Heuristics for Enforcing Service Level Agreements in a Public Computing Utility

by

Balasubramaneyam Maniymaran

A Thesis

Submitted to the Faculty of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Department of Electrical and Computer Engineering

University of Manitoba
Winnipeg, Manitoba, Canada

“To my father and mother

who sacrificed their pleasant days
to build my future pleasant”

Abstract

With the increasing popularity of consumer and research oriented wide-area applications, there arises a need for a robust and efficient wide-area resource management system. Even though there exists number of systems for wide area resource management, they fail to couple the QoS management with cost management, which is the key issue in pushing such a system to be commercially successful. Further, the lack of IT skills within the companies arouses the need of decoupling service management from the underlying complex wide-area resource management. A *public computing utility* (PCU) addresses both these issues, and, in addition, it creates a market place for the selling idling computing resources.

This work proposes a PCU model addressing the above mentioned issues and develops heuristics to enforce QoS in that model. A new concept called *virtual clusters* (VCs) is introduced as semi-dynamic, service specific resource partitions of a PCU, optimizing cost, QoS, and resource utilization. This thesis describes the methodology of VC creation, analyses the formulation of a VC creation into an optimization problem, and develops solution heuristics. The concept of VC is supported by two other concepts introduced here namely *anchor point* (AP) and *overload partition* (OLP). The concept of AP is used to represent the demand distribution in a network that assists the problem formulation of the VC creation and SLA management. The concept of overload partition is used to handle the demand spikes in a VC.

In a PCU, the VC management is implemented in two phases: the first is an off-line phase of creating a VC that selects the appropriate resources and allocates them for the particular service; and the second phase employs on-line scheduling heuristic to distribute the jobs/requests from the APs among the VC nodes to achieve load balancing. A detailed simulation study is conducted to analyze the performance of different VC configurations for different load conditions and scheduling schemes and this performance is compared with a fully dynamic resource allocation scheme called *Service Grid*. The results verify the novelty of the VC concept.

Acknowledgement

First of all, I would like to thank my advisor **Dr. Muthucumaru Maheswaran** for his continuous patience, guidance, and support in the completion of this thesis. Also my thanks goes to **Dr. Sylvanus A. Ehikioya** and **Dr. Bob McLeod** for their valuable time spent in being part of my examination committee.

Also, I would like thank **University of Manitoba** and its **Department of Computer Science** and **Department of Electrical and Computer Engineering** for the opportunity and the resources they provided to my research. In addition, I would like to convey my gratitude to **TR*Labs*, Winnipeg** for the ample facilities it provided, without which my research would not have been a reality. Especially I am thankful to all **the staff and colleagues at TR*Labs*** for the direct and indirect support they gave towards my research.

I would like to thank also my friends **Kumaran** and **Vasee** for their valuable feedbacks on this thesis.

At last, but not least, I convey my heartiest thanks to all **my family members** and **friends** for their continuous moral support in my entire life and career.

Contents

Abstract iv

Acknowledgement vi

1 Introduction 1

2 Literature Survey 5

2.1 PCU Related Projects 5

2.1.1 Condor 5

2.1.2 Legion 6

2.1.3 Océano 7

2.1.4 Shared Hosting Platforms 7

2.1.5 Cluster-on-demand 8

2.1.6 Virtual Data Center 9

2.1.7 Grid 10

2.2 Service Level Agreements 12

2.3 Other Related Works 14

2.3.1 WebOS 14

2.3.2 Service Grid 15

2.3.3 Self-Organizing Network Services 16

2.3.4 Spawning Networks 17

2.3.5 Cluster Reserves 17

2.4 Optimization Techniques in Wide-Area Networks 19

3 Public Computing Utility Model **20**

4 Virtual Clusters **23**

5 Heuristics for Enforcing the SLAs in PCUs **26**

5.1 Mathematical Formulation 26

5.1.1 Practical Validity of the Formulation 29

5.2 Solution Methods and the Drop Heuristic 30

5.3 VC Creation – An Example 32

5.4 Request Scheduling 36

6 Simulation, Results, and Discussions **38**

6.1 Assumptions 38

6.2 Simulation Setup 39

6.3 Load Prediction and Active Capacity 41

6.4 Results and Discussions 42

7 Conclusion **52**

7.1 Contributions 52

7.2 Limitations 53

7.3 Future Works 53

A Abbreviations **55**

Bibliography **56**

List of Tables

5.1 The preferred server list headed by the “home servers”. 35

6.1 Performance of the Service Grid with mean request interval. 43
6.2 Performance of the VC without OLP with mean request interval using greedy server acquisition. 44
6.3 Performance of the VC without OLP with mean request interval using probabilistic server acquisition. 44
6.4 Performance of the Service Grid with mean request length. 45
6.5 Performance of the VC without OLP with mean request length using greedy scheduling. 45
6.6 Performance of the VC without OLP with mean request length using probabilistic scheduling. 46
6.7 Performance of the VC with OLP with mean request interval. 46
6.8 Performance of the VC with OLP with mean request length. 47
6.9 Performance of the Service Grid for windowed loading. 47
6.10 Performance of the VC without OLP for windowed loading. 47
6.11 Performance of the VC with OLP for windowed loading. 47
6.12 Performance of the Service Grid for asymmetrical loading. 48
6.13 Performance of the VC without OLP for asymmetrical loading. 49
6.14 Performance of the VC with OLP for asymmetrical loading. 49

List of Figures

3.1 Resource management in a PCU 20
3.2 An overall picture of PCU operation 21

4.1 VC creation process on a PCU system. 23

5.1 The modified drop heuristic. 31
5.2 The initial network. 33
5.3 The created VC with OLP 34

6.1 Resource utilization within a cluster. 50
6.2 Unit utilization cost of a cluster. 51

1 Introduction

Tremendous cut down in the cost of the IT appliances has increased the availability of powerful appliances to average consumers. Further, mobile devices have become popular and web-enabled, creating a pervasive Internet. Following these observations, *Wide-area applications*, whether they are commercial services like *video-on-demand* or scientific applications like distributed computing, have started exploiting the Internet to reach a large consumer audience. As a result, Internet population has increased to hundreds of million that mostly enjoy the services through the Internet than telneting remote servers. Through these changes, the Internet that has been viewed as a network to connect computers across the world, has become a "service network" that is a base of varieties of wide-area services. In addition, with state-of-the-art devices easily available to the end-users, the users expect much higher *quality of service* (QoS) from those services. These observations impose a number of challenges on a wide-area service management infrastructure.

Wide-area services have to consume a large amount of resources to satisfy the customer population that spans the globe. Often the resources to build the services need to be distributed globally to reach the target population with required QoS. Generally the *service originators* (SOs) of the services buy and organize their own resources in local area or wide area clusters. But, this is a costly option and often those resources are underutilized since the clusters are generally designed for near worst case scenarios and demand fluctuations in wide-area services are very high.

Guaranteeing performance over a wide-area is a difficult task, mainly because the Internet is not designed to consider QoS aspects, but just to provide connectivity. In the Internet achieving QoS is hindered by number of aspects: (a) *first mile bandwidth* problem (generally the link from end-user to the edge of the Internet is of low bandwidth); (b) unreliable peer-to-peer links causing congestion providing varying performances for different customers; (c) the back bone switches and routers getting overloaded causing content loses; and (d) lack of fault tolerance deteriorating the service.

Another interesting observation in the field of wide-area services is the lack of technical skill within organizations. The service originators have the service and capital but not the technical skill to manage a complex resource base beneath. Therefore, generally SOs want to manage just the services but hand over the responsibility of managing the resources to a third party.

There exists number of solutions that address the above mentioned issues. Content and application *outsourcing* is a solution for acquiring enough resources to host a wide-area service and it is becoming almost a norm for web-hosting. Outsourcing works by an SO handing over the service content and the responsibility of hosting the service on the Internet to a third party and pay the money for that service. The hosting agent uses its abundance of widely distributed resources to host multiple services. Akamai [Aka02] is a popular example for this which hosts number of famous web sites such as Yahoo, MTV, AOL, Victoria's Secret, and many more. Outsourcing is also a solution for alleviating the lack of IT skills in a company. Another approach to gain a large resource pool is to use the idling resources in the Internet. Condor [Con02] and *Internet computing* like SETI@home [SET02] are examples for this. This is motivated by the observation of a huge amount of computational power idling on the Internet while some applications suffering from the scarcity of enough computing power. Projects like Legion [LeV02] are dedicated to *metacomputing* which orchestrate resources over a wide geographical span to perform distributed computing. Akamai's *edge delivery* system promises a QoS guaranteed deployment of a service. Edge delivery is a technique of moving the service and content to the "edge" of the Internet avoiding the in-network loses and congestion. Grid architecture collectively addresses some of the issues in wide-area resource management such as aggregating multi-owner resources, resource dissemination and discovery, providing metacomputing functionality, and supporting wide-area services. Projects like Océano [ApF01] handle the large demand fluctuations in wide-area services with dynamically varying resource allocation.

Even though the above mentioned solutions address a number of issues towards realizing a QoS-centric wide-area resource management, each of them alone could not provide an efficient commercial infrastructure that can host various types of wide-area services, because they are narrowly focused, research or academic based, or lacking number of components to realize them as commercial solutions. Further, these solutions fail to consider the trade-off between the cost of service instantiation and the QoS provision, that decides the practical feasibility of a solution. In this thesis a *public computing utility* (PCU) model is proposed as an infrastructure to host various wide-area services considering cost and QoS aspects.

A public computing utility is like any other public utility such as water supply or electricity. A PCU aggregates computing power from various owners and, on the other hand, the consumers subscribe the PCU service provider and pay for the service based on usage. The computing power from a PCU can be used in various ways such as to build up a web-server cluster, to perform distributed computing, or to setup a storage area network. The transfer of computing power from owner to consumer is transparent, hiding the origin and the destination. This high level abstraction simplifies the resource management for the consumers.

The vision of a PCU is not new, but dates back to early 1980s. A number of projects such as IBM's Océano and HP's virtual data center are moving in that direction. However, the PCU architecture proposed in this work has its unique aspects towards providing a QoS guaranteed, cost effective, and service specific resource management infrastructure. In summary, in this work:

1. A unique PCU model is introduced that can be the resource management substrate for the wide-area services of the future.

2. New concepts called *anchor points* (APs), *virtual cluster* (VC), and *overload partition* (OLP) are introduced to facilitate the above concepts. Anchor points provides an abstraction of demand intensity in a network; virtual cluster is the QoS abiding resource collection for a service; and overload partition is the protection layer of the virtual cluster to handle demand spikes.

3. An optimization based mathematical model is presented for VC creation. Also a

solution heuristic is developed, that results in a cost effective and the same time QoS guaranteed resource partition.

4. Two different request scheduling schemes are analyzed to balance the load among VC nodes.

5. A simulation study has been carried out to validate the solution heuristic and to compare the performance of the VC concept with another concept called Service Grid [LeW01].

The purpose of this research work is not to implement a complete PCU model, but to establish the concept of such a PCU model and to develop some heuristics that enforce QoS in such a model. I took this research direction because I strongly felt that enforcing QoS measures is the primary key for such a model to be commercially successful.

This thesis is organized as follows: Chapter 2 provides concise descriptions of related research works and presents the uniqueness of the ideas presented here compared to the existing work. The proposed PCU model is further described in Chapter 3 and Chapter 4 presents the concept of virtual cluster. The Chapter 5 formulates the mathematical model for the creation of a virtual cluster, discusses its practical validity, and presents the solution heuristic. It also discusses two scheduling schemes for load balancing. The details of the simulation study and the results obtained are given in Chapter 6 with detailed discussion. The Chapter 7 concludes this thesis. In Appendix A, a list of abbreviations used in this thesis is given.

2

Literature Survey

This chapter provides brief descriptions of different research works in the field related to my work and highlights the unique aspects of my work from the existing ideas. The first section discusses some projects that have direct connection with or share the vision of our PCU concept. The Section 2.2 talks about the service level agreement and relevant projects. Number of other projects that share some ideas of our PCU infrastructure (even though they do not fall into the utility computing category) are given in Section 2.3. The final section of this chapter talks about some existing approaches that uses optimization techniques for the wide-area resource management in computer networks.

2.1 PCU Related Projects

A number of research efforts have been put forwards in addressing various issues towards implementing various visions of a computing utility. The following sections describes some of the research works that try to implement a multi-service, multi-ownership resource management infrastructures.

2.1.1 Condor

Condor [Con02] is the earliest project that had a vision towards a PCU. It was introduced [LiL88] as a mechanism for *high throughput computing* which can exploit the idling computing powers of different owners in a cluster of an organization to increase the overall computing throughput of that cluster. Hence, Condor is the first in the history to address resource sharing specifically among multiple autonomous owners. Thus, Condor is designed with the three principles: (1) Condor jobs should have no impact on the availability and QoS to the owner of the resources; (2) Condor job execution should be transparent to the users; and (3) Condor should require no special programming and preserve the operating environment of the machines.

To obey these principles, the Condor's system design includes the following components:

Scheduling structure: The scheduling structure of Condor is a hybrid of centralized and distributed approaches. While a *central manager* is responsible for resource discovery and notifications, localized daemons are responsible for scheduling locally generated jobs in the Condor system and executing the submitted Condor jobs locally.

Remote Unix facility: Remote Unix (RU) turns idle workstations into cycle servers. When RU is running on a workstation, it can fork *shadow* processes which can do the scheduling or execution of the Condor jobs.

