Proc. 2004 Computer Measurement Group Conf., Las Vegas, NV, Dec. 7-10, 2004.

QUALITY OF SERVICE ASPECTS AND METRICS IN GRID COMPUTING

Daniel A. Menascé*
Dept. of Computer Science
George Mason University
Fairfax, VA 22030, USA
menasce@cs.gmu.edu

Emiliano Casalicchio†
Dipt. Informatica Sistemi e Produzione
Univ. Roma "Tor Vergata"
Rome, Italy
casalicchio@ing.uniroma2.it

Abstract

Grid computing promises to become the future computing paradigm for enterprise application after having shown to be a quite effective computing paradigm for resource-intensive scientific applications. Large scale grids are complex systems, composed of a large number of components belonging to disjoint domains. Planning the capacity to guarantee quality of service (QoS) in these environments is a challenge because global Service Level Agreements (SLA) depend on local SLAs, i.e., SLAs established with components that make up the grid. These components are generally autonomous and join the grid as part of a loose federation. This paper investigates some of the relevant issues that must be considered in designing grid applications that deliver appropriate QoS: definition of metrics, relationship between resource allocation and SLAs, and QoS-related mechanisms.

1 Introduction

Grid computing is not only a mainstream computing paradigm for resource-intensive scientific applications but it also promises to become the future computing paradigm for enterprise applications. A recent survey by Forrester Research indicates that 37% of the companies surveyed are in some stage of piloting or implementation of grid; 41% use it for financial analysis and modeling, 41% use it for typical business applications, and 26% use it for scientific or engineering calculations [32].

The grid enables resource sharing and dynamic allocation of computational environments composed of components from different domains, thus increasing access to data, promoting operational flexibility and collaboration, and allowing service providers to efficiently scale to meet variable demands. All these factors contribute to increase the productivity and the utilization of an enterprise's infrastructure.

Large scale grids are complex systems, composed of large numbers of components belonging to disjoint domains. Planning the capacity to guarantee quality of service (QoS) in these environments is a challenge because global Service Level Agreements (SLA) depend on local SLAs, i.e., SLAs established with components that make up the grid. These components are generally autonomous and join the grid as part of a loose federation. Only if all these partial SLAs are satisfied, will the global SLA be satisfied. To understand how this can be achieved and how the global and partial SLAs can be specified and then met, it is essential to understand the architecture of the grid, the impact of grid resource allocation on SLAs, the dependencies among grid QoS metrics, the relationships between the QoS and SLAs mechanisms, and the main mechanism to achieve good QoS.

To address the above issues we use an example-driven approach. We consider an insurance company that decides to adopt a grid infrastructure to run complex and rich risk assessment models that will allow the company to be competitive and to increase customer retention and acquisition. From this motivating example, we define the grid-related QoS metrics, the relationship between QoS and SLAs, and the mechanisms to achieve QoS in large scale grid systems. Section two provides a brief introduction to grid architecture, focusing on its multi-layer characteristics. Section three presents a motivating example. Section four analyzes grid-related QoS metrics and discusses how each metric has a different meaning depending on the layer at which it is considered. Section five discusses the impact of resource allocation on grid SLAs. Section six identifies the main mechanisms that must be applied at each level of the grid architecture to achieve desired QoS levels. Finally, section seven presents some concluding remarks.

*This work was supported in part by the National Geospatial-Intelligence Agency (NGA) contract NMA501-03-1-2033.

†This work was partially funded by the PERF project supported by the Italian MIUR under the FIRB program.

2 What is Grid Computing?

The grid is defined [24] as a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. It became the main computing paradigm for resource-intensive scientific applications but it also promises to be a successful platform for commercial environments [25, 26]. This trend is confirmed by an alliance of leading companies (AMD, Ascential Software, EMC, Cassatt, Citrix, Data Synapse, Enigmatec, Force 10 Networks, Fujitsu, Siemens Computers, HP, IBM, Intel, NEC, Network Appliance, Novell, Oracle, Optena, Paremus, SAS, Sun Microsystems, and Topspin) in different consortiums with the purpose to research and develop standards and open grid technologies [4, 33], to develop enterprise grid solutions, and to accelerate the deployment of grid computing in enterprises [2].

