; it can be done but only if the client is willing to build its own file system within the byte array. Another difference is that IBP allows both permanent and volatile allocations. NeST does not have permanent lots but users are allowed to indefinitely renew them and best-effort lots are analogous to volatile allocations. However, there does not appear to be a mechanism in IBP for switching an allocation from permanent to volatile while lots in NeST switch automatically to best-effort when their duration expires.

Like NeST, LegionFS also recognizes the importance of supporting the NFS protocol in order to allow unmodified applications the benefit of using Grid storage resources. However LegionFS builds this support on the client side while NeST does so at the server side. LegionFS's client-based NFS allows an easier server implementation but makes deployment more difficult as the Legion-modified NFS module must be deployed at all client locations.

Although NeST is the only Grid storage system that supports multiple protocols at the server, PFS [PFS] and SRB [BMRW98] middleware both do so at the client side. We see these approaches as complementary because they enable the middleware and the server to negotiate and choose the most appropriate protocol for any particular transfer (*e.g.*, NFS locally and GridFTP remotely).

9. CONCLUSION

We have presented NeST, an open-source, user-level, software-only storage appliance. NeST is specifically intended for the Grid and is therefore designed around the concepts of flexibility, adaptivity, and Grid-awareness. Flexibility is achieved through a virtual protocol layer which insulates the transfer architecture from the particulars of different file transfer protocols. Dynamic adaptation in the transfer manager allows additional flexibility by enabling NeST to run effectively on a wide range of hardware and software platforms. By supporting key Grid functionality such as storage space guarantees, mechanisms for resource and data discovery, user authentication, and quality of service, NeST is Grid-aware and thereby able to integrate cleanly with distributed computing systems.

Acknowledgments

This chapter is an updated version of the paper which appeared in HPDC 11. Some of the experimental results have been removed and replaced with a more functional description of using NeST to build Grid services. This work is sponsored by NSF CCR-0092840, CCR-0098274, NGS-0103670, CCR-0133456, ITR-0086044, and the Wisconsin Alumni Research Foundation.

We would like to thank the members of the Condor team, too numerous to list, and the members of the WiND group, who are not: Nathan C. Burnett, Timothy E. Denehy, Brian C. Forney, Florentina I. Popovici and Muthian Sivathanu. However, we would like to specially mention Erik Paulson, Douglas Thain, Peter Couvares and Todd Tannenbaum of the Condor team. All of these people, as well as our anonymous reviewers, have contributed many useful suggestions specifically for this paper or for the development of the NeST project in general. Also, we would like to extend our gratitude to the members of our Computer Systems Lab who do such an outstanding job keeping our computers running and our networks up.

Chapter 22

COMPUTATION SCHEDULING AND DATA REPLICATION ALGORITHMS FOR DATA GRIDS

Kavitha Ranganathan[1] and Ian Foster[1,2]

[1]*Department of Computer Science, The University of Chicago*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract Data Grids seek to harness geographically distributed resources for large-scale data-intensive problems such as those encountered in high energy physics, bioinformatics, and other disciplines. These problems typically involve numerous, loosely coupled jobs that both access and generate large data sets. Effective scheduling in such environments is challenging, because of a need to address a variety of metrics and constraints (e.g., resource utilization, response time, global and local allocation policies) while dealing with multiple, potentially independent sources of jobs and a large number of storage, compute, and network resources.

We describe a scheduling framework that addresses these problems. Within this framework, data movement operations may be either tightly bound to job scheduling decisions or performed by a decoupled, asynchronous process on the basis of observed data access patterns and load. We develop a family of job scheduling and data movement (replication) algorithms and use simulation studies to evaluate various combinations. Our results suggest that while it is necessary to consider the impact of replication on the scheduling strategy, it is not always necessary to couple data movement and computation scheduling. Instead, these two activities can be addressed separately, thus significantly simplifying the design and implementation of the overall Data Grid system.

1. INTRODUCTION

A Grid is a distributed collection of computer and storage resources maintained to serve the needs of some community or *virtual organization* (VO) [FK99b, FKT01]. Any of the potentially large number of authorized users within that VO has access to all or some of these resources and is able to submit jobs to the Grid and expect responses. The choice of algorithms used to schedule jobs in such environments depends on the target application. Our

focus here is on scheduling algorithms suitable for large-scale data-intensive problems, such as those that arise in the high-energy physics experiments currently being developed at CERN [CMS] that will generate petabytes of scientific data by 2006. In these experiments, a community of hundreds of physicists around the world will ultimately submit millions of jobs, with each job accessing some subset of that data.

Scheduling is a challenging task in this context. The data-intensive nature of individual jobs means it can be important to take data location into account when determining job placement. Replication of data from primary repositories to other locations can be an important optimization step to reduce the frequency of remote data access. And the large number of jobs and resources means that centralized algorithms may be ineffective. Thus, for example, scheduling algorithms that focus only on maximizing processor utilization by mapping jobs to idle processors (disregarding costs associated with fetching remote data) are unlikely to be efficient.

To address this problem, we define a general and extensible scheduling framework within which we can instantiate a wide variety of scheduling algorithms. We then use simulation studies to explore the effectiveness of different algorithms within this framework.

We assume a system model in which many users submit requests for job execution from any one of a large number of sites. At each site, we place three components: an external scheduler (ES), responsible for determining where to send jobs submitted to that site; a local scheduler (LS), responsible for determining the order in which jobs are executed at that particular site; and a dataset scheduler (DS), responsible for determining if and when to replicate data and/or delete local files. The choice of algorithms for each component defines a particular scheduling system.

Within this framework, we have defined a family of five ES and four DS algorithms, LS algorithms being widely researched in the past [SHK95]. Our ES algorithms dispatch jobs to a random site, the least loaded site, a randomly selected less loaded site, the local site, or a site where required data already exists. Our DS algorithms perform no explicit replication (only caching), or alternatively, choose a random or the least loaded neighbor for replication of popular datasets (we shall use file and dataset interchangeably for the rest of the chapter). In the case of no replication, a job execution is preceded by a fetch of the required data, leading to a strong coupling between job scheduling and data movement. By contrast, the other replication strategies are loosely coupled to job execution.

To study the effectiveness of these different scheduling algorithms, we have developed a modular and extensible discrete event Data Grid simulation system, ChicagoSim (the Chicago Grid Simulator) [Chi]. In this chapter, we synthesize and extend simulation results presented in other articles [RF03, RF02].

Our simulation results show a marked improvement in Grid performance when the right combination of loosely coupled replication and scheduling policies are used. Our results also show that evaluating scheduling algorithms on their own, without considering the impact of replication techniques, can lead to suboptimal choices.

The outline of the chapter is as follows. Section 2 reviews relevant work in the arenas of Web caching and replication, distributed file systems, and Grid resource management. In Section 3, we briefly touch upon alternative Grid scheduling frameworks and provide details of our proposed model. Section 4 describes the scheduling and replication algorithms that we evaluate. Simulation details are discussed in Section 5 while Section 6 contains results. We conclude and point to future directions in Section 7.

2. RELATED WORK

Our work is related to two distinct areas: replication/caching on the Web and data management in distributed file systems. We discuss related work in these areas and their applicability to data management in Grids. We also talk about related research in the arena of Grid computing.

2.1 Data Management on the Web

Replication/caching of files has proved beneficial to reduce access latency, server load, and network bandwidth consumption on the Internet. Dynamic replication strategies are useful when user behavior changes over time, thus facilitating automatic creation and deletion of replicas.

Push caching, proposed by [GS95], uses access history to replicate files on remote servers. A server knows how popular its files are and so it decides when to push one of its popular files to a remote *friend* server. The network is divided into subnets, and a server has a record of how many requests for a file were generated by each subnet. Depending on the number of requests, a server calculates the optimal subnet to replicate a file. A similar strategy can be used for Grids. An entity (such as the proposed dataset scheduler in our framework) can keep track of file popularity at storage centers in the Grid and replicates popular files to less loaded data centers or to data centers near potential users, thus reducing hotspots and access latency.

Both [BC96] and [RA99] propose techniques to exploit geographical and temporal locality exhibited in Web client access patterns, by moving popular data closer to potential users. Bestavros et al. use the information available in the TCP/IP record route option is used to put together a tree of nodes. The server (host) is the root, the various other proxies (replication sites) form the nodes of the tree, and the clients form the leaves. The server keeps track of the popularity of each document and where the requests come from in the tree. It then periodically creates replicas of the files down the tree depending on

a popularity function. RaDar [RA99] (Replicator and Distributor and Redirector) uses the information available in routers to move data closer to clients. When a request for data is generated from a client to a server, the route taken by the request is noted and used to generate what is known as a *preference path*. The host computes preference paths periodically from information from the system's router. If one particular node appears frequently on a file's preference path, it is considered a good choice to replicate the file at that node.

Of particular importance is the issue of whether Grid user access patterns exhibit some degree of temporal, geographical, or spatial locality. Such information can then be exploited for intelligent replication strategies, similar to those described above. Initial studies of potential Grid user logs [IR03] suggest that interesting patterns of locality do exist. Earlier [RF01] we studied different file replication strategies for Grid scenarios, with the aim of exploiting locality properties in user behavior, but we did not consider job scheduling. Here we consider the effect of using such replication strategies in combination with certain job scheduling strategies.

2.2 Distributed File Systems

Projects such as OceanStore [KBC+00], Pangaea [SK01] and Freenet [CSWH00] aim to span the globe and facilitate large-scale data sharing, with an emphasis on security, reliability, availability, or anonymity. Others such as MojoNation [MOJ], Gnutella [Gnu], and Napster [Nap] focus on peer-to-peer file-sharing without much concern for security or locality. We discuss here some of the data management techniques employed by two such systems and their applicability to Grid data management.

OceanStore uses two approaches, cluster recognition and replica management, to increase the efficiency of the system. Cluster recognition involves periodically running a clustering algorithm that attempts to identify closely related files. This helps to pre-stage clusters of files that are likely to be used together. Replica management involves tracking the load created by a particular file and creating more replicas for that file on nearby nodes when access requests overwhelm it. Both methods are applicable to data management in Grid computing.

Pangaea focuses on achieving a high availability of files by massive, decentralized and optimistic replication [SK01]. Since the developers of Pangaea believe that predicting access patterns for wide-area transfers is difficult, Pangaea aggressively creates a replica of a file, whenever and wherever it is accessed. Pangaea maintains a sparely connected graph of all replicas for each file that aids in updating or deleting replicas. Replicas are deleted when the disk is out of space or a replica is inactive. To ensure a minimum number of replicas for a file, Pangaea classifies replicas as either gold or bronze and tries to maintain gold replicas on the disk for as long as possible.

Aggressive replication in Grids may be a viable option, especially if access patterns cannot be predicted accurately. Since files are replicated only when they are accessed, unneeded data transfers are avoided. Moreover, disk space is occupied only as long as there is no better use for it. Widespread replication in a Grid could, however, cause thrashing of disks. If a job was sent to a site because of the data it contained, and that data was quickly expunged (to make room for new replicas) before the job could run, the job would have to be transferred elsewhere, or the files would have to be fetched again.

2.3 Resource Management for Grids

A number of projects deal with Grid resource management. Thain et al. [TBAD+01], for example, describe a system that links jobs and data by binding execution and storage sites into I/O communities that reflect physical reality. An I/O community consists of a storage device and several compute devices associated with that storage device. Jobs that run on those compute elements are encouraged to use their own community's storage device for accessing and storing data. Thus, similar applications could form a community and reduce usage of wide-area resources. Since the work of Thain et al. presents building blocks for such communities but does not address policy issues, our work on scheduling policies complements that effort.

Execution Domains [BLM00] is a framework that defines bindings between computing power and data resources in a Grid such that applications are scheduled to run at CPUs that have access to required data and storage. Again, since this work is concerned with building such a system as opposed to defining scheduling policies, our work can be put to use here.

Another recent project, the DaP (Data Placement) Scheduler [DAP], aims to intelligently manage and schedule data to decrease I/O wait time and response time and increase disk usage and throughput in Grid communities. AppLeS (Application Level Scheduling) [BWF+96], involves scheduling from the perspective of an application. Information such as the computation/communication ratio, memory required, and nature of application data structures is all taken into account while generating schedules. Casanova et al. [COBW00] describe an adaptive scheduling algorithm XSufferage, for parameter sweep applications in Grid environments, under the AppLeS guidelines. Their approach is to place files strategically for maximum reuse. The basic difference between their work and ours is that our heuristic also actively replicates/pre-stages files. In addition, while [COBW00] makes scheduling decisions centrally, we concentrate on a decentralized and presumably more scalable model, where each site decides where and when to place its job and data.

3. ARCHITECTURES FOR SCHEDULING ON DATA GRIDS

One can envision three basic categories of scheduling architectures [HSSY00] for a distributed wide-area community: centralized, hierarchical, and decentralized. In *centralized* scheduling, all jobs, no matter where they originate, are submitted to a central scheduler for the whole Grid. The central scheduler then decides which machine to run each job on, depending on the state of different remote machines. The advantage of a centralized scheme is potentially very efficient schedules, since global state knowledge is used to generate workloads. A drawback of the centralized scheme is the bottleneck caused by all the scheduling functionality being present at one entity. As the size of the Grid grows, the central scheduler must manage more and more computing elements and thus does not scale well. Another disadvantage of a centralized scheme is the conflict in administrative domains. Local administration must give all machine handling rights to the central scheduler that decides what job to run on any machine in the Grid.

A solution to the administrative problem is to use a *hierarchical* scheme. In a hierarchical scheme, both local and global policies can be in place. Jobs are submitted to a central global scheduler, which then sends the jobs to a local scheduler present at a site. The local scheduler allocates those jobs to its machines depending on its local policy.

In the *decentralized* framework, jobs are submitted to more than one global or external schedulers. These external schedulers then select a local scheduler for each job. An advantage is that the failure of a single component does not adversely affect the whole system. Also, this scheme scales well, an important feature when we have thousands of users simultaneously submitting jobs. The disadvantage is that schedulers may have to reply on partial information or a restricted view of the Grid to make their decisions. Coordination among the external schedulers may help minimize the so-called herd behavior [Dah99] of multiple entities submitting simultaneously to a desirable site, causing it to overload.

A typical Data Grid may be composed of many different administrative domains and may ultimately serve hundreds of sites with thousands of users. Intelligent data management techniques can play a crucial role in maximizing the efficiency of Grid resources for such a Data Grid. Keeping the above factors in mind, we define a Grid scheduling framework that facilitates efficient data management, allows priority to local policies, and is decentralized with no single point of failure. We adopt a decentralized architecture, but the proposed framework could also handle the other two architectures described earlier. The scheduling logic is encapsulated in three modules (see Figure 22.1).

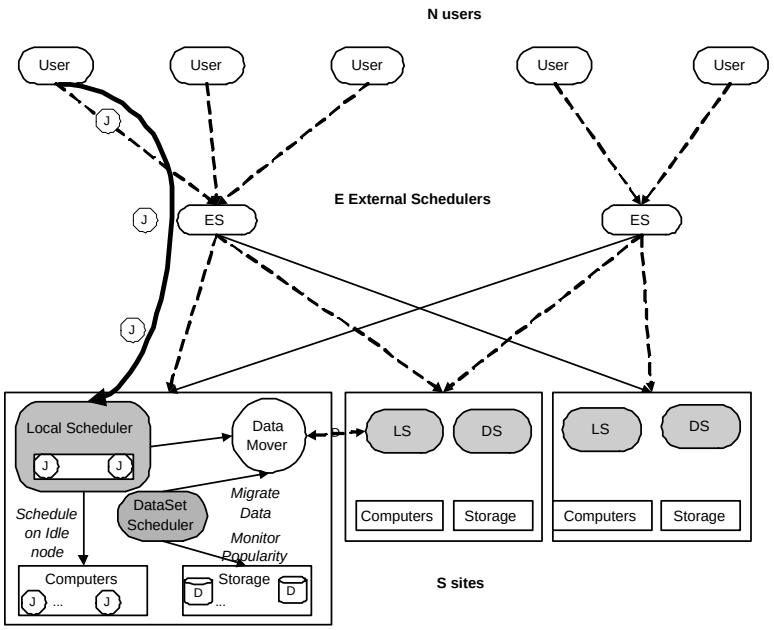


Figure 22.1. Interactions among Data Grid components.

- *External Scheduler (ES)*: Users submit jobs to the external scheduler they are associated with. The ES then decides which remote site to send the job to depending on some scheduling algorithm. It may need external information such as the load at a remote site or the location of a dataset in order to make its decisions.
- *Local Scheduler (LS)*: Once a job is assigned to run at a particular site (and sent to an incoming job queue), it is then managed by the local scheduler. The LS of a site decides how to schedule all jobs allocated to its using its local resources.
- *Dataset Scheduler (DS)*: The DS at each site keeps track of the popularity of each dataset locally available. It then replicates popular datasets to remote sites depending on some algorithm. The DS may need external information such as whether the data already exists at a site or the load at a remote site before making a decision.

Different mappings between users and external schedulers lead to different scenarios. For example, in a one-to-one mapping between external schedulers and users, the users make scheduling decisions on their own, whereas having a single ES in the system involves a central scheduler to which all users submit their jobs. For our experiments we assumed one ES per site. We plan to

study other mappings in the future. The external information a module needs can be obtained either from an information service (e.g., the Globus Toolkit's Monitoring and Discovery Service [CFFK01] or the Network Weather Service [Wol97]) or directly from the sites.

4. SCHEDULING AND REPLICATION ALGORITHMS

We are interested in two distinct functionalities: external scheduling and data replication. For each, we define and evaluate a range of different algorithms.

Our framework allows each site to have its own local scheduling policy that is implemented by the LS. Management of internal resources is a problem that has been widely researched [FR01, SHK95]. We use FIFO (first-in first-out) as a simplification.

An external scheduler selects a remote site to which to send a job, based on one of five algorithms:

- *Random*: A randomly selected site. Each site has an equal probability of getting selected, and the expected number of jobs assigned to each site is the same.

- *LeastLoaded*: The site that currently has the least load. Various definitions for load are possible; here we define it simply as the least number of jobs waiting to run, that is, the shortest job queue.

- *RandLeastLoaded*: A site randomly selected from the n least-loaded sites. This is a variation of LeastLoaded with the aim of minimizing any hotspots caused by the symmetry of actions of various sites.

- *DataPresent*: A site that already has the required data. If more than one site qualifies, the least loaded one is chosen.

- *Local*: The site where the job originated. That is, a job is always run locally.

In each case, any data required to run a job is fetched locally before the task is run, if it is not already present at the site. For the dataset scheduler, we define four algorithms:

- *DataCaching*: No active replication takes place. Datasets are pre-assigned to different sites, and no dynamic replication policy is in place. Data may be fetched from a remote site for a particular job, in which case it is cached and managed using a least recently used (LRU) algorithm. A cached dataset is then available to the Grid as a replica.

- *DataRandom*: The DS keeps track of the popularity of the datasets it contains (by tracking the number of jobs that have needed a particular dataset); and when the site's load exceeds a threshold, those popular datasets are replicated to a random site on the Grid.

- *DataLeastLoaded*: The DS keeps track of dataset popularity and chooses the least loaded site as a new host for a popular dataset. Again it replicates only when the load at its site exceeds a threshold.

- *DataRandLeastLoaded*: This is a variation of DataLeastLoaded where a random site is picked from the top n least loaded sites to avoid symmetry of behavior among sites.

In all cases, data is also cached, and the finite storage at each site is managed by using LRU.

We thus have a total of 5x4 = 20 algorithms to evaluate.

5. SIMULATION STUDIES

We have constructed a discrete event simulator, ChicagoSim, to evaluate the performance of different combinations of job and task scheduling algorithms. ChicagoSim is built on top of Parsec [PARb], a C-based simulation language. We describe in turn the simulation framework and experiments performed.

5.1 Simulation Framework

We model a Data Grid as a set of sites, each comprising a number of processors and a limited amount of storage; a set of users, each associated with a site; and a set of files, each of a specified size, initially mapped to sites according to some distribution. We assume that all processors have the same performance and that all processors at a site can access any storage at that site. Each user generates jobs according to some distribution. Each job requires that a specified set of files be available before it can execute. It then executes for a specified amount of time on a single processor.

In the absence of real traces from real Data Grids, we model the amount of processing power needed per unit of data, and the size of input and output datasets, on the expected values of CMS experiments [Hol01], but otherwise generate synthetic data distributions and workloads, as we now describe.

Table 22.1 specifies the simulation parameters used for our study. Dataset sizes are selected randomly with a uniform distribution between 500 MB and 2 GB and with initially only one replica per dataset in the system. Users are mapped evenly across sites and submit jobs according to a Poisson distribution with an inter-arrival time of 5 seconds. Each job requires a single input file and runs for 300D seconds (estimated job characteristics for CMS experiments),

where D is the size of the input file in gigabytes. The transfer of input files from one site to another incurs a cost corresponding to the size of the file divided by the nominal speed of the link. Since job output is often much smaller than input we disregard output optimization for now.

The jobs (i.e., input file names) needed by a particular user are generated randomly according to a geometric distribution with the goal of modeling situations in which a community focuses on some datasets more than others. We note that we do not attempt to model changes in dataset popularity over time.

Table 22.1. Simulation parameters used in study.

Number of sites/users	30/120
Compute elements per site	2-5
Total number of datasets	1000
Connectivity bandwidth	10 MB/sec
Size of workload	6000 jobs

5.2 Experiments

A particular *Data Grid execution* (DGE) is defined by a sequence of job submissions, allocations, and executions along with data movements. A DGE can be characterized according to various metrics, such as elapsed time, average response time, processor utilization, network utilization, and storage utilization. The scheduling problem for a Data Grid is to define algorithms that will produce DGEs that are both correct and good with respect to one or more metrics.

We use the following metrics for our experiments:

- Average amount of data transferred (bandwidth consumed) per job
- Average job completion time (max (queue time, data transfer time) + compute time)
- Average idle time for a processor

The amount of data transferred is important from the perspective of overall resource utilization, while system response time is of more concern to users. Since the data transfer needed for a job starts while the job is still in the processor queue of a site, the average job completion time includes the maximum of the queue time and transfer time, in addition to the execution time.

The idle time of processors helps measure the total utilization of the system under the different algorithms. A processor is idle because either the job queue of that site is empty or the datasets needed for the jobs in the queue are not yet available at that site.

We ran each of our 5x4 = 20 pairs of scheduling algorithms five times, with different random seeds in order to evaluate variance. In practice, we found no significant variation.

6. RESULTS AND DISCUSSION

Figures 22.2, 22.3, and 22.4 show the average response time, data transferred, and average idle time of processors for the system parameters of Table 22.1 for the different combinations of the data replication and job scheduling algorithms. The results are the average over five experiments performed for each algorithm pair. The access patterns follow a geometric distribution with $\alpha = 0.98$.

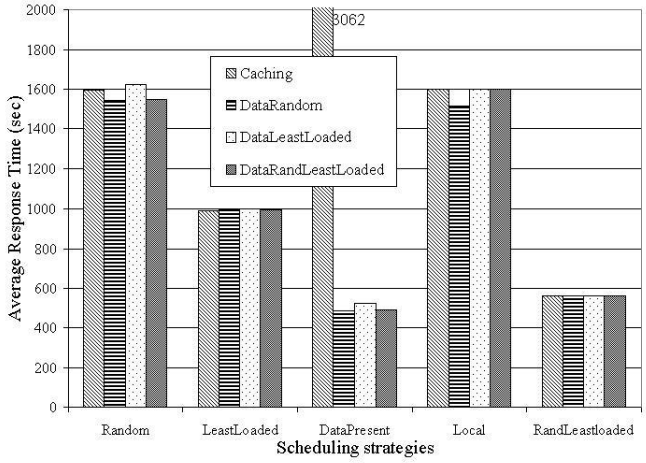


Figure 22.2. Average response time for different combinations of scheduling and replication strategies.

When plain caching (DataCaching) is used, algorithm RandLeastLoaded (send job to a randomly selected least loaded site) performs the best in terms of response time, and algorithm DataPresent (compute where the data is) performs the worst. Random and Local perform worse than the least loaded algorithms but significantly better than DataPresent. This result can be explained as follows. Although data is uniformly distributed across the Grid, the geometric distribution of dataset popularity causes certain sites to contain often-used datasets. When the algorithm DataPresent is used, these sites get more jobs than others and hence tend to get overloaded. This overloading leads to long queues at those particular sites and hence a degradation in performance. Since there is no active replication, the amount of data transferred is zero in this particular case (Figure 22.3). Also, RandLeastLoaded effectively minimizes the

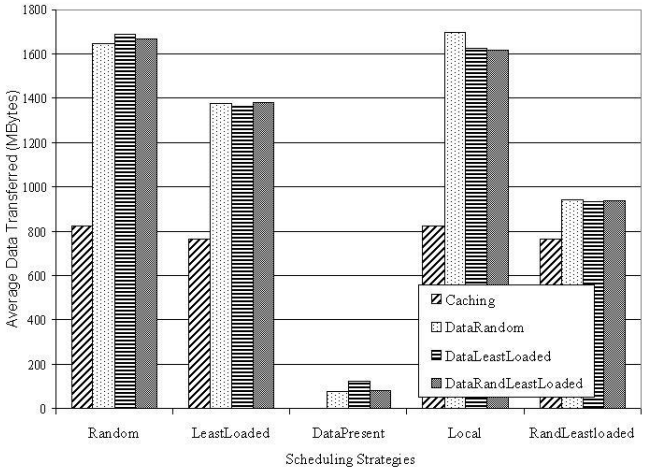


Figure 22.3. Average data transferred per job for the different scheduling and replication combinations.

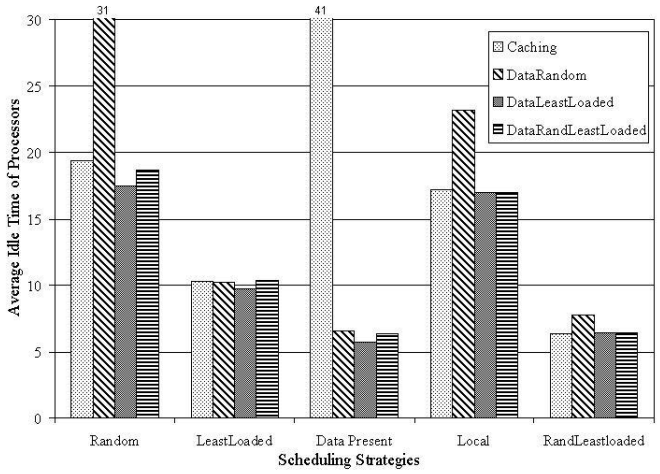


Figure 22.4. Average idle time of processors for the different strategies.

negative impact of symmetry of decisions made by different sites. This fact explains the significant improvement in response times of RandLeastLoaded as compared with LeastLoaded.

Once we introduce a replication policy, however, DataPresent performs better than all the other alternatives (Figure 22.2). It does not seem to matter which particular replication strategy is chosen, as long as one of them is em-

ployed. In terms of data transfer (Figure 22.3), the best combination is again DataPresent and DataRandom since the amount of bandwidth used in this case is almost ten times less than the other job allocation choices. Similarly, the idle time of processors is significantly smaller (Figure 22.4) for DataPresent with replication.

Clearly, in this case, dynamic replication helps reduce hotspots created by popular data and enables load sharing. The significantly better performance of strategy DataPresent when combined with any replication policy can be explained as follows.

The scheduling strategy by itself generates minimal data transfer, since jobs are scheduled to the site where the data they need is already present. Datasets are moved only as a result of explicit replication. This strategy ensures that the network does not get overloaded. Moreover, since the input data is already present at the site, jobs are not held up waiting for the required data to arrive; hence, response times are shorter. We note that once we extend our work to consider jobs that require more than one dataset, these statements may not always be true, since a job may be sent to a node that has only some of the datasets required by that job. The replication policy ensures that jobs do not accumulate at a few sites that contain popular data (by replicating popular data to other sites). Thus the computing power at a site does not cause a bottleneck. However, the replication algorithms studied do not have any significant effects on the other three scheduling algorithms.

As Figure 22.3 illustrates, the difference in the average amount of data transferred between algorithm DataPresent and the others is large. Clearly, if data locality issues are not considered, even the best scheduling algorithms fall prey to data transfer bottlenecks. These results point to the following. If the scheduling algorithms are studied by themselves (using plain caching), RandLeastLoaded appears to be the best choice. When the algorithms are studied along with the effect of replication strategies, however, we see that DataPresent works much better than any other choice. Similarly, a replication policy that might work well by itself may not guarantee the best overall performance of the Grid. Only by studying the effects of the combination of different replication and scheduling policies were we able to come up with a solution that works better than each isolated study.

In terms of idle time, DataRandom performs much worse than the other two replication policies DataLeastLoaded, and DataRandLeastLoaded.

6.1 Effect of Bandwidth

The large amounts of data transfers that take place seem to imply that the bandwidth available to processes has a direct impact on performance. Indeed, we find that if network bandwidth is decreased from 10 MB/sec to 1

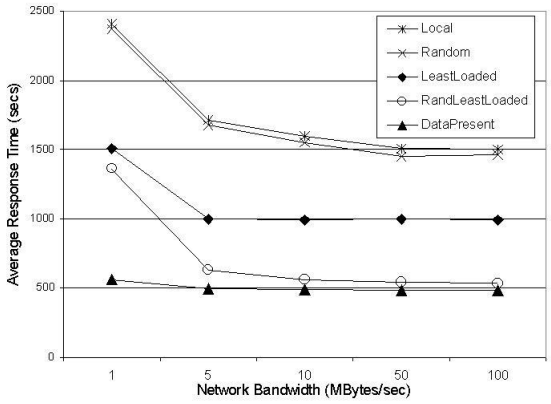


Figure 22.5. Response times of job scheduling algorithms for different bandwidth scenarios (replication algorithm used is DataRandLeastLoaded).

MB/sec (Figure 22.5), the performance of all algorithms that involve extensive data transfer (Random, LeastLoaded, RandLeastLoaded, and Local) degrade sharply. DataPresent performs more or less consistently, since it does not involve a large amount of data movement. Similarly, when the network bandwidth is increased, the four algorithms with large data transfers perform much better. The point to be noted here is that under these new conditions of ample bandwidth (100 MB/sec), RandLeastLoaded performs almost as well as DataPresent. Thus, while we believe that the system parameters of Table 22.1 are realistic for a global scientific Grid, we must be careful to evaluate the impact of future technological changes on our results.

7. CONCLUSIONS AND FUTURE WORK

We have addressed the problem of scheduling job executions and data movement operations in a distributed Data Grid environment with the goal of identifying both general principles and specific algorithms that can be used to achieve good system utilization and/or response times. In support of this investigation, we have developed a modular and extensible Data Grid scheduling framework. We have instantiated this framework with five different job scheduling algorithms and four different replication algorithms and then used a Data Grid simulation system, ChicagoSim, to evaluate the performance of different algorithm combinations.

Our results are as follows. First, the choice of scheduling algorithm has a significant impact on system performance. Second, it is important to address both job and data scheduling explicitly: for example, simply scheduling jobs to idle processors, and then moving data if required, performs significantly

less well than algorithms that also consider data location when scheduling. Third, and most interesting, we can achieve particularly good performance with an approach in which jobs are always scheduled where data is located, and a separate replication process at each site periodically generates new replicas of popular datasets. We note that this approach has significant implementation advantages when compared with (say) approaches that attempt to generate a globally optimal schedule: first, it effectively decouples job scheduling and data replication, so that these two functions can be implemented and optimized separately, and second it permits highly decentralized implementations.

These results are promising, but in interpreting their significance we have to bear in mind that they are based on synthetic workloads and simplified Grid scenarios. In future work, we plan to investigate more realistic scenarios (e.g., multiple input files) and real user access patterns. We are currently working on using workloads from Fermi Laboratory [FNA].

We also plan to validate our simulation results on real Grid testbeds, such as those being developed within the GriPhyN project [GRIb] and by participants in the International Virtual Data Grid Laboratory [iVD].

Furthermore, we plan to explore adaptive algorithms that select algorithms dynamically depending on current Grid conditions. For example, slow links and large datasets might imply scheduling the jobs at the data source and using a replication policy similar to those we used for our studies. On the other hand, if the data is small and networks links are not congested, moving the data to the job.

Acknowledgments

We thank Jennifer Schopf and Mike Wilde for their valuable discussions and feedback, and Koen Holtman for his input on physics workloads. This research was supported by the National Science Foundation's GriPhyN project under contract ITR-0086044.

VI

QUALITY OF SERVICE: QOS

Chapter 23

GARA: A UNIFORM QUALITY OF SERVICE ARCHITECTURE

Alain Roy[1] and Volker Sander[2]
[1]*Department of Computer Science, University of Wisconsin-Madison*
[2]*Central Institute for Applied Mathematics, Forschungszentrum Jülich GmbH*

Abstract Many Grid applications, such as interactive and collaborative environments, can benefit from guarantees for resource performance or quality of service (QoS). Although QoS mechanisms have been developed for different types of resources, they are often difficult to use together because they have different semantics and interfaces. Moreover, many of them do not allow QoS requests to be made in advance of when they are needed. In this chapter, we describe GARA, which is a modular and extensible QoS architecture that allows users to make advance reservations for different types of QoS. We also describe our implementation of network QoS in detail.

1. INTRODUCTION

Many computing applications demonstrate increasingly voracious appetites, consuming ever more resources. From USENET to the spread of the World Wide Web to peer to peer file sharing, the demand for bandwidth on the Internet has been steadily increasing. Similarly, scientific programs used to measure their speed in megaflops, but now strive for teraflops and process terabytes instead of gigabytes [FK99b].

Just as data seems to expand to fill any size hard drive one can buy, today's most demanding applications strain the capacities of the networks, computers, and storage devices they use. When these applications must share their resources with other applications, they may be unable to perform to the satisfaction of their users. The problem is twofold: the resources are limited and the amount of a resource available to a particular application fluctuates depending on conditions beyond its control.

If an application does not have enough resources available to meet its performance needs, there are only two general solutions: the capacity of the resources

available to the application can be increased (such as buying more bandwidth), or the need for the resource can be decreased (such as decreasing the resolution of a streaming video). Sometimes resources have sufficient capacity for one application, but the actual capacity available to that application fluctuates because the resource is being shared with other applications. The most common example of this is a network, which is almost always shared between multiple applications. If we have such a shared resource and we cannot reliably have constant and sufficient service from it, there are two general strategies we can use. First, an application can adapt to the amount that is available. For example, a video streaming application may decrease the resolution of the video it sends when less bandwidth is available. Second, the resource may provide a guarantee that it will provide a certain quality, such as an upper boundary for the end-to-end delay, to the application. When a resource is able to offer such a guarantee, it is said to offer *quality of service*, or *QoS*.

While some applications are capable of easily adapting or may need only a single type of QoS, such as network QoS, others are very demanding and run in complex environments. They may require combinations of several types of QoS including network, CPU, and storage. Managing multiple resources with QoS can be difficult for applications because each type of QoS is typically controlled by a completely different system with different interfaces, capabilities, and behavior. Yet this management is important, because without combining different types of QoS, some applications may fail to operate well enough to meet users' expectations.

Additionally, applications often need to be scheduled. Sometimes an application needs to run at a particular time, perhaps to perform a demonstration or to be coordinated with some other activity. Other times, it is merely necessary to find a time when different QoS constraints can be simultaneously satisfied. In these examples, it is advantageous to be able to schedule the reservations for QoS in advance of when they are needed. We call these advance reservations. More precisely, an advance reservation is obtained through a process of negotiating a possibly limited or restricted delegation of a particular resource capability from the resource owner to the requester over a defined time interval.

To address this demand, we believe that there must be a resource management framework that is capable of providing a uniform interface to advance reservations for different types of QoS. To fill this need, we have developed such an architecture, the General-purpose Architecture for Reservation and Allocation (GARA) to allow demanding applications to easily manage quality of service for the various resources used by the application. GARA is a modular and extensible QoS system architecture that integrates different QoS mechanisms.

2. A UNIFIED ARCHITECTURE FOR QUALITY OF SERVICE

GARA provides a uniform mechanism for programmers to request QoS for different types of QoS. Once such uniform mechanisms are in place, it simplifies life for more than just the application programmer. It becomes possible to easily create higher-level services that can manage multiple simultaneous QoS requests on behalf of users. Perhaps the most important reason for having a uniform architecture for QoS is that it allows for relatively easy expansion of the services provided to users. As we will see below, GARA has a layered architecture that allows developers to easily add new QoS providers.

GARA's uniform architecture allows it to be easily used and extended by Grid users and developers. GARA has four key features:

- A uniform interface makes it easy to build services on top of GARA that provide new features to end-users, such as the ability to make coordinated reservations, or co-reservations.
- The ability to request advance reservations, in order to schedule applications against other constraints, or in order to find a time when all the QoS constraints will be able to be simultaneously met in the future.
- A layered architecture that allows for easy extensions as new QoS reservation mechanisms become available. For example, a graphical application that makes CPU and network reservations can easily add reservations for graphic pipelines if that ability is added to the lower layers of GARA. It is easy to add to the lower layers, and it does not require deep understanding of the higher layers in order to do so.
- GARA operates in a Grid infrastructure that includes a security infrastructure so that all reservation requests are securely authenticated and authorized. Security is an important aspect for a system that allows reservations, yet many QoS systems do not provide security. The Grid infrastructure that GARA currently uses is Globus.

2.1 Architecture

2.1.1 A Generic Framework for Advance Reservation in Grid Environments

GARA has a four-layer architecture, as illustrated in Figure 23.1.

The layer that most programmers would use is the *GARA layer*, which provides uniform remote access via the GARA Application Programmers Interface (API). This layer provides three essential services. First, it allows reservation requests to be described in a simple, uniform way. Second, when a

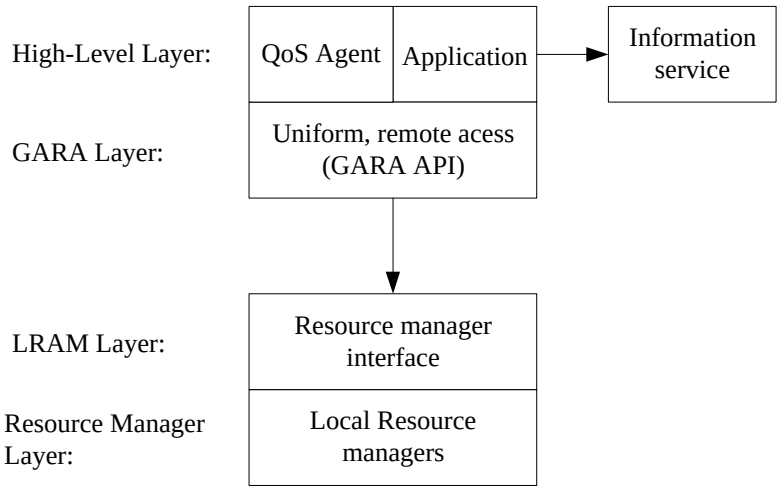


Figure 23.1. GARA's four-layer architecture. Applications and higher-level services use the GARA API, which communicate securely with the local resource layer, which in turn communicates with resource managers. Applications also communicate with an information service to find out information about resources for which they can make reservations.

reservation has been granted, it is described by a unique, data structure called a handle that can be stored, transmitted, and used to refer to the reservation. Third, it communicates with reservation services that may be located remotely or locally.

Applications also communicate with an information service that can inform them about likely reservations that can be made, and what to contact to make them. By combining resource reservation and allocation with the ability to search the information service, GARA offers a flexible framework for the construction of higher-level scheduling services.

The GARA layer communicates with the *LRAM layer*. The LRAM layer provides a resource manager interface that is responsible for authenticating and authorizing that the user is allowed to make a reservation. This layer is unaware of the specifics of the reservation, so it can only provide coarse authorization such as "Alice can make reservations", but not "Alice can only make reservations for bandwidths less than ten percent of the available bandwidth". That fine-grained authorization happens at a lower-level because it often depends on specifics of the available resource.

The LRAM layer is a "mostly uniform" interface to resource managers. It is not completely uniform because it is unnecessary: this layer provides a thin shim between the resource interface layer and the resource manager level beneath. It is the responsibility of this layer to translate all incoming requests so

that they can be presented to the resource managers that actually provide the QoS reservations.

The resource managers in the *resource manager layer* are responsible for tracking reservations and enforcing them by communicating with the lower-level resources, such as the network.

Instead of applications, there may be higher-level services in a *high-level layer*. These handle QoS requests for applications, often interacting with the information service and making multiple reservations at the same time. Such services are discussed in Section 3.

This four layer architecture allows for a uniform interface at the top, secure access to remote resources, and any number of QoS implementations.

2.1.2 Resource Reservations: A High-Level Interface for Grid Applications

Let us take an example of how a programmer may make a reservation for network bandwidth needed tomorrow afternoon. First, the program needs to make a list of the attributes needed for the reservation. GARA uses a text-based attribute-value representation of a reservation. The representation language we currently use—the Globus Resource Specification Language, or RSL [CFK+98b]—is schema-free, but GARA has some standard attributes. A typical reservation request may look like:

```
&(reservation-type=network)
 (start-time=953158862)
 (duration=3600)
 (endpoint-a=140.221.48.146)
 (endpoint-b=140.221.48.106)
 (bandwidth=150)}
```

The first three fields (reservation-type, start-time, and duration) are used for all reservations. The last three fields are unique to network reservations.

To request the reservation, the programmer does:

```
(error, handle) = reservation-create(resource-name, resv-desc);
```

Assuming there is no error, the reservation has been made. It can be queried at any time to find out the status of the reservation:

```
(error, status) = reservation-status(handle);
```

There is also an asynchronous event delivery service that can inform a program about reservation related events. These events are sent by the resource manager. Example events are a notification that a reservation time has begun or that an application is sending data faster than the reservation allows [FRS00].

When a program is ready to use a reservation, it sometimes needs to inform GARA of the information that was not previously available. For example, a network reservation needs to provide the port numbers used by the TCP or UDP connection so that the network routers can provide QoS guarantees, but these port numbers are not known in advance. Providing this information is known as binding the reservation:

```
bind_params = "(endpoint-a-port=1234)"
            + "(endpoint-b-port=5678)";
error = reservation-bind(handle, bind_params);
```

When the program is done with a reservation, it can be canceled:

```
error = reservation-cancel(handle);
```

Note that the information passed within a bind request is always related to the requested type of service. GARA uses RSL to provide a generic interface. As much as possible, these parameters are kept consistent in GARA, but they must change to reflect the underlying properties of the QoS. Beyond this difference though, GARA present a uniform interface to the underlying QoS providers.

2.1.3 Resource Managers: Service Provisioning for Grid Resources

A *resource manager* translates requests for QoS into actions that need to be taken to ensure that the QoS is provided to the application. For instance, a resource manager may configure a router to ensure that an application receives the bandwidth that it requested.

GARA was designed to make it easy to integrate new resource managers written by other people, but we also created several resource managers just for use in GARA. For GARA, we created a network QoS resource manager that uses differentiated services, a prototype disk space QoS resource manager, and a CPU QoS resource manager that uses process priorities. We also created two hybrid resource managers: one interacts with the Dynamic Soft Real-Time (DSRT) CPU scheduler [CN99] and adds advance reservations, another interacts with the Portable Batch System (PBS) which already provides advance reservations. A collaborator created a resource manager for graphic pipelines. Although we worked with a number of resource managers, most of our focus was on the network resource manager.

To be used in GARA, resource managers need to have a few common features:

- *Advance Reservations.* Each resource manager must support advance reservations. If a resource manager does not support advance reservations, support can be added by using a hybrid resource manager on top

of the resource manager, similar to the DSRT example mentioned above. Within GARA, we developed a simple but effective *slot table manager* to manage reservations that are considered as slots in time. Each slot represents a single capacity delegation as a "slot" of time. These slot tables can be used by any resource manager to keep track of reservations. In addition to the provision of basic slot table operations such as creating, modifying, and deleting an entry, the manager can also deliver asynchronous events when a reservation begins or ends. Therefore, it offers an implementation framework for implementing advanced notification services as described above and can be reused in different resource managers.

- *Interaction with Resource.* Each resource manager needs to interact with the underlying resource in order to enforce the QoS. If the resource manager does not have complete control over the QoS, then reservations cannot be guaranteed.

- *External Interface.* Services provided by resource managers need to be accessed. Because GARA incorporates the interface of resource managers into a Grid resource management framework, it depends on the ability to interface directly with the resource manager. Note that GARA was implemented within the Globus framework which provides user delegation. It therefore knows which user is accessing a resource manager, and all interaction happens as that user.

Note that resource managers do not necessarily need to provide authentication or remote access, since that is provided through the higher levels in GARA. However, because GARA understands users and uses delegation when users authenticate, resource managers can do additional authentication.

3. CO-RESERVATIONS

Most QoS research has concentrated on single types of reservations, whether network reservations [FV90], CPU reservations [LRM96], or disk reservations [MNO+96]. However, it is often important to use different reservations at the same time. When multiple reservations are made at the same time, we call them coordinated reservations, or *co-reservations*.

Consider, for example, the scientific visualization application shown in Figure 23.2. Here we have an application reading experimental results from disk, rendering the results by creating lists of polygons, and sending the results to a remote computer which then visualizes the results. If the entire system is manually reserved to be used by the application alone, perhaps by a phone call to a system administrator, then no QoS mechanism is necessary. However, if we are using shared systems, any portion of the system could experience con-

tention, slowing down the scientific visualization. In particular we could have contention for:

- the disk system where the experimental results are stored,
- the CPU doing the rendering,
- the network used for sending the rendered data,
- the CPU displaying the final results.

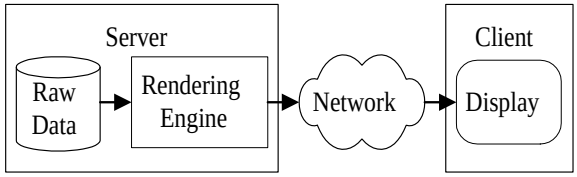


Figure 23.2. An application that could benefit from co-reservation. An example of an application that could benefit from co-reservation. Because the reservation pipeline uses several different potentially shared resources, it is likely to be beneficial for the application to make a reservation for each resource: disk, graphic pipeline, computer, display, and network.

Any one or a combination of these systems could require the use of QoS. We need to make reservations for each system to ensure that everything works smoothly when we cannot predict what contention will occur in the future.

Figure 23.3 shows a concrete example of the usefulness of co-reservation. In this example, an application is attempting to send data at 80 Mb/s using TCP. Because the application is sending at a high rate, it may delay in sending data if the CPU is busy. Because of TCP's sliding window mechanism, this may result is significantly lower bandwidth. In the experiment, the application experienced two types of congestion. First, there was network congestion beginning at about time 15 and continuing to the end of the experiment. A network reservation was made at time 40 to request for an appropriate bandwidth. Second, there was contention for the CPU at about time 60 and continuing for the rest of the experiment. A CPU reservation was made at time 80 to correct for this. From time 80 to 120, both reservations were active, and the application was able to send data at its full rate. co-reservation.

Although the application shown in Figure 23.3 was an experiment and not performed with a real application, it reinforces our point that it is often important to combine different types of reservations.

Because GARA has a uniform interface to multiple types of underlying reservation systems, it is fairly easy to build co-reservation agents that manage the multiple reservations on behalf of a user. We have built such co-reservation agents, and they are described in [Roy01].

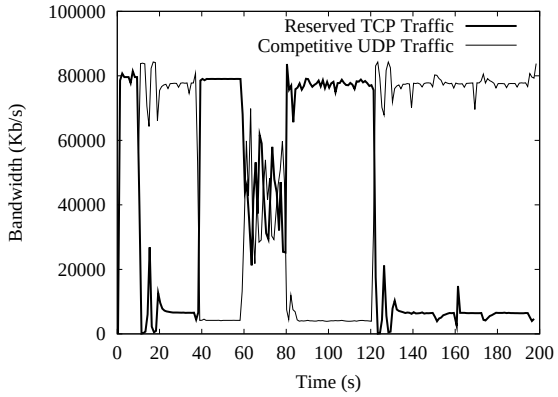


Figure 23.3. Combining DSRT and differentiated services reservations.

4. NETWORK RESERVATIONS

In a Grid environment, networks are essential to the smooth operation of the Grid, and they are also often the most shared resources. Therefore, when we developed GARA we spent considerable effort in ensuring that GARA could effectively manage advance reservations and QoS for applications that made demanding use of networks.