Checkpointing: Since Condor always gives priority to the owners of the workstations, any remote Condor job can be preempted any time. Condor enforces a *checkpointing* mechanism to avoid data loss at the time of preemption

The Condor system that is originally designed for achieving high throughput within an organization expands its idea to span over multiple autonomous organizations through the *flocks of Condors* project [EpL96]. Flocks of Condors provides different level of access rights of computing power in an intra-organization and inter-organization levels.

2.1.2 Legion

Legion [LeV02, GrW94] is another project similar to Condor (section 2.1.1) but truly implementing a metacomputing system. It identifies eight areas of developments to develop a *nationwide virtual computer* : achieving high performance via parallelism, managing and exploiting component heterogeneity, resource management, file and data access, fault-tolerance, ease-of-use and user interfaces, protection and authentication, and exploitation of high-performance protocols. Legion differentiates itself from Condor mainly by providing facilities to job parallelization and parallel execution. Also, Legion's implementation structure differs from Condors being completely object oriented.

2.1.3 Océano

IBM's *Océano* project [ApF01] is the one among the few that incorporate management of service level agreements (Section 2.2) into the design. But, it deviates from perception of PCU described in this thesis (even though they categorize it as a computing utility) in such a way that it does not provide an infrastructure to collect resources from multiple owners, but to maintain a shared server farm to host multiple services.

Océano infrastructure consists of three tiers: (1) front-end IP sprayers for load balancing among the selected servers; (2) a large pool of *"dolphin"* servers; and (3) a small pool of *"whale"* servers. Whale servers are permanently allocated to services to guarantee the minimum level of agreed QoS, while dolphin servers are dynamically included into and expelled from the service-specific server pools depending on the load conditions of the services. SLAs are observed by event driven mechanisms. Whenever a SLA is breached, *monitoring agents* trigger a *violation events* and a *correlation engine* will analyze the root-course for the event and notifies the *resource director* to carry out corrective measures.

IBM further expands their idea on computing utilities by invoking self-managing services and considering virtualization of resources [ApE02]. The *configuration policy* component can change the functionality of a resource dynamically (for example from a web-server to a fire-wall node) which helps the virtualization of resources.

2.1.4 Shared Hosting Platforms

The work [UrS02] presents another shared server farm (the authors call it as *shared hosting platform*) infrastructure similar to the Océano's (Section 2.1.3), but it differs from Océano in number of ways. It fails to address the SLA management directly as Océano does, however, it provides other novel features such as (a) a method for automatic derivation of QoS requirements; (b) a notion of *overbooking* to increase the revenue per resource; and

(c) algorithms for partitioning of a single resource to gain performance guarantee in finer granularity.

The overbooking mechanism is one of the important features this work proposes: the authors prove that provisioning cluster partitions with a slight yield-off factor in QoS guarantee will enable many times larger number of services that can be hosted in a fixed resource pool, than provisioning partitions with 100% performance guarantee. Not like Océano, the granularity of resource sharing is smaller here making multiple services sharing individual resources. Performance guarantee within a resource is assumed to be achieved by resource reservation mechanisms such as *sand-boxing* provided by the operating systems. The amount of share (named *capsule*) each service should enjoy in every resource node is found by a *placement algorithm*.

2.1.5 Cluster-on-demand

Cluster-on-demand (COD) [MoC02] is another shared cluster solution. The authors use the same phrase *virtual cluster* as I do in my work (Chapter 3). However, here a virtual cluster implies an isolated and secure collection of resources, but not any actual virtualization as my work defines for the same term.

COD provides a mechanism to create virtual clusters for services according to the customer defined *node configuration templates*. When a node is included in a virtual cluster, the COD system configures it according to this template. Virtual clusters are dynamic, in the sense that the number of nodes it contains can change with time depending on the demand. In addition to the computing power the virtual cluster provides, a common storage layer below the cluster pool facilitates a persistent storage pool for the customers. The resource selection for a virtual cluster is made solving a *constraint satisfaction problem*, which just match the requested *node classes* onto the available resources and selecting the resources by their desirability (combination of hardware specification, current usage, and priority information).

2.1.6 Virtual Data Center

Hewlett-Packard's concepts of *virtual data center* [Kot01], *planetary-scale computing* [AnG02], and *service-centric system organization* [VaK01] share our vision of PCU (Chapter 3) in a number of ways. Importantly, the virtual data center realizes the need of virtualization of resources to achieve both the scalable and simplified system manageability and high resource utilization. This virtualization enables services to be hosted transparently on the underlying computer architectures, which is a step towards the next-generation service-centric Internet.
HP's general approach to facilitate such a system is divided into a number of directions:

- Usage of *control layers* – *resource layer* manages the physical resources providing virtual resources while *service layer* allocates the virtual resources to the groups of applications.
- Providing higher granularity of the control resources – underlying details are hidden and abstractions are given at higher levels to reduce the number of resources to be managed.
- Using uniform recursive structure – both the control and controlled infrastructure share the same type of hierarchical structure which enables a service to spawn multiple services or a virtual resource to be built up from multiple virtual resources, eventually enabling a easy way of building up a complex system.
- Providing self-control – flexible mechanisms and algorithms in the participating entities provide autonomous self-analysis and self-governing of system parameters and states.

Similar to the approach proposed in my work (Chapter 3 and 4), HP's infrastructure also uses optimization techniques to utilize the resources economically, prevent overloading, achieve high resource utilization, and provide high-availability and fault-tolerance. However, the parameters it tries to optimize differs from my work; their objectives are to balance the server load keeping the utilization within desired range, to keep the communication demands between services within the capacity of the links between the servers, and to minimize the overall network traffic. It uses an distributed optimization technique using an *agent-based control infrastructure*. Here, small software components called *ants* walk through servers collecting the informations on the way. Upon the completions of the walks the managing agent takes decision of server selection based on the informations the ants sent.

2.1.7 Grid

Grid [FoK01, KrB02], in a way, is the present day standard for utility computing. The goal of a Grid is to provide a highly flexible resource sharing relationship among multiple autonomous entities to create *virtual organizations* (VO). Creating a VO from resources governed by different autonomous policies requires an excellent interoperability, and hence, the design of the Grid is centered around this issue. Grid proposes three different components to provide an efficient interoperability: (a) protocols to provide standard rules for the interactions of heterogeneous, multi-owner resources; (b) services that enhances the protocol operations by providing some functions like access to computation, access to data, and resource discovery; (c) application programming interfaces (APIs) and software development kits (SDKs) to enable high level programming that exploits the underlying grid infrastructure.

Grid is basically a protocol architecture. Protocols simplifies the interoperability of various components by providing standards. Grid architecture is composed of four protocol layers:

Fabric layer: This layer provides a standard low level access to the underlying resources such as computing resources, storage resources, network resources, catalogs, and sensors. The resources can also be logical entities such as a distributed file system or a computer cluster.

Connectivity layer: It defines the communication and authentication protocols required for Grid-specific transactions. Grid communications require services like naming and routing. Authentication protocols requires security services.

Resource layer: The resource layer protocols provide the functionalities to manage individual resources. They can include (a) information protocols to obtain resource informations such as status, load level, and policy informations; and (b) management protocols to specify resource requirements, launch jobs, or access data.

Collective layer: The protocols in this layer helps to orchestrate multiple resources for a single application or a VO. They provide services for co-allocation, scheduling, brokering, monitoring, and many more.

Above these layers, the Grid application layer launches the Grid applications exploiting the APIs and SDKs the layers below provide.

A number of implementations of this Grid architecture are available at present. Among them, the *Globus toolkit* [FoK99] is the most popular one partially because it is created by the pioneers of the Grid concept. Globus provides tools for each of the above layers: GARA for fabric layer, GSI for connectivity layer, GRAM for resource layer, and GIIS, GRIS, and GRRP for collective layer.

The above view of Grid is mainly intended for implementing it as a resource management infrastructure for scientific or technical applications. Later, with the vivid potential of such infrastructure in commercial applications, the Grid technology is now evolving in another axis as *Open Grid Service Architecture* (OGSA) [FoK02]. As the name implies, this is an architecture for orchestrating wide area services using the Grid infrastructure. As the underlying Grid is mainly concerned about providing standard protocols for resource sharing, OGSA provides standard *interfaces* to access wide area services. Interfaces virtualize the underlying resources enabling efficient service management. In addition to the base Grid protocols, OGSA also borrows a number of tools from *web services* [Kre01] for interface management. OGSA architecture consists of *GridServices*, *Factories* that generate services, *Registries* where Factories register themselves, and *HandleMaps* which map the services to the resources.

The above mentioned projects shares a number of aspects with the PCU infrastructure proposed in this thesis. Condor addresses the policy management aspect of a PCU.

And with Legion, they address wide-area resource discovery mechanism that is similar to a PCU. However, these are only dedicated to the metacomputing applications, not for a service-based infrastructure. Océano addresses such service-centric structure and even provides SLA management aspects, but fails to address the multi-ownership of a shared-resource base. Further, it did not specifically address the issues of managing a shared pool distributed over a wide geographical span. Shared hosting platforms shares the same view of Océano failing to address multi-ownership of resources and wide-area distribution of resources. However it employ some algorithm to increase the number of services to be hosted to increase the revenue. Cluster-on-demand just provides some mechanism to partition resources in a shared pool. It also have the same set-backs as the other above mentioned shared-hosting platform solutions have. HP's virtual data center closely resembles our idea of virtualization and resource partition, but fails to cover other aspects such as SLA management, cost of service, and resource utilization. Grid technology with OGSA also shares a number of functionalities of a PCU. But, I feel that the need of manual interactions while using a Grid, failure to provide the true virtualization as needed by a PCU, lack of SLA management infrastructure, and lack revenue-driven structure are some major issues in the Grid technology for it to be a true PCU.

This thesis addresses some important issues in a PCU, that the above existing projects failed to consider:

- The concept of true virtualization of resources (HP's virtual data center in a way addresses this).
- Cost driven resource allocation – instantiation and removal of services at a node result in a service cost, which all the above schemes fail to address.
- demand-driven resource allocation – the allocation process should consider the demand locality of the services to provide the required delivered performance.

2.2 Service Level Agreements

When a *service originator* (SO) outsources its service or content to a third party (a *hosting service*), both the parties have to negotiate and come to an agreement on the terms of delivered performance level, cost of outsourcing, duration of service, acceptable demand conditions, and so on. This negotiation generally results in a *service level agreement* (SLA) which, there onwards, controls the business relationship between the SO and the hosting service.

The notion of SLA came into the field from the service hosting business. At present, the SLAs are generally static, negotiated off-line, and mostly used as a high-level business relationship in the area of content/application outsourcing. But, SLA is an important component of business for any resource sharing systems, and so for a PCU. In a PCU like structure, the only way an SO can control the underlying resource base is through the SLA. Therefore, SLA management in a PCU is not only a business issue, but importantly a technical issue. SLA management in a PCU is much complicated due to (a) multi-owner resources; (b) wide-area deployment; and (c) dynamic behavior of the resources (they can be up and down in a ad-hoc fashion). Until now a little work have been done in the SLA management perspective. Below I am briefly commenting on some works on the SLA management.

Bouillet et al, in their work [BoM02], try to enforce SLA in the managed network, especially by controlling the traffic in the network links. It provides a complete SLA management infrastructure from the off-line design and crafting algorithms to real-time route selection, measurement, and monitoring algorithms. Its monitoring includes *revenue* and *penalty* functions; revenue function provides the incentive to accommodate more traffic in the system, while penalty function discouraging the system violating the SLA.

[FuV02] is a similar work that enforcing revenue and penalty functions, but the resources it manages are the server bandwidths. While [BoM02] makes the dynamic decisions on selecting different routes to manage the bandwidths, [FuV02] implements a *squeeze* algorithm to distribute the available server bandwidth among the assigned services attaining maximum profit depending on the revenues and penalties.

The SNAP protocol and related framework proposed in [CzF02] tries to address the SLA issue in a shared resource infrastructure in a more composite manner. The authors allow the resources to be managed be any computing resources from CPU cycles to bandwidth. The SLA is subdivided into (1) *task service level agreement* that negotiates for the performance of an activity or task; (2) *resource service level agreement* that negotiates for the right to consume a resource; and (3) *binding service level agreement* that negotiates for the mapping of an task to a resource. The composite of these three parts nicely fits into a PCU service infrastructure and virtualization.

Other than these works, the Océano project (section 2.1.3) also can be quoted here for its work on SLA management.

My work does not specifically deal with SLA management, but I assume that there are mechanisms to build the SLA and convert it to input arguments to be fed into my solution heuristics (Chapter 5.2). Once the input parameters are provided according to the SLA, the proposed algorithm guarantees required performance.

2.3 Other Related Works

Here, I discuss some related works that do not directly fall into the PCU vision but share some common research issues such as dynamic resource allocation, load balancing, movement of resource allocation, and virtualization of resources.

2.3.1 WebOS

WebOS [Web02, VaA98] is a mechanism for wide area resource management having some PCU functionality such as dynamic resource allocation, high availability, and capability to find best resources. As the name implies, WebOS provides an illusion of single operating system spanning over wide area resource and hence issuing a job in a WebOS enabled remote machine becomes as easy as issuing the job locally.
The WebOS includes various mechanisms towards its final goal:

Global naming: This component includes algorithms for mapping a service name to servers and load balancing and maintains enough state to perform fail-over if a server becomes unavailable. These operations are performed through *Smart Clients* which are basically Java applets that can be downloaded from service-specific servers.

