Enterprise Grids: Challenges Ahead[⋆]

R. Jiménez-Peris[1], M. Patiño-Martínez[1], and B. Kemme[2]

[1] Facultad de Informática, Universidad Politécnica de Madrid (UPM), Spain
{rjimenez,mpatino}@fi.upm.es
[2] McGill University, School of Computer Science, Montreal, Quebec, Canada
kemme@cs.mcgill.ca

Abstract. Grid technologies have matured over the last few years. This level of maturity is especially true in the field of scientific computing in which grids have become the main infrastructure for scientific problem solving. Due to its success, the use of grid technology rapidly finds its introduction into other fields. One of such fields is enterprise computing in which grids are seen as a new architecture for data centers. In this paper, we describe the vision of enterprise grids, current scientific achievements that will leverage this vision, and challenges ahead.

1 Introduction

Grids have succeeded in providing an infrastructure for deploying parallel applications in a distributed setting with a high degree of automation. The definition of grids has been redefined along the years. Initially grids were defined as an infrastructure to provide easy and inexpensive access to high-end computing [17]. Then, it was refined in [11] as an infrastructure to share resources for collaborative problem solving. More recently, in [12] the grid definition evolves to an infrastructure to pool and virtualize resources and enable their use in a transparent fashion.

Grid infrastructure exhibits several interesting features. One of the main functionalities is that a grid hides the heterogeneity of its underlying components such as hardware, operating systems or storage systems, and serves as a middleware that provides seamless connectivity of the components it manages. Another interesting characteristic is the transparent pooling of many kinds of resources such as computing power, storage, data, and services. This pooling enables applications deployed on the grid to transparently share resources and utilize the provided capacity more effectively. A related attribute is the ability to allocate and reserve virtualized resources. Virtualization facilitates the sharing of resources: it allows the preservation of quality of service guarantees for time critical applications, and at the same time offers true resource scavenging by taking advantage of unused resources to perform batch computations.

These features make grid computing attractive for enterprises which want to convert it into their infrastructure for deploying enterprise applications. This interest has

[⋆] This work has been partially funded by the European Adapt and S4ALL projects, the Spanish Ministry of Education and Science (MEC) under grant #TIN2004-07474-C02-01, the Madrid Regional Research Council (CAM) under grant #0505/TIC/000285, and the Spanish Ministry of Industry under grants #FIT-340001-2005-8 and # FIT-340000-2005-144.

resulted in minting a new term, *Enterprise Grids*. Enterprise Grid computing reflects the use of grid computing within the context of a business or enterprise rather than for scientific applications. A proof of interest for Enterprise Grids is the recent creation of Enterprise Grid Alliance with the mission of developing a common Enterprise Grid reference model [9] and fostering the use of grids for enterprise computing.

Despite the progress in grid technologies and the will to address the needs of enterprise applications [12] there is still a significant gap between what is required by enterprise grids and what is currently provided by grid technology, with significant technical and scientific hurdles in the way. This gap is mainly due to the many differences between scientific and enterprise applications. Examples are the stateful nature of business applications, the typical underlying multi-tier architecture, the existence of transactional data, and the need for transactional interaction between the different application components. In this paper, we aim at identifying the gap between what grid infrastructures currently provide and what enterprise grids need to offer. From there, we survey recent achievements in the field of distributed computing that will help to fill this gap, and identify the open challenges to realize the enterprise grid vision.

The paper is structured as follows. First, in Section 2 we identify some of the main requirements in Enterprise grids and contrapose them with current functionality provided by grid systems to characterize the gap between them. Then, we describe some of the recent advances in distributed systems that will help to fill this gap in Section 3. We continue in Section 4 by identifying the existing open scientific challenges to realize the Enterprise Grid vision. Finally we present our conclusions in Section 5.

2 The Gap between Grids and Enterprise Grids

Gap 1: Support for Online Applications.