Grid applications have a wide-variety of network QoS needs which are much more challenging than other network QoS applications such as voice over IP. This is because Grid applications use a wide variety of network flows, including low-bandwidth but low-latency control flows, high-bandwidth and low-latency data transfers for applications such as visualization, and high-bandwidth but not necessarily low-latency data transfers for moving large amounts of data for analysis and other purposes. This combination of types of network flow places strong requirements on the network QoS layers. When we examined the needs of various applications [FK99b] we saw the need to support two basic services: a premium service offering a low-delay virtual leased line and a guaranteed rate service. Because of the wide variety and complexity of network demands coupled with the requirement to incorporate the resource "network" into our Grid resource management framework, GARA has a flexible resource manager that addresses the particular requirements of emerging Grid applications in IP-based networks.

Initially, our efforts focused on providing network QoS within a single network domain, which considerably simplified the problem. Later, we investigated providing network QoS between multiple network domains. All of these efforts were implemented in GARA at the resource manager level.

4.1 Single Domain Network Reservations

We initially built a GARA resource manager that could provide reservations within a single network domain. This facilitated a rapid prototyping, because it is possible to give it access to all of the relevant network resources in order enforce the QoS.

The first problem we considered was the mechanism to use to implement the network QoS. QoS in an Internet Protocol (IP) network can be provided in different ways. Over the years, two primary approaches have been used and are exemplified by two standards published by the Internet Engineering Task Force's (IETF). One of these is Integrated Services [Wro97, BCS94] which provides service guarantees based on a flow-based packet differentiation in each router, and the other is Differentiated Services, or diffserv, which differentiates only the treatment of classes of packets called aggregates instead of individual reservations. Integrated Services has been largely abandoned in favor of diffserv, and we do not discuss it further here.

The diffserv architecture [BBC+98] is a reaction to the scalability problems of the flow-based approach of the Integrated Services architecture, and does not provide reservations. Instead, packets are identified by simple markings in the type of service field of the IP-header [NBBB98] that indicate how routers should treat the packets. In the core of the network, routers need not to determine which flow a packet is part of, only which aggregate behavior they should apply. In this scenario, one needs to decide which packets get marked–this is how a higher-level service can provide reservations. To do this, a particular resource manager called a *bandwidth broker* is used. A bandwidth broker is a middleware service which controls and facilitates the dynamic access to network services of a particular administrative domain. Our goal for GARA was to design and implement a resource manager which fulfills the functions of a bandwidth broker.

Diffserv allows packets to be marked either by applications or by the first router that receives the packets—the edge router. If applications are allowed to mark packets, QoS cannot be guaranteed, so GARA uses the edge routers to mark packets. In order to enforce the reservation, packets are only marked when they are "within profile"—that is, when the sender is sending within rate given to the reservation.

Core routers (those that are not on the edge) have an easier job because they do not need to identify packets that need marking, nor police packets to ensure they are within profile. Instead, core routers apply a particular packet treatment—called per-hop behavior (PHB)—based only on these markings. Currently, the Internet Engineering Task Force's Differentiated Services Working Group has specified a small set of PHBs [DCB+01, HFB+99].

GARA uses the Expedited Forwarding (EF) PHB, which is intended to be used for a high-priority service with little jitter and queuing delay. The exact definition of EF is rather technical, so to simplify, each router interface can be configured so that traffic marked for the EF aggregate is prioritized over all other packets. To prevent starvation, we have to limit the amount of data which is marked as EF. This admission control is the task of the GARA resource manager. Details of how EF is required to work are defined in [CBB+02]. We have found that when carefully used, EF can provide robust reservations.

In order to implement a premium service based on EF, GARA assumes that each output link is configured to identify EF packets and to prioritize them appropriately by applying priority queuing. Note that this potentially requires a configuration update of all routers in a domain. Fortunately, this only has to be done once. While the admission control procedure uses the slot table manager to respond to reservation requests, reservations also have to be instantiated and policed in the edge routers. GARA dynamically configures the packet classifier, the policer, and the packet marker to appropriately map packets to EF. Figure 23.4 illustrates this scenario. Packet classification is done based on the reservation attributes specified when a user made a reservation. When applied to the scenario we describe here, it is done based on the end-points (address and port number) and the protocol (TCP or UDP).

Once the packets have been classified, a so-called "token bucket" is used to ensure that during the time interval $[t_0, t_1]$ of length $T = t_1 - t_0$ the amount of data sent does not exceed $rT + b$ bytes. Here, r denotes the average rate the at which the token bucket operates and b represents the depth of the token bucket which allows some limited bursts. Packets which fulfill this constraint will be marked to belong to EF. To do this correctly, GARA identifies and configures the relevant edge router every time a reservation is activated.

Note that applications may have a difficult time staying within the limits of their reservations. While monitoring the policing function and providing feedback to the application is appropriate for UDP-based flows, this mechanism does not work well for TCP-based communication. In order to assist applications, a third mechanism, called traffic shaping, is used for the traffic entering the edge router. The idea is to shape the injected TCP-traffic that it injects a smooth rate that conforms to the reservation to the core network. By incorporating this with the relaxed configuration of the policing function, TCP-applications can effectively paced to use their reserved bandwidth. Details of work we have done with this can be found in [SF02].

In previous papers such as [SFRW00], we have described many experiments that demonstrate the details of implementing such schemes successfully. GARA follows a concept which we call the "easy-to-deploy" paradigm, that is, GARA's ability to provide network services to Grid applications does not rely on complex nor unrealistic assumptions. Based on the deployment of a

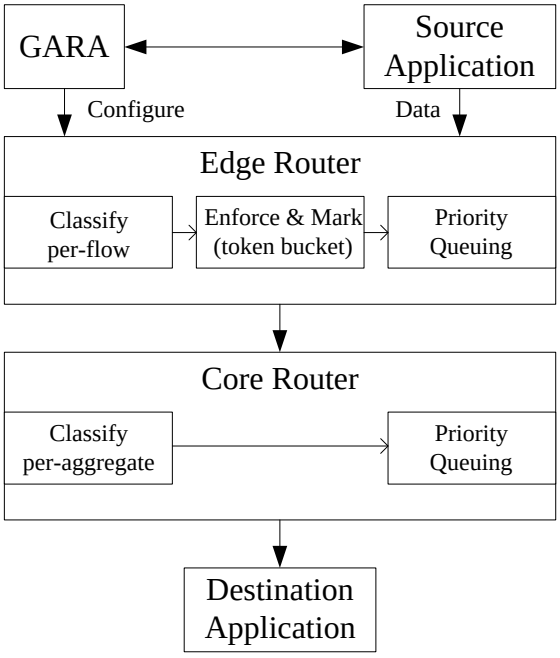


Figure 23.4. A simple network that shows how GARA uses diffserv. GARA configures edge routers to use classification, marking, and enforcement per-flow, and priority queuing is used in the core.

single prioritized PHB, GARA is able to provide a premium and a guaranteed rate service. Furthermore, we do not rely on changes of the transport protocol, nor specific advanced capabilities of the operating system, such as the support of traffic shaping. A comprehensive discussion can be found in [San03].

4.2 Multi-Domain Network Reservations

As mentioned above, the GARA network resource manager that implements network QoS is an example of what is commonly known as a bandwidth broker. Because of the fact that end-to-end guarantees in Grid environments are likely to happen in complex network environments where multiple independent administrative organizations are responsible for the operation of subparts of the network, it is very unlikely that a single bandwidth broker will control more than one administrative domain. Instead, each administrative domain wishes to have control over their resources and will thus operate its own policy decision point.

Therefore, bandwidth brokers must interact with other bandwidth brokers. A network reservation for traffic traversing multiple domains must obtain mul-

tiple network reservations, as shown in Figure 23.5. Here, Alice wants to make a network reservation from her computer in *source domain* A to Charlie's computer in *destination domain* C. Somehow she needs to contact and negotiate a reservation with BB_A and BB_C as well as the *intermediate domain*, BB_B. We have experimented with two approaches to multi-domain reservations: co-reservation and chained bandwidth brokers.

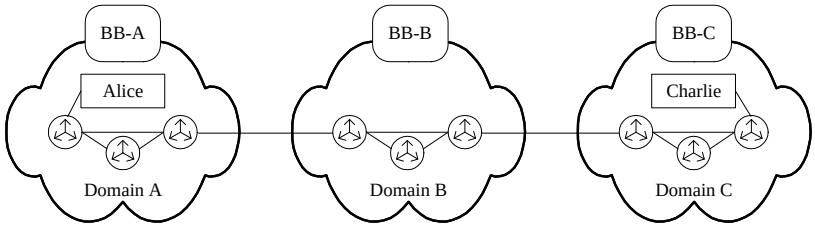


Figure 23.5. The multi-domain reservation problem. Alice needs to contact three bandwidth brokers (BB-A, BB-B, BB-C) to make a network reservation from her computer in domain A to Charlie's computer in domain C.

4.2.1 Using Co-Reservation

Alice, or an agent working on her behalf, can contact each bandwidth broker individually A positive response from every bandwidth broker indicates that Alice has an end-to-end reservation. However, there are two serious flaws with this methodology. First, it is difficult to scale since each bandwidth broker must know about (and be able to authenticate) Alice in order to perform authorization. Furthermore, if another user, Bob, makes an incomplete reservation, either maliciously or accidentally, he can interfere with Alice's reservation. An example of this type of bad reservation is illustrated in Figure 23.6.

The authorization problem could be solved if Alice could acquire some common credential issued by a community wide authorization server. GARA could interoperate with the Community Authorization Server (CAS) [PWF+02] of the Globus project to achieve this. However, the problem of incomplete reservations discouraged us from pursuing network co-reservations further.

4.2.2 Using Chained Bandwidth Brokers

The problems just noted are a motivation for the specification of an alternative approach, in which reservation requests are propagated between bandwidth brokers rather than all originating at the end domain. As shown in Figure 23.7, this means that Alice only contacts BB_A, which then propagates the reservation request to BB_B *only if* the reservation was accepted by BB_A. Similarly,

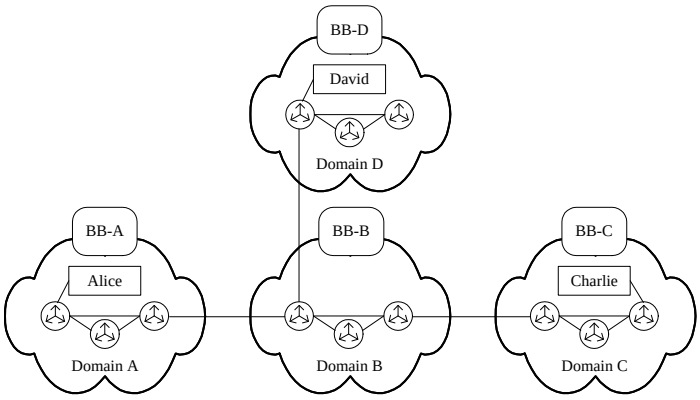


Figure 23.6. Consistency problem of source-domain-based signaling. David, a malicious user in domain D, makes a reservation in domains D and B, but fails to make a reservation in domain C, even though he will be sending his marked packets to Charlie in domain C. Domain C polices traffic based on traffic aggregates, not on individual users, so it cannot tell the difference between David's traffic and Alice's reserved traffic. Therefore, there will be more reserved traffic entering domain C than domain C expects, causing it to discard or downgrade the extra traffic and thereby affecting Alice's reservation.

BB_B contacts BB_C. With this solution, each bandwidth broker only needs to know about its neighboring bandwidth brokers, and all bandwidth brokers are always contacted. In addition to this chained signaling approach, Figure 23.7 also demonstrates the use of the GARA API (see Section 2.1.2) to couple a multi-domain network reservation with a CPU reservation in domain C.

With this chained signaling approach, the bandwidth broker interfaces not only with the high-level GARA interface for application, but also with its peer bandwidth brokers. The Internet2 community [ABC$^+$01] has proposed using a long term TCP connection to establish a stateful communication between peered bandwidth brokers. However, a reservation actuator accompanies reservations during their lifetime. Therefore, there is no need for a long term connection for individual requests. The abstraction of a traffic trunk is the resolution for these heterogeneous demands. While a traffic trunk represents a single reservation for end-domains, it represents the pieces of interest for transient domains: *core tunnels*. A core tunnel is an aggregated uni-directional reservation between the two end-domains. It connects the egress router of the source-domain with the ingress router of the destination-domain by means of the service request parameters. By introducing a traffic trunk for each core tunnel, a reservation actuator accompanies a core tunnel during its lifetime. It subscribes to events signaled by the peered domains and in doing so, it enforces

a TCP connection for all entities which have registered a callback which lifetime is related to the lifetime of the core tunnel. For static Service Level Agreements (SLAs), the proposed model conforms to the SIBBS model, because a static SLA is represented by a set of long term core tunnels. A comprehensive discussion on this approach can be found in [SAFR01, San03].

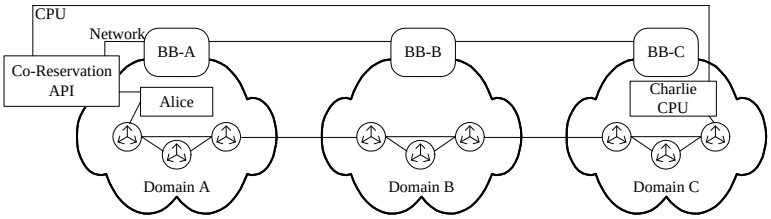


Figure 23.7. Multi-domain reservations with hop-by-hop-based signaling. Hop-by-hop-based signaling of QoS requests is done using an authenticated channel between peered bandwidth brokers along the downstream path to the destination.

4.2.3 Building Per-Domain Behaviors

The purpose of specifying PHBs for aggregates is to establish services. Because diffserv is used in domains in which the specific PHBs are applied, services are established by domains and are provided within domain boundaries. [NC01] defined this more precisely as a Per-Domain Behavior (PDB). It describes the expected treatment that an identifiable or target group of packets will receive from "edge-to-edge" of a diffserv domain. The creation of a core tunnel in transient domains can thus be interpreted as an agreement to serve the related aggregate with a particular service level, or PDB.

Earlier, we described GARA's slot table manager for performing its admission control task. Initially, we used it in a simplistic way, and used a single slot table for a whole domain. This solution limited the offered service to the achievable service of the link with the minimum QoS capability of the domain, that is, we assumed that all requests will flow through this particular link.

Later, we used a more advanced admission control procedure using the knowledge about the network topology and about the routing tables, which is able to identify the actual path of the request in the controlled domain. In this case, the service was not limited by the minimum link capability anymore. However, also this approach does have its limitations. In the fuzzy context of aggregate based scheduling it is hard to provision strict delay and jitter boundaries [CB00]. We therefore respected the ability to incorporate traffic engineering capabilities into the admission control procedure of GARA. In ex-

tending the interaction with the edge router by also controlling the use of the traffic engineering capabilities of the MultiProtocol Label Switching (MPLS) architecture [RVC01, RTF+01], GARA offers a flexible framework for service provisioning in transient domain. Applying network calculus [BT00, Bou96], a formal method for the worst-case analysis of the achievable network service, gives the opportunity to use this feature for the establishment of strong service guarantees [San03, Fid03].

5. AN IMPLEMENTATION FOR THE GLOBUS TOOLKIT

The current implementation of GARA uses the Globus Toolkit as a foundation. It was initially implemented as part of Globus 1.1.3, although a port to Globus 2.0 has been done by a third party.

The four layers of the GARA architecture shown in Figure 23.1 map closely to the layers of the Globus Toolkit. The GARA API, which resides in the remote access layer, corresponds closely with the Globus Resource Allocation Manager (GRAM) API [CFK+98b], which uses the Grid Security Infrastructure (GSI) for authentication (see Chapter 5). The Globus gatekeeper is responsible for authenticating and authorizing all GRAM and GARA interactions with a system. The LRAM layer and the local resource managers do not have exact analogues in the Globus Toolkit, but were implemented completely within the GARA framework.

Because the protocol for communication with the gatekepeer and the security mechanisms were already completely existing within the Globus Toolkit, we were able to easily leverage them without any loss of generality or flexibility in the overall architecture of GARA.

5.1 Security

Globus uses the Grid Security Infrastructure (GSI) [FKTT98]. The GSI normally uses public key cryptography. Users have private keys that they never share, and public keys (called certificates) that anyone can view. An important aspect of GSI is that it allows users to delegate credentials. To do this, a user can create a proxy certificate which has a new public and private key and is signed by the user's private key. However, it usually has a much shorter life time, generally on the order of twelve to twenty-four hours. This proxy certificate can then be used for authentication. If the proxy should happen to be compromised, it will useful for a much shorter time than the user's private key.

5.2 The Gatekeeper Protocol

GARA uses GSI to authenticate with the gatekeeper. After authentication, the gatekeeper passes the network connection to another program called the GARA service. This GARA service uses the Local Resource Manager (LRAM) API to interact with the local resource managers. Each GARA API call is a transaction with the gatekeeper, so each call benefits from the security and remote access capability.

The GARA API allows users to request callbacks that inform the user when changes to the reservation occur. These do not use the gatekeeper for callbacks, but retain the connection originally opened to the gatekeeper, but redirected to another program that provides the callbacks.

5.3 Mapping of Service Requests to Resource Managers

As mentioned above, the GARA service uses the LRAM API. This is similar to the GARA API, but it does not provide remote access or security. It does provide an abstract interface to the resource managers so that the GARA service does not require intimate knowledge of different resource managers. Instead, the LRAM API knows the details of speaking to the resource managers.

The LRAM is implemented as a series of libraries that can be used to communicate with different resource managers. While it was written in C, it is essentially an object-oriented framework that allows for abstract interfaces to a variety of implementations.

6. FUTURE WORK

Although GARA has been demonstrated to be an effective system [SFRW00, FRS00, San03, Roy01], it is a first-generation architecture, and there are important improvements we are planning for GARA.

To improve GARA's functionality in Grid environments, we will extend its uniform API, particularly to include a two-phase commit protocol, which is essential for reliability. We also intend to move GARA to the Open Grid Services Architecture (OGSA) [FKNT02, TCF+03]. For a detailed discussion about this convergence refer to [CFK+02].

GARA needs a better mechanism for helping users find reservations. Currently GARA publishes information about what reservations may be available in an information service, and it is up to the user to ask for a reservation that may work. GARA can only respond with "yes" or "no" when a request is made. A more effective approach is the sort used by *ClassAd matchmaking* [RLS98]. Such a system would allow users to say, "I need 10-15 megabits per second for an hour tomorrow afternoon".

Enhancing GARA's resource managers is driven by user demand. The improvement of the network resource manager is still an ongoing effort. The Path Allocation in Backbone networks (PAB) project [SIF03] funded by the German Research Network (DFN) and the Federal Ministry of Education and Research (BMBF) is developing an optimized admission control procedure. The embedded advanced traffic engineering capabilities can be optimized based on an emerging simulation tool. We intend to integrate the results within GARA's admission control procedures.

Extending GARA's reach to other types of QoS, particularly disk space reservations would be very useful. Our prototype disk space resource manager was sufficient to show that it was interesting, and we believe that interacting with a system such as NeST (see Chapter 21) would work well.

7. CONCLUSIONS

GARA is an architecture that provides a uniform interface to varying types of QoS, and allows users to make advance reservations. In our experience, GARA has provided a useful framework in which to experiment with different types of QoS. In particular, we have experimented heavily with network QoS, but have also investigated providing reservations for computers, CPU time, disk space, and graphic pipelines. We believe that GARA is a promising platform for future investigations into quality of service.

For those interested in further discussion of this topic, please see Chapter 8 and [Roy01, San03].

Chapter 24

QOS-AWARE SERVICE COMPOSITION FOR LARGE-SCALE PEER-TO-PEER SYSTEMS

Xiaohui Gu and Klara Nahrstedt
Department of Computer Science, University of Illinois at Urbana-Champaign

Abstract In this chapter, we present a scalable QoS-aware service composition framework, SpiderNet, for large-scale peer-to-peer (P2P) systems. The SpiderNet framework comprises: (1) service path selection, which is responsible for selecting and composing proper service components into an end-to-end service path satisfying the user's functional and quality requirements; (2) service path instantiation, which decides the specific peers, where the chosen service components are actually instantiated, based on the distributed, dynamic, and composite resource information; and (3) benefit-driven clustering, which dynamically organizes a large-scale P2P system into an overlay network, based on each peer's benefit. Conducting extensive simulations of a large-scale P2P system (10^4 peers), we show that SpiderNet can achieve much higher service provisioning success rate than other common heuristic approaches.

1. INTRODUCTION

In a peer-to-peer (P2P) computing Grid, computers, which are called *peers*, communicate directly among themselves and can act as both clients and servers. Recently, P2P systems have drawn much research attention with the popularity of P2P file sharing systems such as Gnutella [AH00] and Napster [Nap]. Much research work has been done to provide scalable P2P data lookup service [SMK+01, RFH+01]. However, little has been done to solve the P2P service composition problem, which is important for the P2P user to utilize a wealth of application services (e.g., media transcoding, language translation) in P2P systems.

Although service composition has been addressed under different context [CCF+01, RK03, XN02, GN02], previous approaches present the following major limitations when applying to P2P systems. First, they often assume a

global view of the entire system in terms of performance information, which is impractical for large-scale, decentralized P2P systems. Second, they do not consider arbitrary peer arrivals/departures in P2P systems.

In this chapter, we address the above challenges by proposing a QoS-aware, P2P service composition framework, SpiderNet, which is designed in a decentralized and self-organizing fashion. The SpiderNet framework comprises three major components: (1) service path selection; (2) service path instantiation; and (3) benefit- driven peer clustering.

The service path selection component is responsible for choosing correct service components, available in the current P2P system, to compose an end-to-end service path satisfying the user's functional and quality requirements. To satisfy the user's end-to-end quality requirements for the composed application, we provide a quality consistency check algorithm to guarantee the consistency of quality parameters between any two interacting service components.

In P2P systems, each service component can be mapped to multiple service instances, which are provided by different peers. Hence, we introduce a service path instantiation mechanism to achieve load balancing and improve the overall resource utilization in P2P systems. For this purpose, each peer needs to monitor the dynamic and distributed resource information (e.g., CPU load and network bandwidth).

However, for scalability, each peer can maintain the resource information for only a few closely related peers that are called its neighbors. It is important for each peer to wisely choose its neighbors for efficiency. Hence, we introduce the *benefit-driven peer clustering* to dynamically organize a large-scale P2P system into a virtual service overlay network based on each peer's benefit.

The rest of the chapter is organized as follows. We introduce the SpiderNet system models in Section 2. Section 3 describes the design details and algorithms of the SpiderNet framework. In Section 4, we present the performance evaluation of SpiderNet framework using extensive large-scale simulations. Finally, we conclude this chapter in Section 5.

2. SYSTEM MODELS

We first state a number of key assumptions made by the SpiderNet system. Then, we present a component-based application service model to characterize the QoS sensitive distributed applications such as *video-on-demand* and *content delivery*. Finally, we present SpiderNet service composition model for large-scale P2P systems

2.1 Assumptions

First, we assume that service components' meta-data information such as QoS specifications, are accessible to SpiderNet in P2P systems. Several QoS programming frameworks and specification languages have been proposed to allow application developers to provide such QoS specifications [FK98b, LBS+98, GNY+02]. Second, we assume that there exists a translator that can map the application-specific QoS specifications into its resource requirements such as CPU cycles and network bandwidth. Such a translation procedure can be performed by using two major approaches: (1) analytical translations; and (2) offline or online resource profiling, which have been addressed by a wealth of research work [Abd00, LN00, WNGX01, WGN02]. Third, this chapter only addresses the session setup problem to deliver a composed application. We assume that the runtime QoS provisioning of the composed application can be guaranteed by reserving resources for each chosen service instances using resource reservation systems such as GARA presented in Chapter 9.

2.2 Composed Application Service Model

Each composed application delivery is represented by a chain of composable service components, called *service path*. Each service component accepts input data with a quality level Q^{in} and generates output with a quality level Q^{out}, both of which are vectors of application-specific QoS parameters, such as media data format (e.g., MPEG, JPEG) and video frame rate (e.g., [0,20]fps). In order to process input data and generate output data, a specific amount of resources R is required, which is a vector of required end-system resources (e.g., cpu, memory). The network resource requirement, such as bandwidth $b_{A,B}$, is associated with the link between two communicating components A and B. Figure 24.1(a) illustrates such a characterization in terms of QoS parameters and resources.

Formally, we define the input QoS vector Q^{in}, output QoS vector Q^{out}, and resource requirement vector R as follows:

$$Q^{in} = [q_1^{in}, q_2^{in}, ..., q_n^{in}] \tag{24.1}$$

$$Q^{out} = [q_1^{out}, q_2^{out}, ..., q_n^{out}] \tag{24.2}$$

$$R = [r_1, r_2, ..., r_m] \tag{24.3}$$

Intuitively, if a service component A is connected to a service component B, the output QoS of A (Q_A^{out}) must match the input QoS requirements of component B (Q_B^{in}). In order to formally describe this QoS consistency requirements, we define an inter-component relation "$\preceq$", called *satisfy*, as follows:

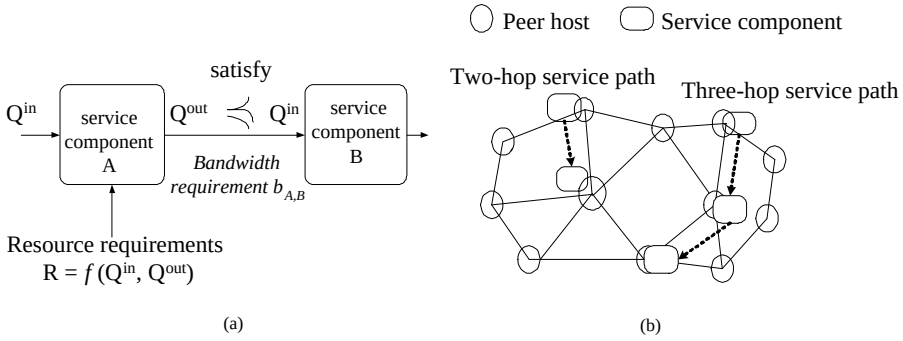


Figure 24.1. Illustration of the application service model for P2P systems.

$Q_A^{out} \preceq Q_B^{in}$ if and only if

$$\forall i, 1 \leq i \leq Dim(Q_B^{in}),\ \exists j, 1 \leq j \leq Dim(Q_A^{out}),$$
$$q_{Aj}^{out} = q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ single\ value;$$
$$q_{Aj}^{out} \subseteq q_{Bi}^{in},\ \ if\ q_{Bi}^{in}\ is\ a\ range\ value. \tag{24.4}$$

The “Dim(Q)" represents the dimension of the vector “Q". The single-value QoS parameters include data format, resolution, and others. The range-value QoS parameters include frame rate ([10fps,30fps]) and others.

If a composed application service involves n service components, we call it an n-hop service path, illustrated in Figure 24.1 (b). For example, a simple content distribution application represents a 2-hop service path: content source $\rightarrow$ content consumer. Note that the hop count represents the number of application-level connections. An n-hop service path consists of $n-1$ virtual application-level connections. Each application-level connection can include many network-level hops depending on the network distance between two peers.

2.3 SpiderNet Service Composition Model

The SpiderNet service composition model can dynamically compose and instantiate a service path to deliver distributed composed applications in P2P systems. For example, Mary wants to stream her honeymoon trip video to her friend Jane who is having lunch in a restaurant and only has a smart cell-phone on hand. Thus, Mary asks the service composition framework to compose and instantiate a composed application with the service path: MPEG-4 encoding $\rightarrow$ scaling for small screen $\rightarrow$ MPEG-4 decoding, to stream the video to Jane.

The SpiderNet service composition model exploits the inherent redundancy property of P2P systems. The redundancy property is represented by the facts

that: (1) the same application service (e.g., video player) can be provided by different service components (e.g., real player, windows media player), each of which has different Q^{in} and Q^{out} parameters; and (2) the same service component (e.g., real player) can have multiple service instances deployed on different physical peer hosts. Hence, the two-tier service composition model solves two key problems for composing services at setup time of each application session: (1) how to choose correct service components to compose an end-to-end service path according to the user function and quality requirements; and (2) how to select proper peers to instantiate the chosen service components according to the current system load and resource availabilities of all candidate peers.

The service composition model includes two cooperating layers: (1) *service path selection*; and (2) *service path instantiation*. Upon receiving a user request, the service path selection component first chooses correct service components to establish a QoS consistent service path satisfying the user's functional and quality requirements, according to the equation 24.4. Next, the service path instantiation component is responsible for selecting among candidate peers, where the chosen service components are instantiated, according to the dynamic, distributed resource information and uptime of each candidate peer. For scalability, the service instantiation decision is made by a distributed algorithm using only local resource information at each candidate peer. The rational of using a two-layer model is that: (1) it allows us to simultaneously achieve both application-specific QoS assurances and load balancing by decoupling application-specific QoS management from resource management; (2) it allows us to achieve scalability by eliminating the requirements of global resource availability information of the whole P2P system. Such a service composition model would be beneficial to a range of quality-sensitive distributed applications in P2P systems such as the critical content delivery and Internet multimedia streaming applications.

3. SYSTEM DESIGN

This section describes the design details of the scalable QoS-aware service composition framework SpiderNet. The SpiderNet framework enables high performance distributed application delivery in P2P systems by meeting the following challenges: (1) Decentralization. The solution must be fully distributed and must only involve local computation based on local information; (2) Scalability. The solution must scale well in the presence of large number of peer nodes; (3) Efficiency. The solution should be able to utilize resource pools provided by P2P systems efficiently so that it can admit as many user requests as possible; and (4) Load balancing. Although each peer makes its own service instantiation decisions based on only local information, the so-

lution should achieve the desired global properties such as load balancing in P2P systems. We first describe the design details for the service path selection and service path instantiation tiers, respectively. Then we introduce the benefit-driven peer clustering algorithm.

3.1 Service Path Selection

In order to compose an end-to-end service path satisfying the user's function and quality requirements, the service path selection tier needs to execute the following protocol steps:

- **Acquire user requirements.** In SpiderNet, the user request is specified by two parts: (1) the requested application in terms of a functional graph; and (2) end-to-end quality requirements. The user can directly specify the functional graph, such as video server $\rightarrow$ image enhancement $\rightarrow$ video player. Alternatively, the user can use the application name such as video-on-demand or distance learning. The functional graph can then be created by various tools, such as Q-compiler [WGN02] and SWORD [PF02]. The end-to-end quality requirements can be expressed by application-specific QoS parameters such as frame rate, response time, and availability.

- **Retrieve service component meta-data information.** Once the user requirements are acquired, a discovery service, based on decentralized P2P data lookup systems (e.g., Chord [SMK+01], CAN [RFH+01]), is invoked to retrieve the meta-data information such as locations (e.g., IP addresses) and QoS specifications (e.g., Q^{in}, Q^{out}, R) of all service component candidates for each required service.

- **Compose an end-to-end qualified service path.** Because of the inherent redundancy property of P2P systems, multiple service components can be discovered in P2P systems, which all deliver the same service functionality but have different Q^{in} and Q^{out} parameters. Thus, we need to select proper service components to compose an end-to-end qualified service path that satisfies the user's quality requirements. We will discuss this step with more details later in this section.

- **Deliver the composed service path to the service path instantiation tier.** After the third step, a qualified service path is generated and delivered to the service path instantiation tier. This tier is responsible for instantiating all service components on the composed service path according to the distributed resource availabilities in the P2P system. We will present the service path instantiation tier in detail in the next section.

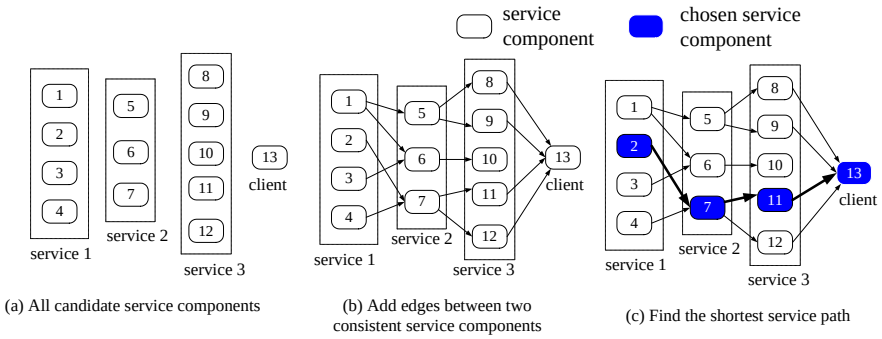


Figure 24.2. Illustration of service path selection.

Among the above four steps, the third step is the key part of the *service path selection*. It addresses two problems: (1) the composed service path must be QoS consistent, which means that the Q^{in} of a service component must be satisfied by the Q^{out} of its predecessor (see equation 24.4); (2) If multiple QoS consistent service paths exist, the *service path selection* component selects the shortest one that has the minimum aggregated resource requirements. Thus, the overall workload of a P2P system is minimized.

We propose the *QCS* algorithm, the acronym for the *QoS consistent and shortest* algorithm, to solve the service path selection problem. It includes the following major operations, illustrated by Figure 24.2: (1) generate a candidate service graph, which lists all candidate service components for each required service, illustrated by Figure 24.2 (a); (2) start from the client component, check the QoS consistency between the current examined service component and all of its candidate predecessors on the service path. Since the output QoS of the client component represents the user QoS requirements, the QoS parameter consistency check and adjustment start from the client component in order to guarantee the output QoS of the client component. If the Q^{out} of the predecessor satisfies the Q^{in} of the current examined service component, we add a directed edge from the predecessor to the current examined service component, illustrated by Figure 24.2 (b); (3) define the cost value on each edge from A to B as a resource tuple $(R^A, b_{A,B})$, where $R^A = [r_1^A, r_2^A, ..., r_m^A]$ represents the required end-system resources (e.g., CPU, memory) of node A and $b_{A,B}$ represents the required network bandwidth from A to B. Because the client component is the common part of all candidate service paths, its required end-system resources are not included in the calculation; and (4) select the shortest path from all possible service paths using the Dijkstra's algorithm, illustrated by Figure 24.2 (c).

In order to apply the Dijkstra algorithm, we define the comparison of any two resource tuples as follows:

DEFINITION 3.1 Given two tuples (R^A, $b_{A,B}$) and (R^C, $b_{C,D}$), they can be compared in the following way:

$$\sum_{i=1}^{m} w_i \cdot \frac{r_i^A - r_i^C}{r_i^{max}} + w_{m+1} \cdot \frac{b_{A,B} - b_{C,D}}{b^{max}} > 0$$
$$\Rightarrow (R^A, b_{A,B}) > (R^C, b_{C,D}) \quad (24.5)$$

where r_i^{max}, b^{max} represent the maximum values of the resource requirements for the ith end-system resource type and network bandwidth, respectively, w_i ($1 \leq$ i $\leq$ m+1, *m* is the number of all end-system resource types) are nonnegative values such that

$$\sum_{i=1}^{m+1} w_i = 1 \quad (24.6)$$

For any end-system resource type r_i (e.g., CPU, disk storage), $\frac{r_i^A - r_i^B}{r_i^{max}}$ is a normalized value ranging between -1 and 1, while w_i represents its significance. Generally, we assign higher weights for more critical resources. For the network resource type, $\frac{b_{A,B} - b_{C,D}}{b^{max}}$ is a normalized value between -1 and 1, where w_{m+1} represents the importance of the network resource.

Thus, if we associate each edge in Figure 24.2 (b) with a cost value (R^x, $b_{x,y}$), we can use the Dijkstra algorithm to find a shortest path from all service components for the source service (e.g., service 1 in Figure 24.2) to the client component. For example, Figure 24.2 (c) illustrates such a QoS consistent shortest path (thick line). The computation complexity of the service path selection algorithm is $O(KV^2)$, where V is the total number of all candidate service components, and K is the number of candidate service components for the source service. The above service path selection algorithm is computed locally and takes less than a few seconds on average.

The final result generated by the service path selection is the functional composition of a service path that satisfies the user-specific functional and quality requirements and also has the minimum aggregated resource requirements. However, because of the redundancy property of P2P systems, each chosen service component can be provided by multiple peers. Hence, we introduce the service path instantiation mechanism to select proper peers to instantiate the service components on the functional composition of the service path.

3.2 Service Path Instantiation

The service path instantiation component selects proper peers to instantiate the chosen service components based on the resource requirements of the ser-

vice components, the distributed resource availabilities, and uptime of different candidate peers. However, there are two difficulties for selecting peers in P2P systems: (1) each peer can only maintain the up-to-date resource and uptime information for a small number of peers because of the scalability requirement; and (2) the required information for peer selection is distributed and includes multiple factors such as the peer's uptime, end-system resources, and network bandwidth. Hence, we make the following design decisions for the scalable and decentralized service path instantiation.

- **Distributed and hop-by-hop service path instantiation.** Since it is impossible for each peer to have the global view of the up-to-date performance information, the service path instantiation has to be performed in a distributed and hop-by-hop manner using a greedy algorithm. Each peer, starting from the client, chooses the most suitable peer among all peers that provide its preceding service component on the service path. The peer selection is based on the current peer's locally maintained performance information of those candidate peers. Then, the peer selected in this step will be responsible for choosing its preceding peer and so on. Note that the service path instantiation is performed in the reverse direction of the service path. In analogy, each step in service path instantiation resembles the server selection in the conventional client-server application model. Although the above decentralized greedy algorithm can give a sub-optimal service path, our experimental results show that it performs well in general case, which will be presented in the Section 4.

- **An integrated and configurable metric for peer selection.** Now we focus on the peer selection at each single step, where current peer needs to choose its preceding peer according to its locally maintained performance information. First, we select among all candidate peers according to the candidate peer's uptime and the expected application session duration. To guarantee the 100% availability of the composed application, the estimated peer uptime must be greater than the expected application session duration. Otherwise, if any selected peer leaves during the session, the application delivery fails. Second, the candidate peer's resource availability must be greater than the service instance's resource requirements. Third, if multiple peers qualify, we use an integrated and configurable metric Φ to choose the best one.

The metric Φ is proposed to solve the problem of composite-value decision-making for peer selection and achieving load balancing in P2P systems. We define $RA = [ra_1, ..., ra_m]$ as the available end-system resources (e.g., CPU, memory) of the candidate peer, and ba as the end-to-end available network

bandwidth from the candidate peer to the current peer. RA represents the same set of resources as the resource requirement vector $R = [r_1, ..., r_m]$ and obeys the same order. Both RA and ba are monitored locally on the current peer through proactive probing. We define b as the network bandwidth requirement from the preceding service component to the current one. Based on the above definitions, the metric Φ can be defined as follows:

$$\Phi = \sum_{i=1}^{m} \omega_i \cdot \frac{r_i}{ra_i} + \omega_{m+1} \cdot \frac{b}{ba} \tag{24.7}$$

where ω_i ($1 \leq$ i $\leq$ (m+1)) are nonnegative values so that

$$\sum_{i=1}^{m+1} \omega_i = 1 \tag{24.8}$$

We choose the best candidate peer that minimizes the value of the metric Φ. For any end-system resource r_i (e.g., CPU, memory), $\frac{r_i}{ra_i}$ is a load balancing ratio, which represents that the more abundant the candidate peer's resource availability is, the smaller is the load balancing ratio, the more advantageous it is to select this peer for achieving load balancing in P2P systems. $\frac{b}{ba}$ represents the same meaning for the network bandwidth. In order to allow customization, we introduce ω_i $(1 \leq i \leq m+1)$ to represent the importance of the ith resource type in making the peer selection decision. They can be adaptively configured according the application's semantics and user's preference. For example, we can assign a higher weight to the CPU resource if the application is CPU-intensive.

However, for scalability, it is impossible for the peer to keep the resource information of all other peers. Hence, in the case that the resource information of the candidate peers is not available, the peer selection falls back to a random policy. At the end of the service path instantiation phase, the distributed application delivery can be started along the chain of the selected peers.

3.3 Benefit-Driven Peer Clustering

In SpiderNet, we require each peer to directly measure the performance information of its neighbors using proactive probing. In order to achieve scalability and avoid flooding of probing messages, we assume that each peer can only probe a small number of peer neighbors whose resource information is most beneficial to the peer. We say that a neighbor is beneficial to the peer if the neighbor provides the service needed by the peer.

We propose *benefit-driven peer clustering*, which dynamically organizes a large-scale P2P system into a dynamic overlay network. Each peer dynamically selects a few other peers as its neighbors based on its own benefits. To be

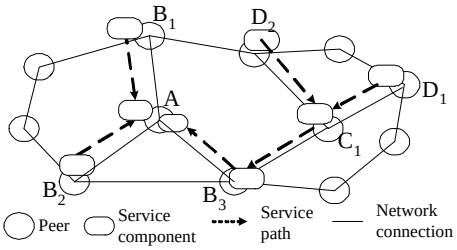


Figure 24.3. Illustration of benefit-driven peer clustering.

specific, if peer B provides services that peer A needs, B should be considered as A's candidate neighbor. If the service that peer B provides is i-hop away from peer A on the service path, B is defined as A's i-hop neighbor candidate. Moreover, if the service that peer B provides is part of an application that A needs, B is defined as the A's direct neighbor candidate. Otherwise, B is regarded as A's indirect neighbor candidate. For example, in Figure 24.3, B_1, B_2, and B_3 are A's 1-hop direct neighbor candidates. C_1 and D_1 are A's 2-hop and 3-hop direct neighbor candidates, respectively. C_1 and D_1 are B_3's 1-hop and 2-hop indirect neighbor candidates, respectively. For scalability, we define an upper-bound for the number of neighbors that can be actively measured. Under the upper-bound constraint, a peer first selects to monitor its 1-hop direct neighbor candidates, then 1-hop indirect neighbor candidates, then 2-hop direct neighbor candidates and so on.

Based on the above benefit-driven peer neighbor selection, the neighbor list of each peer is dynamic and depends on the results from the service path selection. To be specific, after the service path selection component generates a service path, the peer first updates its direct neighbor list to include those direct neighbor candidates that provide the services on the service path. Then the peer notifies those direct neighbor candidates to update their indirect neighbor list to include those peers that provide their preceding services on the service path. The neighbor list at each peer is maintained as soft state information. The above neighbor resolution messages are sent periodically to refresh the soft states as long as the service path is valid and needed.

4. PERFORMANCE EVALUATION

We evaluate the performance of SpiderNet framework by simulations. We first describe our evaluation methodology. Then we present and analyze the simulation results.

4.1 Evaluation Methodology

We simulate a large-scale P2P system with 10^4 peers. Each peer is randomly assigned an initial resource availability $RA = [cpu, memory]$, ranging from [100,100] to [1000,1000] units. Different units reflect the heterogeneity in P2P systems. The end-to-end available network bandwidth between any two peers is defined as the bottleneck bandwidth along the network path between two peers, which is initialized randomly as 10M, 500k, 100k, or 56k bps. The network latency between two peers is also randomly set as 200, 150, 80, 20, or 1 ms according to the recent Internet measurement study [LB01].

The *SpiderNet* algorithms are locally executed at each peer. They include processing user request, composing service paths, instantiating service paths, and periodically probing neighbors that are dynamically selected. The maximum number of neighbors that a peer can probe is 100 in order to control the probing overhead within $100/10000 = 1\%$. During each minute, a number of user requests are generated on each peer. The user request is represented by any of the 10 distributed applications whose service paths have 2 to 5 hops. Each application session has random length ranging from 1 to 60 minutes. Each service instance is randomly assigned values for its Q^{in}, Q^{out} and R parameters. The number of different service components for each service is randomly set between 10 to 20. The number of peers, which all provide a specific service component, is randomly set between 40 to 80. The importance weights for different resource types are uniformly distributed.

The metric that we use for evaluating the performance of the *SpiderNet* algorithm, is the *provisioning success rate*, called ψ. A service provisioning is said to be successful if and only if (1) A qualified service path can be found; and (2) the resource requirements of the service path are always satisfied during the entire session. The metric ψ is defined as the number of successful service deliveries over the total number of user requests. A higher provisioning success rate represents improved service availability and load balancing in P2P systems. For comparison, we also implement two common heuristic algorithms: *random* and *fixed*. The random algorithm randomly chooses a QoS consistent service path (without considering the aggregated resource consumption) and randomly selects peer candidates for instantiating the service path. The fixed algorithm always picks the same qualified service path for a specific application request, and chooses the dedicated peers to instantiate the service path.

The major goals of the SpiderNet framework are to achieve better performance and higher tolerance to the topology variation of P2P systems. Hence, we conduct two sets of experiments to evaluate how well SpiderNet framework meets these goals. In the first set of experiments, we study the performance of SpiderNet compared to the *random* and *fixed* algorithms, for a fixed P2P system. In the second set of experiments, we consider the topology variation in

P2P systems. We study the resilience of SpiderNet to the topology variation, compared with the *random* and *fixed* algorithms. In both sets of experiments, we use ψ as the performance metric.

4.2 Results and Analysis

Figure 24.4 and Figure 24.5 show the simulation results for the first set of experiments, which do not consider the topology variation in P2P systems. In Figure 24.4, the X axis represents different user request rate, calculated by the

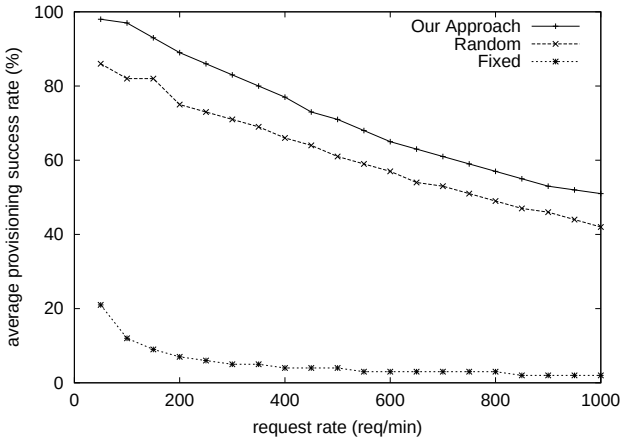


Figure 24.4. Average provisioning success rate under different load conditions, over a period of 400 minutes, without P2P network topology variation.

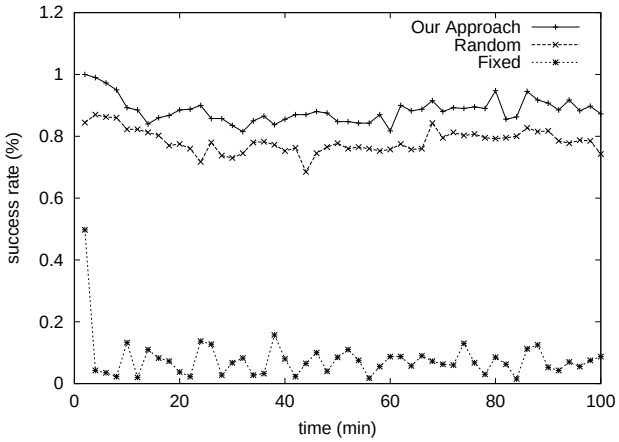


Figure 24.5. Provisioning success rate within a period of 100 minutes, for the request rate = 200 req/min without P2P network topology variation.

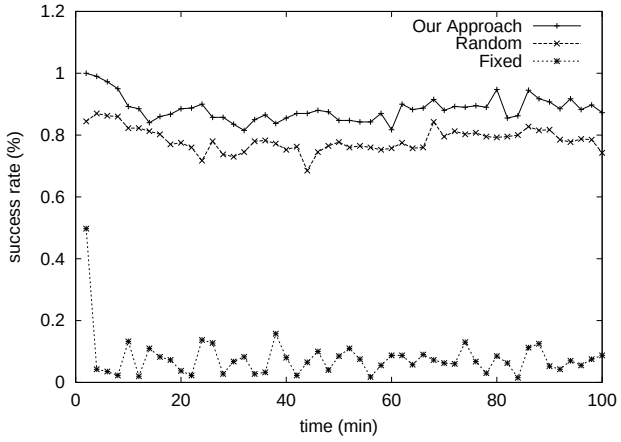


Figure 24.6. Average provisioning success rate under different topology variation rate (peers/min) over a period of 60 minutes with request rate = 100 req/min.