Various types of service providers (SPs) (e.g., web-hosting SPs, content distribution SPs, application SPs, and storage SPs) and enterprise applications involved in B2B processes could benefit from the use of grid solutions for sharing computational resources, storage resources, large datasets, knowledge, and expertise. Enterprises can use a data grid [20] to share large datasets and storage resources internal and/or external to their world-wide networks.

A computing grid can be used to share CPU cycles among mainframes, servers, and workstations in a dynamic fashion. This approach enables scaling of the computational power of an enterprise or service provider as the workload demand grows. A grid allows for the integration and coordination of resources belonging to different domains, characterized by different resource management and security policies, thus providing a secure virtual environment.

The main goal of these dynamically created virtual environments, defined in [27] as virtual organizations (VOs), is to provide nontrivial quality of service both for customers and for enterprise users: faster response to changing business needs, better utilization and service level performance, and lower IT operating costs.

The creation of VOs is based on interoperability, which means common protocols defining the basic mechanisms by which sharing relationships are exploited. In "The Anatomy of the Grid" [27], the authors specify the building blocks of a grid: protocols, services, Application Programming Interfaces (APIs), and Software Development Kits (SDKs). Protocols must be open and standard in order to facilitate extensibility, interoperability, and portability. Standard protocols make it easy to define standard services that provide enhanced capability.

APIs and SDKs make it possible to design grid applications. A grid uses the Internet infrastructure and thus its architecture is multilayered in nature. The layers are: fabric, connectivity, resource, collective, and application. Figure 1 shows the relationship between the grid layers and the architecture of Internet protocols [27].

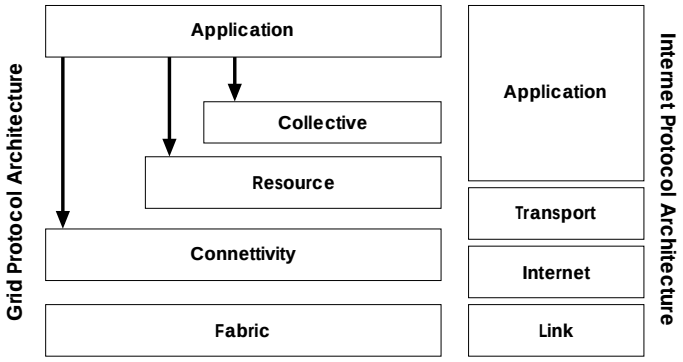

Figure 1: Grid layered architecture.

The *fabric layer* provides the resources to which shared access is mediated by grid protocols. Resources may be physical entities (e.g., computational and network resources, storage systems, sensors) or logical entities (e.g., distributed file systems, clusters, geographically distributed systems). Each resource implements its specific management functionality, to be integrated in a grid environment. For example, computational resources must include mechanisms to start programs and for monitoring and controlling their execution, query functions to determine the state of a resource and resource characteristics. However, if a vendor does not support this functionality (e.g., state query service), the grid platform must provide a capability to try to discover the resource structure and state information and pack them in a standard form.

The *connectivity layer* functionality provides communication and authentication protocols required for grid-specific transactions. Communication protocols allow for the exchange of data between fabric layer resources and include transport, routing, and naming protocols (e.g., IP, ICMP, TCP, UDP, DNS, OSPF, and RSVP). Authentication protocols provide secure mechanisms for verifying the identity of users and resources, and must rely on existing standards (e.g., X.509, TLS, SSL, and Kerberos). Secure environments and high levels of QoS require services and mechanisms such as single sign-on [11], delegation [31, 36], integration with various local security solutions (e.g., Kerberos and UNIX security), and user-based trust relationships.

The Grid Security Infrastructure (GSI) [29] is a public-key based suite of protocols for authentication, data encryption, and authorization. GSI relies on TLS, Kerberos technologies, and on the Generic Authorization and Access Control Interface (GAA-API) [47]

The *resource layer* defines protocols, APIs, and SDKs for secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources. This layer facilitates information gathering and control functionality at the level of individual resources. The Grid Resource Information Protocol (GRIP) and Grid

Resource Registration Protocol (GRRP) are examples of information gathering and management protocols. The Grid Resource Access Management (GRAM) protocol and GridFTP [15] are examples of resource management and access protocols. Protocols at the resource level invoke fabric layer functionality to access and control resources.