Wide area file system: *WebFS* is the underlying wide-area file system of WebOS. It is a cache coherent file system, and, for the backward compatibility, uses URLs as the global name space and HTTP for file system transactions.

Security and authentication: WebOS enforces security guarantee and authentication through its security model called CRISIS. It uses *certificates* signed and counter-signed by authorities trusted by both parties in communication.

Process control: It is responsible for the job performance and fairness in executing tasks. *Resource managers* guarantees performance isolation between various jobs in a single machine by creating *virtual machines* for each job with the physical machine.

The paper [VaA98] explains number of potential applications that can be benefitted from WebOS and also a new concept called *rent-a-server*. Rent-a-server is a concept that uses the Smart Clients to process the load and locational informations within transaction headers for finding the best suitable new server to be included to the service cluster when it experience high load condition.

2.3.2 Service Grid

The Service Grid [LeW01] is an online resource allocation approach. It assumes a shared resource pool which it names as Grid, but *this* Grid need not be exactly in the same context discussed in Section 2.1.7. Since I am comparing the performance of my work with that of Service Grid, the operational description of the Service Grid is given below.
Service Grid has the following major entities:

Clients: These are the consumers of the resources available in the Grid.

Servers: These are the resources of the Grid.

Group manager (GM): It is responsible for allocating servers to the client requests; every client is assigned to a GM. In addition to allocating servers to user requests, the GM is responsible for monitoring the service quality and acquiring new resources or releasing surplus resources.

Resource manager (RM): It is responsible for the complete pool of resources; it allocates parts of the resources to GMs upon request.

When a service is instantiated in Serive Grid, the GM requests a server from the RM. The RM assigns a random server from the global pool of resources. Therefore, at the initial state of the service, all the client requests are directed towards that allocated server by the GM. The GM monitors the request bindings for capacity and/or QoS breaches. When the number of requests receiving poor QoS or serviced under capacity exceeded the predefined thresholds, the GM requests an additional server from the RM. When the GM is already allocated with multiple servers for a service, the GM selects the server that can provide the maximum QoS. The GM also monitors the allocated set of servers and releases a server if it is found out to be idling for more than a predefined threshold. The threshold values are set to avoid oscillations.

From the above description, we can see that Service Grid is a fully on-line resource allocation mechanism. In [LeW01], the authors analyze different QoS metrics, but in my simulation, I use the network distance and response time as the QoS metric.

2.3.3 Self-Organizing Network Services

Self-organizing network services [JaJ99] have the ability to replicate and remove instances of themselves based on dynamically fluctuating demand. The framework proposed in this work is called *Sortie*, and it addresses three issues: (1) adaptability to the demand shifts; (2) prevention of oscillation when the usage pattern is not in equilibrium; and (3) maintenance of stability when the demands are in equilibrium.

This work assumes that all the participating nodes are installed with Sortie-installed so that they can make autonomous decisions based on the usage patterns. When a service is primed in the system, it resides in a *home server*. The existence of the service is disseminated to other nodes. If a demand at a point increases for the service, the node at that point considers replicating the service in itself. Similarly the service is removed from the node if the demand is found out to be low . Oscillation between replication and removal is avoided by using different high and low watermarks (thresholds). These watermarks are adjusted dynamically considering the demand patterns for the other services in the system, in anticipation of achieving high resource utilization.

2.3.4 Spawning Networks

Spawning networks [CaK99] are evolved from the programmable network paradigm to completely replace the existing network architectures. The authors found that the existing architectures do not have the flexibility in adapting to new user needs and hinder realization and deployment of new network structures, and therefore proposes spawning networks as the alternative. Spawning networks provides the flexibility of creating virtual network structures incorporating different node types and network types. Further, it provides a mechanism to spawn child networks that inherits the capabilities of the parents, but have their own network structure.

Genesis kernel is the enabling virtual network operating system of spawning networks. It helps automating a virtual network's life cycles: profiling, spawning, and managing. Profiling captures the blueprint of the intended network architecture, spawning actually sets up the topology, and the managing phase supports the virtual network resource management.

2.3.5 Cluster Reserves

Cluster reserve [ArD00] is a mechanism to enforce performance isolation among *service classes* that share resources in a cluster. A service class is a set of requests to a web-server cluster that requires a particular set of resources to be served. Typically, in a web-server cluster which serves multiple service classes, the performance isolation is achieved by allocating separate nodes for different service classes. But, it generally results in poor resource utilization and sometime poor response time since a possible best server is permanently allocated to another service class. Cluster reserves enforce more fine grained performance isolation by using the *resource containers* as the basic building blocks.

Resource container is an operating system concept for enforcing resource reservation and performance isolation for individual processes within a single machine. Cluster reserves just creates such resource containers in each cluster node such that the aggregated resource reservations build up the required cluster-wide resource allocation for each service class. Hence, assuming resource containers provide node-wise performance isolation, the cluster reserves can provide cluster-wise performance isolation. Further, having this type of fine grain resource allocation enables maximum utilization of each resource by giving the ability to tune the node-wise resource consumptions of each service class.

Hence, the problem of creating cluster reserves that gives maximum resource utilization converges into finding out the percentiles of resource containers in each cluster node. The best way to tune these percentiles is to match for the resource utilization patterns in each cluster nodes for each service class. Cluster reserve approaches this problem as a two-phased optimization problem: in the first step, it finds out what is the possible minimum deviations between provided cluster-wide resource allocation and the required allocation; then in the second step, for the minimum deviation obtained in the first step, the algorithms calculates the node-wide resource allocations that minimize the deviations between node-wise usages and node-wide allocations.

The cluster wide allocations are administrative decision, but the node-wide allocations are tuned to match the usage pattern. Therefore, the cluster reserve technique is well suited for geographically distributed clusters where the usage pattern of different service classes in different cluster nodes can significantly vary. The work done in [ArD00] also validates that the cluster reserves provides good performance isolation for various request conditions such as sparse requests and content-based requests.

Each of the above mentioned works resembles to and at the same time differs from my approach in different dimensions. WebOS enabled rent-a-server, Service Grid, and self-organizing networks provide dynamic resource allocation capability, in a more real-time fashion than my approach. However, except the self-organizing networks, others fails to achieve a high resource utilization. Service Grid actually shows that it attain good resource utilization compared to the fully static allocation. But, it also shows that a hybrid approach provides better results. My technique is naturally a hybrid technique (Chapter 4). Spawning networks provides the functionality to spawn child networks, which is analogous to the VC creation (Chapter 4) in my work. However, it only provides the framework for creating such child systems, but does not describe an efficient way of finding resources to be included in the child network. Further, the practical validity of its argument of totally replacing existing network architectures is still a question. Cluster reserves and VC can be considered as similar concepts. However, whereas VC creation always provides a safety margin to handle sudden demand spikes, cluster reserves always tries to find a tight solution.

2.4 Optimization Techniques in Wide-Area Networks

Computer networking has benefited from a number of optimization algorithms in the past such as Dijkstra algorithm and other algorithms for finding maximum flow and minimum spanning tree [OSP83, GoS98]. However, there are relatively a few works found in the literature that use optimization techniques for wide-area resource management. Below I mention some works that apply ideas from the field of operational research to build up efficient wide-area resource management systems.

Qiu, et al. analyze the performance of an optimization algorithm applied to the problem of web server replica placement with other types of algorithms (greedy, random, and “hot spot”) [QiP01]. Here, the placement problem is approached as a *k-median problem*, and the network topology is assumed to be a tree. Even though, the work concludes that the optimization approach does not perform the best, it shows the feasibility of the application of optimization techniques in wide-area networks. Cidon et al also handle a similar problem that place electronic contents efficiently over a distribution tree [CiK01]. Again the network is considered as a tree, and the objective is to minimize the total cost of communication and storage in the network. The work presented in [JaJ01] consider the network as a graph when again handling the same problem of placing mirrors in the network. Here the placement is attacked as a *min-k-center problem* and different approaches are used to solve the problem.

In my work too, an optimization algorithm from the operational research is used to address the key issue. However, the various assumptions made in the above mentioned works make my work differ from them. These works generally put a limit on the number of replicas to be placed; assume servers are capable of serving any load; consider placement criteria controlled by response time; or assuming a tree based network topology. But, the formulation I present in this thesis (Section 5.1) is more flexible for different scenarios with more practical assumptions.

3

Public Computing Utility Model

A PCU model is introduced here as the resource provisioning framework for wide-area services. It aggregates computing resources from different owners and allocates service specific partitions of those resources to different services. These computing resources can be CPU clock cycles, computer memory, storage space, network bandwidth, or any similar computational resource. Individuals, institutions, or companies can share their idling computing resources with the PCU. PCU respects the individual policies which can stipulate sharing pattern (fully dedicated, partially dedicated, or on-demand based), trusted groups, allowed applications, and cost functions.

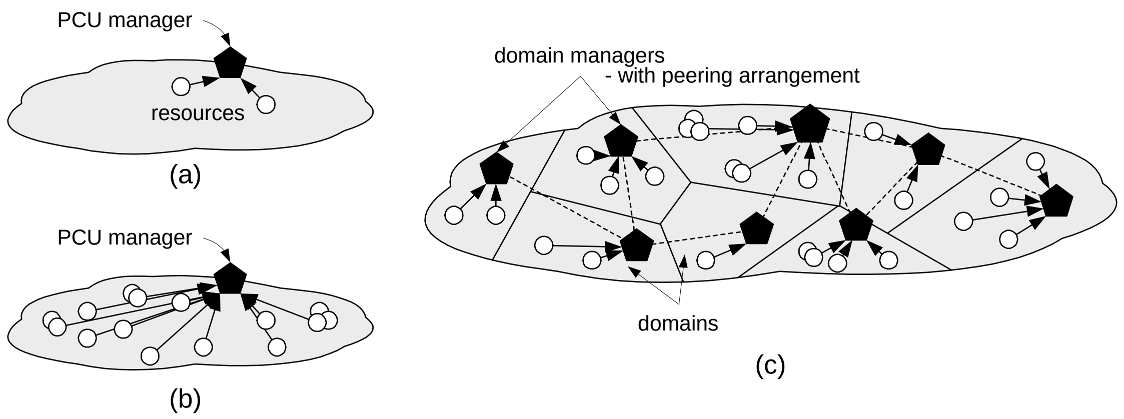

Figure 3.1: Resource management in a PCU

A PCU begins its operations with a resource that acts as the *PCU manager* and accepts the requests for joining from resources (Figure 3.1.a). To address the scalability issue when more resources try to join the PCU from various geographical locations (Figure 3.1.b), the area of PCU administration is divided into multiple *domains* and each domain is instantiated with a *domain manager* (DM) (Figure 3.1.c). Each DM is responsible for the resource management operations within the particular domain. Peering arrangement among the DMs coordinates their operation and with this coordination, the network of DMs can be considered again as a single *PCU manager*, a *virtualization* that is used for the rest of this thesis. The detailed description of the PCU manager operations are out of the scope of this thesis, but it is assumed that there are mechanisms and controlling protocols that provide the PCU manager the control over the participating resources.

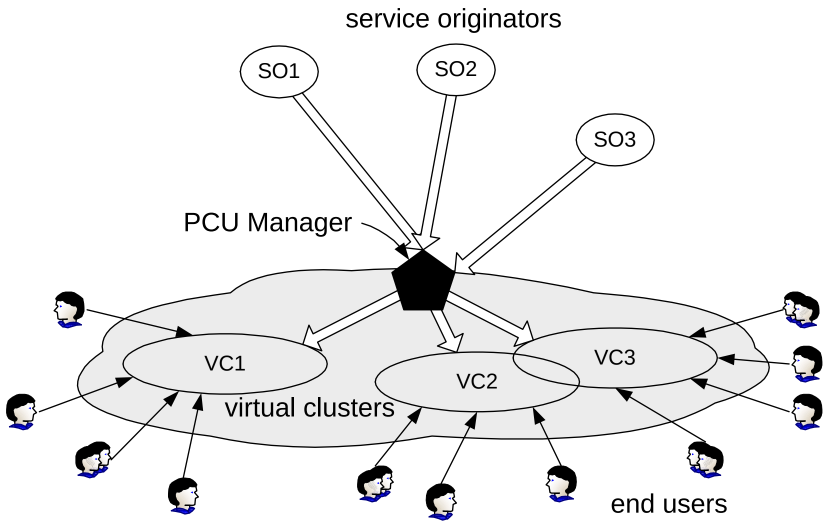

Figure 3.2: An overall picture of PCU operation

The customers of a PCU are generally *service originators* (SOs) who are in need of cost effective and QoS guaranteed wide-area resource deployment. The SOs will contact the PCU manager specifying its requirements and the manager will create resource partitions for these SOs which I name as *virtual clusters* (VC). VCs are service specific, secure collections of resources with minimized cost and maximized QoS. The concept of VC is one of the unique aspects of my work and the detailed description of the VC concept is given in Chapter 4. Despite the fact that VCs can physically overlap, the performance isolation is to be ensured by low level scheduling schemes. The business and service relations between SOs and the PCU are regulated by the negotiated *service level agreements*

(SLA). The ultimate consumers of the resources are the *end users* of the services. For example, an on-line movie streaming company can request resources from the PCU and when the resources are allocated and the contents (movies) are downloaded on to the allocated resources, it will be the customers of the company who watch the movies online are the ultimate consumers (end users) of the resources. In some cases, for example in a distributed computing application, the SO itself can be the end user. Figure 3.2 illustrates this overall operation of a PCU.