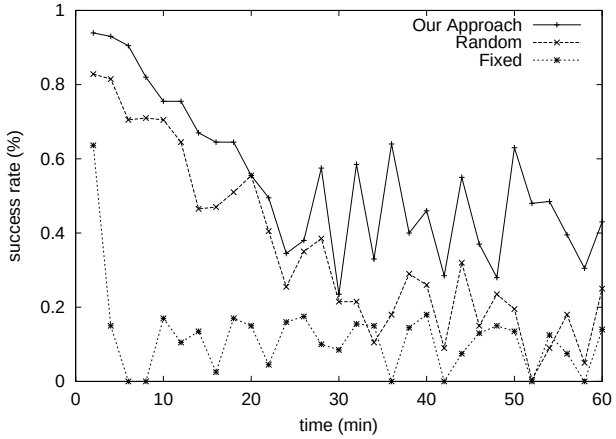


Figure 24.7. Provisioning success rate within a period of 60 minutes, for the request rate = 100 req/min, and topology variation rate = 100 peers/min.

number of requests per minute. The range of request rate is selected to reflect different workload of the P2P system. The Y axis shows the average service provisioning success rate (ψ) achieved by the SpiderNet, random and fixed algorithms, respectively. Each average success rate value is calculated and averaged over a period of 400 minutes. The results show that the average success rate of the SpiderNet algorithm is always higher than the other two heuristics under all request rates. The reason is that the SpiderNet algorithm reduces the overall workload of the P2P system by choosing the service path with mini-

mum aggregated resource consumption. Moreover, the SpiderNet algorithm achieves better load balancing by selecting the peers that have the most abundant resources. The random algorithm achieves lower success rate than the SpiderNet algorithm, but much higher success rate than the fixed algorithm. Such results reflect the prominent advantage of the P2P system brought by its redundancy property.

Figure 24.5 gives a more detailed picture about the success rate under a particular request rate (200 requests/minute). Each run of simulation lasts 100 minutes and the success rate value is sampled every 2 minutes. We observe that the success rate of SpiderNet is consistently higher than those of random and fixed. The former may be higher than the other two by as much as 15% and 90%, respectively.

The second set of simulation results is illustrated in Figure 24.6 and Figure 24.7. In this set of experiments, we consider the topology variation in P2P systems, which is measured by the number of peers leaving or arriving every minute. Figure 24.6 shows the average success rate achieved by SpiderNet, random and fixed under different topology variation rates. Each run of simulation lasts 60 minutes under a fixed request rate (100 requests/minute). Figure 24.7 shows the success rate for a fixed topology variation rate (100 peers/minute) with a particular request rate (100 requests/minute). Both simulation results show that SpiderNet can best tolerate topology variations and uniformly achieve the highest provisioning success rate. The reason is that when SpiderNet selects among peer candidates for instantiating a service instance, it considers the peer uptime measured by the duration that the peer has remained connected to the P2P system, but random and fixed algorithms do not. The SpiderNet always chooses peers already connected to the P2P system for an average uptime that is longer than the expected session duration, in hope that those peers will stay connected to the P2P system for at least the same uptime duration. However, such a heuristic cannot be true all the time. Hence, the results in Figure 24.6 and Figure 24.7 show that the performance of P2P systems is very sensitive to the topology variation, even with a small number of peer arrivals/departures ($\leq$ 2% total peers). Under such circumstances, we do need runtime service path recovery to improve the performance.

5. CONCLUSION

We present a scalable P2P service composition framework, called SpiderNet, for providing dynamic composed QoS-sensitive applications in large-scale P2P systems. The major contributions of the chapter are as follows: (1) identify and solve two key problems, service path selection and service path instantiation in an integrated service composition framework; (2) present a resource- and quality-aware service path selection algorithm, which can gener-

ate a QoS consistent service path with minimum aggregated resource requirements; (3) provide a fully distributed peer selection scheme for instantiating the composed service path; and (4) propose a benefit-driven peer clustering algorithm to organize a large-scale P2P system into an efficient service overlay network. We implement a large-scale simulation testbed and our extensive simulation results show that SpiderNet framework can achieve (1) end-to-end QoS consistency while composing service paths; and (2) much better load balancing and overall resource utilization in P2P systems than other common heuristics.

Acknowledgments

This work was supported by the NASA grant under contract number NASA NAG 2-1406, NSF under contract number 9870736, 9970139, and EIA 99-72884EQ. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NASA, NSF or U.S. Government.

VII

RESOURCE MANAGEMENT IN PEER-TO-PEER ENVIRONMENTS

Chapter 25

A PEER-TO-PEER APPROACH TO RESOURCE LOCATION IN GRID ENVIRONMENTS

Adriana Iamnitchi[1] and Ian Foster[1,2]
[1]*Department of Computer Science, The University of Chicago*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract Resource location (or discovery) is a fundamental service for resource-sharing environments: given desired resource attributes, the service returns locations of matching resources. Designing such a service for a Grid environment of the scale and volatility of today's peer-to-peer systems is not trivial. We explore part of the design space through simulations on an emulated Grid. To this end, we propose four axes that define the resource location design space, model and implement an emulated Grid, evaluate a set of resource discovery mechanisms, and discuss results.

1. INTRODUCTION

When dealing with large sets of shared resources, a basic problem is locating resources in the absence of a naming scheme, that is, dealing with requests that specify a set of desired attributes ("Linux machine with more than 128 MB of available memory") rather than a globally unique identifier (such as "ficus.cs.uchicago.edu"). Attribute-based search is challenging when resource attributes can vary over time (e.g., CPU load, available bandwidth, even software versions) and when the number of resources is large and/or dynamic (as resources join, leave, or fail).

We study the resource discovery problem in a resource-sharing environment that combines the complexity of Grids with the scale and dynamism of peer-to-peer communities. While the two environments have the same final objective—to pool large sets of resources—they emerged from different communities, and hence their current designs highlight different requirements. We believe that the design objectives of the two environments will eventually con-

verge. Consequently, it is important to analyze, compare, and contrast their current requirements and characteristics.

To this end, we discuss the resource location problem in the context of the two resource-sharing environments (Section 2). We identify four critical design objectives for resource discovery (Section 3), and we present a scheme for characterizing resource discovery mechanisms (Section 4) that defines the design space and provides the basis for comparing existing solutions. We then describe an emulated large-scale resource-sharing environment (Section 5), which we use for a preliminary performance evaluation of resource discovery techniques (Section 6). We conclude with a brief summary.

2. GRID AND PEER-TO-PEER ENVIRONMENTS

In recent years significant interest has focused on two resource-sharing environments: Grids and peer-to-peer (P2P) systems. The two systems have followed different evolutionary paths. Grids have incrementally scaled the deployment of relatively sophisticated services, connecting small numbers of sites into collaborations engaged in complex scientific applications. P2P communities, on the other hand, have developed rapidly around unsophisticated but popular services such as file sharing, focusing on scalability and support for intermittent user and resource participation. As a result of these different evolutionary paths, the two systems differ in three respects: target communities, resources, and applications. We discuss each of these below.

Despite these differences, however, we maintain that the two environments are in fact concerned with the same general problem, namely, resource sharing within *virtual organizations (VOs)* that link resources and people spanning multiple physical organizations. Moreover, the two environments seem likely to converge in terms of their concerns, as Grids scale and P2P systems address more sophisticated application requirements [FI03].

2.1 Target Communities and Incentives

The development and deployment of Grid technologies were motivated initially by the requirements of professional communities to access remote resources, federate datasets, and/or to pool computers for large-scale simulations and data analysis. Participants in contemporary Grids form part of established communities that are prepared to devote effort to the creation and operation of required infrastructure and within which exist some degree of trust, accountability, and opportunities for sanctions in response to inappropriate behavior.

In contrast, P2P has been popularized by grass-roots, mass culture (music) file-sharing and highly parallel computing applications [ACK+02, AK02] that scale in some instances to hundreds of thousands of nodes. The "communities" that underlie these applications comprise diverse and anonymous individuals with little incentive to act cooperatively and honestly. Thus, for example, we find that in file-sharing applications, there are few providers and many consumers [AH00]; the operators of SETI@home [SET] devote significant effort to detecting deliberately submitted incorrect results; and people tend to intentionally misreport their resources [SGG02].

The two target communities differ in scale, homogeneity, and the intrinsic degree of trust. The natural tendency of Grids to grow, however, will inevitably lead to less homogeneous communities and, consequently, to smaller degrees of trust. Participation patterns will change also with scale: intermittent participation is likely to become the norm. All these characteristics have a strong impact on defining the assumptions one can make (or, rather, cannot) about the environment. We need support for volatile user communities; and we need to deal with the lack of incentives for and interest in centralized, global administrative control.

2.2 Resources

In general, Grid systems integrate resources that are more powerful, more diverse, and better connected than the "desktop at the edge of the Internet" [Shi00] that constitutes a typical P2P resource. A Grid resource might be a cluster, storage system, database, or scientific instrument of considerable value that is administered in an organized fashion according to some well-defined policy. This explicit administration enhances the resource's ability to deliver desired qualities of service and can facilitate, for example, software upgrades, but it can also increase the cost of integrating the resource into a Grid. Explicit administration, higher cost of membership, and stronger community links within scientific VOs mean that resource availability tends to be high and uniform.

In contrast, P2P systems deal with intermittent participation and highly variable behavior. For example, one study of Mojo Nation [WO02] showed that a node remained connected on average for about 28% of time. Moreover, the connection time distribution was highly skewed, with one sixth of the nodes remaining always connected.

Large-scale Grids will borrow some of the characteristics of today's P2P systems in resource participation: unreliable resources and intermittent participation will constitute a significant share. At the same time, Grid resources will preserve or increase their diversity. Consequently, services - and resource discovery in particular - will have to tolerate failures and adapt to dynamic resource participation.

2.3 Applications

The range and scope of scientific Grid applications vary considerably. Three examples that show the variety of deployed Grid applications are the HotPage portal, providing remote access to supercomputer hardware and software [TMB00]; the numerical solution of the long-open `nug30` quadratic optimization problem using hundreds of computers at many sites [ABGL02]; and the NEESgrid system that integrates earthquake engineering facilities into a national laboratory [PKF+01].

In contrast, P2P systems tend to be vertically integrated solutions to specialized problems: currently deployed systems share either compute cycles or files. Diversification comes from differing design goals, such as scalability [RFH+01, SMK+01, RD01], anonymity [CSWH00], or availability [CSWH00, KBC+00].

Grid applications also tend to be far more data intensive than P2P applications. For example, a recent analysis of Sloan Digital Sky Survey data [AZV+02] involved, on average, 660 MB input data per CPU hour; and the Compact Muon Solenoid [Neg94] data analysis pipeline involves from 60 MB to 72 GB input data per CPU hour. In contrast, SETI@home moves at least four orders of magnitude less data: a mere 21.25 KB data per CPU hour. The reason is presumably, in part at least, better network connectivity.

The variety of Grid applications requires significant support from services. Applications may use not only data and computational power, but also storage, network bandwidth, and Internet-connected instruments at the same time. Unlike in file-sharing systems such as Gnutella, this variety of resources requires attribute-based identification (such as "Linux machine with more than 1 GB memory"), since globally unique names are of no significant use. Also, Grid services must provide stronger quality-of-service guarantees: a scientist that runs data-analysis applications in a Grid is less willing to wait until data is retrieved than is a typical P2P user in search for music files.

3. REQUIREMENTS FOR RESOURCE DISCOVERY

As we have noted, we expect Grid and P2P systems to converge in a unified resource-sharing environment. This environment is likely to scale to millions of resources shared by hundreds of thousands of participants (institutions and individuals); no central, global authority will have the means, the incentive, and the participants' trust to administer such a large collection of distributed resources. Participation patterns will be highly variable: there will be perhaps a larger number of stable nodes than in today's P2P systems, but many resources will join and leave the network frequently. Resources will have highly diverse types (computers, data, services, instruments, storage space) and characteristics (e.g., operating systems, number of CPUs and speed, data of various

sizes, services). Some resources will be shared following well-defined public policies, such as "available to all from 6 pm to 6 am". Other resources will participate rather chaotically, for example, when idle. Technical support will be variable: some participants will benefit from technical support, whereas others will rely on basic tools provided by community (e.g., today's Gnutella nodes run various implementations of the protocol, each with its own particularities).

To perform efficiently in these conditions, a resource discovery mechanism should have the following features:

- Independence of central, global control. This is a departure from previous Grid solutions and a step toward the fully decentralized solutions typical of P2P approaches.

- Support for attribute-based search, a feature not found in current P2P solutions.

- Scalability, which becomes more important to the Grid community with the increase in scale and participation.

- Support for intermittent resource participation, a characteristic frequent in today's P2P systems but rare in current Grid solutions.

In the following sections, we propose a scheme for characterizing resource discovery techniques. Although we are interested primarily in designing mechanisms that adhere to the requirements above, our characterization scheme comprises a more general set of solutions.

4. RESOURCE LOCATION: COMPONENTS AND SOLUTIONS

We assume that every participant in the VO—institution or individual—publishes information about local resources on one or more local servers. We call these servers nodes, or peers. Nodes hence provide information about resources: some advertise locally stored files or the node's computing power, as in a traditional P2P scenario; others advertise all the resources shared by an institution, as in a typical Grid scenario.

From the perspective of resource discovery, a Grid is thus a collection of geographically distributed nodes that may join and leave at any time and without notice (for example, as a result of system or communication failure). Users send their requests to some known (typically local) node. Typically, the node responds with the matching resource descriptions if it has them locally; otherwise it processes the request, possibly forwarding it to one or more nodes.

4.1 Four Axes of the Solution Space

We partition a general resource discovery solution into four architectural components: membership protocol, overlay construction, preprocessing, and query processing. This partitioning helps us recognize the unexplored regions in the solution space. It also provides the basis for comparing previous solutions from both the P2P area and the traditional distributed computing domain, solutions that we present in Section 4.2.

4.1.1 Membership Protocol

The membership protocol specifies how new nodes join the network and how nodes learn about each others (we refer to the latter part of the membership problem as *peer discovery*, although it has multiple names in the literature [KP00]).

Imagine a graph whose vertices are peers and whose edges indicate whether vertices know of each other. Ideally, despite frequent vertex departures and joins, this graph is a clique; that is, every member of the network has accurate information about all participants. In practice, however, this situation is impossible [CHTCB96], and different protocols have been suggested, each involving different tradeoffs. For example, Gnutella uses an aggressive membership protocol that maintains the highly dynamic nodes in the membership graph connected, but at a significant communication cost [RFI02]. More scalable with the number of nodes are membership protocols based on epidemic communication mechanisms [GT92].

4.1.2 Overlay Construction

The overlay construction function selects the set of collaborators from the local membership list. In practice, this set may be limited by such factors as available bandwidth, message-processing load, security or administrative policies, and topology specifications. Hence, the overlay network often contains only a subset of the edges of the membership graph. For example, a Gnutella node maintains a relatively small number of open connections (the average is less than 10, with 3.4 measured as of May 2001 [RFI02]) but knows of many more peers (hundreds) at any given time.

The overlay topology has a significant effect on performance. For example, Barabási and Albert [BA99] show a strong correlation between robustness and the power-law topology; Adamic et al. [AHLP01] give a search algorithm that exploits the power-law topology in a cost-efficient way; and Kleinberg [Kle00] presents an optimal algorithm for search in small-world graphs with a particular topology (two-dimensional lattice) and knowledge about global properties, such as distance between any two nodes. On the other hand, a large number of dynamic, real networks, ranging from the Internet to social and biological net-

works, all exhibit the same power-law and small-world patterns (as surveyed in [AB02] and discussed in detail in [Bar02] and [Wat99]).

4.1.3 Preprocessing

Preprocessing refers to off-line processing used to enhance search performance prior to executing requests. For example, prefetching is a preprocessing technique, but caching is not. Another example of a preprocessing technique is dissemination of resource descriptions, that is, advertising descriptions of the local resources to other areas of the network for better search performance and reliability. A third example of preprocessing is rewiring the overlay network to adapt to changes in usage characteristics.

It is not obvious, however, that such preprocessing strategies work in the dynamic environments that we consider, in which resources may leave the pool and resource characteristics and user behavior may change suddenly. A recent result [CS02] shows that, in a static environment, the optimum replication of an item for search in unstructured networks is proportional to the square root of the popularity of that item.

4.1.4 Request Processing

The request-processing function has a local and a remote component. The local component looks up a request in the local information, processes aggregated requests (e.g., a request for A and B could be broken into two distinct requests to be treated separately), and/or applies local policies, such as dropping requests unacceptable for the local administration.

The remote component implements the request propagation rule. Request propagation is currently an active research topic in the P2P area [RFH+01, SMK+01, RD01, LCC+02, ZKJ01, IRF02, SMZ03]. In some cases, request propagation rules are dictated by other components of the resource discovery mechanism, as with distributed hash tables [RFH+01, SMK+01, RD01, ZKJ01], where the overlay and the propagation rules are strongly correlated. In an unstructured network, however, there are many degrees of freedom in choosing the propagation rule. Various strategies can be employed, characterized by the number of neighbors to which a request is sent and the way in which these neighbors are selected.

4.2 Previous Solutions to Resource Discovery

To provide a basis for our proposed characterization scheme, we discuss here existing solutions and related work from the perspective of the four architectural components presented above.

Many solutions to resource discovery presume the existence of globally unique names. In some cases, this naming scheme is natural (for example,

filenames used as global identifiers in P2P file-sharing systems); in others, it is created to support discovery. In the context of Grid computing it is difficult (if even possible) to define a global naming scheme capable of supporting attribute-based resource identification. We now present resource location solutions that exploit natural naming schemes.

Domain Name Service [Moc87] is perhaps the largest such system that provides name-based location information. Its hierarchical topology dictates the design of all four components: nodes (domains) join at a specified address in the hierarchy, the overlay function maintains the domain-based tree structure, requests are propagated upward in the hierarchy.

Recent contributions to name-based resource location solutions have been proposed in the context of P2P file-sharing systems, such as Gnutella and Napster. The basic mechanism used in Gnutella is *flooding*. Its flooding-based membership component manages a highly dynamic set of members (with median lifetime per node of about 60 minutes [SGG02]) by sending periodic messages. Its overlay function selects a fixed number of nodes from those alive (in most instances, the first nodes of the membership list). Flooding is also the core of the request-processing component: requests are propagated in the overlay until their time-to-live expires. No preprocessing component is active in Gnutella. Answers are returned along the same trajectory, from node to node, to the node that initiated the request. Gnutella's relatively good search performance (as measured in number of hops) is achieved at the cost of intensive network use [RFI02].

Napster uses a centralized approach: a file index is maintained at a central location, while real data (files) are widely distributed on nodes. The membership component is centralized: nodes register with (and report their locally stored files to) the central index. Hence, the request-processing component is a simple lookup in the central index. Napster does not use a distinct overlay function.

Distributed hash table structures such as CAN [RFH$^+$01], Chord [SMK$^+$01], Tapestry [ZKJ01], and Pastry [RD01] build search-efficient overlays. All have similar membership and request processing components, based on information propagation in a structured overlay. Differentiating these four solutions is the definition of the node space, and consequently the overlay function that preserves that definition despite the nodes' volatility: ring in Chord, d-coordinate space on a torus in CAN, Plaxton mesh [PRR97] in Pastry and Tapestry.

The file location mechanism in Freenet [CSWH00] uses a request-propagation component based on dynamic routing tables. Freenet includes both file management and file location mechanisms: popular files are replicated closer to users, while the least popular files eventually disappear.

The solutions discussed above are concerned with locating resources that inherently can be named. Solutions that create an artificial name have been pro-

posed for attribute-based service location (Ninja) and as location-independent identifiers (Globe).

In Ninja's service location service [GBHC00, GWvB$^+$01], services are named based on a most relevant subset of their attributes. Its preprocessing component disseminates lossy aggregations (summaries) of these names up a hierarchy. Requests are then guided by these summaries up or down the hierarchy, in a B-tree search fashion. The fix overlay function (hence, the construction of the hierarchy) is specified at deployment.

The location mechanism in Globe [vSHT99] is based on a search-tree-like structure where the search keys are globally unique names. Its naming service [BvST00] transforms a URL into a location-independent unique identifier.

Among the few attribute-based resource location services is Condor's Matchmaker (Chapter 9, [RLS98]). Resource descriptions and requests are sent to a central authority that performs the matching.

Lee and Benford [LB98] propose a resource discovery mechanism based on request propagation: nodes (called *traders*) forward unsolved requests to other nodes in an unstructured overlay. The overlay function takes into account neighbors' expertise and preference: a node connects to a node that has useful services and/or good recommendations. This evaluation uses information collected by the preprocessing component: traders explore the network off-demand, whenever necessary, and disseminate state changes via flooding.

Another solution is provided by the Globus Toolkit MDS [CFFK01]. Initially centralized, this service moved to a decentralized structure as its pool of resources and users grew. In MDS-2, a Grid consists of multiple information sources that can register with index servers ("nodes" in our terminology) via a registration protocol. Nodes and users can use an enquiry protocol to query other nodes to discover entities and to obtain more detailed descriptions of resources from their information sources. Left unspecified is the overlay construction function, the techniques used to associate information sources to nodes and to construct an efficient, scalable network of index servers.

5. EXPERIMENTAL STUDIES

Our objective is to observe and quantify the synergies emerging from the interaction of the four components of resource discovery in flat, unstructured networks. In the absence of a large-scale, deployed Grid available to test design ideas, we modeled an environment in which we experimented with a set of resource discovery mechanisms. This emulated Grid, while specifically designed to test resource location ideas, can be easily expanded to evaluate other services on large-scale testbeds, such as resource selection and scheduling. More important, it provides a framework for evaluating aggregations of cooperative services, such as resource location, resource selection, and scheduling.

5.1 Emulated Grid

Existing Grid simulators are specialized for certain services, such as scheduling [LMC03] or data replication [RF02]. Others, such as the MicroGrid [SLJ+00], run Grid software and applications on virtual resources. No current simulator is appropriate for or easily extensible to evaluating generic Grid services. We built an emulated Grid that is scalable and is suitable for resource discovery but also is easily extensible to other purposes.

In our framework, nodes form an overlay network. Each node is implemented as a process that communicates with other nodes via TCP. Each node maintains two types of information: (1) information about a set of resources and (2) information about other nodes in the overlay network (including membership information).

The large number of processes needed by our large-scale emulation raises multiple problems, ranging from resource starvation to library limitations. For the preliminary experiments (of up to 32,768 virtual nodes) presented in Section 6, we used 128 systems (256 processors) communicating over fast Ethernet of the Chiba City cluster of Argonne National Laboratory. With minimal modifications, the framework could be used in real deployments.

5.2 Modeling the Grid Environment

Four environment parameters influence the performance and the design of a resource discovery mechanism:

1 *Resource information distribution and density*: Some nodes share information on a large number of resources, whereas others share just on a few (for example, home computers). Also, some resources are common (e.g., PCs running Linux), while others are rare or even unique (e.g., specific services or data).

2 *Resource information dynamism*: Some resource attributes are highly variable (e.g., CPU load or availably bandwidth between two nodes), while others vary so slowly that they can be considered static for many purposes (e.g., operating system version, number and type of CPUs in a computer, etc.).

3 *Request popularity distribution*: The popularity of users' requests for resources varies. For example, studies [BCF+99] have shown that HTTP requests follow Zipf distributions. Our analysis [IR03] of a scientific collaboration, on the other hand, reveals different request popularity patterns, closer to a uniform distribution.

4 *Peer participation*: The participation of peers, or nodes, varies, more significantly in P2P systems than in current Grids. Influenced by incentives, some nodes activate in the network for longer than others.

The failure rate in a large-scale system is inevitably high and hence necessary to model. This factor can easily be captured by two of the parameters just listed, namely, resource information dynamism and peer participation. The effects of failure are visible at two levels: the resource level and the node level. When resources fail, the nodes that publish their descriptions may need to update their local information to reflect the change. Resource failure can therefore be seen as yet another example of resource attribute variation and can be treated as part of resource information dynamism. When nodes fail, not only do their resources disappear, but they cease to participate in maintaining the overlay and processing remote requests. Node failures can therefore be captured in the peer participation parameter as departures. We note, however, that node failures are ungraceful (unannounced) departures. Moreover, such failures may not be perceived in the same way by all peers; for example, in the case of network partitioning, a node may seem failed to some peers and alive to others.

To isolate some of the correlations between the many parameters of our study, we used a simplified, optimistic Grid model characterized by static resource attributes, constant peer participation, and no failures. Thus, we model only the resource and request distributions.

5.2.1 Resource Distributions

In Grids and peer-to-peer environments, the total number of resources increases with the number of nodes, so we model this as well. We assume that the average number of resources per node remains constant with the increase in the network size: in our experiments, we (arbitrarily) chose this constant equal to 5.

New nodes often bring new types of resources, however, such as unique online instruments, new data, and new, possibly locally developed, applications. To account for these, we allowed the set of resource types to increase slowly (5%) with the number of nodes in the system.

In this context, we experimented with two resource distributions of different degrees of fairness, as presented in Figure 25.1: a balanced distribution, with all nodes providing the same number of resources, and a highly unbalanced one, generated as a geometric distribution in which most resources are provided by a small number of nodes.

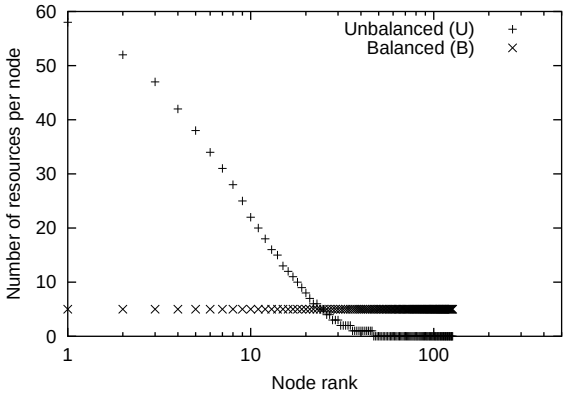


Figure 25.1. Distribution of resources on nodes: balanced (all nodes have equal number of resources) and unbalanced (a significant part of nodes have no resources).

5.2.2 Request Distributions

Although usage patterns can be decisive in making design decisions, we faced the problem of not having real user request logs, a problem inherent in systems during the design phase. We therefore logged, processed, and used one week's requests for computers submitted to the Condor [LLM88] pool at the University of Wisconsin. This pool consists mainly of Linux workstations and hence is a rather homogeneous set of resources. On the other hand, since it is intensively used for various types of computations, the requests specify various attribute values (e.g., for minimum amount of available memory or required disk space). We processed these requests to capture their variety. We acknowledge, however, that despite their authenticity, these traces may not accurately represent the request patterns in a sharing environment that usually comprises data and services in addition to computers.

We also experimented with a synthetic request popularity distribution modeled as a uniform distribution. Figure 25.2 highlights the differences between the two request distributions. The Condor traces exhibit a Zipf-like distribution, where a small number of distinct requests appear frequently in the set of 2000 requests considered. In the pseudo-uniform distribution, on the other hand, requests are repeated about the same number of times. We evaluated various resource location strategies in overlay networks ranging in size from 2^7 to 2^{15} nodes. In our experiments we randomly chose a fixed percentage of nodes to which we sent independently generated sets of 200 requests. The same sets of requests, sent to the same nodes, respectively, were repeated to compare various request-forwarding algorithms.

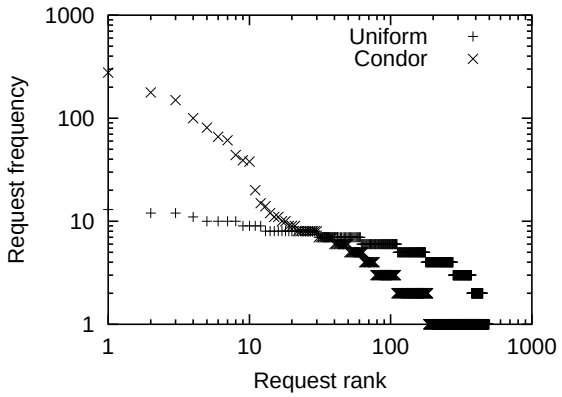


Figure 25.2. Distribution of user requests.

5.3 Resource Discovery Mechanisms

We considered a set of simple resource discovery mechanisms constructed by fixing three of the four components presented in Section 4.1 and varying the fourth: the request-processing component.

For the *membership protocol* we use a join mechanism that is commonly used in P2P systems: a node joins by contacting a member node. Contact addresses of member nodes are learned out-of-band. A node contacted by joining members responds with its membership information. Membership information is passively enriched over time: upon the receipt of a message from a previously unknown node, a node adds the new address to its membership list.

In our design, the *overlay function* accepts an unlimited number of neighbors: hence, we allowed the overlay connectivity to grow as much as the membership information. In this way, we neutralized one more component, aiming to understand the correlations between graph topology and discovery performance. We generated the starting overlay by using a hierarchy-based Internet graph generator [Doa96]. We assumed no *preprocessing*.

Our design of the *request-processing component* is based on forwarding. We assumed simple requests, satisfiable only by perfect matches. Hence, local processing is minimized: a node that has a matching resource responds to the requester; otherwise, it decrements TTL and forwards it (if TTL$>$0) to some other node. Requests are dropped when received by a node with no other neighbors or when TTL=0.

We evaluated four request propagation strategies:

1 *Random walk*: the node to which a request is forwarded is chosen randomly. No extra information is stored on nodes.

2 *Learning-based*: nodes learn from experience by recording the requests answered by other nodes. A request is forwarded to the peer that answered similar requests previously. If no relevant experience exists, the request is forwarded to a randomly chosen node.

3 *Best-neighbor*: the number of answers received from each peer is recorded (without recording the type of request answered). A request is forwarded to the peer who answered the largest number of requests.

4 *Learning-based + best-neighbor*: this strategy is identical with the learning-based strategy except that, when no relevant experience exists, the request is forwarded to the best neighbor.

6. EXPERIMENTAL RESULTS

This section presents preliminary results in two areas: (1) quantification of the costs of simple resource discovery techniques based on request- forwarding (no preprocessing), and (2) effects of resource and request distributions on resource discovery performance.

6.1 Quantitative Estimation of Resource Location Costs

A first question to answer is: What are the search costs in an unstructured, static network in the absence of preprocessing? To this end, we considered time-to-live infinite. The answer is presented in Figures 25.3, 25.4, and 25.5: the learning-based strategy is the best regardless of resource-sharing characteristics, with fewer than 200 hops response time per request for the largest network in our experiment. For a network of thousands of nodes (hence, possibly thousands of institutions and individuals) the average response time is around 20 hops. Assuming 20 ms to travel between consecutive nodes on a path (10 ms. latency in a metropolitan area network and 10 ms. necessary for request processing), then a path of 20 hops takes less than half a second.

Key to the performance of the learning-based strategy is the fact that it takes advantage of similarity in requests by using a possibly large cache. It starts with low performance until it builds its cache.

The random-forwarding algorithm has the advantage that no additional storage space is required on nodes to record history. We also expect it to be the least efficient, however, an expectation confirmed by the results shown in Figure 25.5 (Condor-based user requests, unbalanced resource distribution). For all network sizes in our experiments, the learning-based algorithm consistently performs well, while its more expensive version (learning-based + best neighbor) proves to be rather unpredictable in terms of performance (see, for example, the large standard error deviation for 1024 and 2048 simulated nodes in 25.5).

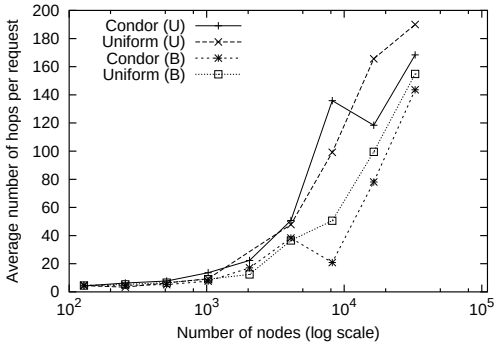


Figure 25.3. Performance (in average number of hops) of learning-based forwarding strategy for the two request distributions (Condor and uniform), in two environments with different resource-sharing characteristics (balanced B and unbalanced U).

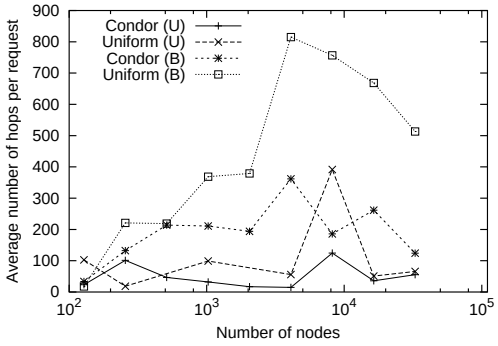


Figure 25.4. Performance (in average number of hops) of the best neighbor request forwarding strategy under different user request and sharing characteristics.

We emphasize that these results do not advocate one strategy over another but give a numerical estimate of the costs (in response time) involved. These estimates are useful in at least two ways. First, they give a lower bound for the performance of resource location mechanisms based on request propagation. They show that more sophisticated strategies (potentially including preprocessing techniques) are needed for efficient resource location in large-scale (tens of thousands institutions) Grids. Second, they can be used in estimating the performance of more sophisticated mechanisms that have a request-propagation component (as is, for example, the solution in [IRF02]).

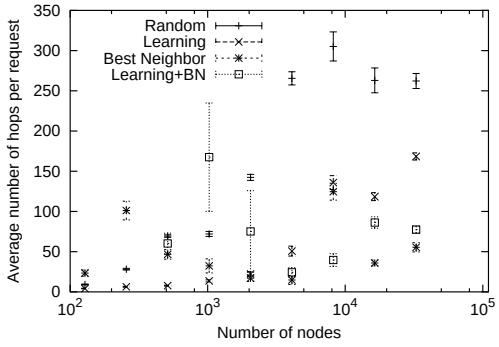


Figure 25.5. Performance of all four request-forwarding strategies for a Condor request load in the unbalanced resource-sharing environments.

6.2 Effects of the Environment

Figure 25.3 highlights the influence of user request popularity distribution on the performance of the learning-based request forwarding strategy. (Of the strategies we considered, this is the most sensitive to user request patterns.) The slightly better performance in the fair-sharing environment is due to the random component of this strategy, employed when no relevant previous information on a specific request exists: random forwarding has a better chance of reaching a useful node when information is distributed fairly on nodes. The learning-based strategy takes most advantage of the Condor request distribution, where a significant part of the requests are repeated (and hence can benefit from previous experience).

The best-neighbor strategy is influenced more strongly by sharing patterns: compared with a balanced environment, in a highly unbalanced environment a node that had already answered a request is more likely to have answers to other requests as well. Figure 25.4 shows the response latency in unbalanced and balanced environments for the two request patterns we considered: the response latency almost doubles in the balanced sharing environment as compared with the unbalanced one. We note that the performance of the best-neighbor strategy is influenced by past requests: the algorithm records the number of requests answered regardless of their type, hence it does not distinguish between nodes that answered same request n times and nodes that answered n distinct requests. This fact explains why the algorithm performs better under a uniform user distribution load than under the Condor traces: since the number of distinct requests in a uniform distribution is larger, the best neighbor identified by this strategy has indeed a larger number of distinct resources.

7. SUMMARY

We estimate that the characteristics and the design objectives of Grid and P2P environments will converge, even if they continue to serve different communities. Grids will increase in scale and inherently will need to address intermittent resource participation, while P2P systems will start to provide more complex functionalities, integrating data and computation sharing with various quality of service requirements. We are therefore studying the resource discovery problem in a resource-sharing environment that combines the characteristics of the two environments: the complexity of the Grids (that share a large diversity of resources, including data, applications, computers, online instruments, and storage) with the scale, dynamism, and heterogeneity of today's P2P systems.

We have identified four components that, we believe, can define any decentralized resource discovery design: membership protocol, overlay function, preprocessing, and request processing.

We have also proposed a Grid emulator for evaluating resource discovery techniques based on request propagation. Our results give a quantitative measure of the influence of the sharing environment (i.e., fairness of sharing) on resource discovery.

Acknowledgments

We are grateful to Daniel C. Nurmi for his help with deploying the Grid emulator on the Chiba City cluster at Argonne National Laboratory. This work was supported by the National Science Foundation under contract ITR–0086044.

Chapter 26

RESOURCE MANAGEMENT IN THE ENTROPIA SYSTEM

Andrew A. Chien,[1] Shawn Marlin,[2] and Stephen T. Elbert[3]

[1] *Department of Computer Science and Engineering, University of California, San Diego,*
[2] *Science Application International Corporation,*
[3] *International Business Machines*

Abstract Resource management for desktop Grids is particularly challenging among Grid resource management because of the heterogeneity in system, network, and sharing of resources with desktop users. Desktop Grids must support thousands to millions of computers with low management overhead. We describe the resource management techniques used in the Entropia Internet and Enterprise desktop Grids, to make the systems manageable, usable, and highly productive. These techniques exploit a wealth of database, Internet, and traditional high performance computing technologies and have demonstrated scale to hundreds of thousands of computers. In particular, the Enterprise system includes extensive support for central management, failure management, and robust execution.

1. INTRODUCTION

For over five years, the largest computing systems in the world have been based on distributed computing the assembly of large numbers of PC's over the Internet. These Grid systems sustain multiple teraflops [SET] continuously by aggregating hundreds of thousands to millions of machines and demonstrate the utility of such resources for solving a surprisingly wide range of large-scale computational problems in data mining, molecular interaction, financial modeling, etc. These systems have come to be called distributed computing systems and leverage the unused capacity of high performance desktop PC's (up to 3 GHz machines [Bre02]), high-speed local-area networks (100 Mbps to 1 Gbps switched), large main memories (256 MB to 1 GB), and large disks (60 to 100 GB). Such distributed computing systems leverage installed hardware capability and are now gaining increased interest within Enterprises to solve

their largest computing problems. For a general background of distributed computing [CCEB03].

In this chapter, we focus on the resource management problem for Enterprise Desktop Grid computing. The focus is on very capable desktops and laptops, which in Enterprise systems often cross the number of tens of thousands with sufficient network bandwidth to enable a wide range of applications. For these PC Grids, unlike server Grids, an integral part of their functionality is to gather unused cycles unobtrusively. Resource management for desktop Grids is particularly challenging because of heterogeneity in system, network, and model of sharing between Grid computations and desktop use. Desktop Grids must exploit large numbers of resources, thousands to millions of computers, to deliver tremendous power, and do so with modest requirements for additional IT administration support. Furthermore, to achieve a high degree of utility, distributed computing must capture a large number of valuable applications – it must be easy to put applications on the platform – and secure the application and its data as it executes on the network. We use the terms distributed computing, high throughput computing, and desktop Grids synonymously to refer to systems that tap vast pools of desktop resources to solve large computing problems – both to meet deadlines and increase resource availability.

The Entropia system provides novel solutions to Enterprise Desktop Grid resource management challenges. These solutions are developed in two systems: an Internet Grid and an Enterprise Desktop Grid system. The result is a flexible, manageable system that is easily accessible to many applications. We describe the system at three stages: the Internet Grid resource management system (Entropia 2000), where innovations include large scale system management, scalable client-server interaction, and flexible scheduling; the Enterprise Grid resource management system (Entropia DCGrid^{TM} 5.0 and 5.1), where innovations include resource pooling, batch-style scheduling , scheduling for reliable and robust completion times, and basic data-locality scheduling; and finally future resource management approaches, including the use of proactive replication and more advanced data locality sensitive scheduling in network-poor environments.

The remainder of the chapter is organized as follows. First, Section 2 describes the unique requirements and challenges for resource management in desktop Grids. Sections 3 and 4 describe the resource management techniques used in the Entropia 2000 Internet Desktop Grid and then the DCGrid 5.0 Enterprise Desktop Grid systems. Section 5 explores future directions in resource management, in particular, extensions to data-intensive applications needed for bioinformatics. Finally, Sections 6 and 7 discuss the related work and a range of future directions.

2. RESOURCE MANAGEMENT REQUIREMENTS FOR DESKTOP GRIDS

Large scale distributed computing using laptop and desktop systems face challenges unlike any other form of Grid computing. A principal source of these challenges is the scale of unreliable, heterogeneous resources: hardware, software, and network, which must be combined to create a reliable and robust resource. Another challenge is creating a secure resource from insecure components. Finally, there is the challenge of matching application requirements to the available resource in an effective manner.

2.1 Resource Management for Internet Grids

A resource manager must characterize the resource environment from hardware information (e.g. processor type, free memory, processor utilization, software configuration such as operating system and version). In general, all of these quantities are dynamic and must be monitored continuously. The information is tagged with an Entropia network identifier for each machine, because system and IP names are transient.

Network characterization is also critical, including bandwidth available to each client, whether its network connection is continuous or intermittent with some pattern, what protocols can be used to communicate, and geographic considerations. Geographic concerns may be performance related – shared bandwidth (many clients at the end of a thin pipe) or legal – (restrictions on cryptographic use, data placement, software licenses, etc.).

The resource manager not only needs to worry about the robust completion of individual work assignments (tasks), but also the functional status of individual clients. Malfunctioning clients must be diagnosed and removed from the list of available clients. Failure to do so can result in sinks or "black holes" for tasks, i.e., reporting completion of large numbers of tasks quickly without producing results. Such behavior is very damaging to the system – preventing other nodes from completing those tasks and resulting in the overall failure of a job.

Once the Grid resources have been characterized, the management of the resources begins as tasks are scheduled, monitored and completed. Tasks must be matched to appropriate clients while observing any additional requirements of priority and task association. To optimize network bandwidth usage, scheduling may need to be data locality aware.

Once a task is scheduled, its progress is monitored to assure completion. Progress monitoring responsibility is split between clients which monitor resource limits (memory, disk, thread counts, network activity, etc.) and the server which is responsible for rescheduling the task if the client fails or takes too long. For performance or reliability, tasks can be scheduled redundantly

(sent to more than one client), requiring the manager to decide which result to accept and when to reassign clients.

2.2 Resource Management for Enterprise Grids

The issues faced in building a Desktop Grid in the Enterprise (e.g. a large corporation, university, etc.) are similar to those on the Internet. Resources must be characterized and tasks must be scheduled against the available resources, tasks monitored and completed robustly with failures properly diagnosed and recovered. Enterprise Grids have one advantage in that network management tools may provide configuration and performance information. While Enterprises may have slightly less system heterogeneity, the reduction is generally not enough to significantly simplify management. The typical case – fast machines with high-speed network connections – represents valuable resources, but they may present server scaling problems.

Enterprise environments do present novel challenges. One is an even greater demand for unobtrusiveness on the desktop and the network. At the same time, a higher degree of desktop control is possible. Obtrusive behavior from applications that use large fractions of available memory and disk may be tolerable during non-business hours.

With always-on clients, centrally initiated scheduling and application data transfer is not only feasible, but also desirable. The work units can be shorter to improve turnaround time, creating corresponding increases in server load and scalability challenges. In general, machines in an Enterprise environment are more reliable than those on the Internet, which is factor in scheduling for performance as well as throughput. Pools of more reliable clients may allow applications to exploit peer-to-peer communication without the fault tolerance concerns normally raised by this style of computing, but with additional requirements to gang schedule the communicating nodes.

Data protection, through digests, and data privacy, through encryption, concerns vary from one Enterprise to another. Accidental data tampering is as much an issue in the Enterprise as it is on the Internet, but privacy issues vary considerably. In some Enterprises the value of the data may be such that any disclosure to unauthorized personnel is unacceptable so encryption is required throughout the system. In other cases the Enterprise may elect to eliminate the overhead incurred with data encryption (significant for some applications).

The administrative effort to manage the system (manageability) is not only more critical in an Enterprise environment, but a Desktop Grid must integrate well with existing Enterprise resource management systems, including cost-recovering accounting systems. Central software configuration management obviates the need for clients to support the self-upgrading used in Internet Grids.

3. ENTROPIA 2000: INTERNET RESOURCE MANAGEMENT

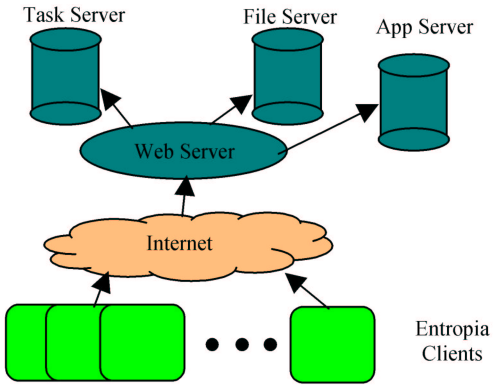


Figure 26.1. Entropia 2000 system overview.

The Entropia 2000 system (see Figure 26.1), an Internet Grid computing platform, consists of four main components; the Task Server, the File Server, the App Server, and the Entropia Client (EC). The EC is the local resource manager on each client and reports machine characteristics to the Task Server, requests assignments, executes the application client specified by an assignment, and ensures the unobtrusiveness of the platform on member (users donating their unused computing cycles) machines. The App Server decomposes an application job into its subcomponents, called application runs, and assigns them to application clients. The Task Server is the heart of the Entropia 2000 system scheduling and resource management capability. The Task Server's main responsibilities are the registration, scheduling, and the administration of EC's. The Task Server database contains all of the information necessary to describe the EC's and the application clients in order to enable efficient scheduling. This approach allows the grouping of online and offline resources to be assigned to App Servers, eliminating the need for an EC to negotiate its assignment when it becomes available.

3.1 Resource Management

The system automatically characterizes each EC (memory size, disk space, processor speed, OS type, network speed, firewall protected, geographical location, and availability) to enable efficient scheduling. An agent on the client collects information from the Windows^{TM} registry and other local sources and

transmits the information to the Task Server periodically and upon startup. The one exception is the EC's geographical location, where the Task Server employs an external service to resolve IP addresses to geographical locations. Several resource attributes are collected from the owner of the computing resource, including system availability, network type/speed, and firewall protected characteristics. Availability indicates when the machine may be used. To allow applications to employ a custom network protocol through firewalls, proxy technology was developed to encapsulate all communications to and from application clients.

3.1.1 Advantages and Limits of Centralized Management

A key feature of the centralized database approach is the ability to group online and offline resources into an Application Pool to be utilized by an App Server. Because the computing resources are not dedicated, Application Pools are configured to provide sufficient compute power.

Another powerful feature is the ability for an administrator to stop, restart, or remove an EC from the platform. Administrators can easily access and analyze a work history for an EC and determine if it was stuck. Because each EC checks in with the Task Server for assignments the EC can be informed to upgrade itself to a specific cached application client to a newer version which is network accessible. Therefore, the central database approach also provides easy deployment of new versions of the EC and application clients, enabling central version management.

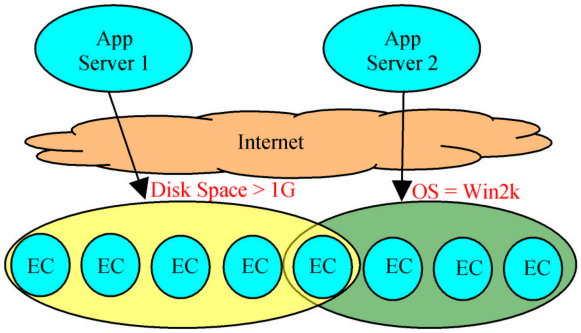


Figure 26.2. Application servers are associated with pools of resources (Application Pools).

3.1.2 Application Pools

Application pools (see Figure 26.2) are the key to Entropia 2000 scheduling. Each App Server (see Figure 26.1) is associated with an application pool (a list of EC's allocated for a specific job , the number of assignments in the job, the number assignments completed, and the pool priority). Pools can group EC's with similar capabilities to deal with heterogeneity. Pre-allocating EC's supports efficient task assignment because policy decisions are pre-evaluated, not done for each task assignment.

3.2 Scheduling

The Application Pools and their priorities determine a large part of the scheduling policy. In the normal case, an EC requests a task from the Task Server, it will find the highest priority Pool of which that machine is a member and send it the application client, the startup parameters needed, and the connection information for the associated App Server.

Across the Internet, intermittent network connectivity is common. There are two major issues: the direction of communication and duration of assignments. Computing assignments must be "pulled" from the Task Server with short-lived connections. If a request fails, the EC delays and retries with random backoff. This supports system scalability. In addition, the amount of work per run is adjusted to fill the useful CPU time between network connections. Choosing the parameters for such backoff is critical for system performance. It was shown statistically that the time between requests for tasks by an EC are not correlated to the length of the computation performed, but instead correlated to machine Internet connectivity characteristics. Empirically, we found that the efficiency of an EC increases with the length of the task being assigned to the EC with diminishing returns around 12 hours. App Servers use this observed behavior to schedule a number of application runs at each interaction to maximize EC efficiency.