The *collective layer* defines protocols, APIs, and SDKs that allow interaction across collections of resources. This level provides services that include directory services to discover resources [22], software discovery services [19, 46], co-allocation, co-reservation, scheduling and brokering services [12, 16, 17, 30], monitoring and diagnostics services, data replication services [14], community accounting and payment services, collaboration, workload management systems and collaboration frameworks services [21, 38], and community authorization services.

On top of the grid stack resides the *application layer* that comprises user applications operating within a VO. Applications are designed using services (through APIs and SDKs) provided by the grid platform, i.e., an implementation of the grid protocol stack. The main grid platforms are the Globus Toolkit [34], the product of the research and development efforts of the Globus Alliance [33] and IBM's grid Toolbox [37], an open solution proposed by IBM and based on the Globus Toolkit (IBM is a member of the Globus Alliance).

Researchers and computational scientists have been heavy users of grid technologies. The following are some success stories. The Grid Portal Development Kit [35] is an open source application to create science portals. CACTUS [18] is an open source problem solving environment designed for scientists and engineers. The Biology WorkBench [49] is a web-based tool that allows biologists to search many popular protein and nucleic acid sequence databases. Entropia Inc's FightAIDSAtHome [23] is a system to analyze AIDS drug candidates (and in 2001, it was used to solve a particular instance of an optimization problem, "Nug30" [8]). Analysis of the many petabytes of data to be produced by the Large Hadron Collider (LHC) and other future high-energy physics experiments will require the marshalling of tens of thousands of processors and hundreds of terabytes of disk space to hold intermediate results [5, 7]. Scientific instruments such as telescopes, synchrotrons, and electron microscopes generate raw data streams that are archived for subsequent batch processing. But quasi-real-time analysis can greatly enhance an instrument's capabilities [6, 9]. Scientists often want to aggregate not only data and computing power, but also human expertise [1, 3, 10].

The Enterprise Grid Alliance (EGA) is an open, independent, and vendor-neutral community addressing near-term requirements for deploying commercial applications in a grid environment. By focusing exclusively on the needs of enterprise users, EGA will enable businesses to realize the many benefits of grid computing such as faster response to changing business needs, better utilization and service level performance, and lower IT operating costs.

IBM designs and develops grid solutions for different kinds of commercial applications such as on-line game providers, insurance companies, business decision maker-companies, and petroleum industries. Oracle's Grid Technology Center developed the Oracle Globus Toolkit to easily integrate Oracle and Globus technologies. SAS joined the Grid Computing Forum (GCF) to develop applications for grid environment.

3 A Motivating Example

To motivate the discussion and illustrate the concepts presented in the remaining sections, we consider a large insurance company (IC) that offers many types of insurance products (e.g., automobile, boat, home, and business) [42]. The primary goal of the IC is to increase its profit by minimizing risks and attracting/retaining more customers. In the insurance business, the premiums paid by customers are a function of the risk posed by the insurance policy. Traditionally, insurance companies use a combination of actuarial data with some minimal amount of personal data to establish the risk associated with a policy and thus its premium. If the risk assessment yields an excessive risk, premiums increase and the insurance company may lose customers.

In an effort to increase its profits, the insurance company in this motivating scenario is moving towards a highly customized risk assessment model (RAM). Under this approach, a much larger number of information sources about a customer are queried to obtain a much richer set of inputs to the RAM. Contrary to the previous model that places a customer under a large category (e.g., all non-smoking male straight-A college students under the age of 25), the new model provides a risk rating specific to a given customer (e.g., John Doe who besides being a non-smoking male straight-A college student under the age of 25, is 24 years old and is a Ph.D. student at the Computer Science Department at Berkeley, is a member of the Association for Computing Machinery (ACM), a member of a programming team that won a regional prize in an ACM-sponsored programming contest, has a very good credit record, undergoes a physical exam every year and is in perfect health, and has a clean record with federal, state, and local law enforcement agencies). Clearly, these customized models are much more sophisticated and significantly more compute- and data-intensive. These models are decomposed into many parallel tasks that run at the various resources provided by the computing grid.