Traditional enterprise computing is usually composed of a mix of batch and online applications. Grids already have a good grip on batch applications and provide sophisticated automation making batch applications run efficiently on large distributed infrastructures. Abstractions such as “task bags” can very well serve as a basis for automating the programming and execution of batch enterprise applications. Additionally, the use of scavenging will enable the use of idle resources for executing them.

However, there is a large fraction of enterprise applications that are inherently interactive, and we refer to them as online applications because an end user is directly connected to the system. Online applications require timely execution of requests and provisioning of appropriate responses to the client. This requires a very well oiled machinery where a single bottleneck can lead to unsatisfying performance. Current grid technology provides little help in this regard since it is more oriented to batch style applications where the emphasis is put on achieving high throughput, i.e., serving a large set of tasks, without paying attention to the response time of individual requests. In contrast, for online applications the main performance metrics is the average response time, often with strict constraints on its statistical distribution. This need is clearly documented in benchmarks for online systems such as TPC-C for database applications, ECPERF for information systems, or TPC-W for web based systems.

Gap 2: Support for Transactional Data.

Business applications are often data-intensive. They access, process and manipulate large amounts of data, most of which resides in database repositories. Access to such data is nearly always in the context of transactions in order to guarantee consistency and durability of changes to the data. Another challenge is the requirement of continuous availability of such data since clients want to have access to their data anytime from anywhere, and lack of availability usually results in financial losses. Current grid technology lacks proper support for scalable, reliable and transactional data processing. Current provisioning in grids focuses mainly on stateless applications, and sometimes on applications that access read-only persistent data. Read-only data is easy to scale by simply providing as many data copies as needed. While data replication can be used even for update intensive transactional data, consistency and scalability are non-trivial challenges not only in wide-area settings but even for localized computing.

There has been some work related to “data grids” but the emphasis has been on data integration and management of large collections of data. The goal of data integration has been to provide a homogeneous view of heterogeneous data schemas containing similar information. Data integration can be an important issue in some enterprise applications in which data from different domains exhibit different schemas and a homogeneous view and access means should be provided. Additionally, there is the trend of converting data access into services. In this context, the OGSA-DAI [31] middleware provides support for integrating relational and XML data and exposing them through a homogeneous web service interface, which can be a very useful abstraction for business applications. Business applications might also take advantage of the support grids provide for automated storage and management of large collections of data in the order of petabytes or higher. This includes fast efficient remote access to such data collections. One of the solutions consists in moving the application close to the data. This technique, also known as function shipping or batching design pattern [35] saves expensive WAN interactions. The idea is that instead of running the computation which accesses the data on a resource which is far from the data storage, the client code is moved to the data domain so it can access the data locally. When the computation is done the result is returned to the client side.

Gap 3: Support for Stateful and Transactional Applications.

Business applications do not only access transactional data residing in backend databases, they often maintain state themselves. This data might also be accessed within the same transactions that control the execution at the backend database. Furthermore, this state often reflects the interactions between an online client and the application (such as session information or a shopping cart). Thus, data consistency across tiers becomes a challenge, typically ignored by current grid technology. Another problem in this context is that a failure of a site can result in losing important application state. Business processes can last for days, weeks, months, or even years. This is very common for workflows and orchestrated web services. The loss, e.g., of a stateful interaction, would be unacceptable in this context. One additional aspect is that in this context, advanced transaction models are often used [18] to relax isolation that is problematic for long running transactions. In the advent of a failure one possibility to attain consistency is to abort all running transactions. However, for long running transactions this is not really an option since they have to progress forward despite failures, unless it is impossible to attain their goal. Grids fail to support transactional applications and stateful applications requiring high availability.