3.3 Pitfalls

Among the most challenging issues in making Entropia 2000 work reliably are those of scale and congestion. If requests are distributed evenly in time, the system could handle one million EC's. However, there were a number of circumstances that led to entrainment or convoys and scaling problems. Causes included: server outage, network outage, EC reconfigurations, application misconfiguration, etc. If a server outage occurred, it caused most of the EC's to be starved for tasks. When the Task Server came up, it gets flooded by task requests from EC's, and crashed again. This type of problem was cited for SETI@Home. A careful design of the backoff scheme to include randomization and exponential backoff achieved stability for Entropia 2000 systems.

4. DCGRID^{TM} 5.0: ENTERPRISE RESOURCE MANAGEMENT

The DCGrid^{TM} 5.0 software platform significantly improves upon the Entropia 2000 platform in the areas of resource management and scheduling efficiency. The App Servers and application clients were replaced with components providing generalized job and subjob management capabilities, simplifying both the system and applications. In DCGrid 5.0, all scheduling and fault tolerance issues are handled generically, enabling applications to ignore most aspects of distributed computing – generic sequential applications work fine.

One of the biggest departures between Entropia 2000 and DCGrid^{TM} 5.0 was that Entropia 2000 was an Internet based platform whereas DCGrid^{TM} 5.0 is an Enterprise based platform. This eased the requirements on resource information gathering. Because of the non –restrictive communications allowed in Enterprise networks, firewalls separating the server side components and the EC's are no longer an issue and therefore become an extraneous piece of information. The assumption of no firewalls in the Enterprise network environment allows for a greater range of applications that assume direct connectivity, to be deployed and allows for a number of communication protocols which could not be deployed on the Internet.

4.1 Adapting the Internet Approach (Layered Scheduling)

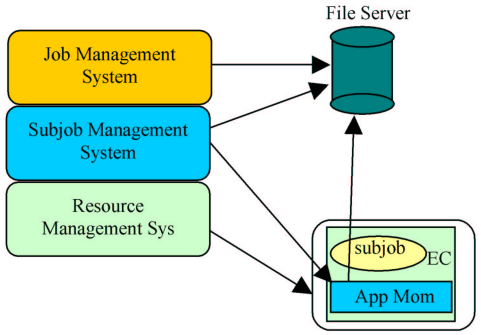


Figure 26.3. DCGrid^{TM} 5.0 components.

As can be seen in Figure 26.3, the scheduling and resource management of DCGrid^{TM} 5.0 has three layers: the Job Management layer, the Subjob Management layer, and the Resource Management layer. The Resource Management layer encompasses what was previously the Task Server and the functions performed by Application Servers and Clients are now handled generically by

the Job Management and Subjob Management layers.This approach cleanly separates physical resource management, job decomposition and management, and subjob scheduling.

Definitions of Work elements:

Job – the original work that is to be performed by a platform user.

Subjobs – a single-machine job component. Jobs are decomposed into subjobs and distributed to the computing resources.

Run – an assignment of a subjob to a specific computing resource. There may be more than one run per subjob.

Applications – the program executables used by subjobs

The Job Management layer is utilized by a platform user to submit a specific problem for the platform to solve, and its constraints. The Job Management layer will decompose the problem to a number of independent subjobs to be worked on by the nodes within the system. After the results of each subjob have been acquired, the Job Management layer will compile them into a single result an present it back to the platform user. The Job Management layer's primary responsibilities include decomposing a job description into the appropriate number of subjob descriptions, submitting subjobs to the Subjob Management layer, and retrieval and reconstitution of subjob results into a job result. The Job Management layer is a purely server side component with an interface with the Subjob Management server side components, the File Server, and the user.

The Subjob Management layer's primary responsibilities include the reception and storage of subjob descriptions and their associated files, the assignment and execution of subjob runs on EC's, individual result retrieval and reporting, and the handling of subjob and systemic failures with respect to subjob scheduling. The Subjob Management layer provides a public interface for subjob submission that is utilized by the Job Management layer, but can be used by any Job Management system as well as provided command line submission utilities.

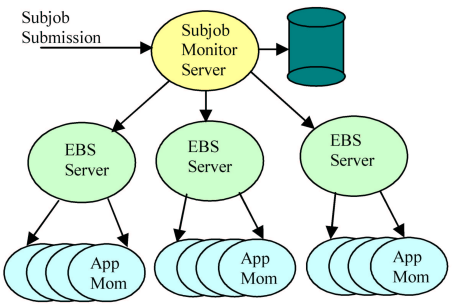


Figure 26.4. Subjob management layer.

As shown in Figure 26.4, the Subjob Management layer is decomposed into its individual components; the Subjob Monitor server, the Entropia Batch System (EBS) servers, and App Moms. The Subjob Monitor server provides the submission interface for subjobs, subjob description storage in a database, scheduling policies to route subjob runs to EBS servers, policies to handle subjob and EBS server failures, and subjob status reporting.

The EBS servers perform assignment of subjob runs to App Mom's, subjob run result reporting, subjob run heartbeat indicating that the subjob is executing, EBS server status interface, and fault tolerance policies to handle subjob run failures effectively. The Subjob Management system provides the capability to expand the number of EBS servers within the system and distribute the App Mom's appropriately across multiple EBS servers.

The App Mom provides the local resource management capabilities for a subjob run. It manages all process created by a subjob run ensuring that they do not run too long, too short, or create zombie processes. The App Mom creates the execution environment specified for a subjob run before execution begins by creating the desired sandbox directory structure, downloading necessary files from the File Server , creating environment variables, and executing the startup script specified by the subjob run description. The App Mom generates the subjob run heartbeat, which allows for quicker response time to errant subjobs and non-responsive resources. At subjob run termination the App Mom transfers the desired results to the file server and cleans up the execution environment.

4.2 Resource Pools

DCGrid^{TM} extends Entropia 2000's application pool abstraction into pool, partition, and queue abstractions to support generalized scheduling. Pools are managed by the Resource Management Server and support allocation EC's to a Subjob Management system – either a general purpose or single application system. When a pool is allocated to a generic Subjob Management system, each EC will instantiate the App Mom and the App Moms will be distributed among the EBS Servers.

Within the Subjob Management system, resources can be partitioned to support performance guarantees. For example, a user might have exclusive access to a partition (group of resources); each a named set of App Moms in a pool of computing resource. Administrators organize the resources by assigning EC's to partitions.

Submission queues are where work is submitted. Users can submit to particular partitions by selecting appropriate submission queues. Each submission queue has an associated priority and partition. These attributes drive schedul-

ing policies. Queues also have default values for time to live (in an EBS server), max time to run, and min time to run associated with them in case they are not specified during submission.

4.3 Scheduling for Robust Execution and Performance

Tables 26.1 and 26.2 describe the failure modes of applications and the system, as well as the algorithms that handle them. We discuss a range of failures, organizing by the subsystem that handles the failure mode. The generic handling of subjob failures is one of the biggest improvements between Entropia 2000 and DCGrid^{TM} 5.0. Scheduling in the DCGrid^{TM} 5.0 platform is performed at each layer of the system: Resource Layer, Subjob Layer, and Job Layer. Errors are handled by their associated subsystem with the exception of systemic failures, which require a coordinated effort to handle them.

4.3.1 Subjob Scheduling

Subjob scheduling in the "go-path" starts with the Subjob Monitor server finding the capacity of each EBS server and the capabilities of their associated computing resources. The capabilities of the EC's are binned; the Subjob Monitor defines the bins. The Subjob Monitor first finds all of the third retry subjobs (subjobs that have previously been submitted twice to an EBS Server without successfully completing) in the highest priority queue and assigns them to an EBS server. The monitor chooses a server with appropriate capacity and capabilities but also considers load balancing. The Subjob Monitor does the same for second retry subjobs and then first try subjobs. Subjob runs that complete successfully are flagged complete to the submitter and subjobs that have three failed runs are reported failed.

Within an EBS server, subjob runs are handled in FIFO with priority for third retry subjobs, then second retry subjobs (subjobs that have previously been submitted only once to an EBS Server without successfully completing), and so on. Subjob run assignments are done periodically. The subjob run at the top of the list is assigned to the least capable available client that can run the subjob. Subjob runs are submitted to the least capable available client to allow for maximum usage of client in a fully loaded system. When no available App Mom's can satisfy a run, it is saved in the queue for the next scheduling period. When a run completes the App Mom reports this to the EBS Server, thereby becoming available. The EBS server in turn reports the subjob status to the Subjob Monitor server.

Table 26.1. Subjob and resource failure definitions.

Subjob Failures	
Machine Usage Failure	A subjob process receives a machine usage exception i.e.: out of memory, disk full, etc, which fails the subjob run.
Subjob Running Too Long	Execution time of the subjob run exceeds its "max time to run". Possible causes include a subjob just running much longer than expected or a real "hang" of the application.
Subjob Running Too Short	Execution time of the subjob run violates its "min time to run". Common causes are errant parameters associated with a subjob or a misbehaving computing resource.
Subjob Stuck in EBS Server	Subjob run resides within the EBS Server without assignment for a time greater than its "time to live". When the Subjob Monitorserver submits a subjob to an EBS server, it first makes sure that the EBS server is managing an active EC with the resource capabilities to satisfy the subjob. If that situation changes, the subjob could get stuck waiting for appropriate resources to become available. The EBS Server will delete the subjob and the Subjob Monitor Server will resubmit the subjob to an appropriate EBS Server.
Zombie Process	A subjob process that does not terminate before the startup script terminates. Subjobs can spawn multiple processes in order to complete their computation, but when the startup script terminates all of its child processes should also have terminated.
File Tampering	Checksums or cryptographic hashes used to ensure data integrity have been detected as incorrect. Data corruption is likely whether from accidental or malicious activities.
Bad App Results in a Subjob Completion	Because a subjob completed with an un-desirable result doesn't mean that the subjob failed from a system perspective. The subjob is a success if it executes within the desired constraints and behaves in an appropriate manner. The quality of the application results of the subjob must be handled at the Job Management layer.
Resource Failures	
Resource Termination	A resource is gracefully shut down and removed from the system.
Unresponsive Resource	When a resource fails to respond to server communication, it is declared unresponsive. This condition could be caused by someone accidentally pulling the network plug on the computing resource, a dirty down caused by a power failure, or a network problem causing the computing host to be temporarily unreachable. This situation may be temporary and upon reestablishing communication the results of a subjob that completed offline can be reported.
Black holes	If subjobs complete too quickly (*Subjob Running Too Short*) on a specific computing resource continuously, then the computing resource is termed a black hole. Unchecked, a black hole can prevent applications from completing by preventing correctly working resources from executing application subjobs. The black hole sucks the subjobs out of the system, reporting them as completed when in fact they are not.

Table 26.2. Job and system failure definitions.

Job Failures	
Incorrect Application Parameters	During the decomposition of a job it must define the parameters for the subjob submission string. If this happens incorrectly, the subjobs that are created will never run correctly on the system and can have serious impact on the platform performance handling large numbers of failures simultaneously. An example of an incorrect parameter might be that the OS type is specified incorrectly and the executables used by the subjobs will not run on that OS.
System Failures	
Network Segmentation	With large networks a router or switch failure could cause whole subnets to become disconnected.
Power Outages	Power outages could cause server side components to fail and/or many EC's, and conversely App Mom's, to become unresponsive and executing subjob runs to fail.
EBS Server Failure	In the event of an EBS Server failure subjobs runs may become stale and timeout as well as App Mom's to go unused during the failure.

Priorities are only enforced on a submission queue basis; subjobs submitted to high priority queue are sent to an EBS server before others. However, subjobs submitted to a high priority queue will not necessarily be assigned to a client. This is subject to the scheduling policy. In the EBS server, scheduling depends on the subjobs ahead of it in the assignment list, the length of time to run of the subjobs ahead of it, and the clients capable of satisfying the request.

Subjob failures affect scheduling through time to live. Once this subjob timeout is reached, the subjob run is failed to the Subjob Monitor and relevant state removed.

4.3.2 Resource Scheduling

Resources are assigned to Subjob Management systems in units of pools. These resources are initially declared "Up". When the Resource Management server has received a heartbeat from an EC for a long period of time, it declares the computing resource "Disconnected". If the EC does not communicate with the resource management server for some time after being declared "Disconnected" the computing resource is declared "Down" and removed from the system.

When a subjob runtime is too short, meaning that its computing time was less than its minimum computing time allowed, an error is signaled and the App Mom notifies the local EC. If the EC has three consecutive subjobs terminating with runtime too short, the EC declares itself a black hole and notifies the resource management server, which then removes the computing resource from the Subjob Management system.

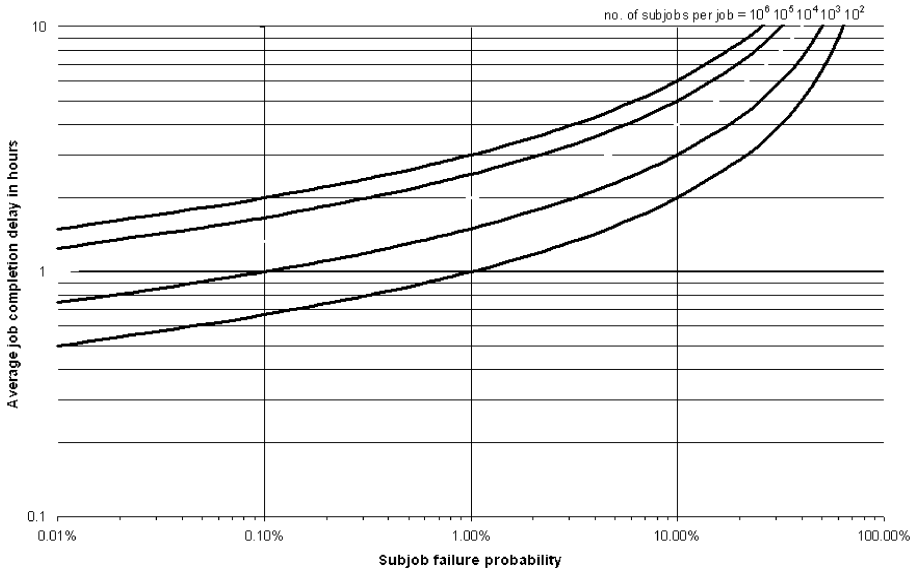


Figure 26.5. Impact of subjob failure probability on job completion time (1-hour subjobs).

4.3.3 Job Scheduling

The Job Management system submits jobs, adjusts their attributes, can choose which queues to submit them to, and if appropriate delete subjobs. Thus, the Job Management system can mediate the access of multiple jobs to a set of resources. The Job Management system periodically checks subjob status to monitor progress. For example, the Job Manager looks for a large number of failed runs for a job. This usually indicates that the subjobs were malformed. In this case, the Job Manager deletes the remaining subjobs and reports an error to the user.

4.3.4 More Aggressive Techniques

In Enterprises, an important capability is timely job completion. However, desktop Grids are known to have unpredictable subjob failures, due to machine shutdowns, network disconnects, etc. Empirical data from Enterprise desktop Grids at several Fortune 500 companies using DCGrid^{TM} 5.0 shows subjob failure rates below one-tenth of one percent. For example, in one 25,000 subjobs job, only 25 failed. Figure 26.5 shows that rerunning failed subjobs produces good job completion times given the observed failure rates. This works well because all retries are rapidly assigned. Over-scheduling , assigning the same subjob to multiple clients, wastes resources, reducing throughput with little benefit [KCWB02]. A better strategy is to schedule subjob retries against the fastest clients available as necessary.

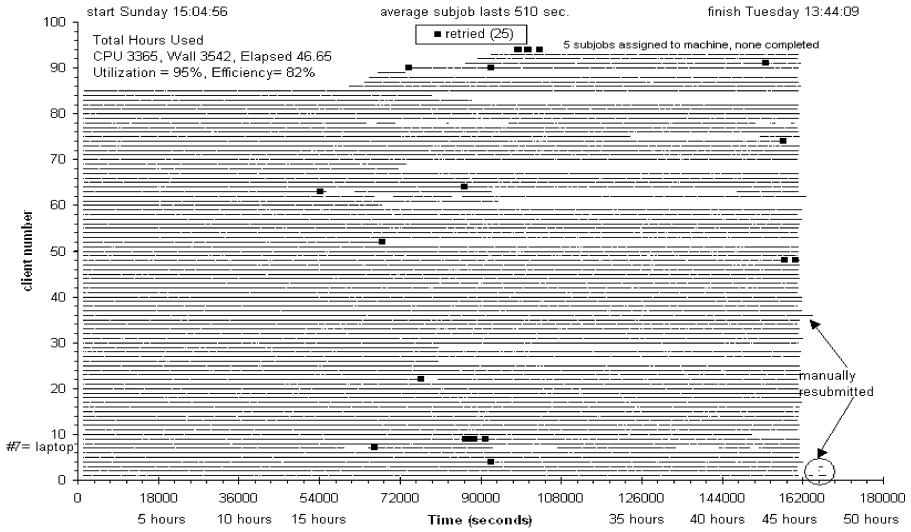


Figure 26.6. Utilization of 93 clients running 25,000 subjobs.

4.4 Scheduling for Network Bandwidth Management

A major concern for Enterprises deploying distributed computing is the system's network usage. The major network usage occurs when downloading files (executables and data) from the File Server for each subjob execution by the Subjob Management layer. Often, these files are used repeatedly by subjobs making file caching an important feature for eliminating network traffic and improving performance. When an App Mom is setting up the execution environment for a subjob, it gets the input file list and checks in its cache if it contains the file. If the file is contained in the cache, the App Mom will get its checksum and will check with the file server to see if it has a different version. If the file server has a different version than the one requested, the file will be downloaded and the App Mom will replace the file in its cache, selecting a file to displace by an LRU algorithm.

4.5 Sample Application Performance

Linear performance scaling of parameter sweep and data parallel applications in a distributed computing environment is well documented. Here we highlight application performance on the Entropia 2000 platform, showing job initiation and completion and the impact of resource volatility.

Figure 26.6 is a Gantt chart showing a Grid execution of a virtual screening application that evaluates 50,000 compounds. The Job Manager partitions the task into 25,000 subjobs of two compounds each. Initially only 85 clients are available but eventually 93 are used. A solid line indicates a client is do-

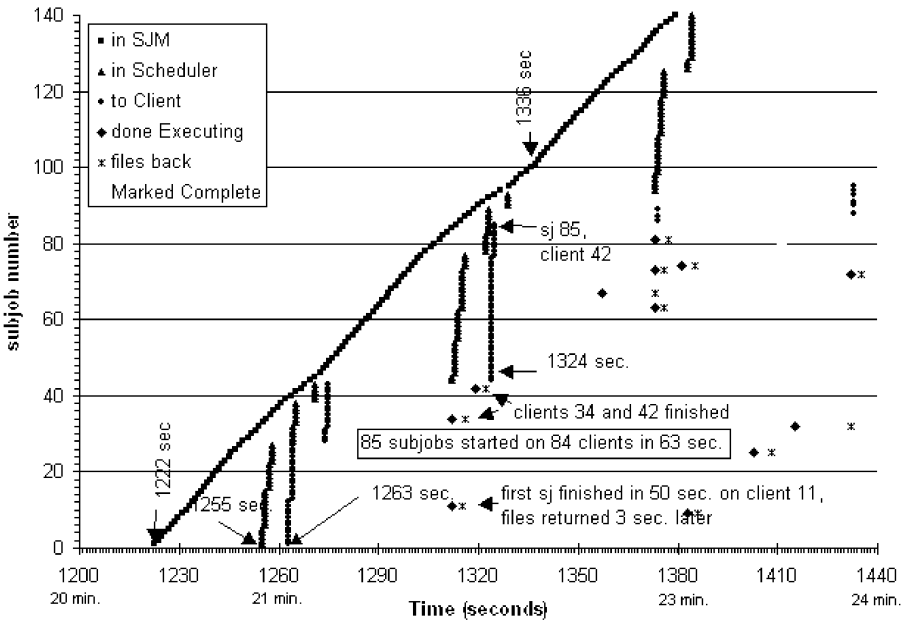


Figure 26.7. Detail of initial 140 subjobs.

ing useful work, and a gap that it is not. The squares indicate points where a client disappears (25 occurrences) and the subjob must be rescheduled. The job was started on a Sunday afternoon and the Grid performed consistently until Monday morning when normal desktop user activity alters the behavior of individual clients. For example, client 7 is known to be a laptop that is connected to the Grid except during commutes on Monday morning and evening and Tuesday morning. The laptop crashed Monday morning resulting in one of the rescheduling events. The 93^{rd} client was misconfigured and after it failed on five different subjobs it was automatically removed from the list of available clients. Five subjobs failed three times, and then were manually resubmitted and completed successfully. As Figure 6 shows, the overall job started and finished in a timely manner as indicated by the nearly flat edges on the left and right of the chart. The total CPU utilization is 95%. The tiling efficiency (fraction of this time used for work by all clients) is 82%.

Figure 26.7 is another Gantt chart of the same job from the perspective of each subjob near the beginning and shows the time evolution of the major events in the processing of a subjob: transfer to the Subjob Manager, insertion into the scheduler, arrival at the client, completion on the client, the return of result data and being marked as complete. Before the Job Manager can create the subjobs, the preprocessing must be completed, so the first subjob is not created until 1093 seconds after the job is submitted and arrives in the

Subjob Monitor 129 seconds later at 1222 seconds. All the charts use the job submission as the zero time. The 100^{th} subjob begins 114 seconds after the first, a rate that is sustained through most of the job. The Subjob Monitor then periodically sends work to the scheduler. The first batch of 27 arrives at 1255 seconds and has begun executing on clients eight to nine seconds later. Another 16 subjobs reach clients eleven seconds later and then there is a pause of 49 seconds during which the first three subjobs complete. In the next assignment period one of the three idle clients (no. 42) receives a new subjob along with 41 new clients. At this point 85 subjobs have been started on 84 clients in 63 seconds. The two idling clients receive work during the next assignment period and steady state has been achieved.

5. BEYOND DCGRID^{TM} 5: DATA SCHEDULING

DCGrid$^{\text{TM}}$ 5.0 has proven to be an effective way to manage applications that have high compute to communicate ratios. Here we consider applications where data locality is an important factor.

The first step is to understand the characteristics of applications that need to process large amounts of data. Table 26.3 considers two applications, a docking (virtual screening) application that is generally considered compute intensive, and BLAST [AGM^{+}90], a sequence analysis code that is generally considered data intensive. For each application we analyze required server bandwidth, assuming all of the data resides on the server. We consider two cases for each application, documenting typical subjob run times, volume of data per subjob, the server side bandwidth needed to support 1,000 clients, and the number of clients that could be supported on a typical 100 Megabit/second network assuming no more than 20% of the bandwidth can be used without becoming obtrusive. The final column is the Compute Intensiveness (CI) index.

We define the Compute Intensiveness index as

$$CI = \frac{4 * \text{Subjob Runtime}}{\text{Kilobytes to be moved per subjob}}$$

The factor of four was chosen as an arbitrary scale factor to make applications that we considered compute-intensive have a CI ≥ 1 and those with a value less than one are data intensive . Clearly, large-scale networks are currently only feasible with compute intensive applications. Data intensive applications scale only to small networks ($\sim$100 clients). Data compression does not significantly alter the situation.

6. DISCUSSION AND RELATED WORK

While a great deal of work has been published on resource management, the scope of resources and range of applications addressed by the Entropia

Table 26.3. Characteristics of Docking and BLAST applications.

Application	Subjob Run Time	Subjob Data In/Out	Server Bandwidth (1000 clients)	Maximum Clients using 20 Mbits/sec	Compute Intensiveness (CI)
Docking	20 min.	1 Mbyte	6.67 Mbits/sec	2,998	4.8
Small Data & Med. Run	10 min.	1 Mbyte	13.3 Mbits/sec	1,503	2.4
BLAST	5 min.	10 Mbyte	264 Mbits/sec	75	0.12
Large Data & Long Run	20 min.	20 Mbyte	132 Mbits/sec	150	0.24

systems is broader than most previous Desktop Grid work. The majority of work on Grids has focused on highly available server resources, not desktops. While the work on large-scale desktop Grids has largely focused on single application systems, the Entropia systems described in this paper combine large-scale heterogeneous unreliable desktop resources with complex multi-application workloads. Key novel resource management techniques aspects of these two systems include scheduling for intermittent connectivity, resource pooling, innovative failure management, and data management.

The idea of distributed computing has been pursued for nearly as long as networks have connected computers. However, most Desktop Grid systems employed simple scheduling systems as they supported only a single application or were not operated as resources for extended periods of time. Systems falling into this category include: GIMPS [GIM01], SETI@home [ACK+02], United Devices [Unib], Parabon Computation [Para], BOINC [BOI], etc. These systems provide only rudimentary resource management, collecting resource capability information and using it only to gather statistics for the aggregate "distributed supercomputer". In practice, scheduling of work units is done naively to all the resources in the system, with replication/retry for lost units. Since there is only one application (for which many pieces are available), there is no incentive to do more sophisticated scheduling. This tradition is continued in new open source infrastructures such as BOINC. These projects have all been single-application systems, difficult to program and deploy, and very sensitive to the communication-to-computation ratio. A simple programming error could cause network links to be saturated and servers to be overloaded.

Another important body of related work involves batch schedulers for dedicated resources such as clusters, supercomputers, and mainframes. Such typically homogeneous resources, or nearly so, are managed by systems like PBS [PBS], the Maui Scheduler [Mau] and its Grid extension Silver [Sil], Turbo Linux [Tur], and Load-leveler [IBM01]. Because the resources managed by these systems are typically highly reliable (and available), and have

limited heterogeneity, these systems do little complex resource management, and deal with resource unavailability in simple ways (as it rarely happens). The resource management (and coordination approaches) typically assumes continuous connection to resources, and doesn't deal systematically with task failure except signaling it. Systems supporting heterogeneous resource management for complex mixed workloads include Condor. The Matchmaking [BL99b] system in Condor manages complex heterogeneous resources, and jobs can specify a "class ad" indicating their resource requirements and priorities (see Chapter 9). However, the resource assignment does not have the same detailed control and optimization possible in a central database system, and has no high level understanding of collections of jobs and their relationship to application (and user) progress.

Several commercial products address the Enterprise Desktop Grid market, including UD's MetaProcessor and Platform's ActiveCluster [Pla]. Because of the difficulty in obtaining accurate information about commercial product capabilities and shortcomings, we omit a comparison of the Entropia system to these other commercial systems.

Grid technologies developed in the research community have focused on issues of security, interoperation, scheduling, communication, and storage. In all cases, these efforts have focused on Unix servers. For example, the vast majority of Globus [FK97] and Legion [NCWD$^+$01] activity has been done on Unix servers. Such systems differ significantly from ones developed by Entropia, as they do not address issues that arise in a desktop environment, including dynamic naming, intermittent connection, untrusted users, etc. Further, they do not address a range of challenges unique to the Windows environment, whose five major variants are the predominant desktop operating system.

7. SUMMARY AND FUTURES

Resource management for desktop Grids is particularly challenging because of the heterogeneity in system, network, and model of use allowed for Grid computations. We describe the resource management techniques used in the Entropia Internet and Enterprise desktop Grids, to make the systems manageable, usable, and highly productive. These techniques exploit a wealth of database, Internet, and more traditional high performance computing technologies and have demonstrated scale to hundreds of thousands of systems. Building on the capabilities of these systems, future systems will include more advanced resource classification, and particularly the management of data movement and locality-based scheduling to optimize network resource consumption.

Acknowledgments

We gratefully acknowledge the contributions of the talented engineers and architects at Entropia to the Entropia system. We specifically acknowledge Kenjiro Taura, Scott Kurowski, Brad Calder, and Wayne Schroeder for contributions to the resource management system.

Chapter 27

RESOURCE MANAGEMENT FOR THE TRIANA PEER-TO-PEER SERVICES

Ian Taylor,[1] Matthew Shields,[2] and Ian Wang[2]
[1]*Department of Computer Science, Cardiff University,*
[2]*Department of Computer Science and Physics and Astronomy, Cardiff University*

Abstract In this chapter we discuss the Triana problem solving environment and its distributed implementation. Triana-specific distribution mechanisms are described along with the corresponding mappings. We outline the middleware independent nature of this implementation through the use of an application-driven API, called the GAT. The GAT supports many modes of operation including, but not limited to, Web Services and JXTA. We describe how the resources are managed within this context as Triana services and give an overview of one specific GAT binding using JXTA, used to prototype the distributed implementation of Triana services. A discussion of Triana resource management is given with emphasis on Triana-service organization within both P2P and Grid computing environments.

1. INTRODUCTION

Grid Computing [FK99b] and peer-to-peer (P2P) computing [Ora01] are both important emerging paradigms for seamless aggregation and utilization of the ever increasing computing resources available today throughout the world. Currently, there are one billion mobile devices worldwide and half a billion Internet users. Further, some mobile phone companies are promising mobile devices with processors in the GHz region and with broadband wireless networking capabilities within the next two years. Potential example applications of this massive distributed resource include: CPU sharing applications and environments, such as SETI@Home [SET] and many others; file sharing Gnutella [Gnu] and data locating services; collaborative technologies, such as AccessGrid [AG]; and high performance scientific applications, such the Cactus Worm [AAF+01].

Recently, there has been significant interest in the field of Grid computing. Viewing the Grid as an infrastructure to support Virtual Organizations with a single sign-on mechanism is a significant and important step. Further, the recent convergence of Grid Computing and Web Services in the form of the Open Grid Services Architecture (OGSA) citeFoster2002b has given rise to an enormous drive in this direction by both industrial [IG02] and academic projects, such as Globus [FK97, GLO]. In parallel, interest in P2P technology is growing rapidly, as evidenced by the popularity of services like Gnutella and SETI@home. Within the past few months, P2P has appeared on the covers of the Red Herring and Wired and was crowned by Fortune as one of the four technologies that will shape the Internet's future. No doubt, there is a lot of hype but in reality there is also significant substance. One recent advance has been the introduction of architectures that support the programming of such networks. One such architecture is project JXTA [JXT], which defines a set of protocols that can be used to create decentralized P2P networks.

In this paper, we describe the distributed implementation of the Triana software environment [Tri]. Triana is a distributed problem solving environment (PSE) that allows users to graphically compose applications from a set of components. Such applications can then be distributed onto the Grid in a variety of ways, with resources being dynamically selected for the distributed execution. Triana is middleware independent through the use of a Grid Application Toolkit (GAT) API [AAG+02]. The GAT provides an application driven API and implements bindings to the various underlying mechanisms for the implementation of this functionality. Further, the GAT can be dynamically switched at run time to utilize the functionality that exists on a particular platform or environment. Current GAT implementations include Web Services (OGSA to follow shortly), JXTA and local services for prototyping applications. The current Triana implementation supports both P2P and Grid computing models for prototyping its distribution mechanisms.

The next section gives a brief overview of Triana and describes the implementation of its distribution mechanisms. Section 3 gives an overview of JXTA and it's binding within the GAT for this current Triana.

2. TRIANA

Triana is an open source problem solving environment, written in Java, which can be used in a variety of ways through the use of its pluggable software architecture. Its current uses include: a workflow management system for Grid applications; a data analysis environment for image, signal or text processing applications; a graphical script editor for workflow composition, e.g. WSFL and BPEL4WS formats; and it can be used as an application designer tool, creating stand-alone application from the composition of components. Triana

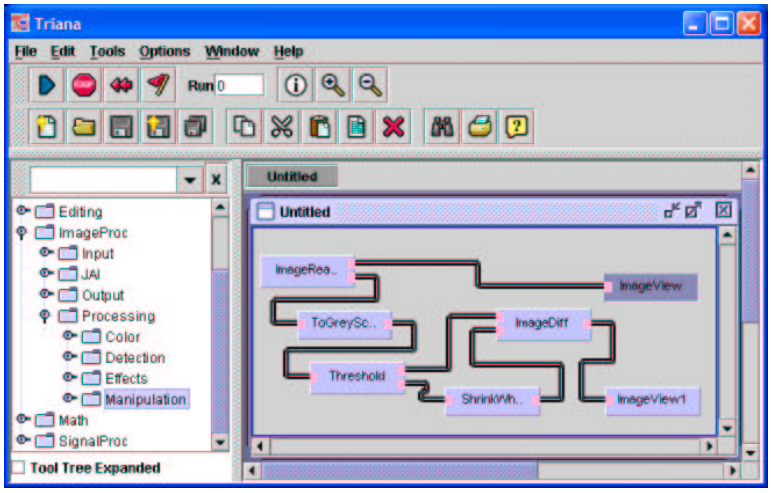


Figure 27.1. A screen shot of the Triana GUI being used for an image processing application for extracting the edges from an image.

consists of a collection of toolboxes (see left panel on Figure 27.1), containing a variety of tools and a work surface, for composition (see right panel).

Applications are written by dragging the required tools onto the work surface and then wiring them together to create a workflow or dataflow for the specific behavior. Triana supports looping constructs (e.g. do-while and repeat-until) and logic units (e.g. if, then etc.) that can be used to graphically control the data-flow, just as a programmer would control the flow within a program by writing instructions directly. In this sense, Triana is a graphical programming environment. Programming units (i.e. tools) are created within specific constraints so that they include information about which data-type objects they can receive and which ones they output. Triana performs dynamic run-time type checking on requested connections to ensure data compatibility between components; this is equivalent to the checking of function call compatibility during program compilation.

2.1 Pluggable Architecture

Triana has modularized pluggable architecture that is composed of a set of flexible interacting components that can be used to create Triana or any subset thereof. The Triana GUI is a light-weight thin client that connects to the Triana engine locally or via the network (see Figure 27.2). Clients can log into a Triana Controlling Service (TCS) via the various readers/writers, remotely build and run a Triana network and then visualize the result on their device (e.g. laptop or PDA) even though the visualization unit itself is run remotely.

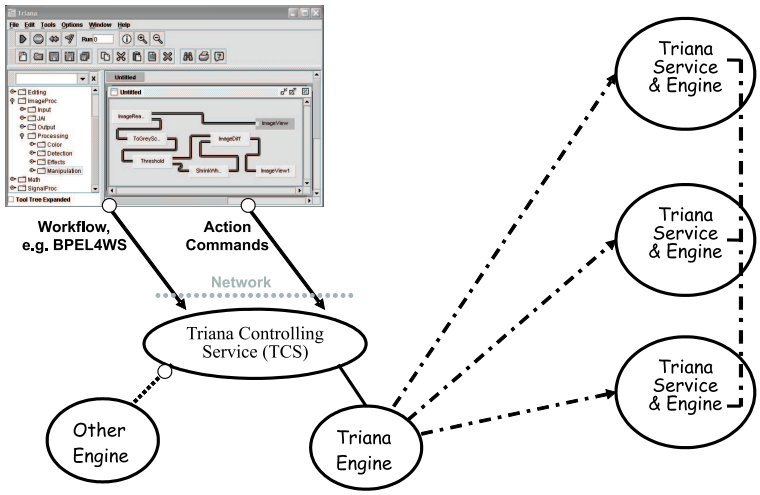


Figure 27.2. The Triana pluggable architecture. Applications can plug into any of the insert points (indicated by the white circles) and each Triana engine can act as a gateway to other Triana services where the task can be distributed to in a hierarchical fashion.

There are two types of communication between the GUI and the Triana Engine:

1 Workflow Updates: The task-graph representing the workflow can be updated incrementally (i.e. at each addition/deletion of units/cables) or can be updated once when the composition is complete. Triana adopts a pluggable architecture for the reading and writing of task-graph formats. Currently, we have implemented a BPEL4WS reader and writer and a Triana proprietary one. Each writer accessed by a GUI implementation has a corresponding reader on the TCS for interpretation. Reader/writers can be switched dynamically at run time depending upon the particular configuration.

2 Control Commands: These are commands that are used to control the non-workflow functionality of the GUI e.g. start algorithm, stop algorithm, save/open network, log on/off, security, etc.

Clients can log off without stopping the execution of the network and then log on at a later stage to view its progress, perhaps using a different device. In this context, Triana could be used as a visual environment for monitoring the workflow of Grid services or as an interactive portal by running the TCS as a servlet on the web server and by running the applet version of the Triana GUI. Further, since any Triana network can be run with or without using the GUI, Triana networks can be run as executables in a stand-alone mode.

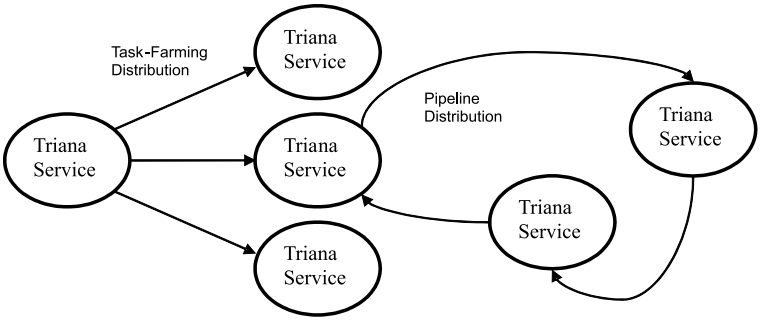


Figure 27.3. Triana Services within a distributed Triana scenario where one service distributes a task-graph to three other Triana services. Each of these Triana Services acts as a gateway and distributes their task graphs to another two services.

Another feature of the pluggable architecture is that it allows programmers to use the system at various levels by being able to plug in their own code at any of the insertion points within the system (indicated in Figure 27.2 by the white circles). At these points, the reader and writer interfaces allow the integration of tools,task-graphs and GUI commands. For example, a programmer can use Triana GUI as a front end to their stand alone application either on a single machine or using the remote control facility.

2.2 Distributed Triana Services

Each TCS has a corresponding Triana engine (or a third party engine). If a Triana engine is used then the user can take advantage of the various distribution mechanisms currently implemented in Triana. Triana implements its distributed behavior through the use of Triana services, which contain engines that are capable of executing complete or partial task-graphs in any of the supported task-graph formats. Each Triana engine can execute the task-graph locally or it can distribute the code to other Triana servers (see Figure 27.2) according to the particular distribution policy set for the supplied task-graph. Further, Triana services can communicate with each other, as indicated, to offer pipelined work-flow distributions or they can act as gateways to distribute the task-graph to finer levels of granularity.

For example, Figure 27.3 illustrates this conceptually using the two distribution policies implemented in Triana (described in the next section). Here, one Triana Service distributes a task-graph to three other Triana services using the task farming distribution policy then each of these Triana services act as a gateway and distribute their task graph to two other services using the P2P distribution policy. This has the capability of offering multiple tiers of distributed granularity.

2.3 Distributed Triana Implementation

The distributed implementation is based around the concept of *Triana group units*. Group units are aggregate tools that can contain many interconnected units. They have the same properties as normal tools e.g. they have input/output nodes, properties etc, and therefore, they can be connected to other Triana units using the standard mechanism. Tools have to be grouped in order to be distributed. However, the way in which groups can be distributed is extremely flexible.

Each group has a distribution policy which is, in fact, implemented as a Triana unit. Such units are called *control units*. A distribution policy is the mechanism for distribution within a group and therefore there is one control unit per group. This flexible approach means that it is easy for new users to create their own distribution policies without needing to know about the underlying middleware or specifics about individual Triana units. There are two distribution policies currently implemented in Triana, parallel and pipelined. Parallel is a farming out mechanism (see Figure 27.3) and generally involves no communication between hosts. Pipelined involves distributing the group vertically i.e. each unit in the group is distributed onto a separate resource and data is passed between them in a pipelined fashion. In practice, control units dynamically rewire the task-graph to reroute the input data to the available Triana services according to the particular distribution policy chosen. The distributed task-graph is created at run-time and therefore does not have to be fixed to a specific set of resources; rather it dynamically reconfigures itself to utilize the available services in the most effective way.

Conceptually, each Triana service shown in Figures 27.2 and 27.3 is in fact a Triana group implemented as a service. There is a one-to one-correspondence with the GUI representation of the group and how this appears on the Triana service. This makes it very easy for the users to visualize the functionality of each Triana service at any level of distribution. The actual representation of the entire distribution can also be logged for visualization.

3. JXTA

Project JXTA [JXT] defines a set of protocols to support the development of decentralized P2P applications. A peer in JXTA is any networked device that implements one or more of the JXTA protocols. Peers can be sensors, phones, PDAs, PCs, servers and even supercomputers. The JXTA protocols define the way peers communicate with each other and are specified using XML message formats. Such protocols are therefore programming-language independent and current bindings include Java, C and Python. At the time of writing, the Java implementation is the most advanced but the C implementation incorporates full edge-peer functionality and is compatible with the Java peers. For exam-

ple, JXTA C peers can be used to implement a JXTA service (e.g. file sharing or CPU sharing) but cannot act as rendezvous nodes (look-up servers) and relays for messages. Each peer operates independently and asynchronously from all other peers, and is uniquely identified by a peer ID. A peer group is a collection of cooperating peers providing a common set of services. Groups also form a hierarchical parent-child relationship, in which each group has single parent.

The six JXTA protocols allow peers to cooperate to form self-organized and self-configured peer groups independently of their positions in the network (edges, firewalls), and without the need of a centralized management infrastructure. Briefly, these protocols are as follows: the *Peer Resolver Protocol (PRP)* is the mechanism by which a peer can send a query to one or more peers, and receive a response (or multiple responses) to that query. The *Peer Discovery Protocol (PDP)* is the mechanism by which a peer can advertise its own resources, and discover the resources of other peers (peer groups, services, pipes and additional peers). The *Peer Information Protocol (PIP)* is the mechanism by which a peer may obtain status information about other peers, such as state, uptime, traffic load, capabilities. *The Pipe Binding Protocol (PBP)* is used to connect pipes between peers. The *Endpoint Routing Protocol (ERP)* is used to route JXTA Messages. Finally, the *Rendezvous Protocol (RVP)* is the mechanism by which peers can subscribe or be a subscriber to a propagation service. Within a Peer Group, peers can be rendezvous peers, or peers that are listening to rendezvous peers. The Rendezvous Protocol allows a peer to send messages to all the listeners of the service. The RVP is used by the Peer Resolver Protocol and by the Pipe Binding Protocol in order to propagate messages.

4. TRIANA RESOURCES ON THE GRID

Triana implements its distributed functionality as Triana services and therefore manages its resources as such. Both in the context of JXTA and OGSA [FKNT02] a service can be defined as a “network-enabled entity that provides some capability” [FKNT02]. The service paradigm is similar in concept to method calls or sub-routines but is more coarse grained, and therefore a better abstraction for Computational Grids. The recent development of Grid Services using OGSA is an important step in this direction. A computational Grid environment is typically composed of a number of heterogeneous resources, which may be owned and managed by different administrators. Each computing resource may offer one or more services and each service could be a single application or a collection of applications. Triana Services is an example of the latter, but they not only provide access to multiple applications but allow these applications to be connected together to form new applications. There-

fore, the interface to a Triana service needs to be flexible to allow the dynamic advertisement of its current functionality and communication specifications. To this end, each Triana unit (or group) has a description of its functionality and each unit (or group) can advertise data types for communication.

Possible deployment mechanisms for Triana Services include JXTA and OGSA, using J2EE or similar. Currently, OGSA is the most likely path for interoperable Grid computing solutions and may indeed circumvent the need for other architectures, such as P2P. However, we believe that P2P systems, such as JXTA, address problems currently not incorporated into mainstream Grid implementations and can therefore provide a useful feedback mechanism into future research for optimum solutions. Further, some systems are already building frameworks for accounting and managing compute power like electricity by using a JXTA-based P2P toolkit [GRIa]. Their Compute Power Market (CPM) Project uses an economics approach for managing computational resource consumers and providers around the world in P2P computing style. For our purpose however, extracting the useful notions from the various underlying middleware mechanisms is key to defining the GAT application-level interface described in the next section. We are currently exploring the JXTA prototype with this view in mind so that the GAT can support the various mappings within the application requirements context. Below, therefore we illustrate some differences between the JXTA P2P technology and Web Services/OGSA.

One significant difference is the notion of a JXTA peer. A JXTA peer can be a client, a server, a relay or a lookup server (a rendezvous node). In Web Services, the lookup server is located on a third party machine using UDDI or similar. Furthermore, JXTA peers can be dynamically organized as the network grows in a number of ways. The organization of JXTA peers is independent to the underlying physical devices and connectivity (see Figure 27.4). JXTA peers are organized within a virtual network overlay which sits on top of the physical devices. They are not required to have direct point-to-point network connections between themselves and they can spontaneously discover each other on the network to form transient or persistent relationships called peer groups. A JXTA peer group is a virtual entity that typically consists of a collection of cooperating peers providing a common set of services. Groups can be hierarchical and therefore can be used to monitor and organize subgroups [VNRS02]. This allows for varied organizations of peers depending on the particular environment. For example, there maybe a case where there is a set of reliable servers such as web servers that can be used to monitor other services on the network which may be much more transient, such as mobile sensors. In this instance, the reliable server could be a Rendezvous node for a group of transient services maintaining a decentralized federated arrangement with centralized lookup services at the end groups. Other situations may be

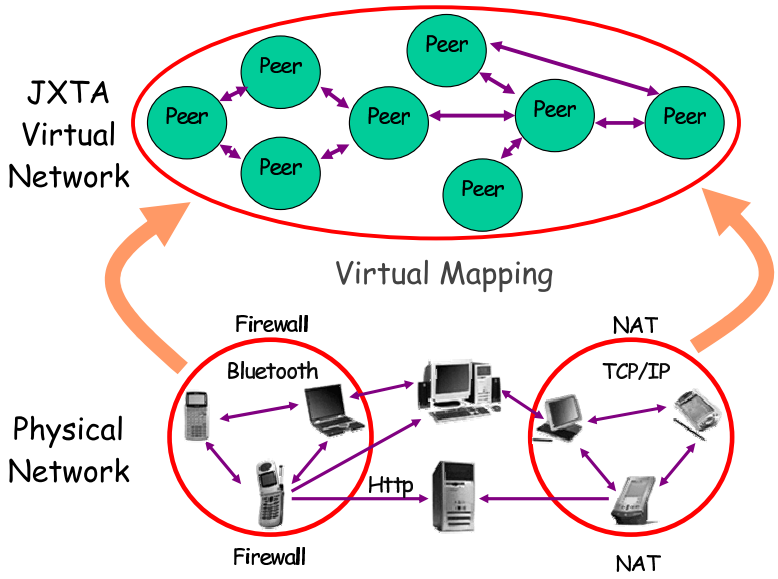


Figure 27.4. The virtual network overlay used within JXTA providing the applications with a virtual view of the physical underlying network.

more stochastic and may require a fullydecentralized solution. JXTA caters for any combination of the above.

Current web services and OGSA specifications do not address these issues to a comprehensive degree. Much can be learned from the P2P research in this respect. Other difference include key JXTA features, such as: the addressing of late binding of IP address in NAT translation systems, for example; and the use of virtual communication channels for communication for traversing multi-hop and multiple transport-protocol networks. Conversely, JXTA does not address how a service is created; it simply assumes such services already exist (as they do with file-sharing software for example). The dynamic creation of JXTA services would not be possible without the aid of an external toolkit, such as Globus.

5. THE GAT

The majority of Grid enabled applications are very much still at the prototype stage with many initiatives worldwide aiming at a wide range of different users. Proposed differing approaches to the architecture of the Grid such as the Globus Toolkit [FK99a], OGSA and Web Services all contribute to a set of emerging standards that overlap in places and provide different functionality. This mix of technologies often leaves the application programmer confused about which architecture to choose. Consequently, this slows the development

of applications for the Grid as the developers wait to see which technology becomes dominant or the most widely accepted. The GridLab project [GL] aims to remove this confusion by producing an application-level API called the GAT (Grid Application Toolkit). The GAT will have two initial reference implementations, written in C and Java. The GAT interface is intended to be used by applications to access core Grid services. It provides application developers with a middleware independent API containing the necessary functionality and implements the necessary hooks to the underlying middleware.

The main focus of the project is to enable applications to easily use the Grid by both abstracting the required functionality and by creating many GridLab services include monitoring, adaptive components, resource brokering, scheduling, security and application management. High-level graphical interface portals are also being developed for submitting, monitoring and viewing progress of the user's application on the Grid. The GAT interface effectively provides an insulation layer between the application and emerging technologies on the Grid. GridLab uses two principal applications to extract the necessary GAT functionality. These applications are Cactus [CAC] and Triana. A number of application scenarios have been defined to identify example uses of these applications.