The IC plans to establish a Web portal through which prospective customers can obtain insurance policy quotes. Users will be able to request three different types of quotes: immediate, non-immediate, and delayed. Immediate requests use simpler RAMs and can return results in a few seconds. Non-immediate requests return results in a few

minutes while the user is still online and use RAMs that are more complex than the ones used by immediate requests. Finally, delayed requests may take hours to process and use fairly sophisticated RAMs. In the latter case, users are notified by e-mail with a link to a Web page that contains the details of their quote. Thus, customers can obtain potentially lower premiums if they are willing to wait longer for sophisticated and complex RAMs to execute.

The IC does not want to invest in additional computing resources to run the new models and decides to use a computing grid to harness unused cycles of all its computers (from desktops to mainframes) connected to its worldwide network. The grid that supports the IC application will schedule resources according to type and computing requirements of the different types of RAMs needed to evaluate the three categories of requests.

The new customized RAMs draw their inputs from a large number of public and private data sources. These sources include health insurance companies, law enforcement agencies, financial organizations, departments of motor vehicles, professional and scientific organizations, federal, state, and local government agencies, and weather-related sources used to assess the risk of home and business insurance policies. When a user logs in to the new application, he/she must present a certificate used for single sign-on [11]. Then, using the delegation and user proxy features of the Grid Security Infrastructure (GSI) [27], the risk assessment application can have access to user's private data on behalf of the user at the various databases owned and managed by a set of organizations selected by the user. This way, the customer selectively trades-in access to some of his/her private data for lower premiums.

We use Fig. 2 (next page) to illustrate a scenario in which an insurance quote is requested by a customer of the IC through its Web portal. The notation used in the figure is patterned after the data mining example presented in "Grid Services for Distributed System Integration" [25]. The Web portal application contacts the factories at the RAM and DB service providers to request the creation of an instance of a RAM and of a database to be used to evaluate the risk of the policy (1). These instances have a specified lifetime, which can be extended by the application if needed. A RAM object is then instantiated along with the appropriate database (2). As the RAM is evaluated, it will generate queries (3) to various database services belonging to financial organizations, law enforcement agencies, health insurance organizations, and others not shown in the figure. The results (4) populate the database at the database service provider. The RAM instance queries this database (5) and uses the results (6) to provide an insurance quote (7) to the Web portal application.

4 QoS Metrics in a Grid Environment

In this section we address issues that involve the understanding of QoS metrics in the context of grid environments. We use the IC scenario to illustrate the concepts discussed here.

We classify QoS metrics for the grid in two dimensions. The first dimension is the *type* of QoS metric and the second is the *layer* of the metric in the grid architecture as discussed in section 2.

We consider three types of metrics:

- *Latency:* this type of metric is related to the time it takes to execute a task and is measured in time units. Latency metrics can be defined at different granularities. For example, in the IC case, one may be interested in the average time it takes to complete immediate quote requests or in the 95-th percentile of the time to respond to non-immediate requests. These two examples are illustrative of QoS metrics at the application layer. Another example of a latency metric is the elapsed time to return a delayed quote to the user. Sophisticated RAMs are typically complex parallel jobs that are decomposed into tasks allocated to different computing resources in the grid. An example of a QoS metric at the collective layer is the average time to perform co-reservation and co-allocation of computing resources in order to run a RAM. At the resource layer, one may be interested in measuring the time it takes to access a specific computing resource through the GRAM protocol. An example of a latency metric at the connectivity layer is the time it takes to perform an authentication on behalf of a customer to a financial organization using the GSI. At the fabric layer, one may be interested in measuring the time taken by a database server at a health insurance organization to reply to a query.

- *Throughput:* is measured in units of work accomplished per unit time. There are many possible throughput metrics depending on the definition of unit of work. At the application layer one may be interested in the number of delayed quote requests processed per second. At the collective layer one may be interested in the number of queries per second that can be handled by directory services used to locate resources across different VOs. Examples of throughput metrics at the resource layer include i) the effective transfer rate in Kbytes/sec under the Grid-FTP protocol used to transfer files from different computes nodes involved in running RAMs and ii) the number of queries/sec that can be processed by the database server of a law enforcement agency needed by the RAM application. At the connectivity layer, one may be interested in measuring the throughput, in Kbytes/sec, of a secure connection between an instance of a RAM model and a database