Gap 4: Support for Multi-tier Architectures

Enterprise applications are typically deployed on multi-tier architectures consisting of web, application server and database tiers. As mentioned in gaps 2 and 3, managing each of the individual tiers on the grid is challenging, in particular because of the state they manage. Providing a grid solution that supports multiple tiers is even more complicated due to the interaction patterns between the tiers. For instance, transactions can span execution across several tiers. This means, gridification of multi-tier applications needs to preserve consistency across tiers despite data and execution dependencies between the tiers. Also failures need to be handled appropriately. That is, failover should be performed in such a way that consistency is still preserved. Certainly, grid systems provide support for multiple applications and servers. However, this support tends to deal with different servers as isolated entities. In order to extend functionality to multi-tier systems, a systematic approach is needed to understand the interaction patterns between adjacent tiers, the execution across all tiers, and the impact of both on the gridification process. Current support in grid systems fails to address multi-tier architectures, what is one of the significant gaps for attaining enterprise grids.

Gap 5: Autonomic support

Another important goal for the Enterprise Grid vision is that of autonomic management [24]. Autonomic management encompasses several attributes such as self-healing, self-provisioning, self-optimizing, and self-configuring. **Self-healing** capabilities, which allow to tolerate failures, provide the Enterprise Grid a continuous availability. Fault tolerance is typically attained by replication. Another facet required to realize a self-healing enterprise grid and the associated high availability is the ability to recover failed sites. Most approaches to recovery of replicas imply stopping system operation in order to obtain a quiescent state and transfer it to the recovering replica. Once the recovery is completed the system operation is resumed. However, this offline recovery goes against the initial goal of high availability. On the other hand, recovery is needed to maintain a particular level of availability. Otherwise, replicas will fail and eventually the system will become unavailable. Therefore, there is a need for performing recovery in a non-intrusive fashion maintaining the system online. Currently, grid systems provide self-healing facilities but mainly for stateless applications. Sites can join and leave, but little is done to keep the consistency between the state of working sites and the sites joining the grid. Therefore, there is still a large gap between the needs of Enterprise Grids and what is provided by current grid systems relative to stateful applications.

Another essential aspect of enterprise grids is performance and the need for **self-optimization**. Underlying middleware and database systems have numerous parameters that enable to tune them to maximize its performance. Unfortunately, these parameters are set manually and require a highly qualified administrator. What is more, the right value for these parameters depends on the actual workload. An enterprise grid cannot rely on human tuning and should be able to tune itself to maximize performance. At best, human administrators will set high level goals in the form of utility functions to hint the system in which dimensions to maximize first. Autonomic performance tuning implies to monitor the system load and performance. When the system detects that performance decreases, tuning parameters need to be adjusted in order to stabilize the system and achieve a configuration that provides optimal performance for the the current workload and its characteristics. This self-optimizing behavior can only be built upon an analytical model of the system that enables to understand the system state by monitoring the relationship between load and performance and and knowing which parameters influence this relationship. This means that the self-optimizing support requires specific work for each kind of server, and in the case of enterprise grids, for each particular tier. Certainly, there are currently no provisions for self-optimization of multi-tier architectures in grids.

Self-optimization at each individual site/server/tier guarantees that its performance is maximized given the received workload. However, individual self-optimization is not enough to maximize performance at the global level. The reason is that some sites might be overloaded whilst others are idle, despite each site being tuned to maximize its individual performance. Thus, processing power is being lost due to the imbalance in the load. This means that an enterprise grid should be able to **self-configure** itself to distribute the load across sites in a balanced way. This load balancing should be performed on a continuous basis and in a non-intrusive fashion. There has been substantial work in providing load balancing solutions for grid environments. However, although load balancing techniques and policies for different kinds of applications and servers share a common background, each server and/or application has typically very specific characteristics which lead to individual load-balancing requirements. Thus, while current grid systems offer load-balancing mechanisms that are well suited for parallel and scientific applications they do not serve well transactional stateful applications with their many constraints in regard to data and execution dependencies.