5.1 The GAT JXTA Binding

JXTAServe is an API we have developed that implements the GAT binding for JXTA. It implements the basic GAT functionality and the hides JXTA specific details from such developers. Furthermore, since JXTA is an evolving system, JXTAServe provides the stability for our implementation in Triana; although JXTA may change the interface to JXTAServe will not. JXTAServe also gives the GridLab project [GL] a concrete prototype implementation to work with. JXTAServe implements a service-oriented architecture within JXTA. It conceptually looks very similar to a Triana group unit, illustrated in Figure 27.5.

A JXTAServe service has a control input pipe, zero or more data input nodes, and zero or more data output nodes. The control input is used for receiving workflow control commands from the TCS, such as initialize/run workflow. To distribute a workflow, Triana splits the workflow into subsections according to specified distribution policy (see Section 2.3), and sends each of these subsections to run on a remote service using that service's control pipe. However, before the workflow subsections are distributed, each of the input and output nodes is labeled by the distribution policy with a unique name denoting the pipe that it will connect to. Each JXTAServe service advertises its input and output nodes as JXTA pipes using their specified name, and, using the JXTA discovery service, a virtual communication channel is established be-

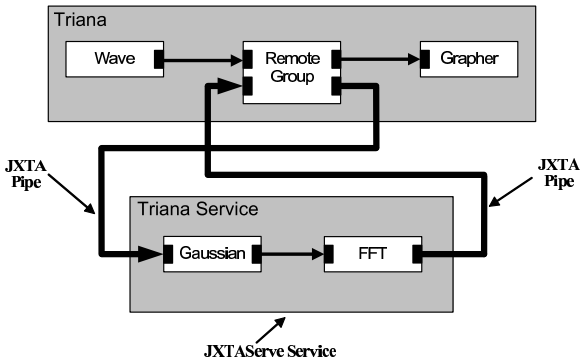


Figure 27.5. An illustration of how JXTAServe maps a Triana group. There is a one to one correspondence between a Triana group unit, a Triana service and a JXTAServe service.

tween the corresponding input and output nodes. In JXTA, the communication channel adapts to a particular communication protocol (e.g. TCP/IP or Bluetooth) depending on the current operating environment. Although our current implementation uses JXTA protocols, the unique labeling of pipes is done by the distribution policy independently of JXTA and is applicable to workflow implementation using other GAT implementations. When the communication pipes between the services are established the connected workflow subsections together form a complete distributed workflow.

As a test case, we integrated a Java galaxy formation code into Triana, which was successfully demonstrated recently [TSP02]. Briefly, galaxy and star formation simulation codes generate a binary data file that represents a series of particles in three dimensions and their associated properties as snap shots in time. The user would like to visualize this data as an animation in two dimensions with the ability to vary the point of view and project specific two dimensional slices and re-run the animation. We used the Triana parallel distribution (task-farming) policy to distribute this simulation temporally on our prototype Grid by splicing up the data into visual frames that could be computed independently from each other.

6. CONCLUSION

In this chapter we have outlined the open source Triana problem solving environment and its applications which include signal analysis, text processing and graphical workflow composition. In particular we discussed how Triana has been extended to enable the graphical creation of workflow-based Grid applications by using a middleware independent Grid Application Toolkit (GAT). The use of a middleware independent GAT enables Triana to seamlessly switch between Grid protocols, such as web services and JXTA, and to be extended

to new technologies, such as OGSA. As distributed Triana is currently based on a JXTA GAT binding, we outlined the salient features of the JXTA model and discussed how P2P concepts in JXTA could influence future Grid architectures. We showed how Triana distributes sub-sections of a workflow to JXTA services and establishes virtual communication channels to reconnect the distributed workflow.

The Triana system is an ongoing project based at Cardiff University; the latest version can be downloaded from *http://www.trianacode.org/*. For those interested in further discussion of this topic, please see [TSP02, AAG+02] and [Ora01].

VIII

ECONOMIC APPROACHES AND GRID RESOURCE MANAGEMENT

Chapter 28

GRID RESOURCE COMMERCIALIZATION

Economic Engineering and Delivery Scenarios

Chris Kenyon and Giorgos Cheliotis
IBM Zürich Research Lab

Abstract In this chapter we consider the architectural steps needed to commercialize Grid resources as technical focus shifts towards business requirements. These requirements have been met for conventional utilities resources through commoditization, a variety of market designs, customized contract design, and decision support. Decision support is needed to exploit the flexibility provided by Grids in the context of uncertain and dynamic user requirements and resource prices. We provide a detailed example of how the decision support for users can be formulated as a multi-stage stochastic optimization problem. We derive required architectural features for commercialization using inspiration from conventional utilities and considering the delivery context of Grid resources. We consider two basic delivery scenarios: a group of peers and a group with an external provider. In summary, we provide a conceptual framework for Grid resource commercialization including both the understanding of the underlying resource commodity characteristics and also the delivery context.

1. INTRODUCTION

Current Grid computing technology enables aggregation of resources from across many budget boundaries. Sharing those resources dynamically in a commercial environment requires an appropriate business model. One business model for Grid computing is commercialization at the resource level. *Resource commercialization* means that decisions about resources are made on an economic basis. That is, price is a key deciding factor in resource use. Thus price dynamics must be understood and appropriate decision support enabled, complementing support at the traditional systems level. We use the term *economic engineering* to describe these elements related to decision support and design of price dynamics. Economic engineering, i.e. understanding and design of economic elements for Grid resources, complements traditional systems engineering.

In this chapter we first concentrate on the basics of economic engineering for Grid resources and compare Grid resource commercialization to that of existing commercialized resources. We then describe architecture requirements for resource commercialization in two basic delivery scenarios in terms of economic and systems needs: a closed group of peer entities and a group of entities with a provider. Table 28.1 gives a list of some useful economic and financial terminology.

To understand Grid resource commercialization, or utility computing, it is worth looking at conventional utilities and business practices. Conventional utilities, e.g. electricity and gas, are based on open standards and on commodities. The open standards aspect has been well understood and vigorously proposed in the Grid space but the lessons from conventional (resource) commodities are generally missing except for a few significant exceptions [BGA01, KC02a, KC03] although the comparison with electricity is widely made [CB02]. Certain aspects of commodities, e.g. resource prices for immediate use, have been used but this is only a small part of what can be applied from an understanding of conventional commodities and markets.

Perhaps the two most significant omissions in the Grid literature on commercialization are an explicit derivation of price dynamics from the nature of the underlying commodity resources and the elucidation of a decision support layer for the exploitation of these commodities by users. We start with these aspects in Section 2. To support industrial scale commercial deployment equivalently robust decision support must be provided. This demands serious technical depth in the economic engineering, just as the systems engineering builds on deep foundations. We provide the reader with an example that details the level of available and appropriate economic technology. In Section 3 we describe an appropriate comprehensive architecture. This architecture also includes many supporting functions around resource contract handling.

When implementing a Grid for resources the most important economic question is whether the Grid will be run on a cost-recovery basis or as a profit center. Many details follow from this decision. However, we expect that the adoption of commercial Grids will not start with public Grids but from within individual companies, with providers, and between closed groups of companies. These partners will need to make this high level decision. In practice, different partners may make different decisions and the logical structure of the resource delivery will have a large influence on the resulting outcome. This interplay between delivery method, economics and architecture is described in Section 4.

We conclude in Section 5 with a short perspective of how Grid resources may commercialize over the next few years in terms of the development and blending of economic and systems engineering.

2. ECONOMIC ENGINEERING FOR GRID RESOURCES

2.1 Motivation

We have stated above that Grid resource commercialization will start within companies and not at the public level. Thus the use of economics for resource sharing, or acquisition on a utility basis, is an explicit choice by the company or companies over other possibilities. It is worth describing here, briefly, why we expect some companies to make this choice because these reasons provide design criteria. An economic basis for resource sharing can provide flexibility, efficiency, scalability, and feedback for budget and investment decisions:

Flexibility: Resources can be obtained by users when they need them: the degree of need expressed is under user control and can be described with high precision for resource use now and for future times of interest, like before deadlines.

Efficiency: Resource price reflects resource value.

Scalability: New budget entities and users can be added easily, preserving flexibility and efficiency.

Feedback: Prices for resource use and value of resource uses over time can be used to guide management decisions in terms they are familiar with, i.e. money and Net Present Value.

Management of resources using policies provide these attributes very poorly as compared to management of resources using economics. In fact policies scale badly, are difficult to maintain in a dynamic organization, and provide no linkage with investment decisions. They also provide no means of obtaining or providing feedback on any timescale, and users have no ability to express their urgency or lack of it. Using priorities is only slightly better that policies in these respects. Hence we expect economics to become the dominant Grid resource management method.

2.2 Resource Commodity Characteristics and Price Dynamics

Grid resources can be packaged as commodities: provided that resource technicalities, including security, QoS guarantees, reliability and credit-worthiness are met, the ownership of different resources is not important. For maximum convenience of use, commodities also need to be uniformly assembled into interchangeable packages. This is also typical in the development of a commodity: first the base good is introduced, then grades are standardized in

Table 28.1. Financial and economic terminology.

Commodity	A commodity is a good that, once technical specifications are met, is supplier neutral.
Derivative	A contract that depends on, or is derived from, another contract
Portfolio	A set of contracts owned and managed as a whole
OTC Contract	An Over-The-Counter contract is one specially designed for a particular customer requirement
Spot, $S(t)$	Contract bought now for immediate (time now is t) use
Forward, $F(t,T)$, or Futures	Contract bought now for use of resources at a later time, T. A Futures contract is the name for a Forward contract traded on an exchange.
Call Option on a Forward	Contract bought now, agreeing on a price for use at a given later time if the option is exercised. The owner can make the decision only at time T if option is *European* style, or decide at any time up to and including T for *American* style options

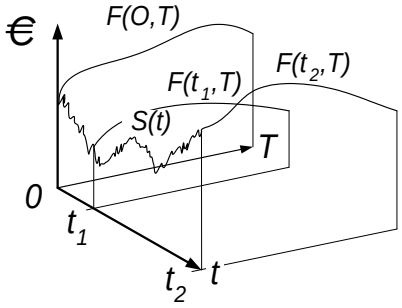


Figure 28.1 Relationship of current prices and reservation (forward) prices. Price for resources now (currently) is $S(0)$, ($S(t)$, jagged line). Reservation prices now (currently) for use later are $F(0,T)$, ($F(t,T)$, smooth curves).

order to permit trading, then markets and derivatives appear to meet the various needs of producers, consumers and intermediaries.

Grid resources are capacity-type resources and so are *non-storable commodities*. That is, unused capacity from yesterday is worthless today. Thus *futures contracts* (reservations) markets will be the basic building blocks in Grid environments, not *spot* (use now) markets. (For terminology see Table 28.1). Spot markets will be present but only as the shortest-dated futures contract. For all types of planning the curve of futures prices (Figure 28.1), however processed into appropriate application-style contracts, will be basic.

The recent deregulation of many electricity markets has driven an upsurge in models for price dynamics of non-storable commodities [CS00, Pil98]. The start of deregulation in bandwidth has also prompted development of new models that deal with Non-storability and also with the inherently distributed nature of network capacity [KC01, KL01, KC02b]. Price modeling in both these new markets tend to be built on interest rate models, particularly those that model the forward price curve directly [HJM92]. Thus there is an excellent basis from which to build models for Grid resource prices.

In markets, uncertainty plays a large part: what will be available when, and for what price? Hence the growth of structured contracts to provide certainty and control from the buyer's point of view. The most basic of these are the forward contract and options on forward contracts. New variations are continually being developed as particular business needs come up [Hul03, KC02b]. A single contract, however structured, is not a scalable solution to resource planning needs although particular designs are be useful building blocks. For industrial scale resource planning by users decision support tools are needed.

2.3 Decision Support

A glance at Figure 28.1 gives a hint of the potential size of resource planning problems for users. Note that while Forward prices $F(t,T)$ and option prices may be available, a set of contracts must still be assembled to match users (application) needs. Users already have day jobs, they do not want to be traders, and they want the ability to have the same level of surety about resource access as when they had the computer boxes in their offices. Users also want to pay for resources only as and when they need them. There are also several different user groups, users that: run applications (i.e. normal users); manage budgets (these users themselves or their managers); manage Grid resources (i.e. IT managers). This is a rough sketch of the problems that a decision support system, or architecture layer, must solve for users to benefit from the flexibility provided at the systems level by Grids. Decision support is not an optional extra for Grid exploitation in a commercial environment.

Manager decisions in a commercial Grid are often similar to portfolio problems in finance that express asset-liability models [ZM98]. These generally lead to multi-stage stochastic optimization problems. The book by Birge [BL97] provides a good introduction to the large body of research that addresses solving such systems.

More detailed decision problems faced by managers relate to offering and pricing application-specific resource packages. This is analogous to the OTC contract market in financial derivatives whereby specific offerings are made in response to customer needs. Again there is an extensive literature in this area [Hul03, BR96, Reb96, CS00].

The general reader with a computer science background is already aware of just how much technology is embodied in systems engineering but what may not have seen is just how much economic technical machinery is available to address the decision support problems in Grid resource management. In Figure 28.2 we give an insider view of an example formulation of the multi-stage Stochastic,optimization problem that expresses a particular user problem. In this problem a user needs to complete a project before a deadline within a given budget. Prices for resources are stochastic as are the day-by-day re-

Data: Uncertainty	
Ω:	sample space of all events over time (i.e. all scenarios)
ω:	sample point, i.e. a single scenario
p^ω:	probability mass function for scenario ω
Ξ_t:	classes of equivalent scenarios at time t

Data: Parameters	
B:	initial budget;
b_t^ω:	amount of money spent so far, $t = 1, \ldots, T$;
P:	required amount of work to finish project;
Γ:	unit penalty for missed project;
d_t^ω:	amount of work done so far, $t = 1, \ldots, T$;
γ:	unit penalty for missed work;
s_t^ω:	unit spot price of computation, $t = 1, \ldots, T$;
n_t^ω:	amount of work needing to be done, $t = 1, \ldots, T$;

Decision variables	
u_t^ω:	amount to spend in time period t in scenario ω

Formulation

minimize $\sum_{\omega \in \Omega} p_t^\omega \Gamma(d_T^\omega - P) + \sum_{t=0}^{t=T} \sum_{\omega \in \Omega} p_t^\omega \gamma(n_t^\omega - u_t^\omega / s_t^\omega)$

subject to

(keep track of project)	$d_t^\omega = d_{(t-1)}^\omega + u_{(t-1)}^\omega / s_{(t-1)}^\omega$	$\forall \omega \in \Omega,\ t = 2, \ldots, T$
(can't overdo day)	$u_t^\omega / s_t^\omega \leq n_t^\omega$	$\forall \omega \in \Omega,\ t = 1, \ldots, T$
(keep track of budget)	$b_t^\omega = b_{(t-1)}^\omega + u_{(t-1)}^\omega$	$\forall \omega \in \Omega,\ t = 2, \ldots, T$
(can't overdo budget)	$b_t^\omega + u_t^\omega \leq B$	$\forall \omega \in \Omega,\ t = 1, \ldots, T$
(non-anticipativity)	$u_t^\omega = u_t^{\omega'}$	$\forall \omega, \omega' \in \Xi_t$
(non-negativity)	$u_t^\omega \geq 0$	$\forall \omega \in \Omega,\ t = 1, \ldots, T$
(spent starts at zero)	$b_1^\omega = 0$	$\forall \omega \in \Omega$

Figure 28.2. User problem formulation as a stochastic optimization problem. User has a project due by T and a budget B. The amount of work each day and resource prices are both stochastic. There are penalties for not accomplishing the work required each day and missing the project deadline. The user's objective is to minimize the penalties, i.e. accomplish the work needed each day and finish the project on time. The user decides each day what to spend on resources.

quirements of the project up to completion. This formulation and its solution method would be contained in the decision support tools that the users have available to them. The decision for the user to optimize is how much to spend on resource each day to minimize the potential penalties for missing the requirements of each day and the overall deadline for the project. This formulation is quite simple in that it does not take into account reservations but should provide a hint as to the level of available economic technology available.

In summary, the decision support problem, while complex, is tractable. The cost of suitable tools in the infrastructure can be amortized over the (large) user population. The building of such tools is practical in business terms.

2.4 Assessment of Current Status

2.4.1 Grid versus Conventional Commodities

Conventional commodity markets, over the past 100 or so years, have evolved into highly successful and effective businesses for efficient transfer, pricing and allocation of resources. They have also succeeded in accommodating many new commodities (e.g. live cattle, government and corporate debt, gas, electricity) with a wide variety of characteristics. Few other businesses have been so adaptable.

Today the allocation of Grid resources to computational tasks and the execution of these tasks resembles the Internet world. It is often based on best-effort service, which means that there is little support for hard QoS guarantees that are the basis of resource surety for users. Typically commercial load-balancing products support job preemption based on a hierarchy of execution priorities. This is similar again to attempts in broadband networking, such as the Diff-Serv initiative, whereby statistical QoS guarantees are supported for premium traffic, based often on priority queues [FH98, DCB+01, SPG97]. In fact, the networking research community as well as telcos and ISP's have made significant progress in defining Service-Level Agreements (SLA), a step forward from current Grid practices (see [CFK+02], or in this book Chapters 23, 8, and 24). An SLA describes in detail the service offered to the customer with a focus on those factors that influence a user's experience the most (response time, downtime, etc).

Most conventional commodities are deterministic, e.g. a barrel of oil has no uncertainty in its description, its just 42 US gallons (with defined composition, at given temperature and pressure, for delivery at specified location on a given date, etc.). However, some commodities are stochastic, e.g. Random Length Lumber (RLL), in that their definition includes a specification of a probability distribution function (pdf). Figure 28.3 shows the guaranteed pdf of the lengths of the lumber contained in a single contract on the Chicago Mercantile Exchange (CME) [CME]. Kenyon [KC03] has shown how similar guarantees

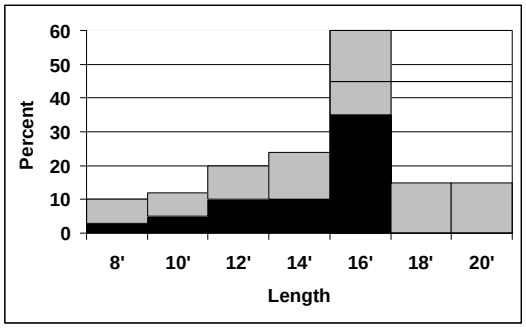


Figure 28.3 Definition of a conventional stochastic commodity: Random Length Lumber. The probability distribution function of lumber lengths under CME rules is shown (min/max percentages in each category and also for the 16'+18'+20' combined category).

can be supported in the Grid world for cycle-harvested resources. Once again the reader should take note of the sophistication that can be brought from complementary fields to address resource management issues for Grids.

In order to move from the sharing of computing resources to commercialized computing on-demand we need to build support for combined QoS of network, storage and computation with clear guarantees.

2.4.2 Economic Engineering

Several Grid-related initiatives include economic aspects in their design or deliverables with some idea of market-based valuation (e.g. Mariposa [SDK+94], Popcorn [RN98], Economy Grid initiative [BGA01]). These market-based projects have demonstrated the basic viability of the approach but have had some limitations to date (lack of dynamic prices, no guaranteed QoS, no reservations, etc.). Additionally the economic designs did not start from commercial requirements.

A number of start-up companies have attempted to commercialize spare cycles on commodity machines but have not addressed the market aspects of valuation. Using spare cycles, or cycle-scavenging, is the subject of non-profit initiatives such as SETI@Home or ZetaGrid, and commercial ventures such as Platform Computing (see Chapter 12), Parabon [Para], UnitedDevices [Unib], Entropia (see Chapter 26) and Avaki [Ava] whose main business is selling (scheduling, clustering and distributed load-balancing) middleware for Grid computing. Quite apart from Grid-related research, the idea of managing computer resources using economics has received much attention and harks back to the 60's and 70's (e.g. [Sut68]) when time-sharing of mainframes was the dominant means of computation and prices were somewhat variable. Not that this has ceased at least as far as supercomputers is concerned. Decision support as described above is basically missing from Grid initiatives to date.

3. REQUIREMENTS FOR A COMMERCIALIZED GRID ARCHITECTURE

Here we describe architecture requirements for Grid commercialization dealing particularly with those elements that are needed from the point of view of economic engineering. In this respect comparison with, and understanding of, conventional commodity markets is very useful ([KC02a] goes into depth).

In Figure 28.4 we present a simplified view of a commercialized Grid computing architecture using three stacks. The Execution stack is where jobs are taken over by local or Grid-level schedulers and executed. It also provides a common interface for discovering, querying and using basic Grid Services. The inclusion of proper QoS mechanisms at this level is one of the requirements for commercialization. We then assume that Grid-enabled applications will access the virtualized infrastructure through the use of application-specific portals that will in effect provide different views of the Grid, depending on particular application needs (thus creating virtual application-specific Grids on top). This is shown in the Application stack.

The main focus of Grid commercialization efforts will be in the development of the Commercialization stack, where computing capabilities are turned into tradable products. It is also here that the decision support for enabling the use of these products is found. As shown in the figure, we envision this stack as independent from the application stack, i.e. as an extra management stack that communicates with lower layers using standard (OGSA) [FKNT02]) calls and whose development does not need to interfere with the Grid-enabling of applications.

3.1 The Execution Stack

A basic requirement in all commodity markets is clear property rights. Property rights imply ownership (appropriately defined for the resource) for the duration of the contract. This is a strong requirement but without it attempts to build an economically functioning market for Grid resources or services will encounter serious commercial resistance. Best effort, or queuing priorities, are insufficient to support wide-scale commercialization.

For Grid resources, commercialization means support for hard quality of service (QoS) guarantees as well as careful attention to the time dimension and an appropriate granularity with respect to application use. Time granularity, i.e. *quantization* of the time attribute, is also vital for planning: resources for peak use can be bought in advance and idle resources offered on the market exactly when they are not needed. Resource blocks in the Grid context will also have a quantized location attribute.

Hard QoS is not required for every application. Statistical QoS is sufficient for some specific cases (notably in the Life Sciences where a long and uncer-

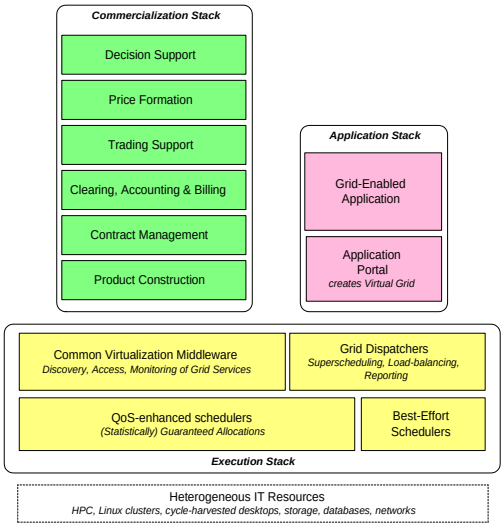


Figure 28.4. Commercialized Grid architecture stacks and layers.

tain processing time is acceptable for a few important applications). However, perhaps paradoxically, in a commercial context even statistical guarantees must be hard, as in the random-length lumber example. That is, the expected QoS as defined by statistical means (e.g. the distribution of available processing times on a set of PC's) must occur in practice. Uncharacterized, non-guaranteed resources have no value. Even poor QoS *provided it is precisely characterized* can have value. Clearly there will be a premium between statistical and deterministic guarantees as determined by market supply and demand.

Commoditized resources are like a set of children's building blocks: from a dozen standard shapes a vast array of structures, all with different attributes, can be built. A basic block in the Grid context starts from a resource with a hard quality of service guarantee and a set of quantized time attributes.

3.2 The Commercialization Stack

This stack embodies the main features required for commercial exploitation of Grid resources. Generally one does not need to implement all the layers of this stack to achieve business advantage from Grid technology. We will first discuss the main point of every layer in this stack and in the next section we will show different Grid delivery configurations where some layers are more important than others.

3.2.1 Product Construction and Reservations

First and foremost, commercialization means turning a service or capability into a product, something that can be bought and sold for a price. Resource virtualization and the transformation of resource control using Web Services interfaces with OGSA [FKNT02] are two welcome developments that provide for a consistent and flexible means of accessing Grid capability. But these developments fall short of turning capabilities into useful commercial services, in fact they were never intended to do so.

We need a layer whose sole responsibility is the packaging of capability into units of commercial value. Value is determined by user needs and budgets. A commercial service should adhere to — and comply with — a strictly defined usage profile, including hard QoS guarantees. Also, typically some service parameters will be flexible, allowing the user to modify them in accordance with changes in needs/budgets. Moreover, certain events may be triggered automatically, for example a commercial service may define that for every hour of downtime in any of its components a certain amount of compensation should be granted to the user. Such features distinguish a product that can be sold and resold from a computing capability.

3.2.2 Contract Management

Any party in the Grid space, besides perhaps some end-users, will be entitled to — or liable for — multiple contracts regarding commercial Grid capability products. These contracts need to be managed at the contract level, not at the level of their computational components. This is similar to the concept of SLA (Service-Level Agreement) management that originated in the world of commercial telecom services. Traditionally, resource management deals with system configuration, monitoring and reporting at the level of raw resources and basic services. But a commercial offering comprises many such elements and a great deal of information regarding all of them needs to be collected, updated, stored and processed to provide users or providers with an accurate view of end-to-end service performance.

3.2.3 Clearing, Accounting and Billing

The Grid market clearing mechanism is the part of the system that actually initiates the actions contained in a contract. A transaction agreed to on a Grid futures market is an event that is intended to happen in the future. A contract on a Grid futures market states that a certain resource will be made available at a future time given in the contract for an amount of money that is also specified. The contract does not result in any action on signing. It is the responsibility of a clearing mechanism to ensure that all the actions agreed in the contract are carried out.

When the event specified in the contract occurs, the clearing mechanism informs both the accounting systems of the respective parties and the access control system (or other appropriate mechanism) of the transaction to take place. The clearing mechanism also checks at the point specified in the resource contract whether the actions specified have taken place and initiates error or penalty routines if necessary.

A particular Grid resource contract may change hands several times before initiation. The clearing mechanism must support this. This will, generally, imply transacting accounting actions and change of ownership and/or liability actions because the commitment to perform, or the commitment to pay, may be sold. Billing will have to follow in the event of consumption (any initialization fees and then unit fees, etc).

3.2.4 Trading Support

In some cases, e.g. inside a single company, financial settlements for service consumption may be rare, but generally we can assume that in many contexts Grid resources will be exchanges across budget boundaries (within or between organizations). If this occurs relatively often it is clear that a marketplace for Grid capacity will emerge, not so much unlike traditional commodity and financial markets (only smaller in transaction volume, at least in the early stages). In order to carry such transactions electronically an e-marketplace infrastructure is needed.

Basic support for trading includes negotiation/auction protocols (dealt with separately in the next layer) and mechanisms for exchanging bid/ask data reliably and securely. Also, in the case of complex products or when multiple bids/asks need to be matched in real time specialized matching engines will be used that can deal well with the load and complexity of some situations. The marketplace can be mostly centralized (core functions in one place, at the server, with simple trading interface on client side), mostly distributed (federation of local marketplaces with synchronization mechanism), or perhaps even entirely distributed (peer-to-peer), in decreasing order of likelihood.

3.2.5 Price Formation

Moving on to this layer, the exchange of bids and asks and the results of matching engines need to be translated into market prices that will be communicated to all participants and form the base for decision support tools.

Prices are generally the combined result of supply and demand, so a price formation mechanism for matching these is required. A large body of research in auctions for a wide variety of goods is available [WWW01, Kle99, Kri02] but for every new market a new mechanism is needed or an older one must be adapted to fit the idiosyncrasy of this market. Extensive work on auction of

bundles is available [San02, PU00, CDS01]; in a Grid environment this would be a combination of storage capacity, CPU cycles, and network bandwidth.

A corollary to futures markets being basic is that prices must be transparent to all market participants. What this means in practice is that a significant number of trades must go through the futures markets so that these are credible reference sources for other deals. (In many financial markets roughly 5% of all deals are done through the futures markets but this is still a sufficient volume.)

3.2.6 Decision Support

This is the layer that users interact with for the management of resources and their use for satisfying application needs. The details have been described above in Section 2.3.

4. DELIVERY SCENARIOS

In order to illustrate the dependency of commercialization requirements on organizational structures, we will briefly describe two basic examples of possible Grid deployments: a closed group of entities and a group of entities with a service provider. By entity we mean an organization with a budget. This could range from an individual to a whole company. Many variations on these two basic themes are possible.

We stated earlier that one advantage of the management of Grid resource using economics was that resource price reflected resource value. This is true provided the market is structured to be competitive. The two delivery scenarios we describe can satisfy this requirement: the key is to have a sufficient number of competing elements to enable a valid price signal to be observed.

We can envision a group of entities forming a closed Grid where resources can be exchanged for money between the members (see Figure 28.5 *left panel*). In such a case a non-public marketplace will be naturally formed. Most layers of the Commercialization stack will be needed to produce a scalable and reliable e-market infrastructure, as shown in Figure 28.5 *right panel*. Actually, a 3rd party or a prominent member of the group may take over the marketplace, in which case that party can also take over clearing, accounting and billing duties as well, thus easing the load on the other parties which only need to keep track of what they're actually using. This e-business model has been relatively successful in procurement situations.

The second situation is where a service provider is attached to a group of entities that consume the provider's capacity but also participate in a marketplace with their own spare resources. The main challenge in this scenario lies in the implementation of the decision support layers for the provider and for the partnering entities. The provider may not actually be a full market participant but only provide capacity. Even if this capacity is not priced competitively, the

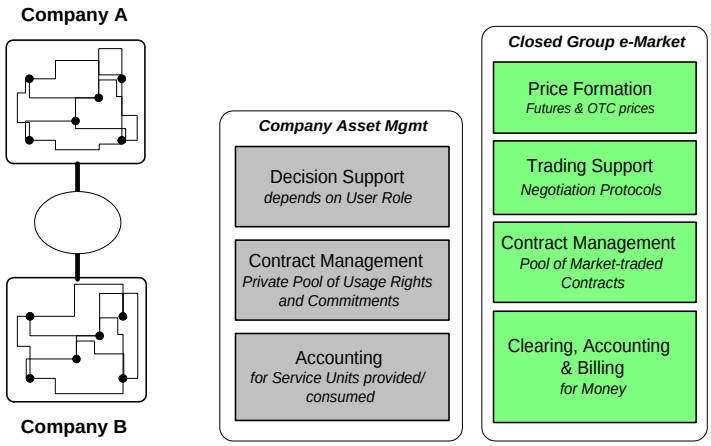


Figure 28.5. Group of entities exchanging Grid resources and architecture requirements.

choice of whether to use that capacity can be decided by a comparison with the competitive prices between the entities participating in the marketplace.

5. CONCLUSIONS

Economic engineering of Grid deployments, complementing technical systems engineering, is fundamental to their business success. To advance Grid resource commercialization we need architecture layers implementing business-type functionality, i.e. layers corresponding to the Commercialization Stack proposed here. There are many similarities with conventional commodity markets and management but specifics depend on the details of the resource delivery scenario. Significant steps toward Grid resource commercialization have indeed already been taken. Combining these steps with utility computing initiatives from industry indicates that resource commercialization is now underway. We have presented here a conceptual framework for the next steps as technical focus shifts to business enablement.

Chapter 29

TRADING GRID SERVICES WITHIN THE UK E-SCIENCE GRID

Steven Newhouse,[1] Jon MacLaren,[2] and Katarzyna Keahey[3]

[1]*London e-Science Centre, Imperial College London*
[2]*Manchester Computing, The University of Manchester*
[3]*Mathematics and Computer Science Division, Argonne National Laboratory*

Abstract

The Open Grid Services Architecture (OGSA) presents the Grid community with an opportunity to define standard service interfaces to enable the construction of an interoperable Grid infrastructure. The provision of this infrastructure has, to date, come from the donation of time and effort from the research community primarily for their own use. The growing involvement of industry and commerce in Grid activity is accelerating the need to find business models that support their participation. It is therefore essential that an economic infrastructure be incorporated into the OGSA to support economic transactions between service providers and their clients. This chapter describes current standardization efforts taking place with the Global Grid Forum and the implementation of such an architecture within the UK e-Science Programme through the Computational Markets project.

1. INTRODUCTION

The term computational Grid is an intended analogy to electrical power grids: a vision of computational power available on tap, without the user needing to really care about precisely where and how the power was generated. For this vision to become a reality, Grid users, or consumers, must be able to access appropriate computational power; similarly, resource providers must be able to receive payment for the use of their resources.

Resource brokering is the process of discovering suitable resources for the consumer's purpose. By definition, resource brokering is the act of an intermediary responding to the immediate needs of its consumers, while collating information from the resources it represents. The provision of a brokering service is predicated on the existence of an interoperable standards-driven infrastructure for representing resources and their corresponding services, as well as on standard payment protocols. Without these capabilities there is no economic incentive to provide a resource brokering service, since different resource in-

frastructures have to be abstracted within the broker and, without standard payment mechanisms, there is no generated revenue for the organization providing that service.

The recent moves within the Grid community through the Open Grid Services Architecture (OGSA) [FKNT02] to standardize on a framework specification as opposed to a service implementation has provided a generic mechanism for resource virtualization that will enable resource brokering. Due to the steady increase in Internet trading, or e-commerce, a number of reputable organizations already provide secure on-line payment services (e.g., WorldPay [Wor]).

With standardized schemes to describe electronic money and to virtualize the underlying resource as services through OGSA, the outstanding requirement is to provide standardized mechanisms to describe the protocols needed to set the cost of using the service. Currently, this requirement is the focus of the Grid Economic Services Architecture Working Group (GESA-WG) within the Global Grid Forum [GES] (of which we are the chairs).

This chapter outlines a set of motivating use cases for the provisioning of services either through direct invocation or through a resource broker. We then examine how the demands of such an infrastructure could be met by the emerging Open Grid Services Architecture by extending its standard Grid Services with interfaces to support economic activity. We also describe activity taking place within the U.K.'s e-Science Programme to build such an infrastructure using the OGSA.

2. ECONOMY-BASED GRIDS

The marketing of computational services for economic reward has been the subject of much research activity over the past decade as the availability and power of distributed computing resources have evolved. One example of early work in exploiting distributed computing infrastructures was Spawn, which demonstrated how different funding ratios could be used to guide resource allocation and usage [WHH+92]. The growth of Grid infrastructures, such as the Globus Toolkit® [FK97, GLO], UNICORE [UNIa], and Condor [LLM88], has promoted further discussion as to how economic paradigms may be used not only as an approach to resource allocation but as a means for making money. For instance, Nimord/G has shown how historical execution times and heterogeneous resource costs can be used for the deadline scheduling of multiple tasks within a fixed budget [ASGH95].

The key to trading in the real world is a medium of exchange that is deemed to have value and goods whose value can be assessed for exchange. Bringing an economic model into Grid computing presents two opportunities: using an economic paradigm to drive effective resource utilization, and motivating service provisioning for real economic gain by third-party service providers.

3. MOTIVATING USE CASES

The availability of flexible charging mechanisms that are fully integrated into the Grid infrastructure presents many commercial opportunities for independent service suppliers. One of the many architectural possibilities offered by OGSA is that of service provisioning through hierarchical encapsulation of service workflow and offering the encapsulated service as a single service to the user. The infrastructure provided by OGSA, when coupled with an economic mechanism, offers considerable scope for new service-oriented markets. These have recently been explored in a series of use cases being developed within the Global Grid Forum's GESA-WG [GES].

3.1 Coordination Between Services

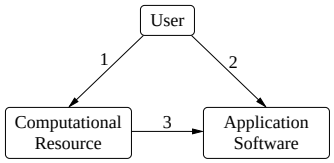


Figure 29.1. Coordinated use of application software on hardware.

Consider a simple scenario (shown in Figure 1.1) of a user wishing to use a commercial third-party application to analyze a self-generated dataset by using a computational resource. (We set aside for the moment the important factors that drive the selection of these services.) The user must obtain a quotation and reservation on the *computational resource provider* (1) before approaching the *application software provider*(2) to obtain a quotation for the use of that particular software on the computational resource. Once an acceptable quotation has been found from the compute and application providers—and this may be an iterative process because the cost of the software may depend on the class of computational resource and the time the data may take to process—the quotations and reservations are confirmed, and the computational resource may download and install the application software as required (3).

This process has already placed several requirements on the Grid infrastructure from both an economic and a general usage perspective. These requirements include a multiphase commitment to a resource reservation (one such approach using service-level agreements is described in Chapter 8) and iterative negotiation to converge on an acceptable pricing for the resource reservation. Additional requirements such as authentication, authorization, and impersonation (of the user by the computational resource provider in order to retrieve the application software) should be met through the basic core middleware.

3.2 Service Aggregation

The process just described exposes the user to the potential complexity of negotiating and reserving resources between different service providers. Alternatively, an organization can provide this combined functionality directly to the user (see Figure 1.2). This form of resource broker can be described as an application service provider because it provides a complete service—running the user's data using application implementation on an arbitrary resource.

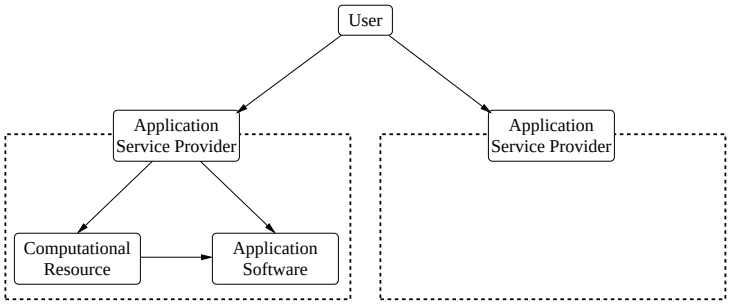


Figure 29.2. Service aggregation and virtualization.

Whereas previously the user was exposed to the full complexity of the underlying resource, in this scenario the application service provider had aggregated the services to supply a complete package. Two mechanisms can be used for providing this package. The application service provider can provide the computational infrastructure and application software through off-line purchases of the relevant equipment and software, as would normally be expected. In this case, the service provider has full control of the costs and can offer a service directly to the user. Alternatively, the application service provider can dynamically acquire these resources in much the same way as the user did in the earlier scenario.

A natural question is, "What are the economic benefits to the user?" The answer rests in part with the fact that the application service provider is able to derive potential economies of scale through the bulk purchase of computer resources and software licenses, by using the economic Grid infrastructure. These economies of scale can be passed on to the user as reduced costs, while the service provider still retains a profit margin for the service aggregation. Moreover, the service aggregator has the flexibility to switch suppliers as long it continues to deliver any contracted service levels. From the user's perspective, then, the service aggregator may be able to offer better pricing, faster discovery (since only a single aggregated service needs to be discovered, as opposed to several compatible services), and faster service delivery (as software may be pre-installed).

3.3 Service Brokering

In addition to these direct benefits, service aggregation can be viewed as a form of service (or resource) brokering that offers a convenience function—all the required services are grouped under one roof. But how does a user determine which of several application service providers should be selected for a particular application? The user could retain the right to select an application service provider service based on those that have been discovered from a registry service. Alternatively, this decision could be delegated to a service broker, which maintains an index of available application service providers.

The service broker is able to add value to its registry of application service providers by providing extra information about the services. This information may be as simple as cost, or it may include information about the reliability, trustworthiness, quality of service or service-level agreements, and possible compensation routes. Much like a financial consultant, the broker does not provide this added value service for free. Indeed, it may have a role in the financial transaction to provide an escrow account, acting as a trusted third party and holding the fee until the job is complete.

4. ARCHITECTURAL REQUIREMENTS

The preceding example of application service provision does not illustrate all of the features that may be required from an economic Grid services architecture. Indeed, many of the requirements from the scenario are a feature of a service-oriented architecture rather than that of an economic pricing mechanism. The emergence of the Open Grid Services Architecture from the Grid community is providing a service infrastructure upon which a variety of economic models may be developed and explored.

In this section we outline the basic mechanisms required to support such an infrastructure. We assume that economic models, dealing with issues such as price setting and Grid Services market creation, will be provided by other work in this area (see Chapter 28). Our goal is to define an open infrastructure to enable the application of these pricing models to generic Grid Services.

4.1 Exploiting the Open Grid Services Architecture

The OGSA builds on the established Web Services infrastructure provided through the eXtensible Markup Language (XML) [XML], the Simple Object Access Protocol (SOAP) [BEK+00], and the Web Services Description Language (WSDL) [CCMW01]. It provides an infrastructure to securely create, manage, interact with, and destroy transient Web Service instances within distributed hosting environments [FKNT02].

The Grid Service Specification defines the interface and the semantic behavior that must be supported by the Web Service for it to classed as a Grid Service [TCF+03]. This specification is under development and is being standardized within the Open Grid Services Infrastructure Working Group (OGSI-WG) of the Global Grid Forum [TCF+03].

A Grid Service has three features of interest in constructing an economic framework to trade resources:

- The *Grid Service Handle* (GSH) provides a unique identifier to a service instance running in a service environment.
- Each Grid Service has a *service data element*(SDE)—an XML document—that describes the internal state of the service. The Grid Service provides standard ports to support updating, searching, and so forth of the SDE by other entities.
- A Grid Service may support a *factory port* (or interface) that allows new service interfaces to be instantiated within the hosting environment.

The GESA-WG [GES] is analyzing the architectural requirements of an economic infrastructure within the context of the OGSA.

4.1.1 Grid Service Handle

The GSH is used by the client-side code to contact the specified service or factory instance. By assuming that the economic architecture is able to embed the cost of a transient Grid Service as one of the SDEs of a service factory (not an unreasonable assumption), the GSH effectively provides an identifier to a cost quotation for the use of the service. This price can also be advertised by other Grid advertising mechanisms; however, we assume here that the factory is a reliable source of such quotations. This service price quote may vary depending on factors such as the time the service will be performed, the time the quote is requested, the identity of the requestor, the level of Quality of Service (QoS) factors with which the service should be performed, and the guarantee with which those QoS representation can be delivered.

4.1.2 Service Data Elements

The application service provider scenario has illustrated that many of the issues relating to the selection of services within an economic architecture concern service rather than function:

- Does this service offer any bulk purchase discounts?
- Can I trust this service to deliver on its commitments?

- Is my data secure while it is residing on the remote server?
- Will I be compensated if anything goes wrong?

Such service metadata is encapsulated within the SDE structure provided by OGSA and may be collected from many service instances for presentation within an advertising service. This metadata may be static (extracted from the Grid Services Description Language document that defines the service interface) or dynamic (generated by the service or inserted from other authorized services). Standardization of the required and optional elements of this metadata that is one of the challenges now facing the community.

4.1.3 Factory Ports and Service Level Agreements

The factory model of service generation used within the OGSA provides a powerful abstraction to deal with pricing of Grid Services. We encapsulate the cost of using the service within the instance produced by the service factory (which can be referenced by the user through the GSH). This approach strengthens the link between the GSH acting as a quotation to the cost of invoking a service. Every quotation is created with an expiration time that puts time limits on its use.

The Chargeable Grid Service contains additional port types (to set prices, etc.), thereby extending the capability of the Grid Service being offered for sale. This approach enables existing client-side code to use the economic Grid Service without having to regenerate these interfaces.

In addition to providing quotes on service prices, the factory needs to support the negotiation for services with concrete QoS specifications as well as the creation of such services. Extending the concept of the OGSA factory to allow negotiation of service-level agreement (SLAs), as shown in [KM03], provides this capability. As a result of the negotiation process, concrete and well-defined SLAs are issued to concrete clients.

An SLA is a bilateral agreement [KKL+02] between the client and the service provider (represented by the factory) specifying the level of QoS with which the service will be provided (including the price or a pointer to price-generating mechanisms if the price should change during the lifetime of the contract), monitoring mechanisms that can determine whether the QoS requirements are met, and corrective actions to be taken if the requirements are not being met. Corrective actions may include adaptive scheduling, such as preemption of other executions, or QoS adjustments, such as price cuts or other kinds of compensation. Furthermore, the QoS conditions listed in the SLA should specify exhaustively the actions taken to provide overall QoS. For example, it is not enough to say that data will be secure; the composition of different security mechanisms used throughout the process should be specified. The SLA will usually be digitally signed by both parties agreeing to it.

A typical scenario might look like the following. Having obtained from the factory (or other advertising mechanisms) a quotation for the use of a transient Grid Service, a client negotiates with the factory for an SLA based on this quotation. If the conditions have not changed, the SLA requested by the client is issued. If the conditions have changed (the price of the service changed, or resources on which the QoS was predicated became unavailable), the SLA is renegotiated. At the time and manner specified in the SLA, the service is provided. If the QoS promised in the SLA is not provided according to the agreed-upon monitoring mechanisms, corrective actions are taken.

4.1.4 An Example: Application Service Provider

We continue with our motivating example of the coordinated use of a computational resource and an application software services. The user searches a community registry for service instances that support these capabilities. The user may specify additional nonfunctional requirements, such as a certain refund policy, or a particular architecture. The user's client contacts the factory port on each service and requests a particular level of resource use from both services (e.g., 16 processors with an interconnect greater than 100 Mbs running a Solaris 2.8 operating system and a compatible version of the application software) and a minimum termination time of the reservation.

The factory generates a new service instance for each requested service use and returns these to the user. By querying the SDEs of the newly created services, the user can obtain the agreed price for using the service and the agreed terms and conditions. The SDE of the newly created service may differ from that of the original because the latter may support multiple approaches to setting the price of the software while the created service describes only the agreed-upon protocol. If the user is unhappy with the offered reservation, the GSH may be discarded (or retained until it expires) and the process restarted from the original service. Alternatively, the price-setting protocol may allow the price to be adjusted through the newly created service, which will again generate a new GSH for use in further negotiation steps.

These transient service reservations will be destroyed when their lifetime expires. If the user takes up the reservation, by invoking part of the underlying Grid Service, then the reservation will be confirmed, any subsequent resource consumption will be monitored and recorded in a Resource Usage Service, and charging will then take place when the service invocation is complete.

4.2 The Grid Economic Services Architecture

The constructs provided by OGSA enable a Chargeable Grid Service to be built that can encapsulate an existing Grid Service with the mechanisms needed to set the cost of using a service and to offer it for sale. This approach exploits

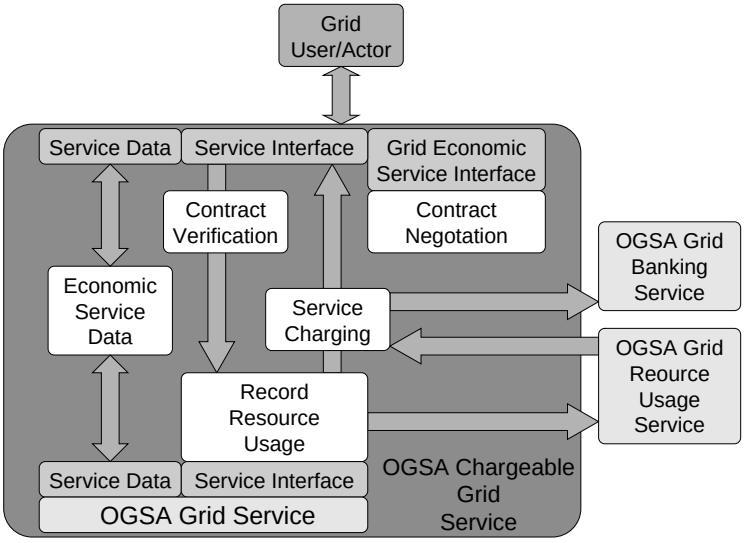


Figure 29.3. The current Grid economic economic architecture.

the basic infrastructure within OGSA for transient Grid Services while retaining considerable flexibility as to the eventual economic model that is used to set the cost of using the service.

Figure 29.3 shows the internal structure within a Chargeable Grid Service. The service data elements are composed from those contained by the underlying Grid Service and from the additional elements generated by the Chargeable Grid Service to describe the economic state of the service. This information is accessible through the standard Grid Service ports. An invocation by an authorized client on the service interface is verified and passed to the underlying service. On completion of the service invocation, the resources used by the service are recorded in an external service—the Resource Usage Service. The resources consumed during the service invocation (e.g., memory, disk space, CPU time) may be charged per unit of consumed resource rather than per service invocation. The cost of using the service is passed to an external service—the Grid Banking Service—for later reconciliation.