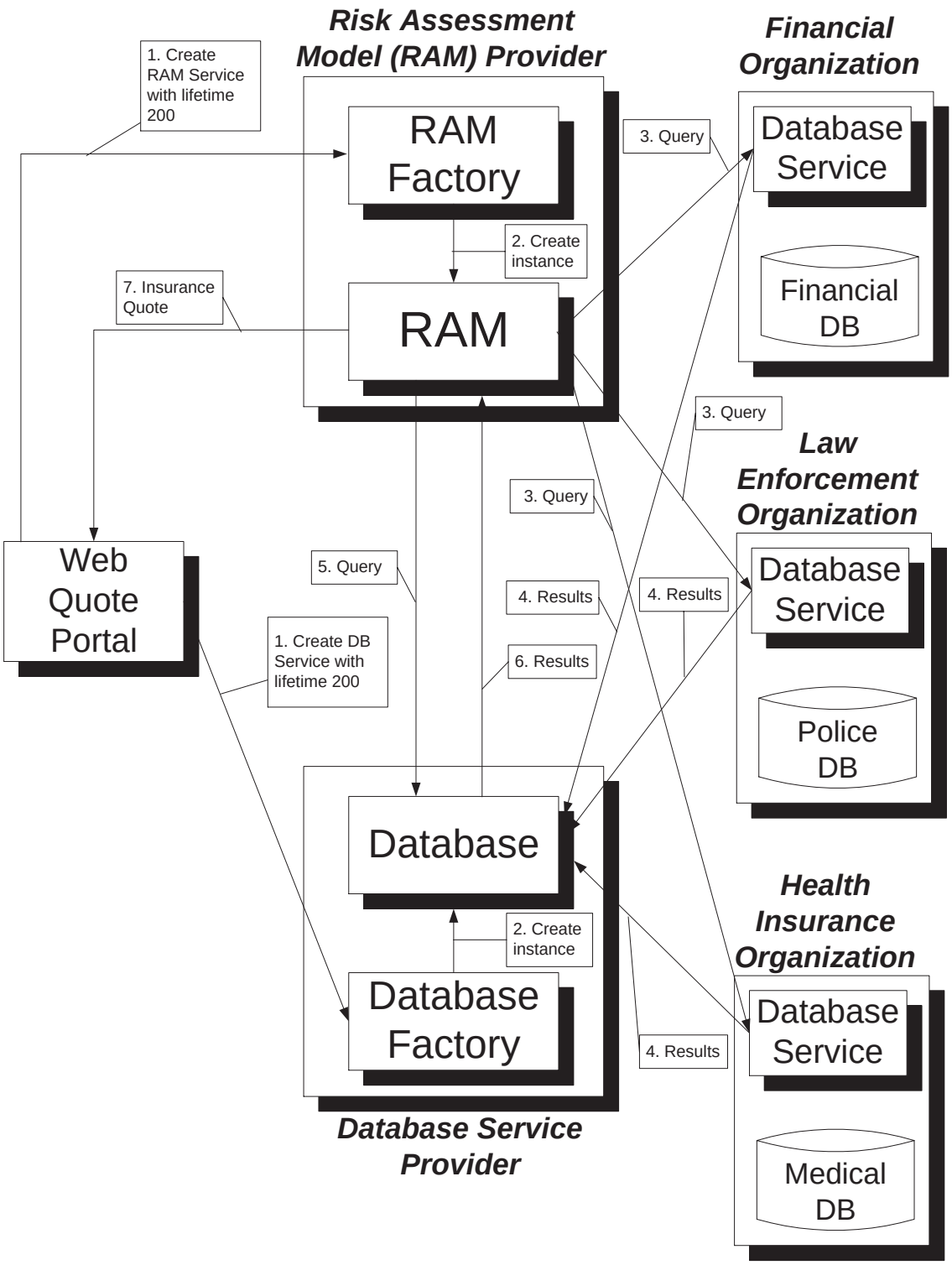

Figure 2: Quote request scenario.

Application	Customized Risk Assessment Model
Collective (application)	Checkpointing, failover, staging
Collective (generic)	Co-reservation and co-allocation of computing and storage resources
Resource	Access to data, access to computers, reservation and allocation of data storage resources
Connectivity	Authentication, authorization, processing resource discovery, data storage discovery, information source discovery, delegation, single sign-on
Fabric	Mainframes, servers, desktops, storage systems, networks

Table 1: Mapping of the IC application to the various layers of the grid architecture.

service that provides financial information about a customer. Finally, an example of a throughput metric at the fabric layer could be the number of CPU cycles per second obtained from a machine involved in processing a task that is part of a parallel RAM evaluation job.