Another interesting challenge arises if the computing power requirements of an enterprise application change over time which will require some form of **self-provisioning**. If more processing power is needed, more resources need to be added to be able to cope with the load. This should happen autonomously. Thus, a pool of free sites needs to be available. When load submitted increases and the system feels that the given processing power will soon be unable to handle this load, the system should self-provision itself and increase the amount of devoted resources. Conversely, if the load is reduced the resources should shrink to free unneeded resources. However, self-provisioning in enterprise grids is very challenging due to their stateful nature. Adding a new replica means that state should be first transferred to it. This means that online recovery is needed to enable self-provisioning as it was needed for attaining the self-healing property. Grid systems have achieved significant progress in self-provisioning. This infrastructure can be generic enough to be reused in the context of Enterprise Grids, although support for dealing with stateful applications will have to be integrated. However, the recovery solutions for self-healing purposes can be reused in this context.

Finally, it should be realized that autonomic management in an Enterprise Grid cannot just rely on a component or server-based approach. Instead, autonomic management in Enterprise Grids should adopt a holistic approach. This means that there will be an entity in charge of management, what is known as Grid Management Entity (GME) in the Enterprise Grid Architecture (EGA). This GME is a recursive component. There is a GME at the global Enterprise Grid level. Then, each subsystem with the Enterprise Grid will have its own GME that will interact with the higher level GME and the lower level GMEs corresponding to the contained subsystems. For instance, in a multi-tier architecture, there will be a GME for the full system, and then a GME for each gridified tier, e.g., web tier, application tier, and database tier. There will also be a GME within each instance of the server in a tier. That is, if there are n servers in the application server tier, each of them will have its own GME for local autonomic management. In this area, the gap with what is currently provided by grids is significant, since grids typically deal with independent servers or at least deal with them as if they were independent.

Gap 6: Support for Service Level Agreements

An important class of enterprise applications provide service provision governed by service level agreements (SLAs). SLAs set a contract between client and provider regarding the offered service, quality of service, cost vs. service tradeoffs, tracking of the quality service offered, etc. One of the main concerns is that by sharing resources across different clients, the demand from one client can lead to violations of the SLA of another client. This problem can be addressed by resource virtualization. The idea is to split the resources of a site into multiple isolated execution environments that can be assigned to different clients. The performance isolation achieved by virtualization enables to adopt policies to maximize the fulfillment of SLAs. Grid computing has produced many results regarding resource virtualization that can be exploited to attain Enterprise Grids. However, there is still an important gap regarding how to handle SLAs and the mapping of resource virtualization to clients, especially when the resources are spread across different tiers in a multi-tier architecture.

3 Filling the Gap: Recent Advances in Distributed Systems

Database Replication.

Database replication is important to support the gridification of stateful and transactional online applications and the database tier itself. Database replication has been an active research topic for many years. One of the challenges is to keep replicas consistent despite updates. The first textbook solutions for data replication [4] defined one-copy serializability (1CS) as the universal correctness criteria for replicated, transactional data and provided protocols satisfying consistency but very poor performance and no scalability. The famous paper from Jim Gray et al. [15] analytically characterized the lack of scalability of these solutions. This paper triggered research in different directions trying to overcome the shortcomings of the existing solutions.

Several different approaches looked at lazy replication where updates are executed only at one replica and updates a propagated only after commit to the other replicas. For instance, in [5], serializability is provided by assigning each object a master and only the master can accept and propagate updates; furthermore assignment of objects to masters and the propagation topology is restricted to guarantee correct executions. A different research line aimed at making the lack of consistency explicit to the user, providing different degrees of data freshness to the user [32, 41, 33]. Freshness measures the amount of updates that might have been missed by transactions and therefore quantifies the staleness of the data read by a transaction. Another set of lazy approaches, such as IceCube [25] and its clustered extension [27], has been devoted to reconciliation in optimistic replication. Replicas are allowed to proceed in parallel without any measure to prevent conflicts. Later, conflicts due to concurrent accesses are fixed by means of semantic corrective actions.

A different line of research looked at scalable solutions for eager data replication, where updates are propagated before commit, hence providing much better consistency guarantees. Most of these approaches look at solutions providing 1CS. May of them, such as [37, 34] built on group communication to guarantee consistency in advent of any failure scenario. They additionally exploit asymmetric update processing [20] in which update transactions are only fully processed at one replica (typically, different transactions are fully processed at different replicas), while other replicas only apply the updated tuples. This asymmetric processing is essential to attain scalability for update workloads [20]. Other eager approaches were based on schedulers [1] that provided the necessary message ordering and consistency guarantees. All these approaches attained some reasonable scalability up to several tens of nodes while providing full consistency.