5. BUILDING THE U.K.'S COMPUTATIONAL MARKETPLACE

The UK e-Science Programme started in April 2001 as an ambitious £120M three-year effort to change the dynamic of the way science is undertaken by exploiting the emerging Grid infrastructures to enable global collaborations in key areas of science [Tay02]. Within this multidisciplinary activity a core pro-

gram focused on developing the key middleware expertise and components that would be needed by the U.K. science and business communities to encourage adoption.

As the global Grid infrastructure started to emerge and its commercial adoption started to become a reality, the lack of an economic infrastructure to motivate the provision of Grid Services started to become a barrier to adoption. This situation was recognized within the U.K.'s e-Science Core Programme, and in response the Computational Markets project [MAR] was formed to develop and explore the potential of such an infrastructure within the academic and commercial Grid communities. Project participants include the regional e-science centers in London (lead site), the North West, and Southampton; a variety of commercial partners including hardware vendors, application software vendors, and service providers; and end users within the engineering and physics communities. The U.K.'s Grid Support Centre will deploy the infrastructure developed through the project throughout the UK e-Science Grid.

The project has two main goals: to develop an OGSA infrastructure that supports the trading of Grid Services, and to explore a variety of economic models within this infrastructure through its deployment across a testbed between the e-science centers involved in the project. This will include the instantiation of the Chargeable Grid service, the Resource Usage Service, and the Grid Banking Service, as outlined previously.

One possible long-term outcome from the project is to change the model of resource provisioning within the U.K. for computational, and implicitly data, services. Currently, investigators requiring use of the U.K.'s high- performance computing resources (after passing a peer review) are awarded a budget for the use of the service. This budget is tied to a particular set of resources at a center and cannot be used to purchase general compute or data capability from other providers. Future models for resource provisioning could see this budget available for expenditure on the resources at university computing centers or through the provision of local compute clusters. The ability of researchers to flexibly acquire the most appropriate resources as they are needed would ensure transparent use of these resources and reduce the barriers to new entrants in the provisioning of these resources within the UK community. Key to any form of economic activity is a trustworthy medium of exchange. Within this project this capability is encapsulated in the Grid Banking Service, which records financial transactions and checks that the customer has the ability to pay. In reality we expect this service abstraction to be implemented by trusted third parties such as credit card companies, since we consider the development of an e-currency to be outside the scope of this project. The banking service will also be able to define a conversion mechanism between different currencies if required.

A key goal within the Computational Markets project, and within the wider U.K. e-Science Programme, is that the project's activity contribute to building international standards within the Grid community. It is envisaged that the project will produce a reference implementation of the economic architecture defined through various activities within the Global Grid Forum.

6. ACTIVITY WITH THE GLOBAL GRID FORUM

We focus here on three working groups within the Global Grid Forum's Scheduling and Resource Management area that are actively contributing to the definition of the economic architecture described earlier.

The *Grid Economic Services Architecture Working Group* (GESA-WG) is capturing a set of motivating use cases to identify the requirements for the underlying economic service architecture defined earlier in this chapter. A key element within the overall architecture is the consumption of resources. The Resource Usage Service within GESA exposes the consumption of resources within an organization by a user. Many of these resources (e.g., CPU and memory) might be used to determine the cost of having used the service.

The controlled sharing of resource usage information that has been captured by the underlying service infrastructure is becoming an increasing priority with virtual organizations around the world. A service interface is being defined by the *Resource Usage Service Working Group* (RUS-WG) [RUS] that will enable the secure uploading of consumed resource information and the extraction from the service by authorized clients.

An assumption with the RUS-WG activity is a standard mechanism to interchange data between different Grid entities. The resource information (its values, quantities, and structure) that may need to be exchanged between different centers is being defined within the *Usage Records Working Group* (UR-WG) [UR]. Several possible interchange formats (including XML) are envisaged for this information.

7. THE FUTURE

The past few years have seen the early adoption of Grid infrastructures within the academic and business community. While the use of Grid mechanisms is not yet widespread, their adoption will certainly be accelerated by the Open Grid Services Architecture and its use of Web Services as its service infrastructure. Future Grid environments may therefore comprise thousands of Grid Services exposing applications, software libraries, compute resources, disk storage, network links, instruments, and visualization devices for use by their communities. Nevertheless, while this vision of a pool of Grid Services available for general use is appealing, we emphasize that it is not realistic, as such a service infrastructure would have to be paid for by its users.

We foresee, instead, the emergence of resource brokers that add value to the basic service infrastructure by finding annotating services with information relating to their capability and trustworthiness. Users will be able to obtain their required services from these brokers, who may offer a guarantee as to their capability. Alternatively, users may seek out and discover their own services. These services need not be provided for free; indeed, for widespread acceptance of the Grid paradigm, organizations must have a mechanism for defining and connecting revenue from service provision.

The Internet has brought us ubiquitous access to data and simple services for little or no cost. The Grid offers the possibility of ubiquitous access to more complex services, but their appearance will be predicated on the service provider receiving an income for its provision. The proposed economic architecture is in its early stages of development but will build upon OGSA to be open and extensible across many deployment scenarios and economic models, thereby providing an infrastructure that will enable utility computing. Within this architecture we can see the speculative purchase of resources by services for later resale (a futures market), customer-dependent pricing policies (Grid miles), and other mechanisms to encourage the maximum utilization of resources by maximizing revenue generation.

Acknowledgments

This work is being supported in part by the Computational Markets project funded under the UK e-Science Core Programme by the Department of Trade and Industry and the Engineering and Physical Sciences Research Council (`http://www.lesc.ic.ac.uk/markets/`), and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

Chapter 30

APPLYING ECONOMIC SCHEDULING METHODS TO GRID ENVIRONMENTS

Carsten Ernemann and Ramin Yahyapour
Computer Engineering Institute, University Dortmund

Abstract Scheduling becomes more difficult when resources are geographically distributed and owned by individuals with different access and cost policies. This chapter addresses the idea of applying economic models to Grid scheduling. We describe a scheduling infrastructure that implements a market-economy approach, and we evaluate the efficiency of this approach using simulations with real workload traces. Our evaluation shows that this economic scheduling algorithm provides average weighted response-times as good or better than a common scheduling algorithm with backfilling. Our economic model has the additional advantages of supporting different price models, different optimization objectives, varying access policies, and Quality of Service demands.

1. INTRODUCTION

Online job scheduling for parallel computers and parallel jobs is a complex task, but scheduling for the Grid is made even harder by the fact that different participants will likely have different preferences. Resource owners must be allowed to have local control and to define their own objective functions for the use of their resources, while users will also have individual preferences.

Many installations of parallel computers use scheduling algorithms such as first-come first-served and backfilling [FRS+97, Lif96]. These conventional algorithms are well known in terms of worst-case behavior and competitive analysis, and some have been adapted and analyzed for Grid computing [EHS+02, HSSY00]. However, these approaches do not take into account the different scheduling and management preferences of users and resource owners. In this area, a market economy approach can provide support for in-

dividual access and service policies, both to the resource owners and to Grid users, which is very important for future Grids.

Several additional problems occur in Grid computing that can be solved by market methods (also referred to as market-oriented programming in computer science). Examples for those important problems are as follows [CFK+98b]:

- Site autonomy problem
- Heterogeneous substrate problem
- Policy extensibility problem
- Co-allocation problem
- Cost management problem
- Various optimization objectives problem

A supply-and-demand mechanism can enable a system to optimize different objectives for different participants. An economic-based system can be expected to provide a robust and flexible approach to handle failures, as well as allow adaptivity during changes. The underlying definitions of market, market methods, agents, and auctions can be found in [TB96, WWWMM98, Ygg98]. Additional information on the background of market economic approaches to Grid scheduling can be found in Chapters 28 and 29 and in [BAGS02, EHY02].

In comparison with other economic systems [WHH+92, SAWP95], our model uses individual utility functions for the different Grid participants. In addition, the model is not restricted to single parallel computers but is able to establish co-allocations of resources from different sites without violating remote policy rules. Our model is similar to the Enterprise system [MFGH88], in which machines create offers for jobs to use their resources, jobs describe their own requirements, and then a job selects the best offer. However, the Enterprise model is limited to a single user preference in which the offer with the shortest response time is chosen.

2. INFRASTRUCTURE MODEL

Our economic scheduling model has been implemented within the NWIRE (**N**et-**Wi**de-**Re**sources) management infrastructure [SY99]. Within this structure, the local management provides remote access to resources that are represented by CORBA objects. Resources in a domain are locally controlled by a *MetaManager* that acts as a broker or trader with MetaManagers from other domains; see Figure 30.1. Site-autonomy is maintained because the local MetaManager is responsible only for the corresponding domain and can

be set up according to local policy rules. The MetaManager is able to explore its neighborhood using a directory service or peer-to-peer strategies. Local requests are answered by the MetaManager and, if necessary, forwarded to other domains.

The main advantages of this approach are the independence of each domain, increased reliability and fault tolerance. A failure at one site will have only local impact if the overall network is still intact. Within the system, different scheduling implementations at different sites are possible using local policies. The information exchange between the different sites is implemented using requests and offers in a description language, described in Section 3.2. This enables the use of any scheduling implementations locally.

3. ECONOMIC SCHEDULING

This section describes the market-oriented scheduling algorithm that has been implemented for the presented scheduling infrastructure. We introduce the concepts of a user request and offer, in order to provide the necessary background for the information exchanges between different sites and the potential specification of job requests and the corresponding offers.

3.1 General Flow of the Scheduling Process

The application flow is presented in Figure 30.1. In contrast to [BGA00], our scheduling model is not restricted to a single central scheduling instance. Moreover, each domain can act independently and may have individual objective policies. Also, each job request can include an individual objective function. We have defined a description language to formulate objective functions that are then evaluated to scalar values at run time. The scheduling system combines the different objective functions to find the *equilibrium* between supply and demand. More details about equilibration and the existence of the general equilibrium is given by Ygge [Ygg98]. His model is a derivation of the previously mentioned Enterprise approach as well as the WALRAS model [Bog94].

All users submit their jobs to their local MetaManager. This MetaManager is responsible for the whole equilibration process. During job submission, the user may specify requirements for a job, such as a special operating system or the amount of memory needed to run the job. The specifications can be described within the request-and-offer description presented in the next section. In our model, each job request specifies an estimated run time, earliest start time, and a latest completion time for the job execution. Most current job scheduling systems already require the specification of a maximum run time by the user, since this information is required for some scheduling strategies,

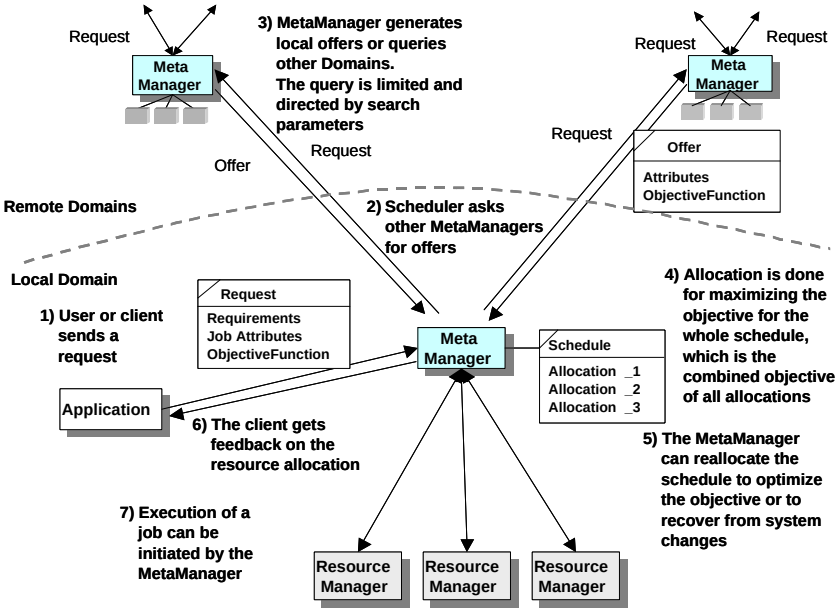


Figure 30.1. Market economy scheduling architecture.

for example backfilling. If the job finishes earlier than estimated, the idle resources can be used by other submitted jobs.

After the local MetaManager has received a new job, the local domain scheduler first analyzes the requirements of the job. Next, local offers are generated if the local resources match the job specifications. Only the best local offers are kept for further processing. The job is forwarded to other connected domains. To prevent the system from permanently forwarding job request, a user can restrict the number of hops made or specify a time-to-live for the request forwarding. In addition, none of the domains answers a request a second time if the same request has been received before. The remote domains create new offers and send their best offers back to the original domain. In order to prevent a live-lock of the system, the original request includes a deadline after which no offer will be accepted. Finally, the best of all incoming and all locally generated offers is selected as the final offer.

Note that this process is an auction with neither a centralized nor a decentralized auctioneer. Moreover, the objective functions of all participants are put into equilibrium. During this process, the utility value, $UV_{i,o}$, for each potential offer o corresponding to the job i is evaluated and returned to the originating domain with the offer. The utility value UV is calculated by the user-supplied utility function, UF_i, for the job i. We define the parameter set

$\vec{P_u}$ to be the possible request and offer settings. Additionally, we define $MV_{i,j}$ to be the machine value for job i on machine j.

$$UV_{i,o} = UF_i(\vec{P_u}, MV_{i,j}) \quad (30.1)$$

$$MV_{i,j} = MF_j(\vec{P_m}) \quad (30.2)$$

The machine value results from the machine objective function $MF_j(\vec{P_m})$ that depends on the parameter set $\vec{P_m}$, as shown in Figure 30.6, later in the chapter.

3.2 Request-and-Offer Description

A basic component of an economically driven approach is a description language that is used to specify requests, resources, and offers. It is essential that requests for offers be flexible. Our description language allows arbitrary attributes that are specified as nested key-value pairs in combination with the ability to specify several cases and constraints. Only a few keys (or attributes) are specific for the management and scheduling environment, but additional keys can be used.

The description language can be used for requests as well as for offers and resource descriptions. The syntax is similar in all cases. We allow complex statements in the formulation of the values. These can include expressions and conditions to allow parameterized attribute specification that can be evaluated at run time. Examples are the utility function and the job length, both of which may be derived from other attributes such as the job cost or the number of available processors and their speed.

An important feature of this language is the ability to describe resources, requests, and offers, as well as individual objective functions, with the same basic statements. The description language is not limited to a certain resource type, so new resource classes can be added easily. The list of parameters that are available for the request formation is presented in Table 30.1. The description language allows arbitrary key-value pairs not only as a list but as part of complex statements such as expressions and conditions.

Values of a key can be represented by numbers, strings, or mathematical expressions that can include other keys. The evaluation of a key is possible only if all included keys can be evaluated. Operators such as addition, subtraction, multiplication, division, and other more complex mathematical operations (modulo, square-root, exponents, logs, etc.) are also allowed. The resulting expression can be used for mathematical calculations, for example, as part of the objective function for the schedule. Moreover, the logic operators `AND`, `OR`, and `NOT` are available, as well as the conditional statements `ISDEF` and `ISNDEF` that check for the existence of attributes. These are commonly used

Table 30.1. Scheduling parameters.

Parameter	Description
Hops	This attribute limits the query depth to remote domains.
RequestID	This is a unique number that denotes the current request.
MaxOfferNumber	This is the maximum number of offers a user wants to receive. A value of 0 specifies that the MetaManager should automatically select the best offer according to the UtilityValue.
OfferBudget	This specifies the budget that is available for offer generation.
ReservationTime	This is the time until which the available resources should be reserved.
StartTime	A job must not be started before this date.
EndTime	A job must not end after this date.
SearchTime	The scheduling system can search for offers until this time instance.
JobBudget	This parameter specifies the maximum execution cost.
ReservationBudget	This parameter specifies the maximum reservation cost.
RunTime	This parameter specifies the execution time of a job.
UserName	This parameter specifies uniquely the submitting user.
Memory	This is the memory requirement per processor (in kBytes).
NumberOfProcessors	This is the number of requested resources.
UtilityValue	This value denotes the marginal gain from the user's point of view.

to allow the presence of a variable to specify when a different utility function should be used.

To better illustrate the description language, we give a brief example for a request formulation in Figure 30.2. This example request includes assignments for a set of keys (*Hops*, *MaxOfferNumber*, ..., *JobBudget*) and a utility function. The utility value depends on two conditions: the operating system and the number of available processors. The system tries to maximize the *UtilityValue*, which for our example means that the job should be started as soon as possible and that the job costs are minimized.

3.3 Local Offer Creation

In Figure 30.3 we describe how local offers are generated. In step 1, an incoming job request is made. In step 2, the request is checked by first deciding whether the local resources can meet the job's requirements and then whether the user's budget is sufficient to process the job locally.

As an additional part of step 2, the necessary scheduling parameters are extracted from the job request, including the run time, the earliest start time, the latest end time, the needed number of resources, and the maximum search time. Also extracted is the utility function, which is evaluated in later steps.

If the local domain does not have enough resources to fulfill the job requirements, a co-allocation process is initiated that may result in a multisite scheduling in step 3. Additional details on co-scheduling are in Section 3.4.

In step 4, we search for free intervals within the schedule. The scheduler tries to find all potential time intervals that satisfy the needs of the job. Figure 30.4 (1) shows a small example for a parallel computer with seven dedicated processors. The black areas are already allocated to other jobs. Assume that for our example we have a new job request that requires three processors,

```
REQUEST "Req001" {
  KEY "Hops" {VALUE "HOPS" {2}}
  KEY "MaxOfferNumber" {VALUE "MaxOfferNumber" {5}}
  KEY "StartTime" {VALUE "StartTime" {900000}}
  KEY "EndTime" {VALUE "EndTime" {900028}}
  KEY "SearchTime" {VALUE "SearchTime" {899956}}
  KEY "JobBudget" {VALUE "JobBudget" {900.89}}
  KEY "Utility" {
     ELEMENT 1 {
       CONDITION{ (OperatingSystem EQ "Linux")
         && ((NumberOfProcessors >= 8)
         && (NumberOfProcessors <= 32))}
       VALUE "UtilityValue" {-StartTime}
       VALUE "RunTime" {43*NumberOfProcessors}
     }
     ELEMENT 2 {
       CONDITION{(OperatingSystem EQ "AIX")
         && ((NumberOfProcessors >= 8)
         && (NumberOfProcessors <= 64))}
       VALUE "UtilityValue" {-JobCost}
       VALUE "RunTime" {86*NumberOfProcessors}
     }
  }
}
```

Figure 30.2. Request description example.

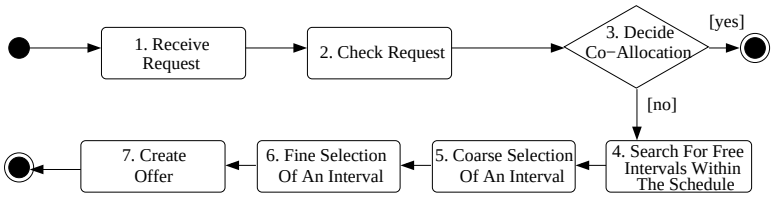


Figure 30.3. Local offer creation.

has a runtime smaller than (C-B), an earliest start time of A, and a latest end time of D.

First, the free-time intervals for each single resource are evaluated, and the potential solutions are created by combining the free-time intervals of the single resources. The result is a list of triples in the form {time, processor number, free/not free}. We then apply the algorithm given in Figure 30.5 to this list to try to find potential solutions.

At this point it remains to be shown how the offers are created from the elements in the used *tempList*, the temporary list that is the result of the pseudocode in Figure 30.5. Step 4 searches for free intervals within the schedule. The goal is to find a set of combined time intervals with sufficient resources for the application to run. During the combination process, the earliest starttime and latest endtime of the corresponding job are dynamically adjusted. We use a derivation of bucket sort to combine the intervals. In the first step, all intervals with the same start time are collected in the same bucket. In the second step, for each bucket, we collect the elements with the same end time into new buckets. As a result, each bucket contains a list of resources available between the same starttime and endtime. In our example, this algorithm creates three buckets as shown in Figures 30.4 (2), 30.4 (3), and 30.4 (4) where, for example, bucket one represents resources 1, 2, 5, and 6 with a start-time of A and an end-time of C.

After the creation of these buckets the suitable offers are generated. If there are enough resources in a single bucket to run the application, we use these resources build a solution. If no bucket can meet the application request individually, we must combine buckets. In our example, bucket 1 can fulfill the

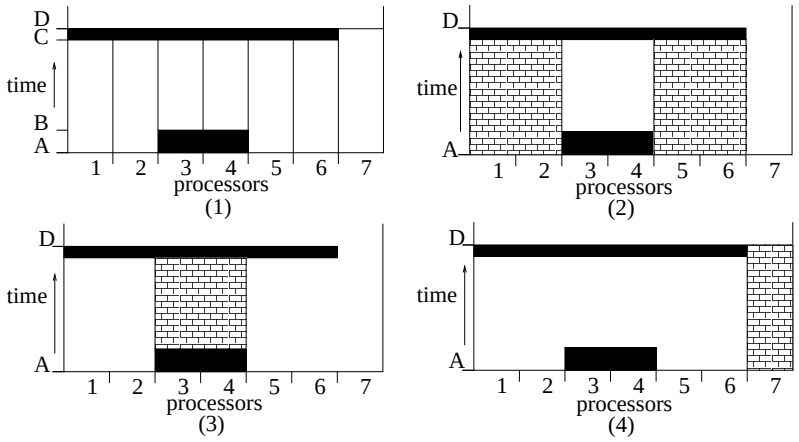


Figure 30.4. Buckets (1)-(4).

requirements alone, and therefore an offer can be built using resources 1, 2 and 5.

To consider combinations of buckets, we modify the contents of the buckets that consist of enough resources to generate a solution alone to contain one resource less than the required number of resources. We then examine additional solutions that are generated by combining elements from different buckets.

Care must be taken if solutions are created by combining resources from different buckets. The free-time intervals of the buckets can differ in start and end time. Therefore, the resulting start time of a combination is the maximum of all the start times of the combined buckets. Similarly, the end time is assigned as the minimum of all the end times of the combined buckets. The resulting time window of free resources must be sufficient for the request. In our example, the full set of possible solutions is {{1,2,5}, {1,2,3}, {1,2,4}, {1,2,7}, {1,3,4}, {1,3,7}, {1,4,7}, {2,3,4}, {2,3,7}, {3,4,7}}.

Given a set of intervals, we now need to select one. Step 5 involves making a coarse selection of intervals. The best solution can be found only when all possible time steps are considered; however, in practice this is not practiced because of the long running time of the algorithm.

A heuristic is used to reduce the number of combinations to be examined. To this end, we evaluate the earliest start time for each of the resource combinations and select the solution with the highest utility value. However, the actual start time in the time interval is not yet determined at this point because

```
list tempList; LOOP:while(generatedList not empty)
  {
    get the time t of the next element in the generatedList;

    test for all elements in tempList whether the difference
    between the beginning of the free interval and the time t
    is greater or equal to the run time of the job;

    if(number of elements in tempList that fulfill the time
       condition, is greater or equal the needed number of
       processors){
      create offers from the elements of the tempList;
    }
    if(enough offers found){
      finish LOOP;
    }
    add or subtract the elements of the generatedList to or
      from tempList that have time entry t;
   }
```

Figure 30.5. Pseudo-code to find potential solutions.

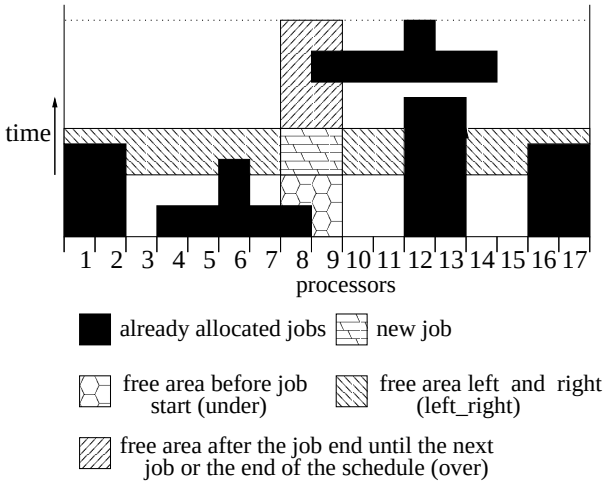


Figure 30.6. Parameters for the calculation of the owner utility function.

the intervals are equal to or larger than the requested allocation time. Therefore, different start times are examined to maximize the utility value in the step 6 of Figure 30.3. For this process, a parameter can be defined to specify the number of different start times to be considered within the given time interval. Note that we did not pose any requirements on the type of utility function (e.g. monotonic or continuous). However, the selection of a time interval is much simpler when the utility functions are restricted to be monotonic and continuous. After this phase the algorithm finishes and possible offers are generated.

The machine owner and the user can define their own utility functions as part of our economic model. Our implementation supports any mathematical formula that uses valid time and resource variables. Overall, we minimize the resulting value for the user's utility function, but the minimization problem can be transformed into a maximization problem and vice versa if needed.

The link between the objective functions of the user and the machine owner is created by the price for the machine use that is equal to the machine owner's utility function. The price may be included in the user's utility function as well.

The owner of the machine can specify additional variables in the utility function in our implementation, as shown in Figure 30.6. The variable *under* specifies the quantity of resources (processors) that are unused before the job is allocated. The variable *over* determines the quantity of unused resources after the job runs until the next job is started on the resources, or until the end of the temporarily schedule. The variable *left_right* specifies the idle resources during the run of the job. The variable *utilization* specifies the utilization of the machine if the job is allocated, as defined by the sum of the utilized resources

in relation to the available resources from the current time instance to the end of the schedule.

Note that networks have not been explicitly considered. However, our model can be easily extended to include network dependencies in the selection and evaluation process similar to the co-allocation process.

3.4 Co-Allocation (Multisite Scheduling)

When no single machine can meet the resource request by itself, we must consider multisite scheduling in order to generate a co-allocation between several machines. To accomplish this, a MetaManager requests resources from other remote MetaManagers. This process is initiated only if no domain is able to generate a solution using a single machine. We do not limit the co-allocation process to resources in a single domain; therefore, several aspects within the Grid must be taken into account, for instance, the site-autonomy, co-allocation, and cost management. For example, because sites are autonomous, an outside agent may not be able to discover the full state of nonlocal resources. Therefore, missing resources are queried by the scheduler (which is at another domain) for more limited information, namely, a fixed time frame.

During the initiation process two heuristics are used: one to fix a job part size and one to estimate start times. We divide the job into several smaller parts as specified by using two parameters, the minimum and maximum number of resources a job part may be allowed to use. These parameters can be specified by each domain. The second heuristic estimates the start times for the entire job. All job parts must be executed at the same time, but the initiating scheduler may have only limited information about the schedules on the other resources. Therefore several different start times are tried within the possible time interval given the job request. The number of tries can be specified for each domain separately. This process ends when a solution is found. As we do not have complete information about all local schedules, a solution may exist but not be found by our approach.

4. PERFORMANCE EVALUATION

In this section we evaluate the performance of our approach and compare it to the performance of a conventional scheduler with backfilling adapted to a Grid environment [EHS+02, EHSY02].

4.1 Resource Configuration

For our simulation experiments we define four different resource configurations, each with 512 processors, but with the processors distributed to the simulated sites in different ways, as shown in Table 30.2. These resource configurations are compatible to those used in [EHS+02, EHSY02].

Table 30.2. Examined resource configurations.

Identifier	Configuration	Maximum Size	Sum
m128	$4 \cdot 128$	128	512
m256	$2 \cdot 256$	256	512
m384	$1 \cdot 384 + 1 \cdot 64 + 4 \cdot 16$	384	512
m512	$1 \cdot 512$	512	512

The m128 and m256 configurations are an example of cooperation between several equally sized sites. In contrast, the m384 configuration represents a central processing center that is connected with several smaller client sites. The m512 configuration represents a single parallel machine for contrast.

In addition to the number of processors at each site, a local objective function is necessary for the equilibration process. We use six different objective functions in our experiments. Further research is needed to evaluate other utilization functions.

$$MF_1 = (NumberOfProcessors \cdot RunTime + over + under + left\_right) \cdot (1 - left\_right\_rel) \quad (30.3)$$

$$MF_2 = NumberOfProcessors \cdot RunTime + over + under + left\_right \quad (30.4)$$

$$MF_3 = (NumberOfProcessors \cdot RunTime + over + under) \cdot (1 - left\_right\_rel) \quad (30.5)$$

$$MF_4 = (NumberOfProcessors \cdot RunTime + left\_right) \cdot (1 - left\_right\_rel) \quad (30.6)$$

$$MF_5 = (NumberOfProcessors \cdot RunTime + over + left\_right) \cdot (1 - left\_right\_rel) \quad (30.7)$$

$$MF_6 = (NumberOfProcessors \cdot RunTime + under + left\_right) \cdot (1 - left\_right\_rel) \quad (30.8)$$

We derive objective functions 30.4 through 30.8 from Equation 30.3, so we use this basic formula for discussion.

The first term

$$NumberOfProcessors \cdot RunTime$$

describes the resource consumption by the job.

The second term

$$over + under + left\_right$$

represents the remaining idle resource time before and after the job on the same processors, as well as the concurrent idle times on the other resources (see Figure 30.6).

The last term of the first objective function

$$1 - left\_right\_rel$$

describes the relationship between the concurrent idle times and the resources consumption of the job itself, where

$$left\_right\_rel = \frac{left\_right}{RunTime \cdot NumberOfProcessors}.$$

A small value of $left\_right\_rel$ indicates that this assignment will leave few resources idle.

4.2 Job Configuration

The main problem when running simulations for Grid environments is the lack of real workloads for Grid computing. We derive a workload from real parallel machine traces from the Cornell Theory Center (CTC) IBM RS6000/SP parallel computer with 430 nodes. The workload is available from the standard workload archive maintained by Feitelson [Fei]. One of the reasons for choosing this workload was the existence of detailed analysis for the trace and the configuration by Hotovy [Hot96].

One of the problems when adapting the workload traces from a parallel computer to a Grid environment is the assignment of the jobs to the different sites. We use a round robin assignment of the jobs to the different machines in order to simulate local job submissions. In the CTC workload trace, node requirements with a power of two are favored, so when we adapted the traces to a Grid configuration we defined the number of nodes of each simulated machine to be a power of two.

Our configuration has a total of 512 processors, and there were only 430 nodes in the original configuration. The jobs in the trace still proved to be more than adequate for the conventional scheduling system to take advantage of backfilling, which needs a sufficient backlog of jobs in order to use idle resources [HSSY00].

Four workload sets, as shown in Table 30.3, are used in our experiments. The first three workload sets are extracts from the real CTC workload, each

Table 30.3. The used workloads.

Identifier	Description
10_20k_org	An extract of the original CTC traces from job 10000 to 20000.
30_40k_org	An extract of the original CTC traces from job 30000 to 40000.
60_70k_org	An extract of the original CTC traces from job 60000 to 70000.
syn_org	The synthetically generated workload derived from the CTC workload traces.

consisting of 10,000 jobs, or approximately three months in real time. The last workload set is synthetically generated [KSY99] in order to avoid any singular effects, for example caused by machine down time.

In addition to the information from the CTC workloads, each job must have a utility function associated with it to represent the user preferences. We define five user utility functions (UF) for our experiments:

$$UF_1 = (-StartTime) \tag{30.9}$$
$$UF_2 = (-JobCost) \tag{30.10}$$
$$UF_3 = (-(StartTime + JobCost)) \tag{30.11}$$
$$UF_4 = (-(StartTime + 2 \cdot JobCost)) \tag{30.12}$$
$$UF_5 = (-(2 \cdot StartTime + JobCost)) \tag{30.13}$$

The first user utility function (30.9) indicates a preference to start a job execution as soon as possible, so the start time is minimized and the processing costs are ignored. In contrast, the second user utility function considers only the costs of using the resource to run the job caused by the job and ignores the start time completely. The other three user utility functions are combinations of the first two with different weights.

4.3 Results

We used discrete event-based simulations on the configurations and settings. Combinations of all the user utility functions and machine functions were used in conjunction with the set of workloads and resource configurations defined in the previous section. In this section we present only the best results for the conventional first-come first-serve/backfilling approach with the best results for the economic model in this section. A more detailed analysis of our results can be found in [EHY02].

To compare approaches, we use the average weighted response time, defined as the sum of the corresponding run and wait times weighted by the resource consumption (the number of resources multiplied with the job execution time). The resource consumption weight prevents any preferring of smaller to larger jobs with respect to the average weighted response time [SY98]. For the evaluation we assume that a smaller average weighted response time is anticipated from the user. Note that the user utility functions as well as the owner machine functions in the figure may vary between configurations and workloads. In some cases, good results have been achieved by using the user utility function UF_1(30.9) (fastest start time) in combination with machine function MF_1 (30.3) (the combination of earliest start time, minimizing idle time and minimizing resources).

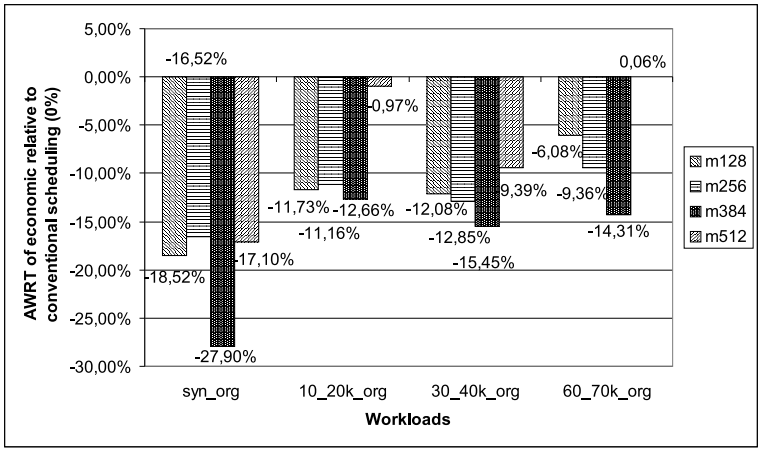


Figure 30.7. Comparison between Economic and Conventional scheduling.

Figure 30.7 shows that the economic scheduling outperforms the conventional first-come first-served/backfilling strategy. This is because the economic model can place jobs without regard to the submission order. In [EHY02] a much deeper analysis of the results can be found with additional analysis of the effects of the chosen resource configurations and the user objective/owner machine functions.

5. CONCLUSION

In this chapter we introduced an economic scheduling approach to Grid environments. We described the architecture and scheduling process, and we presented an evaluation of the strategy. The results of the simulations for different workload sets, different resource configurations, and several different parameter settings for the objective functions show that our economic scheduling system is competitive with the conventional scheduling systems in terms of the average weighted response time.

The economic approach provides several additional advantages over conventional approaches, such as allowing for site autonomy and the ability to use heterogeneous resources. The utility/objective functions for each Grid user and resource owner can be defined separately for each job request. The utility/objective functions we used were only first attempts, so additional work in defining objective functions may improve our results. However, the achieved results readily motivate the use of economic-based scheduling systems.

In future research, data and network management will be integrated to allow network data to be considered. This integration is especially important since the network is a limited resource especially during the execution of multisite

jobs. In addition, the scheduler will be able to automatically transfer the needed data over the network. QoS features and advance reservations may be exploited for network and storage resources as well.

For those interested in further discussion of this topic, please see Chapters 28 and 29. Additional information can be found in [EHY02, BAGS02, MFGH88, TB96].

References

[AAD+92] A. Abramovici, W. Althouse, R. Drever, Y. Gursel, S. Kawamura, F. Raab, D. Shoemaker, L. Sievers, R. Spero, K. Thorne, R. Vogt, R. Weiss, S. Whitcomb, and M. Zucker. LIGO: The Laser Interferometer Gravitational-wave Observatory (in large scale measurements). *Science*, 256(5055):325, 1992.

[AAF+01] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus worm: Experiments with dynamic resource discovery and allocation in a Grid environment. *International Journal of High Performance Computing Applications*, 15(4):345–358, 2001.

[AAG+02] G. Allen, D. Angulo, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, M. Russell, T. Radke, E. Seidel, J. Shalf, and I. Taylor. GridLab: Enabling applications on the Grid. In *Proceedings of the Third International Workshop on Grid Computing (Grid2002)*, November 2002.

[AB02] R. Albert and A. L. Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[ABB+02a] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in high-performance computational Grid environments. *Parallel Computing Journal*, 28(5):749–771, 2002.

[ABB+02b] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. GridFTP protocol specification. Technical report, Global Grid Forum GridFTP Working Group, September 2002.

[ABC+01] Andy Adamson, Giovanni Bennini, Phil Chimento, Larry Dunn, Reinhard Frank, Rüdiger Geib, Hermann Granzer,

Sue Hares, Haci Mantar, Will Murray, Rob Neilson, Ibrahim Okumus, Francis Reichmeyer, Alain Roy, Volker Sander, Dave Spence, Ben Teitelbaum, Andreas Terzis, and Jerr Wheeler. QBone signaling design team final report, 2001. Available from `http://qos.internet2.edu/wg/documents-informational/20020709-chimento-e%tal-qbone-signaling`.

[Abd00] T. F. Abdelzaher. An automated profiling subsystem for QoS-aware services. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, June 2000.

[ABG02] D. Abramson, R. Buyya, and J. Giddy. A computational economy for Grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*, 18(8), October 2002.

[ABGL02] K. Anstreicher, N. Brixius, J.-P. Goux, and J. T. Linderoth. Solving large quadratic assignment problems on computational Grids. *Mathematical Programming*, 91(3):563–588, 2002.

[ABH+99] G. Allen, W. Benger, C. Hege, J. Masso, A. Merzky, T. Radke, E. Seidel, and J. Shalf. Solving Einstein's equations on supercomputers. *IEEE Computer Applications*, 32(12):52–58, 1999.

[ABL+95] Jose Nagib Cotrim Arabe, Adam Beguelin, Bruce Lowekamp, Erik Seligman, Mike Starkey, and Peter Stephan. Dome: Parallel programming in a heterogeneous multi-user environments. Technical Report CMU-CS-95-137, Carnegie Mellon University, School of Computer Science, 1995.

[AC02] G. Aloisio and M. Cafaro. Web-based access to the Grid using the Grid Resource Broker Portal. *Concurrency and Computation: Practice and Experience, Special Issue on Grid Computing Environments*, 14:1145–1160, 2002.

[ACK+02] D. P. Anderson, J. Cobb, E. Korpella, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45:56–61, 2002.

[ACK+04] Malcolm Atkinson, Ann Chervenak, Peter Kunszt, Inderpal Narang, Norman Paton, Dave Pearson, Arie Shoshani, and

Paul Watson. Data access, integration, and management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[ACPtNt95] Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW team. A case for Networks of Workstations (NOW). *IEEE Micro*, February 1995.

[ADAD01] Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau. Information and control in gray-box systems. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, October 2001.

[ADD+03] Gabrielle Allen, Kelly Davis, Konstantinos N. Dolkas, Nikolaos D. Doulamis, Tom Goodale, Thilo Kielmann, Andre Merzky, Jarek Nabrzyski, Juliusz Pukacki, Thomas Radke, Michael Russell, Ed Seidel, John Shalf, and Ian Taylor. Enabling applications on the Grid: A GridLab overview. *International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications*, August *to appear*, 2003.

[ADF+01] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting efficient execution in heterogeneous distributed computing environments with Cactus and Globus. In *Proceedings of SuperComputing (SC'01)*, 2001.

[ADF+03] Daniel E. Atkins, Kelvin K. Droegemeier, Stuart I. Feldman, Hector Garcia-Molina, Michael L. Klein, David G. Messerschmitt, Paul Messina, Jeremiah P. Ostriker, and Margaret H. Wright. Revolutionizing science and engineering through cyberinfrastructure: Report of the National Science Foundation blue ribbon advisory panel on cyberinfrastructure. Technical report, NSF, 2003. Available from `http://www.communitytechnology.org/nsf_ci_report/report.pdf`.

[Adv93] Vikram S. Adve. *Analyzing the Behavior and Performance of Parallel Programs*. PhD thesis, University of Wisconsin-Madison, December 1993. Also available as University of Wisconsin Computer Sciences Technical Report #1201.

[AFF+01] Karen Appleby, Sameh Fakhouri, Liana Fong, German Goldszmidt, Michael Kalantar, Srirama Krishnakumar, Donald

Pazel, John Pershing, and Benny Rochwerger. Oceano: SLA based management of a computing utility. In *Proceedings of the Seventh IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[AG] AccessGrid. `http://www-fp.mcs.anl.gov/fl/accessgrid/`.

[AGK00] D. Abramson, J. Giddy, and L. Kotler. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, May 2000.

[AGM+90] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[AH00] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5, 2000. Also available from `http://www.firstmonday.dk/issues/issue5_10/adar/`.

[AHLP01] L. Adamic, B. Huberman, R. Lukose, and A. Puniyani. Search in power law networks. *Physical Review E*, 64:46135–46143, 2001.

[AJF+04] Jim Austin, Tom Jackson, Martyn Fletcher, Mark Jessop, Peter Cowley, and Peter Lobner. Predictive maintenance: Distributed aircraft engine diagnostics. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[AK97] J. L. Ambite and C. A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[AK02] D. P. Anderson and J. Kubiatowicz. The worldwide computer. *Scientific American*, March 2002.

[AKvL97] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. In E. H. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley, 1997.

[AL97] E. H. L. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, 1997.

[AMS97] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[ARC] GGF Architecture Area (ARCH). `http://www.ggf.org/5_ARCH/arch.htm`.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the Twentieth International Conference on Very Large Databases (VLDB'94)*, 1994.

[AS04] Gabrielle Allen and Ed Seidel. Collaborative science: Astrophysics requirements and experiences. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infastructure (Second Edition)*. Morgan Kaufmann, 2004.

[ASGH95] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A tool for performing parameterized simulations using distributed workstations. In *Proceedings of the Fourth IEEE International Symposium on High-Performance Distributed Computing (HPDC-4)*, August 1995.

[AV93] Vikram Adve and Mary Vernon. The influence of random delays on parallel execution times. In *Proceedings of Sigmetrics '93*, 1993.

[Ava] Avaki. `http://www.avaki.com`.

[AVD01] D. C. Arnold, S. Vadhiyar, and J. Dongarra. On the convergence of computational and data Grids. *Parallel Processing Letters*, 11(2-3):187–202, September 2001.

[AY97] C. C. Aggarwal and P. S. Yu. On disk caching of Web objects in proxy servers. In *Proceedings of the Internationals Conference Info and Knowledge Management (CIKM'97)*, pages 238–245, 1997.

[AZV+02] James Annis, Yong Zhao, Jens Voeckler, Michael Wilde, Steve Kent, and Ian Foster. Applying Chimera virtual data concepts to cluster finding in the Sloan Sky Survey. In *Proceedings of SuperComputing (SC'02)*, 2002.

[BA99] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[BAG00] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in

a global computational Grid. In *Proceedings of the Fourth International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, 2000.

[BAGS02] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15):1507–1542, November - December 2002.

[BAJZ98] J. Bonkalski, R. Anderson, S. Jones, and N. Zaluzec. Bringing telepresence microscopy and science collaboratories into the class room. *TeleConference Magazine*, 17(9), 1998.

[Bar78] B. Austin Barry. *Errors in Practical Measurement in Science, Engineering and Technology*. John Wiley & Sons, 1978.

[Bar02] A. L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[BBADAD02] Nathan C. Burnett, John Bent, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Exploiting gray-box knowledge of buffer-cache management. In *Proceedings of USENIX*, 2002.

[BBC+98] S. Blake, D. Black, M. Carlson, M. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Technical Report RFC 2475, Internet Engineering Task Force (IETF), 1998.

[BBH+99] Jon Bakken, Eileen Berman, Chih-Hao Huang, Alexander Moibenko, Don Petravick, Ron Rechenmacher, and Kurt Ruthmansdorfer. Enstore technical design document. Technical Report JP0026, Fermi National Accelorator Laboratory, June 1999.

[BC96] Azer Bestavros and Carlos Cunha. Server-initiated document dissemination for the WWW. *IEEE Data Engineering Bulletin*, 19, 1996.

[BCC+97] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.

[BCC+01] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, and R. Wolski. The GrADS project: Software support for high-level Grid application development. *International Journal of High-Performance Computing Applications*, 15(4):327–344, 2001.

[BCF+98] Sharon Brunett, Karl Czajkowski, Steven Fitzgerald, Ian Foster, Andrew Johnson, Carl Kesselman, Jason Leigh, and Steven Tuecke. Application experiences with the Globus toolkit. In *Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, pages 81–89, 1998.

[BCF+99] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of InfoCom*, 1999.

[BCS94] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Technical Report RFC 1633, Internet Engineering Task Force (IETF), 1994.

[BDG+98] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing distributed synthetic forces simulations in metacomputing environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 29–42, 1998.

[BDG+03] Jim Blythe, Ewa Deelman, Yolanda Gil, Carl Kesselman, Amit Agarwal, Gaurang Mehta, and Karan Vahi. The role of planning in Grid computing. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 2003.

[BDGK03] Jim Blythe, Ewa Deelman, Yolanda Gil, and Carl Kesselman. Transparent Grid computing: A knowledge-based approach. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, 2003.

[BDM+99] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112:3–41, 1999.

[BEK+00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk, Satish Thatte, and

Dave Winer. Simple Object Access Protocol (SOAP) 1.1. Technical Report Note 08, World Wide Web Consotium (W3C), May 2000. Available from `http://www.w3.org/TR/SOAP/`.

[Ber99] F. Berman. High performance schedulers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 12, pages 279–309. Morgan Kaufmann, 1999.

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote trust-management system version 2. Technical Report RFC 2704, Internet Engineering Task Force (IETF), September 1999.

[BGA00] R. Buyya, J. Giddy, and D. Abramson. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. In *Proceedings of the Second Workshop on Active Middleware Services (AMS 2000), conjunction with the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, August 2000.

[BGA01] R. Buyya, J. Giddy, and D. Abramson. A case for economy Grid architecture for service-oriented Grid computing. In *Proceedings of the Heterogeneous Computing Workshop*, April 2001.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman, 1987.

[BHK01] Ian Bird, Bryan Hess, and Andy Kowalski. Building the mass storage system at Jefferson Lab. In *Proceedings of the Eighteenth IEEE Mass Storage Systems Conference*, 2001.

[BHKL00] B. Bode, D. M. Halstead, R. Kendall, and Z. Lei. The Portable Batch Scheduler and the Maui scheduler on Linux clusters. In *Proceedings of USENIX*, 2000.

[BHL+99] A. Bayucan, R. L. Henderson, C. Lesiak, N. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. Technical Report Release 2.2, MRJ Technology Solutions, November 1999.

[BJB+00] Judy Beiriger, Wilbur Johnson, Hugh Bivens, Steven Humphreys, and Ronald Rhea. Constructing the ASCI Grid. In *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, 2000.

[BKKW96] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff. Charlotte: Metacomputing on the Web. In *Proceedings of the Ninth International Conference on Parallel and Distributed Computing Systems*, 1996.

[BL97] J. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer, 1997.

[BL98] S. Berson and R. Lindell. An architecture for advance reservations in the Internet. Technical report, Information Sciences Institute, University of Southern California, 1998. Available from `www.isi.edu/~berson/advance.ps`.

[BL99a] J. Basney and M. Livny. Deploying a high throughput computing cluster. In *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall, 1999.

[BL99b] J. Basney and M. Livny. Improving goodput by co-scheduling CPU and network capacity. *International Journal of High Performance Computing Applications*, 13, 1999.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific American*, 284(5):34–43, 2001.

[BLM00] J. Basney, M. Livny, and P. Mazzanti. Harnessing the capacity of computational Grids for high energy physics. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP 2000)*, 2000.

[BLRK83] J. Błażewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints. *Discrete Applied Mathematics*, 5, 1983.

[BMRW98] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Proceedings of the Eighth Annual IBM Centers for Advanced Studies Conference (CASCON '98)*, 1998.

[Bog94] N. Bogan. Economic allocation of computation time with computation markets. Master's thesis, Department of Elec-

trical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994.

[BOI] The Berkeley Open Infrastructure for Network Computing (BOINC). `http://boinc.berkeley.edu`.

[Bou96] J.-Y. L. Boudec. Network calculus made easy. Technical Report epfl-di 96/218, Ecole Polytechnique Federale, Lausanne (EPFL), 1996.

[bpr] Bproc: Beowulf distributed process space. `http://bproc.sourceforge.net/bproc.html`.

[BR96] M. Baxter and A. Rennie. *Financial Calculus: An Introduction to Derivative Pricing*. Cambridge University Press, 1996.