A implementation of such a PCU faces a number of issues: it has to provide protocols for resource registration, dissemination, and discovery; a policy management should be devised to respect the rights of the owners on their resources; an SLA management sub-structure is to be built up for SLA creation, monitoring, and modifications; heuristics should be developed for cost-effective and QoS guaranteed resource allocation; and low level scheduling schemes should be enforced to provide low level performance isolation.

The rest of this thesis analyses the means of enforcing QoS guarantee in this PCU model. Since VC is the core idea behind guaranteeing QoS for services, the VC creation process is analyzed in detail and heuristics for the creation process and QoS aware load balancing are developed.

4

Virtual Clusters

Traditionally, a cluster is located at a single site and is designed to handle the expected peak demand for the hosted application. This makes clusters costly because components such as storage, processing, and network have to be provisioned for the highest demand level. Consequently, at average load levels, the cluster resources will be underutilized. The *virtual cluster* (VC) concept introduced in this thesis is an effort to increase the cost effectiveness of the clusters by boosting the overall utilization.

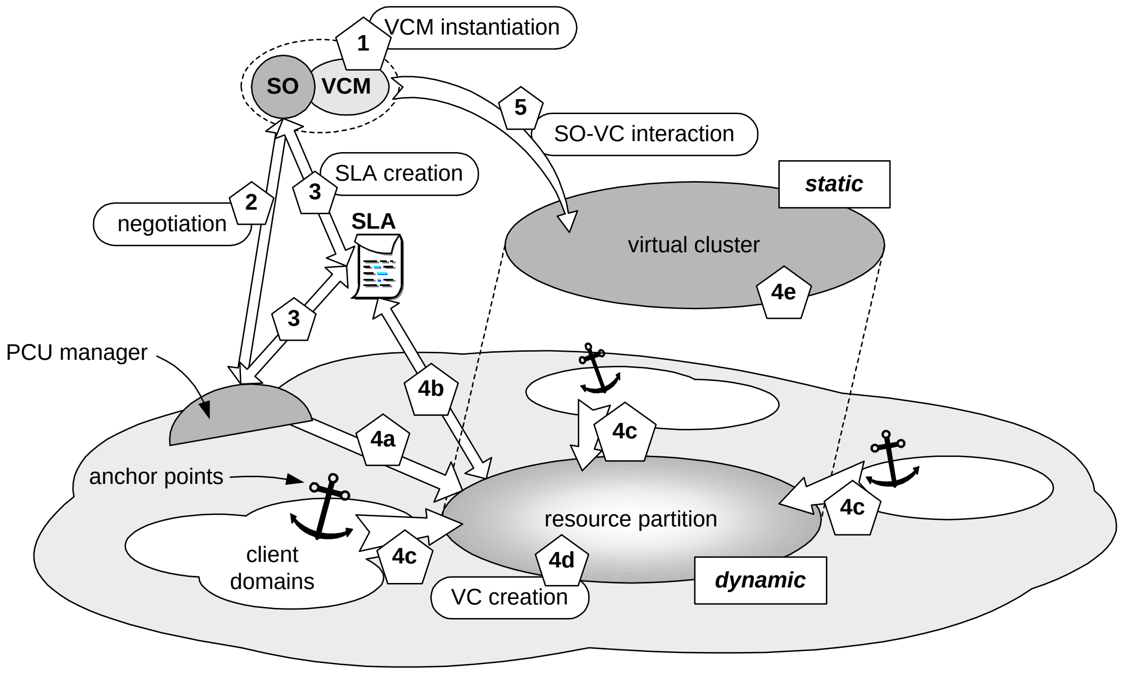

Figure 4.1: VC creation process on a PCU system.

As mentioned previously, the PCU aggregates all available resources into a global resource pool and then allocates partitions of them to different applications. These resource partitions are called VCs as illustrated in Figure 4.1. Because PCU manages the global resource pool from which a VC is allocated resources, a VC is highly flexibly provisioned to handle spikes in demand. Unless all VCs that are handled by a PCU experience demand spikes at the same time, the PCU can dynamically change the resource provisioning level of a VC to handle the changing requirements. This enables a VC that is initially allocated for an estimated demand condition to grow, shrink, or move at run time to accommodate the changing demand.

To support the VC creation process, I introduce a notion called the *anchor points* (APs) (see Figure 4.1). The APs represent *centroids* of the demand distribution in a geographical area. For example, the gateway of a university domain can represent the demand originating from the university. Similarly, a gateway of an *Internet service provider* (ISP) can represent the demand originating from the clients connecting via the ISP. In addition to the locational attributes, APs have other attributes such as demand intensities and type. The VC can be designed with sufficient resources such that the APs are covered at or above a predefined level of service or the AP coverage QoS is maximized.

A VC is created by the PCU for a *service originator* (SO) and is managed by the PCU. The SO uses a proxy called the *VC manager* (VCM) to communicate with the PCU its requirements in the form of a *VC specification* (VCS) that contains a list of APs, acceptable cost, tolerance for variation of delivered service, and others such as access and management requirements. As shown in process 1 in Figure 4.1, the VCM is co-located with the SO. The PCU negotiates with the VCM (process 2 in Figure 4.1) and comes up with a *service level agreement* (SLA). Similar to the SLA introduced in [ApF01], this SLA describes the agreed performance level, tolerance, implementation cost, and other metrics of performance. The PCU creates a VC that satisfies the SLA. As shown in Section 5.1, the VC creation problem can be formulated as an optimization problem, where the cost of creating a VC is minimized while a measure of delivered QoS is maximized. The VC creation is controlled by the PCU manager (process 4a), but it is bound to the constraints from the SLA (process 4b) and anchor point arrangements (process 4c). It results in a collection of resources that can be dynamic (process 4d) with varying demand conditions at APs, but a static virtualization (process 4e) is given to VCM to facilitate its interactions (process 5). This virtualization isolates the complex resource management from the high level service management simplifying the operation of the SO.

At VC creation, each AP is assigned a particular “home server” that is capable of handling nominal demand presented by the AP. However, depending on the conditions present in the SLA the VC can be subjected to reallocations that can be triggered periodically or as demands change by predefined thresholds. Therefore, a VC can be considered as a semi-dynamic resource allocation mechanism.

Because the VC creation process attempts to minimize the set of resources that participate in a given VC configuration, the resultant VC configuration is susceptible overload situations when the demands for the hosted services increase. Although the VC can reorganize itself for the new demand conditions by undergoing reallocations, it will be costly to undergo such reallocations for every significant variation in the VC demand. Further, the delays in obtaining the benefits out of the reallocations make it less attractive.

Therefore, to handle the variations in demand from the expected value that was specified in the VCS, I introduce a notion called the *overload partitions* (OLPs) to handle demand spikes. The OLP can be considered as a protection layer of the actual VC. In the creation process, at first a resource partition is created for the service considering an increased demand conditions. Then the VC nodes are extracted by rerunning the allocation process (for the nominal demand conditions) considering the resource partition obtained from the first step as the global resource pool. The resources outside the VC set, but inside the resource collection from the first step are allocated as the OLP nodes. This ensures that the resource allocated for the VC is contained within the previous set of resources, which keeps the combined cost of VC plus OLP minimum. However, when the VC is created, the resources are dedicated for the exclusive use of the service hosted by the VC, while the resource in the overload partition are shared with other VCs in a best-effort manner. VC nodes and OLP nodes are primed with the appropriate system and application softwares at startup. It should be noted that resources can mean virtualized partitions of a single physical resource. Therefore, even with dedicated allocations a physical resource can be hosting components that belong to different applications.

5

Heuristics for Enforcing the SLAs in PCUs

This chapter develops an optimization model for the VC creation process explained in Chapter 4. The ultimate aim of this optimization model is to mathematically formulate the trade-off between achieving the best QoS and reducing the system cost. Section 5.1 develops the optimization model of a VC creation and analyzes the practical validity of such model. The solution methods of the developed model is analyzed and a heuristic that solves this model is explained in Section 5.2. An example of this heuristic applied is illustrated in Section 5.3. In Section 5.4, different scheduling schemes are discussed in which load balancing and on-line QoS monitoring are given importance.

5.1 Mathematical Formulation

I model the VC creation problem as a *facility location problem* (FLP) [Das95]. In general, a FLP is concerned with optimally placing a number of facilities (equivalent to serving resources) to cover demands at a predefined set of points (equivalent to APs in VCs). There exist several variants of FLPs that differ on the optimization criteria. In this thesis, I develop a model for VC creation that is based on *capacitated fixed-charge location problem* (CFCLP). The CFCLP assumes that placing a facility at a given candidate location incurs a fixed cost and each facility has a limited capacity of covering the demands.

When a server is initiated into a VC, the application or operating system needs to be installed and the initial data should be loaded from the appropriate data sources. This implies a fixed or known cost for priming a server into a VC. Servers that are part of a given VC have a fixed capacity and are capable of handling only a fixed number of concurrent requests at any given time. Therefore, the capacity of the serving nodes should be considered while allocating the serving resources to cover the demand. Further, the serving resources bound by the the VC creation process to a particular AP have to provide the best coverage for the demand originating from that AP. This is achieved by minimizing the covering cost in a CFCLP model.

I model my system as a graph with the servers and APs as the nodes and the network links as the edges. Let V_c be the set of server nodes that can be part of a VC and V_a be the set of APs. In addition, I define the following parameters:

d_{ij}	–	distance/delay between nodes i and j
b_{ij}	–	bandwidth between nodes i and j
m_j	–	capability of server j that denotes attributes such as computing power and storage power
k_j	–	capacity of server j that denotes maximum number of concurrent requests it can handle
c_{ij}	–	cost of covering node i in V_a by node j in V_c as defined below
f_j	–	fixed cost for locating a facility at node j
h_i	–	demand at node i
M	–	required capability for a node to participate in a VC
B	–	intra-VC bandwidth

The covering cost c_{ij} is defined as

$$c_{ij} = \alpha d_{ij} + \frac{\beta}{b_{ij}}$$

where α and β are VC specific and are given by its VCS. For example, by making $\alpha = 0$, we can ignore the inter-node delay and just consider inter-node bandwidth. The covering cost c_{ij} can be considered as the inverse of the delivered QoS with a smaller value of c_{ij} implying a higher value for the delivered QoS.

The decision variables are defined as follows:

$$s_j = \begin{cases} 1 & \text{if node } j \text{ is part of the VC} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if node } j \in V_c \text{ covers node } i \in V_a \\ 0 & \text{otherwise} \end{cases}$$

With these parameters, the optimization problem for creating a VC becomes (with $i \in V_a$ and $j, k \in V_c$):

$$\text{minimize} \quad \sum_j f_j s_j + \sum_i \sum_j h_i c_{ij} a_{ij} \tag{5.1}$$

Subject to

$$m_j \geq M s_j \quad \forall j \tag{5.2}$$

$$s_j + s_k \leq 1 \quad \forall \{j, k | b_{jk} < B\} \tag{5.3}$$

$$a_{ij} \leq s_j \quad \forall i, j \tag{5.4}$$

$$\sum_j a_{ij} \geq 1 \quad \forall i \tag{5.5}$$

$$\sum_i h_i a_{ij} \leq k_j s_j \quad \forall j \tag{5.6}$$

$$s_j = 0, 1 \tag{5.7}$$

$$a_{ij} = 0, 1 \tag{5.8}$$

The objective function given by Equation (5.1) has two parts: first part is based on the number of servers in the VC and the second part is the total demand weighted cost ($h_i c_{ij}$). By minimizing the objective function, the first part reduces the total priming cost of the VC, while the second part reducing the cost of allocation and thus increasing delivered QoS. The demand weighted cost given in the second part is used to place severs of a VC closer to APs that have higher demand. This will reduce the percentage of possible violation of required performance level.

The constraint in Equation (5.2) restricts the VC membership to nodes with given capabilities (e.g., with given CPU, memory, and disk capacities). This constraint can be applied off-line to produce a candidate set of nodes to which the CFCLP can be applied. The intra-VC bandwidth requirement is defined by the constraint in Equation (5.3). The constraint in Equation (5.4) denotes that an AP is covered only by a single server that is part of VC (which is the “home server” of the AP). The constraint in Equation (5.5) ensures that every AP is covered by at least one node in the VC. Finally, the constraint in Equation (5.6) ensures that the number of APs covered by a particular node is within its capacity.

5.1.1 Practical Validity of the Formulation

In this formulation, I assume that the QoS of a service is decided by the network bandwidth and the delay. This assumption is applicable in most cases where the QoS is measured by the response time and the throughput. Even in other cases, this formulation can be still used, but the definition of the c_{ij} will have to be modified to suit the scenario at hand. Further, the above formulation tries to minimize the objective function value and hence, maximize the delivered QoS, but does not guarantee a given QoS level. This becomes an issue when a SO specifies a value for QoS in its VCS. Nevertheless, it can be addressed by preprocessing the c_{ij} matrix such that, the elements c_{ij} that fail to meet the required QoS bound are replaced by a very large number. This transformation of the c_{ij} values will make the solution to the above optimization converge to a solution that satisfies the stipulated bound.

In general, a CFCLP is formulated with $a_{ij} \geq 0$ instead of the constraint in Equation (5.8). When a_{ij} is defined as such, the demand at an AP is *splittable* and can be served by multiple servers. In this case, fractions of server capacities can be used to serve portions of AP demands, resulting in a tighter solution. However, by making a_{ij} a 0-1 variable, each AP is provided a “home server” that naturally provides a safety margin on the available resource capacity. This is because there will be some fraction of total capacity in each server that is not allocated to any AP.