A new breed of protocols and systems exploit a more scalable isolation level, known as snapshot isolation (SI) [44, 8, 26, 39]. SI is the highest isolation provided by some databases such as Oracle and PostgreSQL, and provided by others such as the just released SQLServer. SI is based on multi-version concurrency control and has the advantage that there are only write-write conflicts what is crucial to enable high levels of concurrency, a requirement for high scalability. SI can be extended to a replicated setting creating the notion of one-copy snapshot isolation (1CSI) that has been formally defined in [26]. 1CSI benefits from the high level of concurrency allowed by 1CSI and by the underlying multi-version concurrency control that fits very well in a replicated model in which each replica can be considered a different version.

Clustering in Multi-Tier Architectures.

The research in clustering of multi-tier architectures will be useful to support gridification of multi-tier architectures and the stateful and transactional applications that are typical in these architectures.

The replication of the web server tier has been studied very heavily during the last decade [6]. The research has basically set the ground work for virtualizing IP addresses in different ways, typically through a router. The result is a virtual IP address that can be served by a pool of servers with different physical IP addresses. When moving web clusters to a grid environment, additional problems have to be solved. Currently, most routers guarantee that a request is sent to a working server, but there are no guarantees about serving the request, since the server can crash before completing request execution. What is lacking is a solution that provides end-to-end exactly-once semantics to enable a transparent fault-tolerant and highly available interaction. Furthermore, in order to integrate the self-optimizing and self-provisioning features of the grid system into IP virtualization, it must be possible to change the set of sites serving a virtual IP address dynamically.

A significant body of research has looked into data caching for web servers covering issues such as cache consistency combined with caching of dynamic data. However, distributed and replicated data caching is still an open problem. One of the main difficulties is how to accomplish an independent gridified data caching tier (not recognized as a tier in J2EE), such that its performance and resilience can be increased whilst still providing full data coherence.

In the application server tier, most work has focused on providing availability via primary-backup replication or active replication. Early work started with CORBA (e.g. Eternal [30]) ignoring the interactions with other tiers. The integration of a replicated application server with other, non-replicated tiers, and in particular, the database tier has been first discussed for CORBA and .NET [45, 10, 2] an in general settings [14]. More recently, research on J2EE application servers has yield several promising results in the context of the European Adapt project [3]. For instance, [44] discusses integration of replication of the application server with advanced execution and transaction patterns. Some recent results show that it is possible to provide highly available transactions in which, despite failovers, transactions are not aborted from the client perspective [38]. However, research in this area is still at an early stage in which only solutions for fault tolerance (primary-backup) have been provided. More work is needed which takes self-optimization and self-provisioning into account.

Furthermore, little work has been done when looking at gridification, or at least replication, across several tiers. The X-ability framework [13] allows to reason about correctness in a general multi-tier architecture. [23] looks at different integration approaches when several tiers are replicated. Integration can basically be achieved in two different ways. One is that each tier is replicated independently. That is, the application server tier is replicated independently of the database tier, also known as horizontal replication. The second approach consists of taking a pair of application and database server and replicating this pair. This approach is known as vertical replication [38].

Autonomic Computing.

The sheer size and complexity of modern information systems often overwhelms even the most skilled engineers that are supposed to configure, optimize, secure, and repair them. In fact, if the current trend of complexity growth continues, the administration of the next generation systems will be beyond human capabilities. Even if the complexity does not increase, it is predicted that the number of deployed systems in the near future will be substantially higher that the number of qualified engineers to manage them [24]. In fact, these studies have resulted in launching research initiatives in most large software companies to create a new generation of systems able to self-manage themselves with no human intervention except for setting high level goals or deciding between performance and/or costs tradeoffs. Examples of these initiatives are, among others, IBM autonomic computing, HP Adaptive Enterprise, Microsoft Dynamic Systems, and Intel Proactive Computing. The vision is to attain systems able to manage themselves, that is, systems able to self-heal, self-provision, self-optimize, and self-configure themselves. This vision for self-management and self-adaptation is also part of the Enterprise Grid vision [9].