- *Availability:* is defined as the fraction of time that a resource/application is available for use. In a multi-layer context such as the one described for the grid architecture, the notion of availability differs for each layer. At the application layer, one may define availability according to the type of request. For example, the availability of the immediate quote request application is the fraction of time that this application is available. At the collective layer, one may define availability in terms of the services, such as directories and brokering services, needed to locate computing resources to run a RAM instance. Availability at the resource layer may be measured for example as the fraction of time that a specific computing node is available for allocation to a RAM instance. At the connectivity layer, one may define availability as the percentage of time that a proxy authentication request succeeds in authenticating a user request with a law enforcement agency. Finally, an example of an availability metric at the fabric layer would be the fraction of time that a computing node is available during the the execution of a RAM instance.

It should be noted that QoS metrics at higher layers can often be expressed in terms of QoS metrics of the same type at lower layers of the grid architecture. For instance, the elapsed time to process a delayed quote request depends on many different latency metrics including (i) the time needed to reserve compute cycles for the duration of the evaluation of a RAM, (ii) the time to co-allocate these resources, (iii) the time to authenticate with database servers outside the insurance company, (iv) the time to obtain inputs from external databases, (v) the time to transfer data files to different nodes of a parallel job that will run the RAM, and (vi) the time to execute the RAM.

It is also important to consider the fact that different types of QoS metrics interact with one another [40]. For example, as the load increases, it may be more difficult to allocate resources to evaluate a RAM and some requests may have to be rejected, thus decreasing the availability of the application.

Table 1 shows a mapping of the IC application to the various layers of the grid architecture.

5 Grid Resource Allocation and SLAs

Service Level Agreements (SLA) are contracts between service providers and its users. These contracts specify acceptable levels for various QoS metrics and usually associate a cost with (and sometimes penalties for non-compliance for) a desired level of service. An important challenge in grid environments is that of enforcing SLAs when the resources used to establish a VO are shared by a variety of users. Yet, one of the key aspects of the definition of a grid environment is that non-trivial QoS should be provided.

Application-level SLAs have to be mapped into lower-level SLAs [39]. When negotiating SLAs with lower level resources, one must take into consideration that different service providers (e.g., a database service) may offer services at different levels (e.g., fast and slow response time) at different costs. Thus, an interesting optimization problem in a grid environment is how to select the service providers and the services within these service providers in such a way that the global SLA at the application layer is achieved with minimum cost.

Consider that the evaluation of a RAM requires NC millions of CPU cycles and that it has to be finished in at most $T_{\max}$ time units and the computation cost must not exceed $C_{\max}$ dollars. Assume that there are three available computing resources that can be used by the collective layer to co-reserve and co-allocate the evaluation of the RAM. Let, s_i $(i = 1, 2, 3)$ be the speed of resource i in 10^6 cycles/sec and c_i $(i = 1, 2, 3)$ be the cost, in dollars per second, to use resource i. Consider also that the evaluation of the RAM can be broken down into up to three independent parallel tasks. Thus, there are several possible co-allocations depending on how many and which computing resources are selected: (1), (2), (3), (1,2), (1,3), (2,3), (1,2,3).

Let N be the number of computing resources, NC_i be the number of cycles allocated to computing resource i $(i = 1, \cdots, N)$, T be the execution time of a given allocation, and C the cost of an allocation. Some allocations may not

be feasible in terms of satisfying the global execution time SLA, $T_{\max}$, or the global maximum cost $C_{\max}$. The following constraints apply to this resource allocation problem.