It is further assumed that there exists mechanisms to translate resource capability attributes such as CPU power, memory, and storage into numbers (m_j) to be fed into the formulation and to disseminate node and link informations (such as d_{ij} and b_{ij} matrices) to the central node where the problem is formulated.

5.2 Solution Methods and the Drop Heuristic

Solving CFCLPs is known to be NP-hard [KoP00, Das95]. Several algorithms have been proposed in the literature including local search and Lagrangian relaxation that provide near-optimal solutions [KoP00, Das95]. In this study, my objective is to develop fast solution procedures that can be implemented in a distributed way. Although a VC is not created and deleted at very short time intervals, a fast scheme for VC creation is essential because a PCU can be in a continuous state of VC creation, deletion, and modification in response to requests that arrive at run time. This thesis examines a centralized version of the heuristic as a first step towards solving the VC creation and management problem. We use a modified version of the *drop heuristic* that is used in [Das95] for solving *uncapacitated fixed charged location problems* (UFCLPs). It is a heuristic that is based on the local search paradigm [KoP00].

This heuristic works in two phases. In the first phase, a feasible configuration is found for a VC by considering all available servers that have the required capability. In this allocation, each AP is assigned to a server using a greedy strategy that considers the covering cost c_{ij}, i.e. APs are assigned to servers that are "close" to them by means of c_{ij}. Once the feasible allocation is found, the second phase is initiated to drop excess serving capacity and reduce the cost of the VC.

The dropping phase tentatively drops an already selected server and computes the change in the overall objective function value. The objective function value can increase or decrease. It may decrease because as servers are dropped the first part of the objective function in Equation (5.1) decreases. On the other hand, it may increase because the covering cost given by the second part of the objective function in Equation (5.1) can increase as some APs loose its closest server. Having computed the changes in the value of the objective function resulting from dropping different servers, the server with the largest cost reduction is marked for permanent deletion from the VC. However, before this server is permanently dropped, the APs covered by the server need to be reallocated to other servers. This process can result in capacity overflows at other servers. These capacity overflows are handled by recursively invoking the reallocation procedure to find a viable assignment of APs to servers. If such an assignment is found infeasible or more expensive in terms of the

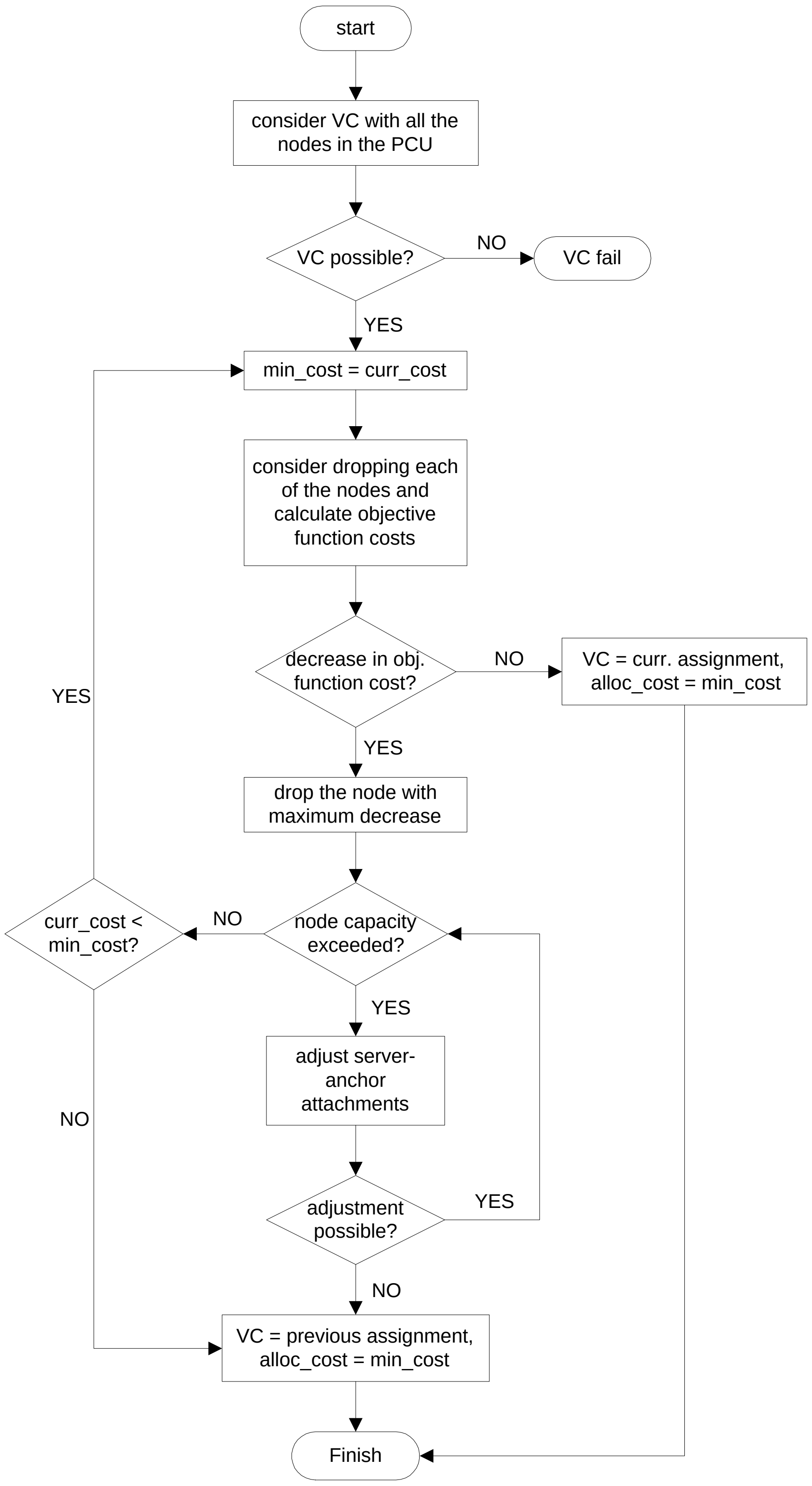
start
consider VC with all the nodes in the PCU
VC possible?
NO
VC fail
YES
min_cost = curr_cost
consider dropping each of the nodes and calculate objective function costs
decrease in obj. function cost?
NO
VC = curr. assignment, alloc_cost = min_cost
YES
drop the node with maximum decrease
YES
curr_cost < min_cost?
NO
node capacity exceeded?
YES
adjust server-anchor attachments
NO
adjustment possible?
YES
NO
VC = previous assignment, alloc_cost = min_cost
Finish

Figure 5.1: The modified drop heuristic.

objective function, then the server marked for deletion is reinstated into the VC and the VC is formed with currently allocated servers. If that assignment is found feasible, the marked server is dropped from VC, and then the VC creation process will go through another drop phase. This iteration will continue until no server is found feasible to be dropped out. The flowchart in Figure 5.1 illustrates this process.

The capability constraint in Equation (5.2) is applied to form a candidate set of servers before the drop heuristic is invoked. Similarly, the intra-bandwidth constraint in Equation (5.3) is applied to produce a connected set of resources with the given bandwidth before the allocation heuristic is applied. The application of this constraint can yield several connected components and the heuristic will be applied to each of these components. The minimum cost VC among all the different connected components is taken as the final VC.

The drop heuristic is a centralized solution method. The practical applicability of such a centralized solution in a wide-area system like PCU is questionable. However, this heuristics can be developed into a distributed form, if we consider the domains (see Chapter 3) as the high level, abstract resources to apply the heuristic. The aggregated capacity of the domains can be used as the formulation parameters. When the high level VC is constructed that is made up of domains, the same heuristic can be applied in each selected domains to select suitable individual resources. Depending on the levels in the domain hierarchy, the heuristic can be recursively applied until the individual resources that participating the VC are found.

5.3 VC Creation – An Example

Here I illustrate an example of a VC creation process on a 56-node network. The interconnection topologies among the resources were generated using an Internet topology generator called the Tiers [Doa96]. The Tiers creates a network with WAN-MAN-LAN topology. Tiers-created network is basically a text file describing node locations (X-Y coordinates), connectivity details, and the delay and bandwidth of the links. The network initialization code written in Matlab accepts this network information file and places the APs at randomly chosen LAN nodes and assume the rest of the nodes to be possible candidates for the location of serving resources. Also it assigns random demands for the APs and random

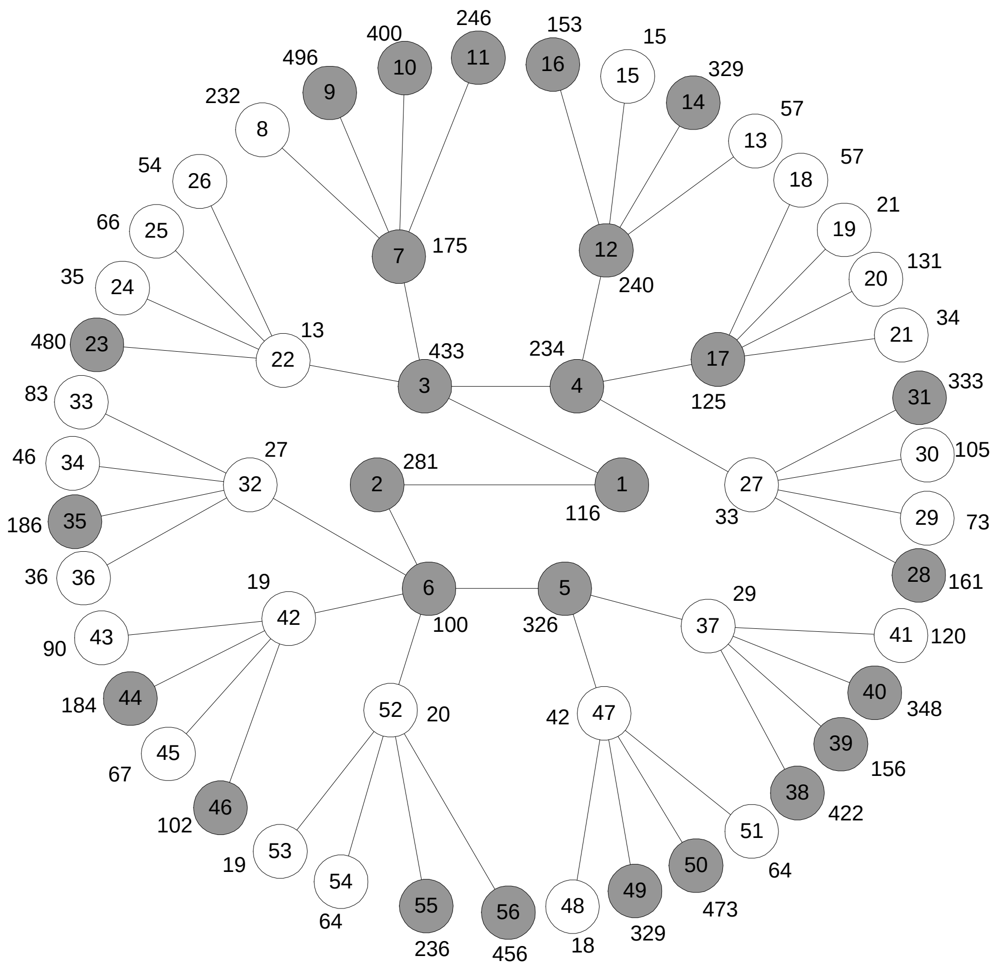

Figure 5.2: The initial network.

capacities and capabilities for the servers and then generates the cost matrix ($h_i c_{ij}$) for the network. The output of this initialization is again an text file and Figure 5.2 illustrates this output. The gray nodes are servers and others are APs and their capacities and demands are given next to them.