As mentioned before, self-healing requires to recover failed replicas or introduce fresh ones. In a stateful context, this requires to transmit the state of working replicas to these new replicas so they can join a grid of servers. Since such recovery might take considerable time it is impossible to stop execution during this process since this would violate the initial goal of high availability. Recent work has looked into online recovery for self-healing databases [22, 19]. [22] proposed different ways to perform online recovery within the database kernel; [19] studies how to implement online recovery at the middleware level providing toggles for a higher level self-configuration entity to adapt the resources devoted to recovery to the current load.

The potential scalability provided by new replication approaches can only be attained by resorting to autonomic protocols that self-optimize locally and self-configure globally to maximize throughput. In the database arena, a significant effort has been made during the last decade to develop self-tuning and self-optimization protocols, that act locally on a particular server. One of the subsystems in a database system is memory management. When the server overloads, the system enters into thrashing without grateful degradation. Several self-management techniques have been proposed for database memory management in [43] such as access pattern aware adaptive caching or speculative, self-adapting pre-fetching policies.

Another important area for self-optimization in databases has been adaptive load control. It aims at choosing the optimal configuration for a changing system load. This is especially needed when the system becomes overloaded to avoid thrashing due to resource contention. One of the most important parameters that affect contention is the multiprogramming level (MPL), i.e., the number of transactions effectively executed concurrently. For instance, if the system has twenty outstanding transactions, it can decide to execute them one by one, two by two, or all twenty concurrently. Some approaches have looked at contention occasioned by locking. [29] determines the optimal degree of conflict and adjusts the MPL to maintain this optimal degree. [16] applies feedback control to adjust the MPL to maximize the throughput of a dataset. [42] uses multi-programming levels as an external scheduling mechanism based on queueing theory and feedback control. [28] proposes a hierarchical self-configuration/optimization for clustered databases. Locally each database monitors the load and the throughput and self-optimizes the MPL to maximize throughput. Since, local self-optimization is not enough, it is complemented with a global level self-reconfiguration in the form of a dynamic load balancing algorithm to maximize throughput at the cluster level.

Another important requirement of Enterprise Grids is the self-provisioning capacity. In the area of workflow, there has been some interesting work, in which replicas of the workflow server are allocated dynamically depending on the system load [36].

4 Challenges Ahead

Boosting the Scalability of Data Grids.

Most of the existing data replication strategies assume full replication (i.e. replicating the full database at each site). However, such approach has a scalability ceiling that might be not enough for future very large data centers. Partial replication just replicates a subset of data at each site and can be a solution for the scalability ([7]). However, there are still many issues to resolve in this context. One of the interesting research problems is whether a combination of full and partial replication can yield a better scalability. Another important problem is that solutions relying on pure partial replication will likely need to resort to distributed atomic commitment that is the current bottleneck of distributed information systems. Recent protocols for high performance distributed atomic commitment such as the commit server [21] can be a solution to leverage partial replication to boost the scalability of database replication.

Low-Latency Gridification across WANs.

A requirement of Enterprise Grids is the inter-connection of data centers across WANs. This will need support for data replication across WANs. WANs differ from LANs mainly in the latency of the communication. This higher latency becomes especially troublesome for group communication based replication approaches in which the required ordering and reliability guarantees might require several rounds of messages. However, purely lazy replication strategies might not be acceptable for some of the transactional, update-intensive data. Thus, new replication protocols are needed with higher levels of scalability and lower latencies while at the same time achieving acceptable levels of consistency.

Reducing the Overhead of Gridification.

Another aspect crucial for the successful gridification of enterprise applications is the reduction of the inherent overhead of the coordination among sites. Modern system area networks provide functionality that might be exploited to reduce this overhead by delegating functions, possibly dynamically, to the underlying hardware.