[Bre02] B. Brewin. Intel introduces 3Ghz desktop chip. *ComputerWorld*, November 2002.

[BS98] L. Breslau and S. Shenker. Best-effort versus reservations: A simple comparative analysis. *ACM Computer Communication Review*, 28(4):3–16, September 1998.

[BSST96] T. Brecht, H. Sandhu, M. Shan, and J. Talbot. ParaWeb: Towards world-wide supercomputing. In *Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.

[BT00] J.-Y. L. Boudec and P. Thiran. *Network Calculus A Theory of Deterministic Queuing System for the Internet*. Springer Verlag, 2000.

[Bur00] E. Burger. *Mastering the Art of Magic*. Kaufman and Company, 2000.

[BvST00] G. Ballintijn, M. van Steen, and A. S. Tanenbaum. Scalable naming in global middleware. In *Proceedings of Thirteenth International Conference on Parallel and Distributed Computing Systems (PDCS-2000)*, 2000.

[BW96] F. Berman and R. Wolski. Scheduling from the perspective of the application. In *Proceedings of the Fifth IEEE International Symposium on High-Performance Distributed Computing (HPDC-5)*, 1996.

[BW99] B. C. Barish and R. Weiss. LIGO and the detection of gravitational waves. *Physics Today*, 52(10):44, 1999.

[BWC+03] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the Grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), April 2003.

[BWF+96] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. In *Proceedings of SuperComputing (SC'96)*, 1996.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. Technical Report RFC 2205, Internet Engineering Task Force (IETF), 1997.

[CAC] Cactus. `http://www.cactuscode.org/`.

[CAS] Castor: the CERN advanced storage manager - Castor architecture. `http://castor.web.cern.ch/castor/DOCUMENTATION/ARCHITECTURE`.

[Cat92] Charlie Catlett. In search of gigabit applications. *IEEE Communications Magazine*, 30(4):42–51, April 1992.

[CB97] M. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5, December 1997.

[CB00] A. Charny and J.-Y. L. Boudec. Delay bounds in a network with aggregate scheduling. In *Proceedings of the International Workshop on Quality of Future Internet Services (QoS 2000)*, 2000.

[CB02] M. Chetty and R. Buyya. Weaving electrical and computational Grids: How analogous are they? *Computing in Science and Engineering*, 4(4):61–71, July/August 2002.

[CBB+02] A. Charny, J. C. R. Bennett, K. Benson, J. Y. Le Boudec, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Kalmanek, and K. K. Ramakrishnan. Supplemental information for the new definition of the EF PHB (expedited forwarding per-hop behavior). Technical Report RFC 3247, Internet Engineering Task Force (IETF), 2002.

[CCEB03] Andrew Chien, Bradley Calder, Stephen Elbert, and Karan Bhatia. Entropia: Architecture and performance of an Enterprise desktop Grid system. *Journal of Parallel and Distributed Computing, to appear,* 2003.

[CCF+01] P. Chandra, Y. Chu, A. Fisher, J. Gao, C. Kosak, T. S. Eugene Ng, P. Steenkiste, E. Takahashi, and H. Zhang. Darwin: Customizable resource management for value-added network services. *IEEE Network Magazine*, 15(1), 2001.

[CCI88] CCITT. Recommendations X.509 - the directory-authentication framework. Technical report, Consultations Committee International Telephone and Telegraph, International Telecommunications Unison, 1988.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, 2001. Available from `http://www.w3.org/-TR/-wsdl/`.

[CCO+03] Shane Canon, Steve Chan, Doug Olson, Craig Tull, and Von Welch. Using CAS to manage role-based VO sub-groups. In *Proceedings of the International Conference for Computing in High Energy and Nuclear Physics (CHEP-2003)*, 2003.

[CDF+01] B. Coghlan, A. Djaoui, S. Fisher, J. Magowan, and M. Oevers. Time, information services and the Grid. In *Proceedings of the British National Conference on Databases*, 2001.

[CDF+02] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney. Giggle: A framework for constructing scalable replica location services. In *Proceedings of SuperComputing (SC'02)*, 2002.

[CDK+02] Ann Chervenak, Ewa Deelman, Carl Kesselman, Laura Pearlman, and Gurmeet Singh. A metadata catalog service for data intensive applications. Technical report, GriPhyN, 2002. Also available from `http://www.isi.edu/~deelman/mcs.pdf`.

[CDO+00] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi. Access Grid: Immersive group-to-group collaborative visualization. In *Proceedings of the Fourth International Immersive Projection Technology Workshop*, 2000.

[CDS01] C. Courcoubetis, M. Dramitinos, and G. Stamoulis. An auction mechanism for bandwidth allocation over paths. In *Proceedings of the Seventeenth International Teletraffic Congress (ITC)*, December 2001.

[CE] S. Cantor and M. Erdos. Shibboleth-architecture draft v05. `http://shibboleth.internet2.edu/draft-internet2-shibboleth-arch-v05.html`.

[Čer85] V. Černy. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[CFFK01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[CFG02] A. Chien, I. Foster, and D. Goddette. Grid technologies empowering drug discovery. *Drug Discovery Today*, 7(20):176–180, 2002.

[CFHB99] Mark Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in Web servers. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1999.

[CFK+98a] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Resource management for value-added customizable network service. In *Proceedings of the Sixth IEEE International Conference on Network Protocols (ICNP'98)*, 1998.

[CFK+98b] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459)*, pages 62–82. Springer-Verlag, 1998.

[CFK99] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational Grids. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, August 1999.

[CFK+02] K. Czajkowski, I. Foster, C. Kesselman., V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS #2537)*, pages 153–183. Springer-Verlag, 2002.

[CGD] Climate and Global Dynamics Division, National Center for Atmospheric Research (NCAR). `http://www.cgd.ucar.edu`.

[CGS] GGF CIM-based Grid Schema Working Group (CGS-WG). `http://www.isi.edu/~flon/cgs-wg/index.htm`.

[Che01] Lap-Sun Cheung. A fuzzy approach to load balancing in a distributed object computing network. In *Proceedings of the First IEEE International Symposium of Cluster Computing and the Grid (CCGrid'01)*, pages 694–699, 2001.

[Chi] ChicSim: The Chicago Grid Simulator. `http://people.cs.uchicago.edu/~krangana/ChicSim.html`.

[Chi04] A. Chien. Massively distributed computing: Virtual screening on desktop computers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[CHTCB96] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, 1996.

[CI97] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[CJ89] Dah-Ming Chiu and Raj Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN*, 17(1):1–14, June 1989.

[CKKG99] S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw. Resource management in Legion. *Future Generation Computing Systems*, 15:583–594, October 1999.

[CME] Chicago Mercantile Exchange. `http://www.cme.com`.

[CMPT04] Jon Crowcroft, Tim Moreton, Ian Pratt, and Andrew Twigg. Peer-to-peer technologies. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[CMS] Compact Muon Solenoid (CMS). `http://cmsinfo.cern.ch/Welcome.html/`.

[CN99] H. Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1999.

[COBW00] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS parameter sweep template: User-level middleware for the Grid. In *Proceedings of SuperComputing (SC'00)*, 2000.

[Cof76] E. G. Coffmanm, editor. *Computer and Job-Shop Scheduling Theory*. John Wiley and Sons, New York, 1976.

[CON] Condor project. `http://www.cs.wisc.edu/condor`.

[Cro58] G. A Croes. A method for solving traveling salesman problems. *Operations Research*, 6:791–812, 1958.

[CS92] C. Catlett and L. Smarr. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[CS00] L. Clewlow and C. Strickland. *Energy Derivatives: Pricing and Risk Management*. Lacima Publications, 2000.

[CS02] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proceedings of the SIGCOMM*, 2002.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[DAG] The Condor Directed-Acyclic-Graph Manager (DAGMan). `http://www.cs.wisc.edu/condor/dagman/`.

[Dah99] M. Dahlin. Interpreting stale load information. In *Proceedings of the Ninteenth International Conference on Distributed Computing Systems*, 1999.

[Dai01] H. J. Dail. A modular framework for adaptive scheduling in Grid application development environments. Technical Report CS2002-0698, Computer Science Department, University of California, California, San Diego, 2001.

[DAM] GGF Discovery and Monitoring Event Data Working Group (DAMED-WG). `http://www-didc.lbl.gov/damed/`.

[DAP] Dap scheduler. `http://www.cs.wisc.edu/condor/dap`.

[DBC03] H. Dail, F. Berman, and H. Casanova. A decoupled scheduling approach for Grid application development environments. *Journal of Parallel and Distributed Computing, to appear*, 2003.

[DBG+03a] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbee, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto Grid environments. *Journal of Grid Computing*, 1, 2003.

[DBG+03b] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Scott Koranda, Albert Lazzarini, and Maria Alessandra Papa. From metadata to execution on the Grid: The Pegasus pulsar search. Technical Report 2003-15, GriPhyN, 2003. Available from `http://www.griphyn.org/documents`.

[DCB+01] B. Davie, A. Charney, J. C. R. Bennett, K. Benson, J. Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding PHB. Technical Report RFC 3246, Internet Engineering Task Force (IETF), 2001.

[DFJ+96] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proceedings of the Twenty-second Conference on Very Large Data Bases (VLDB'96)*, 1996.

[DH77] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6), June 1977.

[DI89] Murthy Devarakonda and Ravishankar Iyer. Predictability of process resource usage: A measurement-based study on

UNIX. *IEEE Transactions on Software Engineering*, 15, December 1989.

[Din99] P. A. Dinda. The statistical properties of host load. *Scientific Programming*, 7:3–4, Fall 1999.

[Din01] P. Dinda. Online prediction of the running time of tasks. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.

[Din02] P. A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3), 2002.

[DKM01] Ewa Deelman, Carl Kesselman, and Gaurang Mehta. Transformation catalog design for GriPhyN, prototype of transformation catalog schema. Technical Report 2001-17, GriPhyN, 2001. Available from `http://www.griphyn.org/documents`.

[DKM+02] Ewa Deelman, Carl Kesselman, Gaurang Mehta, Leila Meshkat, Laura Pearlman, Kent Blackburn, Phil Ehrens, Albert Lazzarini, Roy Williams, and Scott Koranda. GriPhyN and LIGO: Building a virtual data Grid for gravitational wave scientists. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[DKPS97] M. Degermark, T. Kohler, S. Pink, and O. Schelen. Advance reservations for predictive service in the Internet. *Multimedia Systems*, 5(3):177–186, 1997.

[DL03] A. Dittmer and I. Lumb. Building a complete resource management solution: Solaris SRM and Platform LSF. In *Proceedings of the Sun User Performance Group*, May 2003.

[DM90] C. Darken and J. Moody. Fast adaptive k-means clustering: Some empirical results. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, pages 233–238. IEEE Neural Networks Council, 1990.

[DO00] P. A. Dinda and D. R. O'Hallaron. Realistic CPU workloads through host load trace playback. In *Proceedings of the Fifth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR 2000)*, 2000.

[Doa96] M. Doar. A better model for generating test networks. *IEEE Global Internet*, 1996.

[DOEa] DOE Grids CA. http://www.doegrids.org/.

[DOEb] DOE Science Grid. http://doesciencegrid.org/.

[Dow97] A. Downey. Predicting queue times on space-sharing parallel computers. In *Proceedings of the International Parallel Processing Symposium (IPPS)*, 1997.

[DP96a] Jay Devore and Roxy Peck. *Statistics: The Exploration and Analysis of Data*, page 88. Duxbury Press, 1996.

[DP96b] Jay Devore and Roxy Peck. *Statistics: The Exploration and Analysis of Data*, page 567. Duxbury Press, 1996.

[DR99] T. Dierks and E. Rescorla. The TLS Protocol, version 1.1. Technical Report RFC 2246, Internet Engineering Task Force (IETF), January 1999.

[DRM] GGF Distributed Resource Management Application API Working Group (DRMAA-WG). http://www.drmaa.org/.

[DS81] N. R. Draper and H. Smit. *Applied Regression Analysis (Second Edition)*. John Wiley and Sons, 1981.

[EDGa] European DataGrid Project. http://www.eu-datagrid.org.

[EDGb] European DataGrid CA. http://marianne.in2p3.fr/datagrid/ca/ca-table-ca.html.

[EH99] Thomas Eickermann and Ferdinand Hommes. Metacomputing in a gigabit testbed west. In *Proceedings of the Workshop on Wide Area Networks and High Performance Computing*, Lecture Notes in Control and Information Sciences, pages 119–129. Springer-Verlag, 1999.

[EHS+02] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. On advantages of Grid computing for parallel job scheduling. In *Proceedings of the Second IEEE International Symposium on Cluster Computing and the Grid (CCGrid'02)*, pages 39–46, 2002.

[EHSY02] C. Ernemann, V. Hamscher, A. Streit, and R. Yahyapour. On effects of machine configurations on parallel job scheduling in computational Grids. In *Proceedings of the International*

Conference on Architecture of Computing Systems, (ARCS 2002), pages 169–179, April 2002.

[EHY02] C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in Grid computing. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS #2537)*, pages 129–152. Springer-Verlag, 2002.

[EJ01] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). Technical Report RFC 3174, Internet Engineering Task Force (IETF), 2001.

[Ell99] C. Ellison. SPKI requirements. Technical Report RFC 2692, Internet Engineering Task Force (IETF), 1999.

[EMC] EMC Corporation. `http://www.emc.com`.

[EP04] M. Ellisman and S. Peltier. Medical data federation: The biomedical informatics research network. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[EUR] Eurogrid. `http://www.eurogrid.org`.

[FAS] FASTA package of sequence comparison programs. `ftp://ftp.virginia.edu/pub/fasta`.

[FB96] S.M. Figueira and F. Berman. Mapping parallel applications to distributed heterogeneous systems. Technical Report UCSD CS Tech Report # CS96-484, University of California, San Diego, June 1996.

[FC90] R. F. Freund and D. S. Conwel. Superconcurrency: A form of distributed heterogeneous supercomputing. *Supercomputing Review*, 3(10):47–50, October 1990.

[Fei] D. G. Feitelson. Parallel workloads archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[Fei95] D. Feitelson. A survey of scheduling in multiprogrammed parallel systems. Technical Report RC 19790, IBM T. J. Watson Research Center, October 1995.

[FFR+02] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. End-to-end quality of service for high-end applications. *Computer Communications, Special Issue on Network Support for Grid Computing*, 2002.

[FG03] I. Foster and D. Gannon. Open Grid Services Architecture: A roadmap. Technical report, Open Grid Services Architecture Working Group, Global Grid Forum, February 2003. Available from `http://www.ggf.org/ogsa-wg/ogsa_roadmap.0.4.pdf`.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical Report RFC 2616, Internet Engineering Task Force (IETF), June 1999.

[FGN+96] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY high-performance distributed computing experiment. In *Proceedings of the Fifth IEEE International Symposium on High-Performance Distributed Computing (HPDC-5)*, pages 562–571, 1996.

[FGT96] I. Foster, J. Geisler, and S. Tuecke. MPI on the I-WAY: A wide-area, multimethod implementation of the Message Passing Interface. In *Proceedings of the 1996 MPI Developers Conference*, pages 10–17, 1996.

[FGV97] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. *Multimedia Systems*, 5(3), 1997.

[FH98] P. Ferguson and G. Huston. Quality of service on the Internet: Fact, fiction or compromise? In *Proceedings of Inet '98*, 1998.

[FH02] S. Farrell. and R. Housley. An Internet attribute certificate profile for authorization. Technical Report RFC 3281, Internet Engineering Task Force (IETF), 2002.

[FI03] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and Grid computing. In *Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[Fid03] M. Fidler. Extending the network calculus pay bursts only once principle to aggregate scheduling. In *Proceedings of the*

Proceedings of the Second International Workshop on QoS in Multiservice IP Networks (QoS-IP), 2003.

[FJL+88] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors*. Prentice-Hall, 1988.

[FK97] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–129, 1997.

[FK98a] I. Foster and C. Kesselman. The Globus Project: A status report. In *Proceedings of the Seventh Heterogeneous Computing Workshop*, 1998.

[FK98b] S. Frolund and J. Koistinen. QML: A language for quality of service specification. Technical Report HPL-98-10, HP Labs, February 1998.

[FK99a] Ian Foster and Carl Kesselman. The Globus Toolkit. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 11, pages 259–278. Morgan Kauffmann, 1999.

[FK99b] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann, 1999.

[FK04] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[FKH+99] A. J. Ferrari, F. Knabe, M. A. Humphrey, S. J. Chapin, and A. S Grimshaw. A flexible security system for metacomputing environments. In *Proceedings of High Performance Computing and Networking Europe (HPCN Europe '99)*, 1999.

[FKK96] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 Protocol. Technical report, Netscape Communications Corporation, November 1996.

[FKNT02] I Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed systems integration. *IEEE Computer*, 35(6):37–46, 2002.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. Also available from `http://www.globus.org/research/papers/anatomy.pdf`.

[FKTT98] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational Grids. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security Conference*, 1998.

[Fla98] Gary W. Flake. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press, Cambridge, MA, 1998.

[FN95] D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the First International JSSPP Workshop; LNCS #949)*. Springer-Verlag, 1995.

[FNA] Fermi National Accelerator Laboratory. `http://www.fnal.gov`.

[For94] MPI Forum. MPI: A Message-Passing Interface standard. Technical Report CS-94-230, University of Tennessee, Knoxville, 1994.

[FR01] D.G. Feitelson and L. Rudolph, editors. *Job Scheduling Strategies for Parallel Processing (Proceedings of the Seventh International JSSPP Workshop; LNCS #2221)*. Springer Verlag, 2001.

[FRS+97] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong. Theory and practice in parallel job scheduling. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Third International JSSPP Workshop; LNCS #1291)*. Springer-Verlag, 1997.

[FRS00] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the International Workshop on Quality of Service*, 2000.

[FTF+02] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[Fus] Fusion Grid. `http://www.fusiongrid.org/`.

[FV90] D. Ferrari and D. Verma. A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications, 8(3)*, 1990.

[FVWZ02] Ian Foster, Jens Vockler, Michael Wilde, and Yong Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proceedings of the Fourteenth International Conference on Scientific and Statistical Database Management (SSDBM'02)*, 2002.

[FW98] Dror Feitelson and Ahuva Weil. Utilization and predictability in scheduling the IBM SP2 with backfilling. In *Proceedings of Twelfth International Parallel Processing Symposium and Ninth Symposium on Parallel and Distributed Processing*, 1998.

[FWM94] G. C. Fox, R. D. Williams, and P. C. Messina. *Parallel Computing Works*. Morgan Kaufmann, 1994.

[GAL+03] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf. The Cactus framework and toolkit: Design and applications. In *Proceedings of Vector and Parallel Processing (VECPAR)*, 2003.

[GBE+98] Amin (Grid2001)Vahdat, Eshwar Belani, Paul Eastham, Chad Yoshikawa, Thomas Anderson, David Culler, and Michael Dahlin. WebOS: Operating system services for wide area applications. In *Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, 1998.

[GBHC00] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for Internet service construction. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)*, 2000.

[GCC+04] Greg Graham, Richard Cavanaugh, Peter Couvares, Alan DeSmet, and Miron Livny. Distributed data analysis: Federated computing for high energy physics. In Ian Foster and

Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[GCP] GGF Grid Certificate Policy Working Group (GCP-WG). `http://www.gridforum.org/2_SEC/GCP.htm`.

[GDM] Grid Data Mirroring Package (GDMP). `http://project-gdmp.web.cern.ch/project-gdmp/`.

[GDRSF04] Carole A. Goble, David De Roure, Nigel R. Shadbolt, and Alvaro Fernandes. Enhancing services and applications with knowledge and semantics. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[Gen04] Wolfgang Gentzsch. Enterprise resource management: Applications in research and industry. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[GES] GGF Grid Economic Services Architecture Working Group (GESA-WG). `http://www.gridforum.org/3_SRM/gesa.htm`.

[GFKH99] A. S. Grimshaw, A. J. Ferrari, F. Knabe, and M. A. Humphrey. Wide-area computing: Resource sharing on a large scale. *IEEE Computer*, 32(5), May 1999.

[GGF] Global Grid Forum (GGF). `http://www.ggf.org`.

[Gib97] R. Gibbons. A historical application profiler for use by parallel schedulers. *Lecture Notes on Computer Science*, 1297, 1997.

[Gib02] W. Gibbs. Ripples in spacetime. *Scientific American*, April 2002.

[GIM01] Entropia, researchers discover largest multi-million-digit prime using Entropia distributed computing Grid. Press release, Entropia, Inc., December 2001.

[GKTA02] Sven Graupner, Vadim Kotov, Holger Trinks, and Artur Andrzejak. Control architecture for service Grids in a federation of utility data centers. Technical Report HPL-2002-235, HP Labs, 2002.

[GL] GridLab project. `http://www.gridlab.org`.

[GL97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[GLM] GridLab monitoring. `http://www.gridlab.org/WorkPackages/wp-11/index.html`.

[GLO] Globus Project. `http://www.globus.org`.

[Glo86] F. Glover. Future path for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13, 1986.

[Glo89] F. Glover. Tabu search - part 1. *ORSA Journal of Computing*, 1, 1989.

[Glo90] F. Glover. Tabu search - part 2. *ORSA Journal of Computing*, 2, 1990.

[GLS94] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1994.

[GLU] GLUE schema. `http://www.hicb.org/glue/glue-schema/schema.htm`.

[GMR+98] S. D. Gribble, G. S. Manku, D. Roselli, E. A. Brewer, T. J. Gibson, and E. L. Miller. Self-similarity in file systems. In *Proceedings of SIGMETRICS '98*, 1998.

[GMS01] S. Greco, B. Matarazzo, and R. Slowinski. Rough sets theory for multicriteria decision analysis. *European Journal of Operational Research*, 129(1):1–47, 2001.

[GN02] X. Gu and K. Nahrstedt. Dynamic QoS-aware multimedia service configuration in ubiquitous computing environments. In *Proceedings of the IEEE Second International Conference on Distributed Computing Systems (ICDCS 2002)*, 2002.

[Gnu] Gnutella. `www.gnutellanews.com/information`.

[GNY+02] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based quality of service enabling language for the Web. *Journal of Visual Language and Computing, Special Issue on Multimedia Language for the Web*, 13(1), 2002.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[GP01] F. Giacomini and F. Prelz. Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description. Technical Report DataGrid-01-D1.4-0127-1_0, European DataGrid Project, 2001. Available from `http://server11.infn.it/workload-grid/docs/DataGrid-01-D1.4-0127-1_0.doc`.

[GPS+02] F. Giacomini, F. Prelz, M. Sgaravatto, I. Terekhov, G. Garzoglio, and T. Tannenbaum. Planning on the Grid: A status report. Technical Report PPDG-20, Particle Physics Data Grid Collaboration, October 2002.

[GPS04] Carole Goble, Steve Pettifer, and Robert Stevens. Knowledge integration: In silico experiments in bioinformatics. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[GR94] J. Gray and A. Reuter. *Transaction Processing: Techniques and Concepts*. Morgan Kaufmann, 1994.

[GR96] J. Gehring and A. Reinefeld. Mars: A framework for minimizing the job execution time in a metacomputing environment. *Future Generation Computer Systems*, 12(1):87–99, 1996.

[GRAa] GGF Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG). `http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html`.

[Grab] GrADS runtime support for Grid applications. `http://hipersoft.cs.rice.edu/grads/runtime_description.htm`.

[GRAc] Globus Resource Allocation Manager (GRAM). `http://www.globus.org/gram/`.

[GRBK98] E. Gabriel, M. Resch, T. Beisel, and R. Keller. Distributed computing in a heterogenous computing environment. In *Proceedings of EuroPVM/MPI'98*, 1998.

[GRIa] Gridbus. `http://www.gridbus.org/`.

[GRIb] GriPhyN: The Grid Physics Network. `http://www.griphyn.org`.

[GS95] J. Gwertzman and M. Seltzer. The case for geographical push caching. In *Proceedings of the Fifth IEEE Workshop on Hot Topics Operating Systems (HotOS'95)*, 1995.

[GS99] Roch Guérin and Henning Schulzrinne. Network quality of service. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 479–503. Morgan Kaufmann, 1999.

[GT92] R. A. Golding and K. Taylor. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, Jack Baskin School of Engineering, University of California, Santa Cruz, 1992.

[GTJ+02] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, July 2002.

[Gue99] D. Guerrero. Caching the Web, part 1. *Linux Journal*, 57, January 1999.

[GW97] A. S. Grimshaw and W. A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[GWvB+01] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. D. Joseph, R. H. Katz, Z. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust Internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing*, 35(4):473–497, 2001.

[GWWL94] A. Grimshaw, J. Weissman, E. West, and E. Lyot. Metasystems: An approach combining parallel processing and heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 21(3):257–270, 1994.

[GY93] A. Ghafoor and J. Yang. A distributed heterogeneous supercomputing management system. *IEEE Computer*, 26(6):78–86, June 1993.

[Har98] S. Hartmann. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45, 1998.

[Haw] Hawkeye: A monitoring and management tool for distributed systems. `http://www.cs.wisc.edu/condor/hawkeye`.

[HB99] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Proceedings of ASA-IMS Conference on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, 1999.

[HBD96] M. Harchol-Balter and A. Downey. Exploiting process lifetime distributions for dynamic load balancing. In *Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, 1996.

[HBM02] Phillip Hallam-Baker and Eve Maler. Assertions and protocol for the OASIS Security Assertion Markup Language (SAML). Technical Report cs-sstc-core-01, OASIS, May 2002. Available from `http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf`.

[Hen95] Robert L. Henderson. Job scheduling under the Portable Batch System. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the First International JSSPP Workshop; LNCS #949)*, pages 178–186. Springer-Verlag, 1995.

[HFB+99] J. Heinanen, T. Finland, F. Baker, W. Weiss, and J. Wroclawski. Assured forwarding PHB group. Technical Report RFC 2597, Internet Engineering Task Force (IETF), 1999.

[HFPS02] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. Technical Report RFC 3280, Internet Engineering Task Force (IETF), 2002.

[HJM92] D. Heath, R. Jarrow, and A. Morton. Bond pricing and the term structure of interest rates: A new methodology for contingent claim valuation. *Econometrica*, 60, 1992.

[HKL+00] M. Hadida, Y. Kadobayashi, S. Lamont, H.W. Braun, B. Fink, T. Hutton, A. Kamrath, H. Mori, and M.H. Ellisman. Ad-

vanced networking for telemicroscopy. In *Proceedings of the Tenth Annual Internet Society Conference*, 2000.

[HLM94] D. Hitz, J. Lau, and M. Malcolm. File system design for an NFS file server appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 235–246, 1994.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Hol01] K. Holtman. CMS requirements for the Grid. In *Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP 2001)*, 2001.

[Hor01] P. Horn. The IBM vision for autonomic computing. Technical report, IBM, 2001. Available from `www.research.ibm.com/autonomic/manifesto`.

[Hot96] S. Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Second International JSSPP Workshop; LNCS #1162)*. Springer-Verlag, 1996.

[How] Anthony C. Howe. Bandwidth and request throttling for Apache 1.3. `http://www.snert.com/Software/Throttle`.

[HPS] HPSS: High Performance Storage System. `http://www.sdsc.edu/hpss`.

[HSSY00] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for Grid computing. In *Proceedings of the Seventh International Conference of High Performance Computing*, 2000.

[Hul03] J. Hull. *Options, Futures, & Other Derivatives*. Prentice Hall, Fifth edition, 2003.

[HvBD98] A. Hafid, G. von Bochmann, and R. Dssouli. A quality of service negotiation approach with future reservations (NAFUR): a detailed study. *Computer Networks and ISDN Systems*, 30(8):777–794, 1998.

[IBM01] IBM. Using and administering LoadLeveler for AIX 5L. Technical Report IBM Document #SA22-7881-00, IBM,

2001. Available from http://publibfp.boulder.ibm.com/epubs/pdf/a2278810.pdf.

[ID01] Sitaram Iyer and Peter Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 2001.

[IEE94] IEEE. *IEEE Standard for Information Technology, POSIX 1003.2D*. IEEE, 1994.

[IG02] IBM and Globus. IBM and Globus announce Open Grid Services for commercial computing, 2002. Available from http://www.ibm.com/news/be/en/2002/02/211.html.

[IM98] H. Ishibushi and T. Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 28(3), 1998.

[IM02] IBM and Microsoft. Security in a Web services world: A proposed architecture and roadmap: A joint white paper from IBM Corporation and Microsoft Corporation, version 1.0. Technical report, MSDN, April 2002. Available from http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/%dnwssecur/html/securitywhitepaper.asp.

[IMT96] H. Ishibushi, T. Murata, and H. Tanaka. Multi-objective genetic algorithm and its application to flowshop scheduling. *Computer and Industrial Engineering*, 30(4), 1996.

[IOP99] M. Iverson, F. Ozguner, and L. Potter. Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. In *Proceedings of the Heterogeneous Computing Workshop*, 1999.

[IR03] A. Iamnitchi and M. Ripeanu. Myth and reality: Usage patterns in a large data-intensive physics project. Technical Report TR2003-4, GriPhyN, 2003.

[IRF02] A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[ISP] GGF Information Systems and Performance Area (ISP). `http://www.ggf.org/1_GIS/GIS.htm`.

[iVD] iVDGL: International Virtual-Data Grid Laboratory. `http://www.ivdgl.org`.

[Jaca] Scott M. Jackson. Allocation management with Qbank. `http://www.emsl.pnl.gov:2080/docs/mscf/Allocation_Management_with_QBank%.html`.

[Jacb] Scott M. Jackson. Gold allocation manager specification document. `http://www.csm.ornl.gov/~geist/cgi-bin/enote.cgi?nb=rmwg\&action=view\&page=-3`.

[Jac02] K. Jackson. pyGlobus: a Python interface to the Globus toolkit. *Concurrency and Computation: Practice and Experience*, 14(13-15):1075–1084, 2002.

[Jac03a] David B. Jackson. Maui administrators guide. `http://supercluster.org/maui/docs/mauiadmin.html`, 2003.

[Jac03b] David B. Jackson. Silver administrators guide. `http://supercluster.org/silver/docs/silveradmin.html`, 2003.

[Jas98] A. Jaszkiewicz. Genetic local search for multiple objective combinatorial optimisation. Technical Report Technical Report RA014 /98, Institute of Computing Science, Poznan University of Technology, 1998.

[JGN99] William E. Johnston, Dennis Gannon, and Bill Nitzberg. Grids as production computing environments: The engineering aspects of NASA's Information Power Grid. In *Proceedings of the Eighth IEEE International Symposium on High-Performance Distributed Computing (HPDC-8)*, 1999.

[JMR+01] J. Józefowska, M. Mika, R. Różycki, G. Waligóra, and J. Węglarz. Simulated annealing for multi-mode resource-constrained project scheduling problem. *Annals of Operations Research*, 102:137–155, 2001.

[Joh99] William Johnston. Realtime widely distributed instrumentation systems. In Ian Foster and Carl Kesselman, editors, *The*

Grid: Blueprint for a New Computing Infrastructure, chapter 4, pages 75–103. Morgan Kaufmann, 1999.

[Jon96] James Patton Jones. The NASA Metacenter. In *Proceedings of the NASA High Performance Computing and Communications Program / Computational Aerosciences Workshop (HPCCP/CAS)*, August 1996.

[Jon97a] James Patton Jones. Implementation of the NASA Metacenter: Phase 1 report. Technical Report NAS-97-027, NASA Ames Research Center, October 1997.

[Jon97b] James Patton Jones. PBS technology transfer to Department of Defense sites. Technical Report NASA Ames Quarterly Report, NASA Ames Research Center, October 1997.

[Jon98] James Patton Jones. Designing a metacenter: Recommendations to DoD MSRC ASC and CEWES. Technical Report Technology Transfer Whitepaper, NASA Ames Research Center, March 1998.

[Jon03a] James Patton Jones, editor. *PBS Pro 5.3 Administrator Guide*. Altair Grid Technologies, 2003.

[Jon03b] James Patton Jones, editor. *PBS Pro 5.3 User Guide*. Altair Grid Technologies, 2003.

[JXT] JXTA. `http://www.jxta.org/`.

[Kar96] J. F. Karpovich. Support for object placement in wide area distributed systems. Technical Report CS-96-03, University of Virginia, 1996.

[KBC+00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 2000.

[KC00a] J. D. Knowles and D. W. Corne. A comparison of diverse approaches to memetic multiobjective combinatorial optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Workshop On Memetic Algorithms*, July 2000.

[KC00b] J. D. Knowles and D. W. Corne. M-PAES: A memetic algorithm for multiobjective optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, 2000.

[KC01] C. Kenyon and G. Cheliotis. Stochastic models for telecom commodity prices. *Computer Networks*, 36(5-6), 2001.

[KC02a] C. Kenyon and G. Cheliotis. Architecture requirements for commercializing Grid resources. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[KC02b] C. Kenyon and G. Cheliotis. Forward price dynamics and option prices for network commodities. In *Proceedings of the Bachelier Finance Society, Second World Congress*, 2002.

[KC03] C. Kenyon and G. Cheliotis. Creating services with hard guarantees from cycle-harvesting systems. In *Proceedings of the Third IEEE Symposium on Cluster Computing and the Grid (CCGrid'03)*, pages 224–231, 2003.

[KCWB02] D. Kondo, H. Casanova, E. Wing, and F. Berman. Models and scheduling mechanisms for global computing applications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.

[KDB02] S. Kumar, S. K. Das, and R. Biswas. Graph partitioning for parallel applications in heterogeneous Grid environments. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[Kel63] J. E. Kelley. The critical path method: Resource planning and scheduling. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 347–365. Prentice-Hall, 1963.

[KFB99] N. Kapadia, J. Fortes, and C. Brodley. Predictive application performance modeling in a computational Grid environment. In *Proceedings of the Eighth IEEE International Symposium on High-Performance Distributed Computing (HPDC-8)*, 1999.

[KGJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[KHDC01] O. Kornievskaia, P. Honeyman, B. Doster, and K. Coffman. Kerberized credential translation: A solution to Web access

control. Technical Report 01-5, Center for Information Technology Integration, University of Michigan, 2001. Also available from `http://downloads.securityfocus.com/library/citi-tr-01-5.pdf`.

[Kin92] B. A. Kingsbury. The Network Queueing System (NQS). Technical report, Sterling Software, 1992.

[KKL+02] A. Keller, G. Kar, H. Ludwig, A. Dan, and J. L. Hellerstein. Managing dynamic services: A contract based approach to a conceptual architecture. In *Proceedings of 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, April 2002.

[KL01] J. Keppo and J. Lassia. Pricing options. *Telecoms Capacity*, 1(4), 2001.

[Kle86] S. Kleiman. Vnodes: An architecture for multiple file system types in Sun UNIX. In *Proceedings of USENIX*, pages 151–163, 1986.

[Kle99] P. Klemperer. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3), 1999.

[Kle00] J. Kleinberg. The small-worlds phenomenon: an algorithmic perspective. In *Proceedings of the Thirty-Second ACM Symposium on Theory of Computing*, 2000.

[KM03] K. Keahey and K. Motawi. Taming of the Grid: Virtual application services. Technical report, Argonne National Laboratory, Mathematics and Computer Science Division Technical Memorandum ANL/MCS-TM-262, 2003.

[KMC+00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[KMMC+02] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, R. Aydt, D. Reed, D. Gannon, J. Dongarra, S. Vadhiyar, L. Johnsson, C. Kesselman, and R. Wolski. Toward a framework for preparing and executing adaptive Grid programs. In *Proceedings of NSF Next Generation Systems Program Workshop, International Parallel and Distributed Processing Symposium*, 2002.

[KNP00] K. Kurowski, J. Nabrzyski, and J. Pukacki. Predicting job execution times in the Grid. In *Proceedings of the SGI Users Conference*, October 2000.

[KNP01] K. Kurowski, J. Nabrzyski, and J. Pukacki. User preference driven multiobjective resource management in Grid environments. In *Proceedings of the First IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, May 2001.

[Kol95] R. Kolisch. Project scheduling under resource constraints - efficient heuristics for several problem classes. *Physica*, 1995.

[KP00] S. Kutten and D. Peleg. Deterministic distributed resource discovery. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2000.

[KPF04] C. Kesselman, T. Prudhomme, and I. Foster. Distributed telepresence: The NEESgrid earthquake engineering collaboratory. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[Kri02] V. Krishna. *Auction Theory*. Academic Press, 2002.

[KSW02] P. Keyani, N. Sample, and G. Wiederhold. Scheduling under uncertainty: Planning for the ubiquitous Grid. In *Proceedings of the Fifth International Conference on Coordination Models and Languages (COORD'02)*, 2002.

[KSY99] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the design and evaluation of job scheduling systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fifth International JSSPP Workshop; LNCS #1659)*, pages 17–42. Springer-Verlag, 1999.

[KTF03] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing, to appear*, 2003.

[KTN+03] Gaj Kris, El-Ghazawi Tarek, Alexandridis Nikitas, Vroman Frederic, Jacek R. Radzikowski, Preeyapong Samipagdi, and

Suboh A. Suboh. An empirical comparative study of job management systems. *Concurrency: Practice and Experience, to appear*, 2003.

[KW02] K. Keahey and V. Welch. Fine-grain authorization for resource management in the Grid environment. In *Proceedings of the Third International Workshop on Grid Computing (Grid2002)*, 2002.

[LAP] Liberty Alliance Project. `http://www.projectliberty.org/`.

[LB98] O. Lee and S. Benford. An explorative approach to federated trading. *Computer Communications*, 21(2), 1998.

[LB01] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems*, 2001.

[LBS+98] J. P. Loyall, D. E. Bakken, R. E. Schantz, J. A. Zinky, D. A. Karr, R. Vanegas, and K. R. Anderson. QoS aspect languages and their runtime integration. In *Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR'98) (LNCS #1511)*. Springer-Verlag, 1998.

[LCC+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the Sixth Annual ACM International Conference on Supercomputing (ICS)*, 2002.

[LDM+01] K. London, J. Dongarra, S. Moore, P. Mucci, K. Seymour, and T. Spencer. End-user tools for application performance analysis using hardware counters. In *Proceedings of the International Conference on Parallel and Distributed Computing Systems*, August 2001.

[LF03] C. Liu and I. Foster. A constraint language approach to Grid resource selection. Technical Report TR-2003-07, Computer Science Department, The University of Chicago, 2003.

[LFH+03] M. J. Lewis, A. J. Ferrari, M. A. Humphrey, J. F. Karpovich, M. M. Morgan, A. Natrajan, A. Nguyen-Tuong, G. S. Wasson, and A. S. Grimshaw. Support for extensibility and site autonomy in the Legion Grid system object model. *Journal of Parallel and Distributed Computing, to appear*, 2003.

[Lif96] D. A. Lifka. The ANL/IBM SP scheduling system. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the First International JSSPP Workshop; LNCS #949)*. Springer-Verlag, 1996.

[LIG] LIGO: The Laser Interferometer Gravitational-wave Observatory. `http://www.ligo.caltech.edu`.

[LIT92] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Proceedings of AAAI-92*, 1992.

[LJD+99] Jason Leigh, Andrew E. Johnson, Thomas A. DeFanti, Maxine Brown, Mohammed Dastagir Ali, Stuart Bailey, Andy Banerjee, Pat Banerjee, Jim Chen, Kevin Curry, Jim Curtis, Fred Dech, Brian Dodds, Ian Foster, Sarah Fraser, Kartik Ganeshan, Dennis Glen, Robert Grossman, Randy Heiland, John Hicks, Alan D. Hudson, Tomoko Imai, Mohammed Ali Khan, Abhinav Kapoor, Robert V. Kenyon, John Kelso, Ron Kriz, Cathy Lascara, Xiaoyan Liu, Yalu Lin, Theodore Mason, Alan Millman, Kukimoto Nobuyuki, Kyoung Park, Bill Parod, Paul J. Rajlich, Mary Rasmussen, Maggie Rawlings, Daniel H. Robertson, Samroeng Thongrong, Robert J. Stein, Kent Swartz1, Steve Tuecke, Harlan Wallach, Hong Yee Wong, and Glen H. Wheless. A review of tele-immersive collaboration in the CAVE research network. In *Proceedings of IEEE VR99*, 1999.

[LL90] M. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, 1990.

[LLM88] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, 1988.

[LM86] Richard J. Larsen and Morris L. Marx. *An Introduction to Mathematical Statistics and Its Applications*. Prentice-Hall, 1986.

[LMC03] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: The SimGrid simulation framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, 2003.

[LMN94] C. Lai, G. Medvinsky, and B. C. Neuman. Endorsements, licensing, and insurance for distributed system services. In *Proceedings of the Second ACM Conference on Computer and Communication Security*, 1994.

[LN00] B. Li and K. Nahrstedt. QualProbes: Middleware QoS profiling services for configuring adaptive applications. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000)*, 2000.

[Loh02] Steve Lohr. Supercomputing and business move closer. *New York Times Business/Financial Desk*, February 19, 2002.

[LP] lp_solve. `ftp.es.ele.tue.nl/pub/lp_solve`.

[LR01] Larry Lancaster and Alan Rowe. Measuring real world data availability. In *Proceedings of the LISA 2001 Fifteenth Systems Administration Conference*, pages 93–100, 2001.

[LRM96] C. Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic QoS in Real-Time Mach. In *Proceedings of Multimedia Japan*, 1996.

[LS92] Michael Litzkow and Marvin Solomon. Supporting checkpointing and process migration outside the UNIX kernel. In *Proceedings of USENIX*, January 1992.

[LSF] Platform computing technical documentation for Platform LSF. `http://www.platform.com/services/support/docs/LSFDoc51.asp`.

[Lum01] I. Lumb. Linux clustering for high-performance computing. In *Proceedings of USENIX*, August 2001. Also available from `http://www.usenix.org/publications/login/2001-08/pdfs/lumb.pdf`.

[LW04] David Levine and Mark Wirt. Interactivity with scalability: Infrastructure for multiplayer games. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[LWW03] David Levine, Mark Wirt, and Barry Whitebook. *Practical Grid Computing for Massively Multiplayer Games*. Charles River Media, 2003.

[LYFA02] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for Grid applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[MAR] DTI e-Science Core Technology Programme Computational Markets project. `http://www.lesc.ic.ac.uk/markets/`.

[Mau] Maui scheduler. `http://www.supercluster.org/maui`.

[MBHJ98] D. Marinescu, L. Boloni, R. Hao, and K. Jun. An alternative model for scheduling on a computational Grid. In *Proceedings of the Thirteenth International Symposium on Computer and Information Sciences (ISCIS'98)*, pages 473–480, 1998.

[MDS] Globus Monitoring and Discovery System (MDS2). `http://www.globus.org/mds`.

[Mes99] Paul Messina. Distributed supercomputing applications. In *The Grid: Blueprint for a New Computing Infrastructure*, pages 55–73. Morgan Kaufmann, 1999.

[MFGH88] T. W. Malone, R. E. Fikes, K. R. Grant, and M. T. Howard. Enterprise: A market-like task scheduler for distributed computing environments. In B. Huberman, editor, *The Ecology of Computation: Volume 2 of Studies in Computer Science and Artificial Intelligence*, pages 177–255. Elsevier Science Publishers, 1988.

[Mic92] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

[MIS96] A. Mehra, A. Indiresan, and K. Shin. Structuring communication software for quality-of-service guarantees. In *Proceedings of Seventeenth Real-Time Systems Symposium*, December 1996.

[MLH95] Shikharesh Majumdar, Johannes Lüthi, and Günter Haring. Histogram-based performance analysis for computer systems with variabilities or uncertainties in workload. Technical Report SCE-95-22, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, November 1995.

[MMCS+01] M. Mazina, J. Mellor-Crummey, O. Sievert, H. Dail, and G. Obertelli. GrADSoft: A program-level approach to using the Grid. Technical Report GrADS Working Document 3, GrADS, March 2001. Available from http://hipersoft.cs.rice.edu/grads/publications_reports.htm.

[MNO+96] C. Martin, P. S. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini multimedia storage server. In S. M. Chung, editor, *Multimedia Information Storage and Management*. Kluwer Academic Publishers, 1996.

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system, section e.2.1. Technical Report Project Athena Technical Plan, MIT, 1987.

[Moc87] P. Mockapetris. Domain names–concepts and facilities. Technical Report RFC 1034, Internet Engineering Task Force (IETF), 1987.

[MOJ] MojoNation. http://www.mojonation.net.

[MOP] MOP: A system for monte carlo distributed production. http://www.ppdg.net/pa/ppdg-pa/mop/mop.pdf.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1953.

[MSH+01] K. Myers, S. Smith, D. Hildum, P. Jarvis, and R. de Lacaze. Integrating planning and scheduling through adaptation of resource intensity estimates. In *Proceedings of the Sixth European Conference on Planning (ECP-01)*, 2001.

[Nab99] Jarek Nabrzyski. Knowledge-based scheduling method for Globus. In *Proceedings of the 1999 Globus Retreat*, 1999. Also available from http://www.man.poznan.pl/metacomputing/ai-meta/globusnew/index.htm.

[Nab00] J. Nabrzyski. *User Preference Driven Expert System for Solving Multiobjective Project Scheduling Problems*. PhD thesis, Poznan University of Technology, 2000.

[Nap] Napster. `http://www.napster.com`.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers. Technical Report RFC 2474, Internet Engineering Task Force (IETF), 1998.

[NC01] K. Nichols and B. Carpenter. Definition of differentiated services per-domain behaviors and rules for their specification. Technical Report RFC 3086, Internet Engineering Task Force (IETF), 2001.

[NCN98] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications. *Journal on High-Speed Networking, Special Issue on Multimedia Networking*, 8(3-4):227–255, December 1998.

[NCWD+01] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. A. Humphrey, A. D. Fox, A. S. Grimshaw, and C. L. Brooks III. Studying protein folding on the Grid: Experiences using CHARMM on NPACI resources under Legion. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[Neg94] M. D. Negra. CMS collaboration. Technical Report LHCC 94-38, CERN, 1994.

[NeS] NeST storage appliance. `http://www.cs.wisc.edu/condor/nest`.

[NHG02] A. Natrajan, M. A. Humphrey, and A. S. Grimshaw. The Legion support for advanced parameter-space studies on a Grid. *Future Generation Computer Systems*, 18(8):1033–1052, October 2002.

[NI01] K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In P. Hansen and C. Ribeiro, editors, *Essays and Surveys in Metaheuristics*. Kluwer Academic Publishers, 2001.

[Nov02] J. Novotny. The Grid Portal Development Kit. *Concurrency and Computation: Practice and Experience*, 14(13-15):1145–1160, 2002.

[NRW03] J. Novotny, M. Russell, and O. Wehrens. GridSphere: A portal framework for building collaborations. In *Proceedings of The*

First International Workshop on Middleware for Grid Computing, 2003.

[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12), December 1978.

[NS96] K. Nahrstedt and J. M. Smith. Design, implementation and experiences of the OMEGA end-point architecture. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Distributed Multimedia Systems and Technology*, 14(7):1263–1279, September 1996.

[NSZ02] K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2002.

[OAS] OASIS. `http://www.oasis-open.org`.

[OGSa] GGF Open Grid Services Architecture Working Group (OGSA-WG). `http://www.ggf.org/ogsa-wg/`.

[OGSb] Open Grid Services Architecture Database Access and Integration (OGSA-DAI). `http://umbriel.dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI/`.

[OGSc] GGF Open Grid Service Interface Working Group (OGSI-WG). `http://www.gridforum.org/ogsi-wg/`.

[OK96] I. H. Osman and J. P. Kelly. *Metaheuristics: Theory and Applications*. Kluwer Academic Publishers, 1996.

[OOS02] Ekow J. Otoo, Frank Olken, and Arie Shoshani. Disk cache replacement algorithm for storage resource managers in data Grids. In *Proceedings of SuperComputing (SC'02)*, 2002.

[Ope02] OASIS Open. Oasis extensible access control markup language (XACML). Technical report, OASIS, December 2002. Available from `http://www.oasis-open.org/committees/download.php/1642/oasis`.

[Ora01] A. Oram, editor. *Peer-to-Peer. Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[PAD+02] Norman W Paton, Malcolm P Atkinson, Vijay Dialani, Dave Pearson, Tony Storey, and Paul Watson. Database access and integration services on the Grid. Tech-

nical report, U.K. National eScience Center, 2002. Available from `http://umbriel.dcs.gla.ac.uk/Nesc/general/technical_papers/dbtf.pdf`.