The VC creation code, also written in Matlab, contains codes for creating connected node sets (Section 5.2) and for implementing Drop heuristic. The VC creation with OLP is carried out in two stages: in stage one, the VC creation code is run with the demand values of the APs increased by the factor of 1.5; in the second stage, candidate server set is reduced

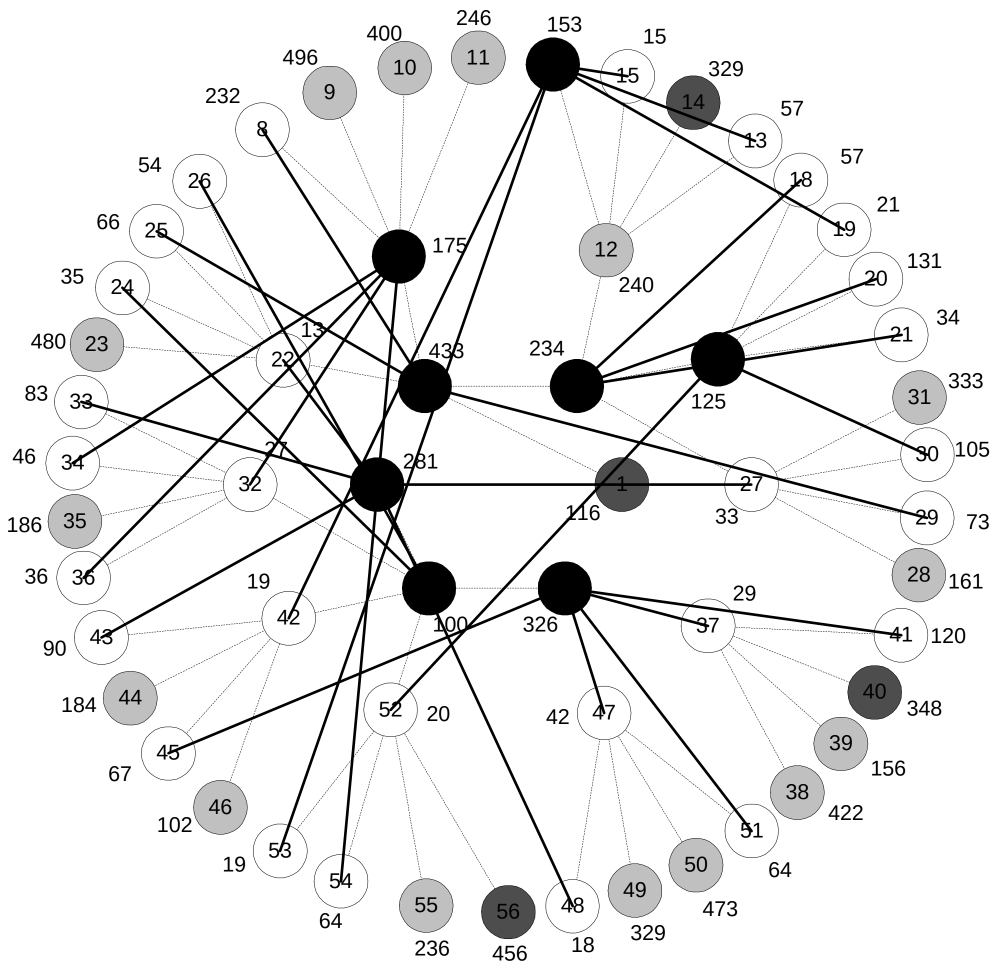

Figure 5.3: The created VC with OLP

to the servers output from the first stage, demands are set back to the nominal values, and the program is rerun. The servers output in the second stage is considered as the VC nodes and the rest of the servers from the first stage as the OLP nodes. The program output is the input text file appended with the list of the selected VC nodes and preferred server list as described in 5.4. The output files from both stages are used as the inputs for the simulation study (Section 6.2). Figure 5.3 shows the created VC with OLP in the network with the "home server"-AP attachments. The black nodes are the dedicated VC nodes and the dark gray nodes are OLP nodes. The preferred list is shown in Table 5.1.

8	13	15	18	19	20	21	22	24	25	26	27	29	39	32	33	34	36	37	41	42	43	45	47	48	51	52	53	54
3	16	16	4	16	4	4	2	6	3	6	2	3	17	7	2	7	7	5	5	16	2	5	5	2	5	17	16	7
40	6	5	14	14	4	14	2	56	56	56	2	16	3	3	3	17	56	2	5	7	2	7	3	7	5	3	56	56
7	4	4	17	17	17	17	3	3	3	3	4	4	4	6	6	6	6	5	40	6	6	6	5	5	6	6	6	6
3	17	17	4	4	3	4	1	1	1	1	17	17	17	2	2	2	2	40	6	2	5	2	6	6	2	2	2	2
1	3	3	3	3	1	3	7	7	7	7	3	3	1	5	5	5	5	6	2	5	1	5	2	2	40	5	5	5
2	1	1	1	1	7	1	6	2	2	2	1	1	7	1	1	1	1	1	1	1	3	1	40	40	1	1	1	1
6	7	7	7	7	2	7	5	6	6	6	7	7	2	7	7	3	3	3	3	3	7	3	1	1	3	7	3	3
5	16	16	2	2	6	2	4	5	5	5	6	2	6	40	40	7	7	7	7	40	40	40	7	3	7	40	7	7
4	2	2	6	6	5	6	17	4	4	4	5	6	5	4	4	40	40	4	4	4	4	4	4	4	4	4	40	40
17	5	6	5	5	16	5	40	17	17	17	16	5	16	17	17	4	4	17	17	17	17	17	17	17	17	17	4	4
16	14	14	16	16	40	16	16	40	40	40	40	40	40	56	56	56	17	56	56	56	56	56	56	56	56	56	17	17
56	40	40	40	40	14	40	56	16	16	16	14	14	14	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
14	56	56	56	56	56	56	14	14	14	14	56	56	56	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14

Table 5.1: The preferred server list headed by the “home servers”.

5.4 Request Scheduling

In a VC a probing order is specified at the creation process for every AP by a preferred list of servers that is headed by the "home server." The preferred list is formed based on the values for $h_i c_{ij}$. The home server will be the destination of the requests from the AP for under loaded conditions. A request scheduling strategy has to make two different types of decisions: (a) determining the best server when capacity is available on different servers and (b) determining the server to queue when capacity is exceeded on all servers. The requests are scheduled using two different strategies: (a) greedy server acquisition with home server queuing, and (b) probabilistic server acquisition with minimum loaded server queuing.

In the greedy approach, requests from an AP are sent to the home server unless the home server is overloaded. If the home server is overloaded, the servers in the preference list is sequentially examined and the search is restricted to the servers that provide a $h_i c_{ij}$ cost smaller than or equal to the maximum tolerable service cost. (This confirms the PCU operation always abide by the SLA). When all servers are found to be serving above their capacities, the request is queued at the home server. Our rationale behind the home server queuing is that because the home server is determined by the off-line assignment process to be the "best" server to handle the bulk of the load from the AP, it should be the single best server to handle excess demand. Besides, queuing at the home server should reduce the situations where a server is inundated by requests from APs that are not primarily handled by this server.

In the probabilistic approach, I use an *escape probability*, p_e, with which a request *escapes* from being assigned to a particular server in the sequential search through the server list. p_e is defined as

$$p_e = \begin{cases} \frac{C_t - c_a}{C_t} & \text{for a server that belong to the dedicated partition} \\ \frac{C_t - c_a}{C_t \cdot p_e^{hs}} & \text{for a server that only belongs to the OLP} \end{cases} \tag{5.9}$$

Where C_t is the total capacity of a server measured in terms of the total number of concurrent requests it can handle; c_a is the active capacity (i.e., capacity that is available at any given time) of the server, and p_e^{hs} is the escape probability at the home server for the particular AP under consideration. $c_a \in [0, C_t]$ is equal to C_t at no-load condition and 0 at overloaded conditions.

With the escape probability defined as above, as the active capacity decreases the requests escape more frequently. This attempts to prevent the situation where a server is fully loaded while the rest of the servers are running below capacity. In other words, it encourages the requests to diffuse to other servers without concentrating on a particular server. In this scheduling process, once a request escapes through the full list of servers in the preference list, it is assigned to a server with minimum load (maximum active capacity or minimum queue length). As in the greedy approach, QoS bounds are considered here too at the time of scheduling requests.

As explained in Chapter 4, the resource allocation for a service is made of two parts: the dedicated VC nodes and the non-dedicated OLP nodes. By including the escape probability at the home server in the denominator of the expression for the escape probability at a OLP server, I bias the requests to escape the OLP servers more until the dedicated VC nodes are almost fully loaded (i.e., the selection process favors to assign the request to a server in the dedicated partition). This motivation stems from the fact that the OLP resources should be used sparingly and not used excessively under nominal loading conditions, because performance isolation is not guaranteed at these OLP nodes.

When a VC without OLP is considered (as I did in my simulation study), only the first part of Equation 5.9 is used for scheduling requests.

6

Simulation, Results, and Discussions

This chapter explains the simulation study I carried out to evaluate the performance of different VC configurations and two different scheduling schemes developed in Section 5.4. Also the performance of the VC is compared with an existing on demand cluster creation mechanism called the Service Grid described in Section 2.3.2. The Section 6.1 discusses the assumptions I made in building up the simulation. Section 6.2 describes the overall arrangement of the simulation study. The load prediction mechanism used in the simulation is explained in Section 6.3, and the obtained results and detailed discussion are given in Section 6.4.

6.1 Assumptions

In response to the demands specified at the APs by the SO, the PCU creates a VC. This VC is essentially a set of compute resources that are interconnected by networks that have at least the specified amount of available bandwidth. These resources can be used to host different types of applications such as high performance computing, streaming media applications, distribution of documents, and transactional applications with multi-tier configurations.

For this study, I assume a web document serving application such as the deployment of an E-news site. The E-news site needs resources such that its popular documents can be served from the "edge" of the network. The interesting edges of the network are specified by the SO using a set of APs. The demand for the services arrive in the form of requests for documents at various points on the network. These requests are routed to the closest serving resources and the servers answer the requests with the appropriate documents. We assume that the computation time at the server is proportional to the request size and there is sufficient bandwidth between the serving resource and client such that no performance bottlenecks are encountered in the network.

In the simulations, I assume that the requests arrive at the APs with exponentially distributed inter-arrival times and have request lengths (or size) on a Gamma distribution. The average demand for service (request intensity) is defined as the mean request length divided by the mean request interval. For different APs, the mean request intervals vary in the range $1 \ldots 10$ seconds and the mean request lengths vary in the range $100 \ldots 500$ seconds. As described in Sections 5.1 and 5.2, the VC can be created with a specific intra-VC bandwidth requirement. Because the Service Grid (Section 2.3.2) that is compared with the VC does not allow the specification of internal connectivity constraints, the minimum intra-VC bandwidth requirement is set to 0.

Also the demand at the APs are assumed to be steady over the run time of the simulation. It may be unrealistic but this type of steady loading is assumed to test the system at its extreme capabilities. For example, when the system is to be tested against overloaded condition, with a realistic type of fluctuating demands the system can not be pushed to its overloaded limits. Following the same line of argument, it is also assumed that each server has infinite-sized buffer to queue the requests. If the buffer sizes are limited at servers, then the run time of the system also will have its effect on the results. Further, when the fluctuating demands also are considered with limited buffer sizes at the servers, the length of overloaded periods also will become a decisive factor in the results. The purpose of this simulation study presented here is not to address such complicated situations but to evaluate the fundamental performance of the proposed system.

6.2 Simulation Setup

The simulation study I carried out is of two phases: in the first phase the centralized off-line drop heuristic is applied to create a VC according to the assumed network, demand, and server attributes; and the second phase simulates the setup by creating the working network model, generating the load and evaluating the performance of the created VC.

The implementation of the first phase is explained in the Section 5.3. For the second phase a discrete event simulator called Parsec [Par02] is used. In this phase a *manager* node is introduced to coordinate the scheduling of requests. The APs send the requests to the manager which implements the stipulated scheduling scheme, selects a server for serving the request, and informs the AP the selected server. Then the AP contacts the server with the request. The manager is established in a server node that is not participating in VC or OLP. However, in real cases, the manager can be distributed in each AP.

The simulator is built up of *entities* which are the templates of different types of nodes in the network – AP, server, and manager. The entity (template) code provides the functionalities of the respective type, such as request generation, communication, scheduling, and request processing. Multiple nodes can be generated from an entity. The communication between entities are in the form of application level *messages*.

This simulation study creates and analyses the performance of the following cases:

1. VC without OLP
    (a) with greedy scheduling
    (b) with probabilistic scheduling
2. VC with OLP
    (a) with probabilistic scheduling

The load conditions for which the VC is created is considered as the nominal load and the performance are analyzed for underload and overload conditions. Also the following styles of load variations are considered in this simulation:

1. Demands in all the APs are increased *or* decreased by the same factor.
2. The demand change factors for APs are *randomly* selected from a "window", so that some of them are above nominal values and others are below. The center of the window is shifted moving the overall loading pattern of the system towards underloaded or overloaded condition.

3. The demands in a small percentage of the APs are increased by a particular factor and the demands in the rest are reduced so that the whole system is at the nominal load conditions.

These different loading schemes are used in anticipation of representing different practical loading scenarios. Demands of the APs are varied either by changing the request intervals or the request lengths.

The performance of the different VC setups for different loading conditions are compared to the performance of the Service Grid. The Service Grid is created with a similar simulator with same APs, servers, and demand and capacity attributes. But the manager entity is removed and two other entities are introduced to represent the GM and RM of the Service Grid (see Section 2.3.2). The scheduling policy used with Service Grid is always greedy assigning server for requests, that gives minimum queue weighted cost (queue_length$\times h_i c_{ij}$).

A capacity prediction scheme explained in the next section is used to predict the capacity of a server at any instance, that assists the scheduling schemes.

6.3 Load Prediction and Active Capacity

A server participating in a VC can receive requests from multiple APs. Therefore, the loading condition of a server is best estimated by the server itself because there does not exist any other single entity that is fully cognizant of the server loading. Even the centralized scheduling manager can not derive the loading conditions of the servers, due to the network delays in the request assignments and the variation of time each request spends on a server. If another entity is interested in the status, the server should update the entity with its status. Because the server's status can change dynamically the updation process could entail significant overhead to maintain accurate state information at a given entity. Further due to communication delay, accuracy of the state information will be diminished due to the staleness in the data values. In this study, I use periodic updates coupled with simple prediction algorithms at the entities to maintain the state at the remote entities. This procedure is used to maintain information regarding the queue lengths (in case of overloaded servers) and active serving capacity (in case of underloaded servers).

The simple prediction algorithm works as follows: at first it calculates the mean μ of the previously obtained five updates with the equation:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{6.1}$$

Where x_i is the previous updates and n is the number of previous updates which is 5 in our case. Then variance σ of the data is defined as:

$$\sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{6.2}$$

Then the autocorrelation r is calculated as:

$$r = \begin{cases} 0 & \text{If } \sigma = 0 \\ \frac{1}{\sigma(n-1)}\sum_{i=1}^{n-1}(x_i - \mu)(x_{i+1} - \mu) & \text{Otherwise} \end{cases} \tag{6.3}$$

Now, the status of the server is predicted as:

$$x_{n+1} = \mu + r(x_n - \mu) \tag{6.4}$$

6.4 Results and Discussions

This simulation study is conducted on a network topology with 56 nodes, where 29 of them are APs and rest are candidate servers. The VC created for APs has 8 servers in the dedicated partition and 4 more servers in the OLP. The simulations are carried out for a fixed duration of time.