Holistic Approach to Virtualization and SLAs.

Despite the advances in virtualization of the different kind of resources (storage, networking, server tier etc.), new research is needed to adopt a holistic approach to virtualization, in which the computational requirements of an application across various tiers are automatically analyzed. From there, the necessary resources are devoted for deploying and running the new application and adjusted autonomically to the changes in the workload and priorities extracted from SLAs.

Scalable and Consistent Gridification of Multi-tier Architectures

The current state of the art has addressed solutions for some of the gridification issues of individual tiers. Much more work has to be done, both in regard to gridification of individual tiers, and across the entire multi-tier architecture, taking into account any form of data and execution dependencies.

Systemic Autonomic Computing.

Most current approaches for autonomic computing focus on individual servers. New approaches addressing the hierarchical self-management of multi-tier systems or complex systems as a whole are needed. Recent advances such as [40], in which performance bottlenecks of complex systems deployed in multi-tier architectures are automatically identified, are a start in this direction.

Accurate Models for Self-Optimization and Self-Provisioning.

Despite the progress attainrf in self-* properties, the attained results do not seem to be enough for the Enterprise Grid vision. One of the aspects that need further research is the modeling of systems with higher accuracy and autonomic protocols with higher guarantees than current best effort approaches.

5 Conclusions

In this paper, we analyzed to what degree current grid systems can help leveraging the Enterprise Grid vision. At the same time, we have identified some of the existing gaps between the current state of the art of grid technology and the envisioned Enterprise Grids. Some recent results in the area of distributed systems and databases have been highlighted which might be the basis for novel mechanisms that will help to bridge the gap between grid reality and the Enterprise grid vision. This paper also aims at fostering cross-fertilization between the grid and the distributed systems communities. Finally, some of the still open challenges to realize Enterprise Grids have been identified.

References

1. C. Amza, A. L. Cox, and W. Zwaenepoel. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In *Middleware*, 2003.
2. R. Barga, D. Lomet, and G. Weikum. Recovery guarantees for general multi-tier applications. In *Int. Conf. on Data Engineering (ICDE)*, 2002.
3. A. Bartoli, R. Jiménez-Peris, B. Kemme, and all. Adapt: Towards autonomic web services. *Distributed Systems Online*, 2005.
4. P. A. Bernstein and N. Goodman. The Failure and Recovery Problem for Replicated Databases. In *PODC*, 1983.
5. Y. Breitbart and H. F. Korth. Replication and consistency: Being lazy helps sometimes. In *ACM PODS*, 1997.
6. V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed Web-server systems. *ACM Computer Surveys*, 34(2), 2002.
7. A. L. P. F. de Sousa, R. C. Oliveira, F. Moura, and F. Pedone. Partial replication in the database state machine. In *IEEE Int. Symposium on Network Computing and Applications*, 2001.
8. S. Elnikety, W. Zwaenepoel, and F. Pedone. Database replication using generalized snapshot isolation. In *SRDS*, 2005.
9. Enterprise Grid Alliance. EGA Reference Model, 2005.
10. P. Felber and P. Narasimhan. Reconciling replication and transactions for the end-to-end reliability of corba applications. In *DOA*, 2002.
11. I. T. Foster. The anatomy of the grid. In *CCGRID*, 2001.
12. I. T. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid. In *Global Grid Forum*, 2002.
13. S. Frølund and R. Guerraoui. X-ability: a theory of replication. In *Symp. on Princ. of Distrib. Comp. (PODC)*, 2000.
14. S. Frølund and R. Guerraoui. e-transactions: End-to-end reliability for three-tier architectures. *IEEE Trans. Software Engineering*, 28(4):378–395, 2002.
15. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *ACM SIGMOD*, 1996.
16. H. Heiss and R. Wagner. Adaptive Load Control in Transaction Processing Systems. In *Proc. of 17th VLDB*, 1991.
17. I. Foster and C. Kesselman. *The Grid*. MKP, 1998.
18. S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer, 1997.
19. R. Jiménez-Peris, M. Patiño-Martínez, and G. Alonso. Non-intrusive, parallel recovery of replicated data. In *IEEE Symp. on Reliable Distributed Systems (SRDS)*, 2002.
20. R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme. Are quorums an alternative for data replication. *ACM Transactions on Database Systems*, 28(3), 2003.
21. R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and S. Arevalo. A Low-Latency Non-Blocking Atomic Commitment. In *DISC*, 2001.