[Para] The Parabon distributed frontier distributed computing system. `http://www.parabon.com`.

[PARb] Parsec: Parallel simulation environment for complex systems. `http://pcl.cs.ucla.edu/projects/parsec`.

[PAS] Microsoft .net Passport. `http://www.passport.net/Consumer/`.

[Pat02] David A. Patterson. Availability and maintainability >> performance: New focus for a new century. Key Note Lecture at the First USENIX Conference on File and Storage Technologies (FAST '02), January 2002.

[Paw82] Z. Pawlak. Rough sets. *International Journal of Information & Computer Sciences*, 11, 1982.

[PBB+01] J. Plank, A. Bassi, M. Beck, T. Moore, M. Swany, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5), September/October 2001.

[PBD+01] A. Petitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar. Numerical libraries and the Grid. In *Proceedings of SuperComputing (SC'01)*, 2001.

[PBS] PBS: The Portable Batch System. `http://www.openpbs.org/`.

[PDZ99] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable Web server. In *Proceedings of the USENIX Technical Conference*, 1999.

[PF02] S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for building composite Web services. In *Proceedings of the Eleventh World Wide Web Conference*, 2002.

[PFS] The Pluggable File System. `http://www.cs.wisc.edu/condor/pfs`.

[PG] Platform Globus. `http://www.platform.com/products/globus/index.asp`.

[Pil98] D. Pilipović. *Energy Risk: Valuing and Managing Energy Derivatives*. McGraw-Hill, 1998.

[PKF+01] T. Prudhomme, C. Kesselman, T. Finholt, I. Foster, D. Parsons, D. Abrams, J.-P. Bardet, R. Pennington, J. Towns, R. Butler, J. Futrelle, N. Zaluzec, and J. Hardin. NEESgrid: A distributed virtual laboratory for advanced earthquake experimentation and simulation: Scoping study. Technical report, NEESgrid, Technical Report, 2001. Available from `http://www.neesgrid.org/documents/NEESgrid_TR.2001-01.pdf`.

[pki] IETF public-key infrastructure (X.509) (pkix) working group. `http://www.ietf.org/html.charters/pkix-charter.html`.

[PL88] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proceedings of the National Academy of Sciences of the United States of America*, 1988.

[PL95] J. Pruyne and M. Livny. Parallel processing on dynamic resources with CARMI. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the First International JSSPP Workshop; LNCS #949)*. Springer-Verlag, 1995.

[Pla] Platform Computing. The Platform ActiveCluster desktop computing system. `http://www.platform.com/products/wm/ActiveCluster/`.

[PLL+03] S. T. Peltier, A. W. Lin, D. Lee, S. Mock, S. Lamont, T. Molina, M. Wong, L. Dai, M. E. Martone, and M. H. Ellisman. The telescience portal for tomography applications. *Journal of Parallel and Distributed Computing*, 2003.

[PPD] PPDG: Particle Physics Data Grid. `http://www.ppdg.net`.

[PR85] J. Postel and J. Reynolds. File transfer protocol (FTP). Technical Report RFC 959, Internet Engineering Task Force (IETF), October 1985.

[PRR97] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[PRY99] Carmen Pancerella, Larry Rahn, and Christine Yang. The diesel combustion collaboratory: Combustion researchers

collaborating over the Internet. In *Proceedings of SuperComputing (SC'99)*, 1999.

[PU00] D. Parkes and L. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the 17th National Conference on Artificial Intelligence, (AAAI-00)*, 2000.

[PWF+02] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the IEEE Third International Workshop on Policies for Distributed Systems and Networks*, 2002.

[pyG] Python GMA. `http://sourceforge.net/projects/py-gma/`.

[Qba] Qbank: A CPU Allocations Bank. `http://www.emsl.pnl.gov:2080/capabs/mscf/?/capabs/mscf/software/listjob%mgt_qbank.html`.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, pages 81–106, 1986.

[RA99] M. Rabinovich and A. Aggarwal. RaDaR: A scalable architecture for a global Web hosting service. In *Proceedings of the Eighth International World Wide Web Conference*, 1999.

[RAD+02] M. Russell, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf, and G. von Laszewski. The astrophysics simulation collaboratory: A science portal enabling community software development. *Cluster Computing*, 5(3):297–304, 2002.

[Ram00] R. Raman. *Matchmaking Frameworks for Distributed Resource Management*. PhD thesis, University of Wisconsin-Madison, 2000.

[RD01] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of Middleware '01*, 2001.

[Reb96] R. Rebonato. *Interest Rate Option Models*. John Wiley & Sons, 1996.

[Ree93] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[RF01] K. Ranganathan and I. Foster. Identifying dynamic replication strategies for a high-performance Data Grid. In *Proceedings of the Second International Workshop on Grid Computing (Grid2001)*, 2001.

[RF02] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data intensive applications. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[RF03] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for DataGrids. *Journal of Grid Computing, to appear*, 2003.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM 2001*, 2001.

[RFI02] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *Internet Computing*, 6, 2002.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back propagating errors. *Nature*, 323:533–536, 1986.

[RIF01a] M. Ripeanu, A. Iamnitchi, and I. Foster. Cactus application: Performance predictions in a Grid environment. In *Proceedings of European Conference on Parallel Computing (EuroPar)*, 2001.

[RIF01b] M. Ripeanu, A. Iamnitchi, and I. Foster. Performance predictions for a numerical relativity package in Grid environments. *International Journal of High Performance Computing Applications*, 15(4):375–387, 2001.

[RIK] RIKEN Institute of Physical and Chemical Research, Computational Science Division. `http://atlas.riken.go.jp/en/index.html`.

[Riv92] R. Rivest. The MD5 message-digest algorithm. Technical Report RFC 1321, Internet Engineering Task Force (IETF), April 1992.

[RK03] B. Raman and R. H. Katz. An architecture for highly available wide-area service composition. *Computer Communications Journal, special issue on Recent Advances in Communication Networking*, May 2003.

[RLS98] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, 1998.

[RLS99] R. Raman, M. Livny, and M. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2), 1999.

[RLS03] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gang matching. In *Proceedings of the Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[RN98] O. Regev and N. Nisan. The Popcorn market online markets for computational resources. In *Proceedings of the First International Conference On Information and Computation Economies*, 1998.

[RN99] T. Ryutov and C. Neuman. Generic authorization and access control application program interface c-bindings. Technical report, Internet Engineering Task Force (IETF), 1999. Available from `http://www.globecom.net/ietf/draft/draft-ietf-cat-gaa-cbind-01.html`.

[Roy01] A Roy. *End-to-End Quality of Service for High-End Applications*. PhD thesis, The University of Chicago, 2001.

[RS96] V. J. Rayward-Smith, editor. *Modern Heuristics Search Methods*. John Wiley & Sons Ltd, 1996.

[RS02] A. Roy and V. Sander. Advance Reservation API. Technical Report GFD-E.5, Global Grid Forum (GGF), 2002.

[RTF+01] E. Rosen, D. Tappan, G. Fedorkow, Y. Rokhter, D. Farinacci, T. Li, and A. Conta. MPLS label stack encoding. Technical Report RFC 3032, Internet Engineering Task Force (IETF), 2001.

[RUS] GGF Resource Usage Service Working Group (RUS-WG). `http://www.gridforum.org/3_SRM/rus.htm`.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. Technical Report RFC 3031, Internet Engineering Task Force (IETF), 2001.

[RVSR98] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive control of distributed applications. In *Proceedings of the Seventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-7)*, 1998.

[SAFR01] V. Sander, W. Adamson, I. Foster, and A. Roy. End-to-end provision of policy information for network QoS. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.

[San99] Thomas W. Sandholm. Distributed rational decision making. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 201–258. MIT Press, 1999.

[San02] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135, 2002.

[San03] V. Sander. Design and evaluation of a bandwidth broker that provides network quality of service for Grid applications. Technical Report Volume 16 of NIC-series, ISBN 3-00-010002-4, John von Neumann Institute for Computing, 2003.

[SAWP95] I. Stoica, H. Abdel-Wahab, and A. Pothen. A microeconomic scheduler for parallel computers. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the First International JSSPP Workshop; LNCS #949)*. Springer-Verlag, 1995.

[SB98] J. Schopf and F. Berman. Performance prediction in production environments. In *Proceedings of Fourteenth International Parallel Processing Symposium and the Ninth Symposium on Parallel and Distributed Processing*, 1998.

[SB99] J. Schopf and F. Berman. Stochastic scheduling. In *Proceedings of SuperComputing (SC'99)*, 1999.

[SC03] O. Sievert and H. Casanova. A simple MPI process swapping architecture for iterative applications. *International Jour-*

nal of High Performance Computing Applications, to appear, 2003.

[Sch97] J. Schopf. Structural prediction models for high performance distributed applications. In *Proceedings of the 1997 Cluster Computing Conference*, 1997.

[Sch99] Jennifer M. Schopf. A practical methodology for defining histograms in predictions. In *Proceedings of ParCo '99*, 1999.

[SD03] GGF Scheduling Disctionary Working Group (SD-WG). `http://www.fz-juelich.de/zam/RD/coop/ggf/sd-wg.html`, 2003.

[SDK+94] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *Proceedings of Third International Conference on Parallel and Distributed Information Systems*, 1994.

[SET] SETI@home: The Search for Extraterrestrial Intelligence. `http://setiathome.berkeley.edu`.

[SF02] V. Sander and M. Fidler. A pragmatic approach for service provisioning based on a small set of per-hop behaviors. In *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2002.

[SFRW00] V. Sander, I. Foster, A. Roy, and L. Winkler. A differentiated services implementation for high-performance TCP flows. *The International Journal of Computer and Telecommunications Networking*, 34, 2000.

[SFT98] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459)*. Springer-Verlag, 1998.

[SFT02] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, May 2002.

[SG01] Alexander Szalay and Jim Gray. The world-wide telescope. *Science*, 293:2037–2040, 2001.

[SG04] A. Szalay and J. Gray. Scientific data federation: The world wide telescope. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[SGE] Sun Grid Engine. `http://wwws.sun.com/software/gridware`.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of SPIE Multimedia Computing and Networking 2002 (MMCN'02)*, 2002.

[SGM02] R. Slowinski, S. Greco, and B. Matarazzo. Axiomatization of utility, outranking and decision-rule preference models for multiple-criteria classification problems under partial incosistency with the dominance principle. *Control and Cybernetics*, 31(4), 2002.

[Sha99] Richard Sharpe. Just what is SMB? `samba.org/cifs/docs/what-is-smb.html`, 1999.

[Shi00] C. Shirky. What is P2P...and what isn't? `http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html`, 2000.

[SHK95] B. A. Shirazi, A. R. Husson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, 1995.

[SIF03] V. Sander, F. Imhoff, and M. Fidler. Path Allocation in Backbone Networks Project. `http://www.pab.rwth-aachen.de`, 2003.

[Sil] Silver scheduler. `http://www.supercluster.org/silver`.

[SK01] Y. Saito and C. Karamanolis. Autonomous and decentralized replication in the Pangaea planetary-scale file service. Technical report, HP, HPL-TR-2001-323, 2001.

[SKT+00] A. S. Szalay, P. Z. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner. Designing and mining multi-terabyte astronomy archives: the Sloan Digital Sky Survey. In *Proceedings of ACM SIGMOD*, pages 451–462, 2000.

[SL94] S. F. Smith and O. Lassila. Toward the development of mixed-initiative scheduling systems. In *Proceedings of the ARPA-Rome Laboratory Planning Initiative Workshop*, 1994.

[SLJ+00] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A scientific tool for modeling computational Grids. In *Proceedings of Super-Computing (SC'00)*, 2000.

[SM00] J. Schneider and A. Moore. A locally weighted learning tutorial using Vizier 1.0. Technical report, CMU-RI-TR-00-18, Robitics Institute, Carnegie Mellon University, 2000.

[Smi99] W. Smith. *Resource Management in Metacomputing Environments*. PhD thesis, Northwestern University, 1999.

[Smi01] W. Smith. A framework for control and observation in distributed environments. Technical Report NAS-01-006, NAS NASA Ames, June 2001.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[SMZ03] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *Proceeding of INFOCOM*, 2003.

[SN02] Jennifer M. Schopf and Bill Nitzberg. Grids: The top ten questions. *Scientific Programming, Special Issue on Grid Computing*, 10(2):103–111, August 2002.

[SOHL+98] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1998.

[SP98] O. Schelen and S. Pink. Resource sharing in advance reservation agents. *Special issue on Multimedia Networking*, 7(3-4), 1998. Also available from `http://www.cdt.luth.se/~olov/publications/JHSN-98.pdf`.

[SPG97] S. Schenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. Technical Report RFC 2212, Internet Engineering Task Force (IETF), 1997.

[SRM] Global Grid Forum Scheduling and Resource Management Area (SRM). `http://www.mcs.anl.gov/~jms/ggf-sched`.

[SSA+02] H. Stockinger, A. Samar, B. Allcock, I. Foster, K. Holtman, and B. Tierney. File and object replication in Data Grids. *Journal of Cluster Computing*, 5(3):305–314, 2002.

[SSG02] A. Shoshani, A. Sim, and J. Gu. Storage resource managers: Middleware components for Grid storage. In *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems (MSS '02)*, 2002.

[SSS] DOE Scalable Systems Software for terascale computer centers. `http://www.scidac.org/ScalableSystems`.

[Sta01] Federal Information Processing Standards. Advanced encryption standard (AES). Technical Report Publication 197, National Institute of Standards and Technology, November 2001. Available from `http://csrc.nist.gov/CryptoToolkit/aes/`.

[Ste04] Rick Stevens. Group-oriented collaboration: The Access Grid collaboration system. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[Sun] Sun. Sun ONE Grid Engine 5.3 administration and user's guide. `http://www.sun.com/gridware`.

[Sun90] V. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice & Experience*, 2(4):315–339, 1990.

[Sut68] I. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6), June 1968.

[SW98] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceedings of ACM International Conference on Supercomputing (ICS)*, July 1998.

[SW02] W. Smith and P. Wong. Resource selection using execution and queue wait time predictions. Technical report, NAS02-003, NASA Ames Research Center, 2002.

[SWDC97] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett. From the I-WAY to the national technology Grid. *Communications of the ACM*, 40(11):50–61, 1997.

[SY98] U. Schwiegelshohn and R. Yahyapour. Analysis of first-come first-served parallel job scheduling. In *Proceedings of the Ninth SIAM Symposium on Discrete Algorithms*, 1998.

[SY99] U. Schwiegelshohn and R. Yahyapour. Resource allocation and scheduling in metasystems. In *Proceedings of the Distributed Computing and Metacomputing Workshop at High Performance Computing and Networking Europe (HPCN Europe '99)*, 1999.

[SY01] U. Schwiegelshohn and R. Yahyapour. Attributes for communication between scheduling instances, scheduling attributes working group. Technical Report GFD-I.6, Global Grid Forum (GGF), December 2001. Also available from `http://ds.e-technik.uni-dortmund.de/~yahya/ggf-sched/WG/sa-wg.html`.

[TAG+03] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A Grid Monitoring Architecture. Technical Report GFD-I.7, Global Grid Forum (GGF), 2003.

[Tay02] John Taylor. Plenary keynote, GGF 5. `http://www.gridforum.org/Meetings/ggf5/plenary/Mon/`, July 2002.

[TB96] P. Tucker and F. Berman. On market mechanisms as a software technique. Technical Report CS96-513, Computer Science and Engineering Department, University of California, San Diego, December 1996.

[TBAD+01] D. Thain, J. Bent, A. Arpaci-Dusseau, R. Arpaci-Dusseau, and M. Livny. Gathering at the well: Creating communities for Grid I/O. In *Proceedings of SuperComputing (SC'01)*, November 2001.

[TBSL01] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The Kangaroo approach to data movement on the Grid. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, August 2001.

[TCF+03] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure (OGSI) 1.0 draft. Technical report, Global Grid Forum (GGF), March 2003. Available from `http://www.gridforum.org/ogsi-wg/`.

[TEF+02] S. Tuecke, D. Engert, I. Foster, M. Thompson, L. Pearlman, and C. Kesselman. Internet X.509 public key infrastructure proxy certificate profile. Technical Report draft-ietf-pkix-proxy-01.txt, Internet Engineering Task Force, 2002. Available from `http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-06.txt`.

[TL02] D. Thain and M. Livny. Error scope on a computational Grid: Theory and practice. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing*, 2002.

[TMB00] M. P. Thomas, S. Mock, and J. Boisseau. Development of Web toolkits for computational science portals: The NPACI HotPage. In *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, 2000.

[TMB+01] M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and S. Sutton. The GridPort toolkit architecture for building Grid portals. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.

[TMEC02] M. Thompson, S. Mudumbai, A. Essiari, and W. Chin. Authorization policy in a PKI environment. In *Proceedings of the First Annual NIST workshop on PKI*, 2002.

[TNS+02] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC based programming middleware for Grid computing. *Journal of Grid Computing*, 2002.

[Tri] Triana. `http://www.trianacode.org/`.

[TSC00] H. Turgeon, Q. Snell, and M. Clement. Application placement using performance surface. In *Proceedings of the Ninth IEEE International Symposium on High-Performance Distributed Computing (HPDC-9)*, 2000.

[TSP02] I. Taylor, M. Shields, , and R. Philip. GridOneD: Peer to peer visualization using Triana: A galaxy formation test case. In *Proceedings of the UK eScience All Hands Meeting*, 2002.

[Tur] TurboLinux. The Enfuzion system. `http://www.turbolinux.com/`.

[TWG+01] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan, Mark Hereld, Ivan Judson, and Rick Stevens. Prophesy: Automating the modeling process. In *Proceedings of the Third International Workshop on Active Middleware Services*, 2001.

[UNA] National Autonomous University of Mexico, The Department of Gravitation and Field Theory. `http://www.nuclecu.unam.mx/~gravit/`.

[UNIa] Unicore. `http://www.unicore.org/`.

[Unib] United Devices. `http://www.ud.com`.

[UR] GGF Usage Record Working Group (UR-WG). `http://www.gridforum.org/3_SRM/ur.htm`.

[USC] US Compact Muon Solenoid (US-CMS) collaboration. `http://www.uscms.org`.

[UT] University of Texas at Austin, The Center for Relativity. `http://wwwrel.ph.utexas.edu/`.

[VAMR01] F. Vraalsen, R. A. Aydt, C. L. Mendes, and D. A. Reed. Performance contracts: Predicting and monitoring Grid application behavior. In *Proceedings of the Second International Workshop on Grid Computing (Grid2001)*, 2001.

[VD02] S. Vadhiyar and J. Dongarra. A metascheduler for the Grid. In *Proceedings of the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[VD03a] S. Vadhiyar and J. Dongarra. SRS - a framework for developing malleable and migratable parallel applications for distributed systems. *Parallel Processing Letters, to appear*, 2003.

[VD03b] S. Vadhiyar and Jack J. Dongarra. A performance oriented migration framework for the Grid. In *Proceedings of the Third*

IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), 2003.

[vLA87] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Reidel, 1987.

[VNRS02] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. Technical report, Sun Microsystems, 2002.

[VOM] VOMS: Virtual Organization Management System. `http://grid-auth.infn.it/docs/VOMS-v1_1.pdf`.

[VS02] S. Vazhkudai and J. M. Schopf. Predicting sporadic Grid data transfers. In *Proceedings of the Eleventh IEEE Symposium on High-Performance Distributed Computing (HPDC-11)*, 2002.

[VS03] S. Vazhkudai and J. Schopf. Using regression techniques to predict large data transfers. *Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Application, to appear,* 2003.

[vSHT99] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A wide-area distributed system. *IEEE Concurrency*, 7(1):70–78, January 1999.

[W3C] W3C architecture domain. `http://www.w3.org/2002/ws/`.

[WASB95] R. Wolski, C. Anglano, J. Schopf, and F. Berman. Developing heterogeneous applications using Zoom and HeNCE. In *Proceedings of the Heterogeneous Computing Workshop*, April 1995.

[Wat99] D. J. Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness*. Princeton University Press, 1999.

[WCB01] Matt Welsh, David Culler, and Eric Brewer. SEDA: An architecture for well-conditioned, scalable Internet services. In *Proceedings of the Eighteenth Symposium on Operating Systems Principles (SOSP-18)*, October 2001.

[Węg99] J. Węglarz, editor. *Project Scheduling - Recent Models, Algorithms and Applications*. Kluwer Academic Publishers, 1999.

[Wei95] J. Weissman. *Scheduling Parallel Computations in a Heterogeneous Environment*. PhD thesis, University of Virginia, August 1995.

[WFP+96] F. Wang, H. Franke, M. Papaefthymiou, P. Pattnaik, L. Rudolph, and M.S. Squillante. A gang scheduling design for multiprogrammed parallel computing environments. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Second International JSSPP Workshop; LNCS #1162)*. Springer-Verlag, 1996.

[WG98] D. Wischik and A. Greenberg. Admission control for booking ahead shared resources. In *Proceedings of IEEE INFOCOM'98*, 1998.

[WGN02] D. Wichadakul, X. Gu, and K. Nahrstedt. A programming framework for quality-aware ubiquitous multimedia applications. In *Proceedings of ACM Multimedia 2002*, 2002.

[WHH+92] C. A. Waldspurger, T. Hogg, B. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, 1992.

[WKN+92] M. C. Wang, S. D. Kim, M. A. Nichols, R. F. Freund, H. J. Seigel, and W. G. Nation. Augmenting the optimal selection theory for superconcurrency. In *Proceedings of the Heterogeneous Computing Workshop*, pages 13–22, 1992.

[WLS+85] D. Walsh, B. Lyon, G. Sager, J. M. Chang, D. Goldberg, S. Kleiman, T. Lyon, R. Sandberg, and P. Weiss. Overview of the Sun network file system. In *Proceedings of the USENIX Winter Conference*, pages 117–124, 1985.

[WM97] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6, 1997.

[WNGX01] D. Wichadakul, K. Nahrstedt, X. Gu, and D. Xu. 2KQ+: An integrated approach of QoS compilation and component-based, runtime middleware for the unified QoS management framework. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[WO02] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[Wol97] R. Wolski. Forecasting network performance to support dynamic scheduling using the Network Weather Service. In *Proceedings of the Sixth IEEE International Symposium on High-Performance Distributed Computing (HPDC-6)*, 1997.

[Wol98] R. Wolski. Dynamically forecasting network performance using the Network Weather Service. *Journal of Cluster Computing*, 1:119–132, January 1998.

[Wor] WorldPay UK. `http://www.worldpay.co.uk/`.

[Wro97] J. Wroclawski. The use of RSVP with IETF integrated services. Technical Report RFC 2210, Internet Engineering Task Force (IETF), 1997.

[WS97] L. C. Wolf and R. Steinmetz. Concepts for reservation in advance. *Kluwer Journal on Multimedia Tools and Applications*, 4(3), May 1997.

[WSF+03] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Cajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. GSI: Security for Grid services. In *Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*, 2003.

[WSH99a] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[WSH99b] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared Unix systems. In *Proceedings of the Eighth IEEE International Symposium on High-Performance Distributed Computing (HPDC-8)*, 1999.

[WW95] Carl A. Waldspurger and William E. Weihl. Stride scheduling: Deterministic proportional-share resource mangement. Technical Report MIT/LCS/TM-528, Massachusetts Institute of Technology, June 1995.

[WWW01] P. Wurman, M. Wellman, and W. Walsh. A parameterization of the auction design space. *Games and Economic Behavior*, 35, 2001.

[WWWMM98] W. Walsh, M. Wellman, P. Wurman, and J. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proceedings of the Eighteenth International Conference on Distributed Computing Systems*, 1998.

[WZ98] J.B. Weissman and X. Zhao. Scheduling parallel applications in distributed networks. *Journal of Cluster Computing*, 1:109–118, 1998.

[XHLL04] Ming Xu, Zhenhua Hu, Weihong Long, and Wayne Liu. Service virtualization: Infrastructure and applications. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufmann, 2004.

[XML] W3 Consoritium XML: eXtensible Markup Language. `http://www.w3c.org/XML`.

[XN02] D. Xu and K. Nahrstedt. Finding service paths in a media service proxy network. In *Proceedings of SPIE/ACM Multimedia Computing and Networking Conference (MMCN'02)*, 2002.

[Xu01] M. Xu. Effective metacomputing using LSF MultiCluster. In *Proceedings of the First IEEE International Symposium of Cluster Computing and the Grid (CCGrid'01)*, 2001.

[Yan03] Lingyun Yang. Load traces. `http://cs.uchicago.edu/~lyang/Load`, 2003.

[YD02] Asim YarKhan and Jack J. Dongarra. Experiments with scheduling using simulated annealing in a Grid environment. In *Proceedings of the Third International Workshop on Grid Computing (Grid2002)*, 2002.

[YFS03] L. Yang, I. Foster, and J. M. Schopf. Homeostatic and tendency-based CPU load predictions. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[Ygg98] F. Ygge. *Market-Oriented Programming and Its Application to Power Load Management*. PhD thesis, Department of Computer Science, Lund University, 1998.

[Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8, 1965.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D Zappala. Rsvp: A new resource reservation protocol. *IEEE Networks Magazine*, 31(9):8–18, September 1993.

[ZFS03] Xuehai Zhang, Jeffrey Freschl, and Jennifer M. Schopf. A performance study of monitoring and information services for distributed systems. In *Proceedings of the IEEE Twelfth International Symposium on High-Performance Distributed Computing (HPDC-12)*, 2003.

[Zho92] S. Zhou. LSF: Load sharing in large-scale heterogeneous distributed systems. In *Proceedings of the Workshop on Cluster Computing*, December 1992.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report CSD-01-1141, Berkeley, 2001.

[ZM98] W. Ziemba and J. Mulvey, editors. *Worldwide Asset and Liability Modeling*. Cambridge University Press, 1998.

[ZWZD93] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. *Software Practice and Experience*, 23(2), 1993.

Index

Abstract Application Resource and Topology (AART), 82
Access control, 350
Access Grid, 7
Access to tentative schedule, 45
Accounting, 39, 475
Advance reservation, 47, 120, 163, 167, 173, 183, 186, 248, 382
Advanced networking, 3
Agent, 121
Aggregate predictions, 226
Akenti, 64
Allen, Gabrielle, 25
Allocation
  cost, 47
  offer, 167
  offers, 46
  properties, 48
  revocation, 48
Angulo, Dave, 73
Apache Web server, 347
Application, 25, 28
  deployment, 31
  frameworks, 38
  level scheduling, 276
  requirements, 19, 28, 82
    compute-related, 28
    data-related, 29
    network-related, 30
  software provider, 481
Arpaci-Dusseau, Andrea C., 340
Arpaci-Dusseau, Remzi H., 340
Artificial intelligence, 3
  AI-based planner, 113
Authentication, 53, 57
Authorization filtering, 18
Authorization, 53
Autocorrelation structure, 208
Available occupancy percentage, 203
Backfill, 243
Bandwidth broker, 386
Benefit-driven peer clustering, 410
Bent, John, 340
Best effort space, 331
Billing, 475
Binder, 75
Binding Service Level Agreements (BSLA), 120, 127
Bioinformatics, 6
Biomedical Informatics Research Network (BIRN), 6
Black hole, 433, 442–443
Blythe, James, 99
Broker, 33, 120–121
  adaptive brokering, 33
Business Process Execution Language for Web Services (BPEL4WS), 452
Butterfly.net, 7
Cache-aware scheduling, 347
Cactus, 7, 34–35, 78, 229
  flesh, 36
  thorns, 36
Casanova, Henri, 73
CERN Advanced Storage Manager (CASTOR), 336
Chargeable Grid Service, 485
Checkpointing, 51, 176
Cheliotis, Giorgos, 465
ChicagoSim, 360, 367
Chien, Andrew A., 431
Chirp, 343
Class object, 148
ClassAds, 77, 82, 255, 343, 350
  abstract syntax, 257, 259, 261
  attribute reference resolution, 257
  canonical unparsing, 263
  concrete native syntax, 257, 261–262
  concrete XML syntax, 257
  constants, 257, 259, 264
  evaluation, 264
  lazy evaluation, 268
  matchmaking (candidate and access), 259
  matchmaking (requirements and rank), 258
  matchmaking, 77
  non-strict operators, 265
  operators, 261
  reserved words, 260
  strict functions, 268

strict operators, 265
undefined and error, 257, 264
using SELECT on a list, 267
using SUBSCRIPT on a record, 267
value identity, 266
Clearing, 475
Client-server, 341
Co-allocation, 120, 132, 174, 180, 501
Co-reservation, 383
Co-scheduling, 174, 180, 189
Collection database, 159
Commercialization stack, 474
Community Authorization Service (CAS), 64
Community scheduler, 121–122
Compact Muon Solenoid (CMS), 36
Monte Carlo Production (MOP), 36
Compilers, 98
Composed application service model, 397
Compromise solution, 274
Computational economies, 479
Computational resource provider, 481
Compute intensiveness index, 447
Concurrency models, 347
Condor, 63, 135, 255, 480
claiming, 139
ClassAds, 77, 82, 255, 343, 350
Condor-G, 38
DAGMan, 351
matchmaker, 138–139, 143, 255, 258, 270
multilateral matchmaking, 270
preemption, 143
NeST, 341
storage management, 341
storage, 341
Configurable Object Program (COP), 75, 81
Conservative scheduling, 215–216
Consistency, 21
Constraint satisfaction, 129
Contract, 119, 475
monitoring, 92
CPU, 217
load prediction, 217
sensing, 201
active, 201
passive, 201
Credentials, 53
Criteria, 279
Cycle harvesting, 186
Cycle stealing, 186
Czajkowski, Karl, 119
DAGMan, 351
Dail, Holly, 73, 217
Data
intensive, 447
locality, 85, 432–433, 447
management, 25, 341, 359
placement, 329
tier architecture, 326
privacy, 434
protection, 434
resource, 322
transfer, 341
Data Grid, 322
Data-parallel, 73
Dataset scheduler, 365–366
policy, 366
Deallocation policy, 47
Decentralized, 459
Decision support, 469
Deelman, Ewa, 99
Delegated credentials, 59
Delivery, 477
Deregulation, 468
Description parameter, 495
Desktop Grid, 432, 434, 448–449
Differentiated services (Diffserv), 386
Dinda, Peter, 217
Direct neighbors, 405
Disk Resource Manager (DRM), 321
Distributed Aircraft Maintenance Environment (DAME), 5
Distributed service path instantiation, 403
Distributed simulations, 32
Distribution
long-tailed, 194, 222
multimodal, 223
normal, 219
Dome, 217
Dongarra, Jack, 73
Drug screening, 6
Durable file, 330
Durable space, 331
Dynamic data, 20
replication, 359
Dynamic Soft Real-Time (DSRT), 382
Economic engineering, 465, 472
Economy-based Grids, 480
Elbert, Stephen T., 431
Electricity, 466
EMC Corporation, 354
Enactor, 151
Encryption, 55
End-users, 274
Enstore, 337
Enterprise Grid, 179, 354
security, 179
job types, 179
monitoring and discovery, 179
scheduling policies, 179
storage, 354
Enterprise
resource management, 7
Entropia, 432, 435, 437–438, 440–441, 445
Entropia 2000, 438

Batch System (EBS), 440
App Server, 435–437
batch system server, 440–443
client, 435–443
DCGrid, 432, 438, 440–441, 444, 447
File Server, 435, 439–440, 445
Task Server, 435–438
Ernemann, Carsten, 491
European Data Grid (EDG), 37, 340
Event
management, 166
notification, 46
Event-driven server, 347
Exclusive allocation, 50
Exclusive control, 45, 177
Execution stack, 473
Execution time prediction, 238
Exponential smoothing, 208
Extensible Markup Language (XML), 483
External scheduler, 364
Factory port, 484–485
FASTA, 79
Fault tolerance, 493
Fermi Laboratory, 373
Figueira, Silvia, 217
File, 123, 329
pinning, 325
release (unpin), 325
replication, 337
staging, 187
transfer service, 121, 123
types, 329
durable file, 330
permanent file, 329
volatile file, 329
First-Come First-Served (FCFS), 243
Fish, 79
Forecasting techniques, 208
Foster, Ian, 3, 73, 119, 359, 413
Full-plan-ahead, 105
Game of Life, 79
Gantt chart, 445–446
Genetic algorithm, 240, 311
Gigabit testbeds, 3
Gil, Yolanda, 99
Global Grid Forum (GGF), 183, 480, 489
Grid Economic Services Architecture (GESA) Working Group, 480, 484, 489
Grid Resource Allocation Agreement Protocol (GRAAP) Working Group, 131Open Grid Services Architecture (OGSA) Working Group|458 Global Grid Forum (GGF),Open Grid Services Architecture (OGSA) Working Group|473 Global Grid Forum (GGF),Open Grid Services Architecture (OGSA) Working Group|480 Global Grid Forum (GGF),Open Grid Services Architecture (OGSA) Working Group|483 Global Grid Forum (GGF),Open Grid Services Infrastructure (OGSI) Working Group,,,
Global Grid Forum (GGF)
Open Grid Services Architecture (OGSA) Working Group|030 Global Grid Forum (GGF)
Open Grid Services Architecture (OGSA) Working Group|131 Global Grid Forum (GGF), 484
Resource Usage Service (RUS) Working Group, 489
Usage Records (UR) Working Group, 489
GlobeXplorer, 7
Globus toolkit, 32, 168, 171, 180, 392, 480
Community Authorization Service (CAS), 64
Dynamically Updated Request Online Co-allocator (DUROC), 120
Globus Architecture for Reservation and Allocation (GARA), 120, 377
Globus Resource Allocation Manager (GRAM), 120, 180
GRAM-2, 120, 131
Grid File Transfer Protocol (GridFTP), 38, 77, 187, 339
Grid Security Infrastructure (GSI), 345
MDS2, 17, 77, 102, 181
Platform Globus, 180
replica catalog, 38
Replica catalog, 334
Replica Location Service (RLS), 102, 334
Resource Specification Language (RSL), 181
security, 63
Service Negotiation and Acquisition Protocol (SNAP), 120
Globus Architecture for Reservation and Allocation (GARA), 120, 377
Local Resource Manager (LRAM), 380
Gold, 167
Goodale, Tom, 25
Grid, 3
applications, 25
architecture, 473
comparison with P2P, 414
definition of, 4
deployed, 416
economics, 465
emulator, 413, 422
reservation, 21
scheduling, 15
services, 479
user communities, 414
Grid Application Development Software (GrADS), 28, 34, 73
Grid Application Toolkit (GAT), 36, 451–452, 459
Grid Data Mirroring Package (GDMP), 38

Grid File Transfer Protocol (GridFTP), 38, 77, 187, 339
Grid Information Services (GIS), 16, 20, 181
Grid Monitoring Architecture (GMA), 17
Grid Physics Network (GriPhyN), 28, 99
Grid Physics Network Project (GriPhyN), 37
Grid Security Infrastructure (GSI), 392
Grid Service Handle (GSH), 484
GridLab, 28, 278, 293
GridSphere, 36
Grimshaw, Andrew S., 145
Guaranteed completion time, 49
Gu, Junmin, 321
Gu, Xiaohui, 395
Hard constraints, 278
Heartbeat, 440, 443
Hierarchical Resource Manager (HRM), 321
High Energy Physics (HEP), 6
High Performance Storage System (HPSS), 337
Host, 151
  object, 152
  properties, 151
Humphrey, Marty A., 145
Iamnitchi, Adriana, 413
Implementation object, 155
In-time global scheduling, 105
In-time local scheduling, 105
Indirect neighbors, 405
Instance-based learning, 241
International Virtual Data Grid Laboratory (iVDGL), 373
Internet Backplane Protocol (IBP), 345, 354
Internet Grid, 431–435, 438
Iterative, 79
Jackson, David B., 161
Jackson, Keith R., 53
Jacobi, 79
JASMine, 337
Job, 15, 435, 437–439, 444
  checkpointing, 169
  configuration, 503
  dependencies, 48
  dependency, 169
  dynamic, 165
  execution, 21
  forwarding, 179
  management, 439, 442, 444
  migration, 169–170
  preemption, 169
  restart, 52, 170
  scheduling, 275
  staging, 121, 124
  state mapping, 181
  state, 181
  submission, 21
  types, 179
Jones, James Patton, 183
Just a Bunch Of Servers (JBOS), 345
JXTA, 451, 456
Keahey, Katarzyna, 479
Kenyon, Chris, 465
Kerberos, 58
Kesselman, Carl, 3, 99, 119
Kurowski, Krzysztof, 271
Least work first (LWF), 243
Legion, 145, 354
  protocol stack, 156
  scheduling process, 157
  security, 63
LeRoy, Nick, 340
Liberty Alliance, 60
Linear systems, 78
Liu, Chuang, 73, 217
Livny, Miron, 135, 255, 340
Load balancing, 217, 395
Local scheduler, 365–366
Local search, 309
Locking, 335
Logical File Name (LFN), 334
Loosely synchronous iterative applications, 219
Lots, 350
Lowest Common Denominator (LCD) tendency, 180
Lumb, Ian, 171
MacLaren, Jon, 479
Makespan, 295
Malleable allocation, 50
Manageability, 341
Mapper, 84
Market equilibrium, 493
Marlin, Shawn, 431
Mars, 217
Master schedule, 153
Matchmaker, 138, 143, 255, 258
  multilateral matchmaking, 270
  preemption, 143
Maui scheduler, 161
Maximum allocation length, 47
Mean absolute error, 209
Mean absolute prediction error, 209
Mean square error, 209
Message digest, 55
Message Passing Interface (MPI), 32, 77, 94, 174
  MPICH-G2, 32, 77
Metacomputing, 32
Metadata service, 112
Metaheuristics, 309
Metascheduling, 73, 95, 121, 158
Migration, 51, 92, 176
Mika, Marek, 295
Miller, Lawrence J., 193
Monitoring, 22, 179
Monte Carlo Production (MOP), 38

Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP), 302
Multi-Mode Resource-Constrained Project Scheduling Problem with Schedule-Dependent Setup Times (MRCPSP-SDST), 303
Multi-process server, 347
Multi-threaded server, 347
Multicriteria decision problem, 271–272
Multicriteria optimization, 282
Multiplayer games, 7
MultiProtocol Label Switching (MPLS), 392
MyGrid, 6
N-hop service path, 398
Nabrzyski, Jarek, ix, 25, 271, 282
Nahrstedt, Klara, 395
Natrajan, Anand, 145
NEESgrid Earthquake Engineering Collaboratory, 6
Negotiation, 151, 153, 157, 160
  transfer protocol, 333
NeST, 341
  access control, 350
  cache-aware scheduling, 347
  Chirp, 343
  ClassAds, 343, 350
  concurrency models, 347
  example scenario, 351
  example usage, 353
  installing, 353
  lots, 350
  proportional-share scheduling, 347
  quality of service, 347
  quotas, 350
  related work, 354
  storage guarantees, 350
  storage management, 350
  virtual protocol, 345
Network appliance, 341, 354
Network reservation, 385
Network Weather Service (NWS), 77, 193, 217–218, 227
Newhouse, Steven, 479
NimrodG, 480
Nitzberg, Bill, 183
Non-storability, 468
Normal distribution, 220
  arithmetic, 220
Numerical relativity, 34
NWIRE, 492
Obertelli, Graziano, 193
Objective function, 493, 502
Offer creation, 496
Offer description, 495
Oleksiak, Ariel, 271
Open Grid Services Architecture (OGSA), 30, 131, 171, 182, 458, 473, 480, 483
Open Grid Services Architecture Distributed Access and Integration (OGSA-DAI), 30
Open Grid Services Infrastructure (OGSI), 484
  Grid Service Handle (GSH), 484
  Service Data Element (SDE), 484
Optimization, 465
Over-scheduling, 444
Owner preemption, 142
Parameter-space studies, 158
Particle Physics Data Grid (PPDG), 37
Partner Grid, 179–180
Passport, 60
Peer clustering, 396
Peer scheduling, 187
Peer-to-Peer, 395, 413, 451, 456, 477
  applications, 416
  characterization, 414
  comparison with Grids, 414
  scheduling, 187
  systems, 420
  user communities, 414
Pegasus, 100, 107
Performance characteristics, 194
Performance
  contract, 75, 92
  evaluation, 501
  modeling, 83
  prediction, 195, 237
Permanent file, 329
Permanent space, 331
Physical File Name (PFN), 334
Pin, 325
  two-phase pinning, 336
Pin file, 325
Pin lifetime, 325
Pin lock, 336
Planning, 99
Platform Globus, 180
  MDS, 181
Platform LSF
  advance reservation, 173
  checkpointing, 176
  event notification, 172
  exclusive control, 177
  job dependencies, 174
  migration, 176
  preemption, 176
  restart, 176
  tentative-schedule access, 176
Platform MultiCluster, 179
  co-allocation, 174, 180
  co-scheduling, 174, 180
  job forwarding, 179
  job types, 179
  Message Passing Interface (MPI), 174
  monitoring and discovery, 179
  resource leasing, 180

scheduling policies, 179
security, 179
Pluggable architecture, 453
Pluggable File System (PFS), 354
Point value, 219
Policy, 187, 366
application, 119
community, 121, 126
distribution, 455
domain bridging, 126
intermediaries, 126
mapping, 126, 131
resource owner, 119
Pool, 432
application, 436–437, 440
resource, 432, 434, 440, 443, 448
Portable Batch System (PBS), 183, 382
PortType, 131
Prediction, 33, 215, 238
CPU load, 217
execution time, 238
load interval, 228
load variance, 229
queue wait time, 242
run time, 238
techniques, 291
tendancy-based, 227
Preemption, 51, 135, 143, 176
matchmaker, 143
owner, 142
user, 142
Preemptive resume scheduling, 135
Preparation tasks, 22
Price, 467–468, 476
dynamics, 467
formation, 476
forward, 468
model, 468
Pricing, 481
Primary predictor, 208
Priority, 164
Process swapping, 94
Products, 475
Proportional-share scheduling, 347
Provisioning, 128
Public key cryptography, 58
Push caching, 361
QBank, 167
Quality of Service (QoS), 347, 377, 395, 473
consistency, 397
Quality-of-information, 17
Queue wait time prediction, 242
Quotas, 350
Raman, Rajesh, 255
Ranganathan, Kavitha, 359
Reliability, 99, 493
Replacement policy, 326
Replica catalog, 334
Replica Location Service (RLS), 102, 334
Replication, 366
algorithms, 366
policies, 366
Web, 361
Request
description, 495
example, 497
manager, 327
planner, 327
Rescheduling, 89
process swapping, 94
Reservation, 161, 465, 481
advance, 47, 120, 163, 167, 173, 183, 186, 248, 382
bandwidth, 125
courtesy, 167
disk space, 125
Resource, 11, 15
administrators, 274
commercialization, 465
configuration, 501
coordination, 119
description language, 278
discovery, 17, 75
components, 418
definition, 413
mechanisms, 419, 425
requirements, 416
leasing, 180
location, 413
monitoring, 196
owners, 274
tuple, 401
virtualization, 125
Resource Service Level Agreement (RSLA), 120, 127
Resource Specification Language (RSL), 278, 381
Resource-Constrained Project Scheduling Problem (RCPSP), 295, 302
Restart, 176
Robust file replication, 337
Round-robin scheduler, 158–159
Roy, Alain, 135, 255, 340, 377
Rule-based system, 281
Run time prediction, 237–238
Run-to-completion, 49
Russell, Michael, 25
Sander, Volker, 377
Satisfiability, 98
Scalability, 21
ScaLAPACK, 78
Schedule, 86, 166
search procedure, 86
tentative, 166
Scheduler, 41, 121, 365, 483

dataset, 365
architecture, 483
higher-level, 41
intermediary, 121
lower-level, 41
round-robin, 158–159
Scheduling, 73, 246, 432–435, 437–441, 443–445, 447–448, 491
algorithms, 366
architecture, 364
attributes, 41, 171–172, 183
allocation-property attributes, 174
available-information attributes, 172
manipulating allocation-execution attributes, 175
resource-requesting attributes, 173
co-allocation, 120, 132, 174, 180, 501
co-scheduling, 48
data locality, 85
decentralized, 364
decision maker, 272
preferences, 272
economic, 491, 493
infrastructure, 492
launch-time, 80
mapper, 84
metascheduling, 95
migration, 92
objective, 47
parameter, 495
performance modeling, 83
policies, 179
policy, 366, 443, 493
problems, 492
rescheduling, 89
simulated annealing, 88
Schema, 17
Schopf, Jennifer M., ix, 15, 183, 215
Schwiegelshohn, Uwe, 41
Search procedure, 86
Secondary predictor, 210
Security, 39, 179
authentication, 53, 57
authorization, 53
credentials, 53
Grid Security Infrastructure (GSI), 345, 392
Kerberos, 58
policy, 53
public key cryptography, 53, 58
X.509 certificates, 58, 185
Seidel, Ed, 25
Selection, 20
Server, 347
event-driven, 347
multi-process, 347
multi-threaded, 347
Service, 11, 479
aggregation, 482
brokering, 483
composition model, 398
composition, 395
Grid, 479
integration, 11
path instantiation, 395
path selection, 395
policy, 326
provider, 477
virtualization, 7
Service Data Element (SDE), 484
Service Level Agreements (SLA), 119, 127, 218, 481, 485
bind, 120
commitment, 129
linkage, 125
ordering, 127–128
resource, 120
satisfaction, 130–131
task, 120
three kinds, 127
violation, 129
Service Negotiation and Access Protocol (SNAP), 182
Shibboleth, 60
Shields, Matthew, 451
Shoshani, Arie, 321
Sievert, Otto, 73
Silver Grid scheduler, 161
Simple Object Access Protocol (SOAP), 483
Simulated annealing, 88, 295, 310
Simulation, 359, 367, 504
ChicagoSim, 360, 367
Simulator, 360
Sim, Alexander, 321
Site URL, 335
Sliding window, 208
Smith, Chris, 171
Smith, Warren, 237
Soft constraints, 278
Software application development, 73
Solomon, Marvin, 255
Space, 331
quota, 326
reservation, 330
types, 331
best effort space, 331
durable space, 331
permanent space, 331
volatile space, 331
Stanley, Joseph, 340
Static vs dynamic data, 20
Stochastic, 469
optimization, 469
scheduling, 215
value, 220

Storage, 341
  appliances, 341
  elements, 321
  guarantees, 341, 350
  management, 321, 341, 350
  resources, 321
  scheduling, 329
Storage Resource Broker (SRB), 354
Storage Resource Manager (SRM), 321, 350–351, 354
Subjob, 439–440, 443–446
  failure, 440–444, 446
  management, 438–440, 443, 445
  monitor, 440–443, 447
  retry, 441
  timeout, 443
Superscheduler, 120–121
Swany, Martin, 193
Tabu search, 295, 311
Tape Resource Manager (TRM), 321
Task Service Level Agreements (TSLA), 120, 127
Task-graph, 455
Taylor, Ian, 451
Temporal locality, 224
Tentative schedule, 45
Thompson, Mary R., 53
Tier architecture, 326
Time balancing, 217
Time series, 208
Tradable Grid services, 479
Trading, 476
Transfer protocol negotiation, 333
Triana, 451–452, 456
Triumvirate, 137
Tuecke, Steven, 119
Two-phase pinning, 336
UK E-Science Computational Markets Project, 487–488
UK E-Science Core Programme, 488
UNICORE, 480
Unix load average, 200
Unpin, 325
URL, 327, 334
Use cases, 481
User, 16
  dynamic account, 124
  preemption, 142
  preferences, 40, 271
Utility function, 502, 504
Utility value, 494
Vadhiyar, Sathish, 73
Variance, 215
Venkataramani, Venkateshwaran, 340
Virtual, 11
  observatory, 6
  organization, 4, 39, 180, 359
    administrators, 274
  protocol, 345
Volatile file, 329
Volatile space, 331
Wait time prediction, 237
Waligóra, Grzegorz, 295
Walltime, 168
Wang, Ian, 451
Web service, 65
Web Service Description Language (WSDL), 483
Web Services Flow Language (WSFL), 452
Węglarz, Jan, ix, 271, 295
Weissman, Jon, 217
Wolski, Rich, 193, 217
Workflow, 451–452, 454, 460
  generation, 104
  management, 99, 295
Workload, 503
  management, 183
World Wide Telescope, 6
WorldPay, 480
X.509 certificates, 58, 185
Yahyapour, Ramin, 41, 491
Yang, Lingyun, 73, 215
YarKhan, Asim, 73