Various performance metrics are used in this study. One of the metrics, *response time* (T_r) is defined as the time taken from the origination of the request at a client to the receipt of the service completion acknowledgment at the client. The response time is comprised of *binding time*, *communication time* (T_c), and *service time*. The binding time is the time taken for a request to be assigned to a server. The communication time is the time spent on transportation and waiting in the queues. The service time is the time the server spends on processing the request. The request length has a direct impact on the response time, because it determines the service time. To isolate this effect, I consider the communication time as a performance metric as well.

In the Service Grid and VC, when a request is assigned to a server that is exceeded its capacity, the request is queued at the server. When a request is queued, its communication time increases with a corresponding increase in the response time. This happens despite the fact that the request may be assigned to a server with a $h_i c_{ij}$ value that satisfies the QoS and hence, the actual QoS will be poor. Therefore, the *percentage of queued requests* (P_q) is considered as another performance metric. With higher the value of P_q, poorer performance with respect to response time can be expected.

In the Service Grid, requests are assigned to the server that provides the best QoS value among, the servers the GM currently manages. But, even this best may fail to satisfy the required QoS. In this study, we assume the QoS value the VC guarantees as the required QoS and uses the *percentage of QoS failed assignments* (P_{QoS}) in the Service Grid as another performance metric. Because the Service Grid is able to utilize as much serving resource as needed and available and the VC is restricted to the chosen set, for a fair comparison, in addition to the above metrics the number of servers used should be considered as well. The tables present two values for the number of servers used: one is the total number of servers used in the system and the other is the maximum number of servers that were being used at a particular instance. All the values presented in the rows of the tables are averages of at least five runs.

Normalized request interval	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_{QoS} (%)	P_q (%)	Total servers used	Total servers at a time
1.5	0.7	462	1.319	283 972	0.01	10.9	15	12
1.1	0.9	461	1.350	398 176	0.01	13.4	21	18
1	1	462	1.359	404 119	0.01	13.6	21	19
0.9	1.1	463	1.914	528 964	0.01	28.4	26	26
0.7	1.5	462	2.091	670 002	0.05	34.1	26	26

Table 6.1: Performance of the Service Grid with mean request interval.

Tables 6.1, 6.2, and 6.3 show the performances of the Service Grid and the VC without

Normalized request interval	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1.5	0.7	461	0.645	267 676	0.00
1.1	0.9	461	0.672	370 741	1.05
1	1	461	0.683	376 725	1.32
0.9	1.1	6 867	6 406	486 727	99.2
0.7	1.5	20 552	20 091	583 762	99.6

Table 6.2: Performance of the VC without OLP with mean request interval using greedy server acquisition.

Normalized request interval	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1.5	0.7	461	0.643	267 996	0
1.1	0.9	461	0.655	370 873	0
1	1	460	0.655	376 761	0.00
0.9	1.1	9 452	8 991	486 472	99.4
0.7	1.5	20 058	23 597	583 872	99.6

Table 6.3: Performance of the VC without OLP with mean request interval using probabilistic server acquisition.

OLP with varying mean request intervals (normalized). Table 6.2 shows the results of using the greedy server acquisition strategy in VCs while Table 6.3 shows the results of using the probabilistic server acquisition strategy. The normalized request interval of 1 denotes the nominal load condition for which the VC was created. The smaller the mean request interval, the higher the load as tabulated in the normalized load column.

From these tables we can make following observations: (a) the performance of the Service Grid in terms of P_{QoS} and P_q are fairly steady; (b) the VC outperforms Service Grid at underload conditions, while the reverse is true at overload conditions in terms of P_q; (c) Service Grid consumes more than double the number of serving resources compared to VC; and (d) even with that much resources consumed, a fraction of requests are always served with poor QoS.

From Tables 6.2 and 6.3, we can observe that the probabilistic server acquisition performs better than the greedy server acquisition process. Further, with probabilistic server acquisition the VC outperforms the Service Grid in the nominal and underload conditions with respect to all metrics.

Normalized request length	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_{QoS} (%)	P_q (%)	Total servers used	Total servers at a time
0.3	0.3	138	0.614	403 878	0.00	0.36	10	8
0.5	0.5	230	0.848	404 826	0.00	6.65	14	11
1	1	462	1.359	404 119	0.01	13.6	21	19
1.1	1.1	507	1.667	403 882	0.01	20.0	22	23
1.5	1.5	694	3.620	404 579	0.03	49.0	26	26

Table 6.4: Performance of the Service Grid with mean request length.

Normalized request length	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
0.3	0.3	138	0.645	376 722	0
0.5	0.5	229	0.645	377 315	0
1	1	461	0.683	376 725	1.32
1.1	1.1	533	26.8	376 717	69.8
1.5	1.5	18 645	17 955	376 747	99.3

Table 6.5: Performance of the VC without OLP with mean request length using greedy scheduling.

Tables 6.4, 6.5, and 6.6 show similar results with varying mean request lengths.

The Service Grid starts off with a single server that is randomly chosen. As requests arrive and accumulate at the current server, further servers are added “on demand” to the serving set. Because the server acquisitions are performed on demand, the system has to run out of serving resources to acquire more resources. Therefore, P_q and P_{QoS} will continue to have non zero (possibly small) values. Delays in acquiring new resources can further increase these values. As a results, under nominal or underloaded conditions, the T_c value tends to be higher with Service Grid than with the VC.

Normalized request length	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
0.3	0.3	138	0.636	377 024	0
0.5	0.5	230	0.636	376 733	0
1	1	460	0.655	376 761	0.00
1.1	1.1	522	16.83	376 470	64.0
1.5	1.5	20 743	20 053	376 386	99.3

Table 6.6: Performance of the VC without OLP with mean request length using probabilistic scheduling.

Moreover, these tables also show the vulnerability of the VC to overload conditions. Because a VC operates with a limited set of resources, when the system is overloaded even by a small amount the requests quickly start queuing at the resources and this queuing dramatically increases T_c and T_r.

Normalized request interval	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1	1	461	0.666	376 967	0
0.8	1.2	461	0.676	454 031	0
0.7	1.5	461	0.697	564 569	0.00
0.6	1.6	461	0.704	602 798	0.00
0.5	2	5 700	5 240	753 214	96.4

Table 6.7: Performance of the VC with OLP with mean request interval.

For underloaded or nominally loaded conditions, the results show that VC *without* OLP with probabilistic server acquisition performs better than Service Grid. The results also show the weakness of the VC under overloaded conditions. To address this problem, I introduced the notion of OLPs. Tables 6.7 and 6.8 show the performance of the VC with OLP that are created for a overloaded condition of 1.5 times the nominal load condition. Even with the OLP, the VC uses only a total of 12 servers. These tables show that the addition of the OLP improves the performance without a dramatic increase in the cost of creating and maintaining the VC.

Normalized request length	Normalized load	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1	1	461	0.666	376 967	0
1.2	1.2	553	0.676	376 882	0
1.5	1.5	691	0.697	376 815	0.00
1.6	1.6	737	0.703	376 665	0.00
2	2	6 601	5 680	377 063	98.0

Table 6.8: Performance of the VC with OLP with mean request length.

Normalized window size	Normalized window center	T_r (sec.)	T_c (sec.)	No. of requests	P_{QoS} (%)	P_q (%)	Total servers used	Total servers at a time
0	1	922	2.940	212 930	0.01	27.4	20	16
1	1	889	2.893	213 217	0.01	25.2	19	15
1.5	1.25	1 038	4.478	213 241	0.02	41.1	21	17

Table 6.9: Performance of the Service Grid for windowed loading.

Normalized window size	Normalized window center	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
0	1	461	0.655	377 018	0.00
1	1	395	0.710	376 918	0.80
1.5	1.25	10 720	10 159	376 571	79.3

Table 6.10: Performance of the VC without OLP for windowed loading.

Normalized window size	Normalized window center	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
0	1	461	0.666	377 042	0
1	1	513	0.672	376 769	0
1.5	1.25	729	86.5	376 260	16.2

Table 6.11: Performance of the VC with OLP for windowed loading.

Because a VC is a large-scale system by itself with geographically distributed presence, it is unrealistic to assume that the demand presented to a VC will be uniform across the VC. In particular, a VC can expect localized hotspots where the demands are much higher than the expected value that are separated by regions where the demands are below the expected value. I simulated the VC configurations under these demand situations to verify whether the VC is able to diffuse the demand to the nodes with sufficient active capacity.

In Tables 6.9, 6.10, and 6.11, the demand varies in a window. The mean request lengths at each AP is selected from a uniformly distributed window. The first rows of the tables report results for demands generated without the windows. Therefore, the first rows of the tables present the results for nominal loading conditions. The second rows of the tables report results for demands generated with a window of width 1 that is centered at nominal loading. Therefore, the maximum loading is 1.5 times the nominal loading and the minimum loading is 0.5 times the nominal loading. The third rows of the tables present the results for demands generated with a window of width 1.5 that is centered at 1.25 times the nominal loading.

The results shown in the above tables indicate that VC with OLP outperforms the Service Grid by about 40% with respect to the response time while using 33% less number of serving resources. The VC is able to achieve this performance benefit by selecting the server locations carefully considering the expected demands at the APs. On the other hand, the Service Grid locates resource on demand and as a result may not place them in the most desirable locations with respect to the majority of the requests.

Normalized overload factor	% of overloaded APs	T_r (sec.)	T_c (sec.)	No. of requests	P_{QoS} (%)	P_q (%)	Total servers used	Total servers at a time
1	1	461	1.360	404 232	0.00	13.6	22	20
1.1	0.25	460	1.344	403 838	0.01	13.0	22	19
1.2	0.25	461	1.362	403 815	0.00	13.5	22	20
1.5	0.25	469	1.450	403 621	0.00	14.8	22	21

Table 6.12: Performance of the Service Grid for asymmetrical loading.

Tables 6.12, 6.13, and 6.14 present results from simulations that applied a different

Normalized overload factor	% of overloaded APs	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1	1	461	0.655	377 128	0.01
1.1	0.25	459	0.655	377 414	0.00
1.2	0.25	461	0.655	376 567	0.01
1.5	0.25	458	0.659	376 385	0.10

Table 6.13: Performance of the VC without OLP for asymmetrical loading.

Normalized overload factor	% of overloaded APs	T_r (sec.)	T_c (sec.)	No. of requests	P_q (%)
1	1	461	0.666	376 551	0
1.6	0.25	458	0.667	376 562	0
1.8	0.25	464	0.663	376 413	0
2	0.25	463	0.666	377 221	0

Table 6.14: Performance of the VC with OLP for asymmetrical loading.

loading scheme called the *asymmetrical loading*. In this loading, a given percentage of the APs are overloaded by a fixed factor and the rest of the APs are underloaded such that the total system is loaded at nominal loading conditions. The APs to be overloaded are randomly selected (provided the total number of selected APs are below the given percentage). As expected, the systems including Service Grid performed very well. However, VC with OLPs was able to perform at the same level as the Service Grid that was using a larger number of serving resources.

The overall resource utilizations among the resources that make up a "cluster" are shown in Figure 6.1. The horizontal axis represents the scaled simulation time with the vertical axis showing the total active capacity remaining in the cluster at the time that corresponds to the value on the horizontal axis. From the figure it is clear that VC without overload partitions provides the best resource utilization. Service Grid have the lowest resource utilization at about 50% overall utilization. The VC with OLP still outperforms the Service Grid (although by a smaller margin). However, when only the dedicated VC

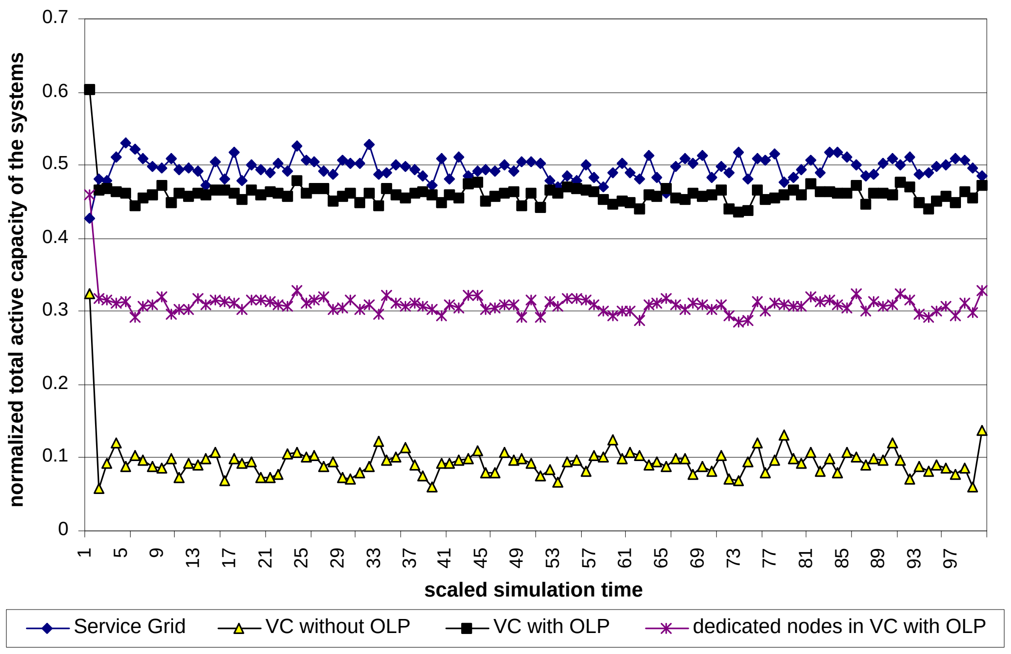

Figure 6.1: Resource utilization within a cluster.

nodes within the cluster is considered, the overall resource utilization improves to about 70%. Because the non-dedicated OLP resources are shared with multiple VCs, they by design should have low utilizations when observed from a single VC (i.e., the OLP resources can expect loading from multiple VCs). The difference between the utilization figures of the dedicated partition resources and OLP resources show that the probabilistic server acquisition scheme meets our objective of maximally using the dedicated partition resources (Section 5.4).