22. B. Kemme, A. Bartoli, and O. Babaoglu. Online Reconfiguration in Replicated Databases Based on Group Communication. In *DSN*, 2001.
23. B. Kemme, R. Jiménez-Peris, and M. Patiño-Martínez and J. Salas. Exactly once interaction in a multi-tier architecture. In *VLDB Workshop on Design, implementation, and deployment of database replication*, 2005.
24. J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer*, 2003.
25. A.-M. Kermarrec, A. I. T. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC*, 2001.
26. Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *ACM SIGMOD*, 2005.
27. V. Martins, E. Pacitti, , and P. Valduriez. A dynamic distributed algorithm for semantic reconciliation. In *WDAS*, 2006.
28. J. Milan, R. Jiménez-Peris, M. Patiño-Martínez, and B. Kemme. Adaptive middleware for data replication. In *Middleware*, 2004.
29. A. Moenkeberg and G. Weikum. Performance Evaluation of an Adaptive and Robust Load Control Method for the Avoidance of Data Contention Trashing. In *VLDB*, 1992.
30. L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. Tewksbury, and V. Kalogeraki. The Eternal System: An Architecture for Enterprise Applications. In *EDOC*, 1999.
31. OGSA-DAI. http://www.ogsadai.org.uk/.
32. E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB Journal*, 8(3), 2000.
33. C. L. Pape, S. Gançarski, and P. Valduriez. Refresco: Improving query performance through freshness control in a database cluster. In *CoopIS*, 2004.
34. M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Middle-R: Consistent Database Replication at the Middleware Level. *ACM TOCS*, 23(4):275–423, 2005.
35. M. Patiño-Martínez, F. Ballesteros, R. Jiménez-Peris, and all. A Design Pattern for Efficient and Flexible Client-Server Interaction. In *PLOPD*, 1999.
36. C. Pautasso, T. Heinis, and G. Alonso. Autonomic execution of web service compositions. In *ICWS*, 2005.
37. F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98, 2003.
38. F. Perez, J. Vuckovic, M. Patiño-Martínez, and R. Jiménez-Peris. Highly Available Long Running Transactions and Activities for J2EE Applications. In *IEEE ICDCS*, 2006.
39. C. Plattner and G. Alonso. Ganymed: Scalable replication for transactional web applications. In *Proc. of the ACM/IFIP/USENIX Int. Middleware Conf.*, 2004.
40. P. Robertson and B. Williams. Automatic recovery from software failure. *CACM*, 2006.
41. U. Röhm, K. Böhm, H.-J. Schek, and H. Schuldt. FAS - a freshness-sensitive coordination middleware for a cluster of olap components. In *VLDB*, 2002.
42. B. Schroeder, M. Harchol-Balter, A. Iyengar, E. M. Nahum, and A. Wierman. How to determine a good multi-programming level for external scheduling. In *Int. Conf. on Data Engineering (ICDE)*, 2006.
43. G. Weikum, A. Christian, A. Kraiss, and M. Sinnwell. Towards Self-Tuning Memory Management for Data Servers . *Data Engineering Bulletin*, 22(2), 1999.
44. S. Wu and B. Kemme. Postgres-R(SI): Combining replica control with concurrency control based on snapshot isolation. In *ICDE*, 2005.
45. W. Zhao, L. E. Moser, and P. M. Melliar-Smith. Unification of replication and transaction processing in three-tier architectures. In *ICDCS*, pages 290–300, 2002.