The overall utilizations shown in Figure 6.1 do not illustrate some of the important features of the different heuristics. Further, a VC is created off the resources managed by the PCU. Therefore, a VC will incur renting costs while it holds the resource or uses the resource to store and execute its programs. The VC and Service Grid operate in different modes as well. For instance, the VC continuously holds the same resources in the dedicated partition until it is reconfigured by a subsequent reallocation process. However, the Service Grid acquires and releases resources in a dynamic fashion. Therefore, I developed a “cost” metric to compare the cost-effectiveness of the different schemes.

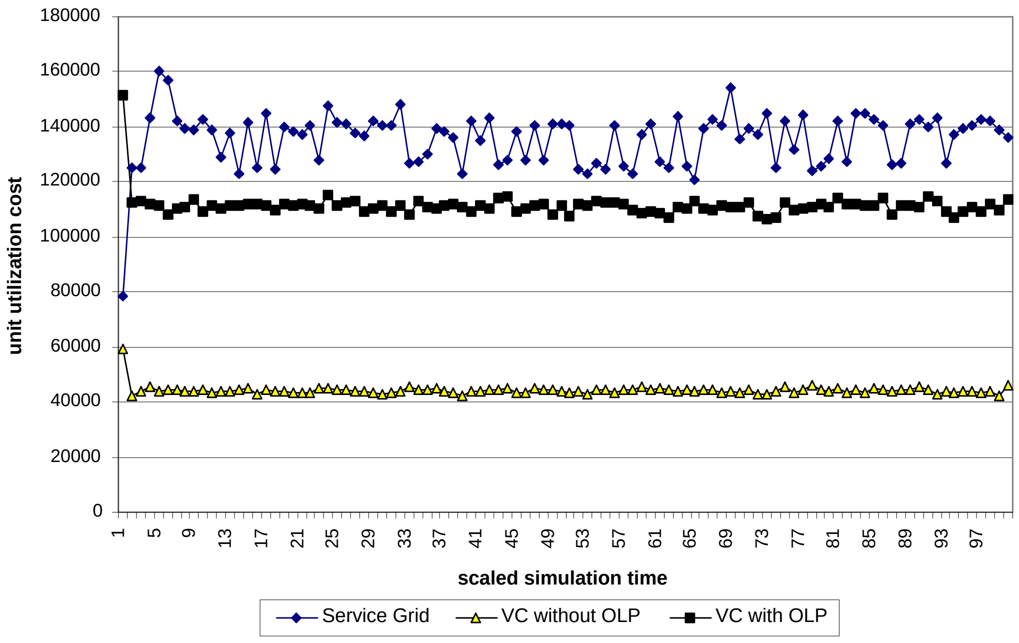

Figure 6.2: Unit utilization cost of a cluster.

Let n be the number of servers acquired and used by a scheme at any given time, f be the fixed cost of using the server per unit time, and τ be the overall resource utilization. Overall resource utilization can be computed from the Figure 6.1 as *1.0 - total remaining active capacity*. With these parameters, a metric called *unit utilization cost* is defined as $(n \times f)/\tau$. A lower value for this metric indicates a cost-effective configuration.

Figure 6.2 shows the variation of this metric with scaled simulation time for the different schemes. It can be observed that the VC configurations hold this metric steady and the Service Grid shows significant oscillations. The oscillations are due to the variations caused by the continuous inclusion and deletion of serving resources from the Service Grid in response to the variations in demand.

7 Conclusion

7.1 Contributions

This thesis makes the following contributions to the wide-area resource management:

A novel PCU model: The PCU model proposed here identifies the key components in making a PCU concept practically feasible and commercially acceptable. The novel features of this model are (a) an ISP like service structure; (b) proposing the resource profiling scheme for resource registration; (c) addressing scalability by developing PCU structure made up of domains; (d) incorporating peering technology for inter-domain information dissemination; and (e) SLA based service instantiation and monitoring.

The concept of virtual cluster: This is the main contribution of this work that has many excellent features: (a) it mathematically formulates the trade-off between achieving the best QoS and reducing the system cost, making it best suitable for commercial infrastructures; (b) even though multiple services can occupy a single resource and the service–resource attachments can change with time, a virtualized static logical resource set exposed to the SO hides the complexity; (c) being a semi-dynamic scheme, a VC can reshape itself matching the varying demand pattern, at the same time the static virtualization to the SO simplifying the service management; and (d) the optimization based VC creation results in better resource utilization.

The concept of anchor point: By providing a representation of demand distribution in a network, the concept of anchor point enables a client-centric resource allocation for wide-area services.

The concept of overload partition: The overload partition nodes have two attributes:

they are selected via an optimization process and they are shared among multiple services. Hence they provides a cost effective, but still QoS obeying solution to handle demand spikes in the network.

7.2 Limitations

The work given in this thesis is bound to the following limitations:
Single service: This work is developed for allocating resources for a single service. If more than one service is to be hosted in the system, the services have to be allocated resource one after anther by a priority order. This work did not extend its mechanism to allocate resources for multiple services simultaneously.
Simulation study is restricted to a single type of service: The service type considered in the simulation study is of a document retrieval type. Other type of applications will need modifications in the Parsec simulator codes. However, the mathematical model and drop heuristic code need no modifications.
Centralized allocation structure: The developed solution heuristic in this work is centralized, making it infeasible to scale a large infrastructure. However, this work does give a proposal for developing the heuristic into a distributed algorithm (Section 5.2).

7.3 Future Works

The novelty of my work can be further extended in the following ways:

- The mathematical model is to be further developed to enable simultaneous resource allocation for multiple services.
- The distributed version of the proposed drop heuristic should be developed to make the allocation process on-line.
- Different service and workload types should be simulated to validate the performance of the VC for various applications.

- The simulation study should be further extended by using real traces instead of the synthetic traces and with limited buffer size at the servers.
- A queueing model can be developed to further improve the scheduling schemes.
- Other mechanisms and components to build a complete PCU should be developed, tested, and implemented to validate the practical feasibility of the proposed PCU architecture.

Despite the limitations of this work mentioned in Section 7.2, the simulation results clearly show the VC mechanism proposed here is a definite break-through towards a QoS guaranteed, cost effective wide-area resource management. Supported by this VC management framework, the proposed PCU architecture will become the potential resource management framework of the future services.

A

Abbreviations

AP	Anchor point
API	Application programming interface
CFCLP	Capacitated fixed charged location problem
COD	Cluster-on-demand
FLP	Facility location problem
GM	Group manager
ISP	Internet service provider
OLP	Overload partition
PCU	Public computing utility
QoS	Quality of service
RM	Resource manger
RU	Remote Unix
SDK	Software development kit
SLA	Service level agreement
UFCLP	Uncapacitated fixed charged location problem
VC	Virtual cluster
VCS	Virtual cluster specification
VoD	Video-on-demand

Bibliography

[Aka02] “Akamai homepage,” http://www.akamai.com.

[AnG02] A. Andrzejak, S. Graupner, V. Kotov, and H. Trinks, “Self-organizing control in planetary-scale computing,” *IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, (Berlin), May 2002.

[ApE02] K. Appleby, T. Eilam, L. L. Fong, G. Goldszmidt, and M. H. Kalantar, “Resource model for self-managing computing utility services,” IBM Research Division, Thomas J. Watson Research Center, Mar 2002.

[ApF01] K. Appleby, L. Fong, G. Goldszmidt, S. Krishnakumar, and et al., “Océano – SLA based management of a computing utility,” *7th IFIP/IEEE International Symposium on Integrated Network Management (IM2001)*, May 2001.

[ArD00] M. Aron, P. Druschel, and W. Zwaenepoel, “Cluster reserves: A mechanism for resource management in cluster-based network servers,” *ACM Sigmetrics 2000 International Conference on Measurement and Modeling of Computer Systems*, June 2000.

[BoM02] E. Bouillet, D. Mitra, and K. Ramakrishnan, “The structure and management of service level agreements in networks,” *IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 4, May 2002.

[CaK99] A. Campbell, M. Kounavis, D. Villela, J. Vicente, H. D. Meer, K. Miki, and K. Kalaichelvan, “Spawning networks,” *IEEE Network Magazine*, Vol. 13, No. 4, July/August 1999, pp. 16–29.

[CiK01] I. Cidon, S. Kutten, and R. Soffer, “Optimal allocation of electronic content,” *INFOCOM*, 2001, pp. 1773–1780.

[Con02] “Condor: High throughput computing,” http://www.cs.wisc.edu/condor/.

[CzF02] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, “SNAP: A protocol for negotiating service level agreements and coordinnating resource management in distributed systems,” *8th Workshop on Job Scheduling Stratergies for Parallel Processing*, (Edinburgh, Scotland), July 2002.

[Das95] M. Daskin, *Network and Discrete Location: Models, Algorithms, and Applications*, John Wiley & Sons, Inc., New York, NY, 1995.

[Doa96] M. B. Doar, “A better model for generating test networks,” *IEEE Globecom*, Nov. 1996, pp. 86–93.

[EpL96] D. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne, “A worldwide flock of condors: Load sharing among workstation clusters,” *Journal on Future Generations of Computer Systems*, Vol. 12, 1996.

[FoK01] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the Grid: Enabling scalable virtual organizations,” *International Journal of Supercomputer Applications*, 2001.

[FoK02] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “The physicology of the Grid: An Open Grid Services architecture for distributed systems integration,” Argonne National Laboratory, IL, USA, 2002.

[FoK99] I. Foster and C. Kesselman, “The Globus project: a status report,” *Future Generation Computer Systems*, Vol. 15, No. 5–6, 1999, pp. 607–621.

[FuV02] Y. Fu and A. Vahdat, “Service level agreement based distributed resource allocation for streaming hosting systems,” *7th International Workshop on Web Content Caching and Distribution (WCW)*, (Boulder, Colorado), Aug. 2002.

[GoS98] M. G. Gouda and M. Schneid, “Maximizable routing mertics,” *6th IEEE International Conference on Network Protocols*, (Austin, Texas), Oct. 1998.

[GrW94] A. S. Grimshaw, W. A. Wulf, J. C. French, A. C. Weaver, and P. F. Reynolds Jr., “Legion: The next logical step toward a nationwide virtual computer,” June 1994.

[JaJ01] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, “Constrained mirror placement on the Internet,” *INFOCOM*, 2001, pp. 31–40.

[JaJ99] H. Jamjoom, S. Jamin, and K. Shin, “Self-organizing network services,” University of Michigan, 1999. Techinal Report CSE-TR-407-99.

[KoP00] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, “Analysis of a local search heuristic for facility location problems,” *Journal of Algorithms*, 2000.

[Kot01] V. Kotov, “On virtual data centers and their operating environments,” Hewlett-Packard Company, 2001. Techreport HPL-2001-44.

[KrB02] K. Krauter, R. Buyya, and M. Maheswaran, “A taxonomy and survey of Grid resource management systems,” *Software Practice and Experiance*, Vol. 32, No. 2, Feb. 2002, pp. 135–164.

[Kre01] H. Kreger, “Web services conceptual architecture,” IBM, May 2001. Whitepaper WSCA 1.0.

[LeV02] “Legion: A worldwide virtual computer,” http://legion.virginia.edu/.

[LeW01] B. Lee and J. B. Weissman, “Dynamic replica management in the Service Grid,” *IEEE 2nd International Workshop on Grid Computing*, Nov. 2001.

[LiL88] M. J. Litzkow, M. Livny, and M. W. Mutka, “Condor – a hunter of idle workstations,” *The 8th International Conference of Distributed Computing Systems*, June 1988, pp. 104–111.

[MoC02] J. Moore and J. Chase, “Technical report: Cluster on demand,” Department of Computer Science, Duke University, May 2002.

[OSP83] “OSPF version 2,” RFC 1583.

[Par02] “Parsec: Parallel simulation enviornment for complex systems,” http://pcl.cs.ucla.edu/projects/parsec/.

[QiP01] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of web server replicas,” *INFOCOM*, 2001, pp. 1587–1596.

[SET02] “SETI@home home page,” http://setiathome.ssl.berkeley.edu/.

[UrS02] B. Urgaonkar, P. Shenoy, and T. Roscoe, “Resource overbooking and application profiling in shared hosting platforms,” *5th Symposium on Operating Systems Design and Implementation (OSDI)*, (Boston, MA), Dec. 2002.

[VaA98] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, and D. Culler, “WebOS: Operating system services for wide area applications,” *The Seventh IEEE Symposium on High Performance Distributed Computing*, July 1998.

[VaK01] V. Kotov, “Towards service-centric system organization,” Hewlett-Packard Company, 2001. Techreport HPL-2001-54.

[Web02] “Webos: Operating system services for wide area applications,” http://www.cs.duke.edu/ari/issg/webos/.