$$\sum_{i=1}^{N} NC_i = NC \tag{1}$$

$$T = \max_{i=1}^{N} \left\{ \frac{NC_i}{s_i} \right\} \leq T_{\max} \tag{2}$$

$$C = \sum_{i=1}^{N} \frac{NC_i}{s_i} \times c_i \leq C_{\max} \tag{3}$$

Equation 1 indicates that a feasible solution must allocate all NC cycles to the N compute resources. Equation 2 is the execution time constraint equation. Since all N tasks run in parallel, the total execution time is the maximum of the run time of the N tasks. Finally, Eq. 3 indicates that the total cost cannot exceed the cost constraint $C_{\max}$.
It is easy to show that if there is no cost constraint, the solution that minimizes the total execution time is the one that allocates more cycles to the faster resources in proportion to its speed:

$$NC_i = NC \times \frac{s_i}{\sum_{j=1}^{N} s_j}. \tag{4}$$

There are two possible optimization problems to be solved by the Connectivity layer of the co-allocation mechanism when cost is considered. The first optimization problem is "Find the feasible solution that satisfies the cost constraint $C \leq C_{\max}$ at minimum execution time." Consider the following numerical example to illustrate this problem: $N = 3$, $NC = 10^{13}$ cycles, $T_{\max} = 4,800$ sec, $C_{\max} =$ $1,500. The speed and cost values for the three compute resources are given in Table 2. We then solve, for each

Resource	Speed (10^6 cycles/sec)	Cost ($/sec)
1	1,000	0.10
2	2,000	0.25
3	3,000	0.60

Table 2: Input parameters for the optimization example.

combination of resource allocations, the optimization problem of finding the allocation of cycles to compute resources that minimizes the execution time T, while satisfying the cost constraint. The solutions, shown in Table 3, were obtained with MS Excel's Solver Tool. However, any other scientific package (e.g., MATLAB or Mathematica) could have been used. The first column of this table shows the compute nodes used, the next three columns indicate the number of cycles allocated to each in the optimal solution (if one exists), and the last two columns shows the resulting execution time and cost.

The first two allocations in Table 3 are not feasible because they violate the execution time constraint of a maximum of 4,800 seconds. The third and fifth allocations violate the maximum cost constraint of $1,500. The best solution is the one shown in the last row of Table 3: it has the lowest possible execution time T of $2,593$ seconds and a cost of $1,352.
A dual optimization problem is "Find the feasible solution that minimizes the cost time C and that satisfies the execution time constraint $T \leq T_{\max}$." The same input parameters used in the previous example yield the solution shown in Table 4. The allocations in rows 1, 2, and 5 of this table are not feasible because they violate the execution time constraint of a maximum execution time of 4,800 seconds. The allocations in rows 4 and 7 are the same and provide the minimum possible cost while preserving the execution time SLA. Note that compute resource 3 was not used in allocation (1,2,3) because of its higher cost relative to the others. Thus, it was feasible to meet the execution time SLA without using this resource. On the other hand, the same allocation in Table 3 uses this compute resource since the goal there is to minimize the response time while meeting the cost constraint.
Table 5 compares the only three feasible allocations for the time and cost optimization problems. The table shows the percentage of the maximum cost and maximum execution time for each allocation. For example, if all three compute nodes are used, the execution time optimization yields a solution with an execution time equal to 54% of $T_{\max}$ at 90% of the maximum cost $C_{\max}$. If the cost optimization problem is solved, the cost is 75% of $C_{\max}$ and results in an execution time equal to 90% of $T_{\max}$.
The size of the solution space for our optimization problem grows in a combinatorial way with the number of available resources. If a computation can be scheduled in any number of resources from 1 to N, the total number of possible allocations that have to be examined is $\left[\sum_{k=1}^{N} \binom{N}{k}\right] - 1 = 2^N - 1$. Thus, the resource broker must adopt heuristics to evaluate the optimal solution.
A real environment is more complex than the example just described. The computation may have a structure that reflects interdependencies among tasks. Other resources, such as network bandwidth, service providers, and data storage nodes, may have to be considered. Communication delays and coordination overheads have to be taken into account in these cases. These more general considerations are addressed in "A Framework for Resource Allocation in Grid Computing" [41].

6 QoS-related Mechanisms

This section presents a multi-level view of the main mechanisms to achieve QoS in grid environments. Each layer of the grid architecture must provide QoS functionalities to

Allocation	Cycle Allocation (NC_i)			T	C
	1	2	3	(sec)	($)
(1)	10,000,000	-	-	10,000	1,000
(2)	-	10,000,000	-	5,000	1,250
(3)	-	-	10,000,000	3,333	2,000
(1,2)	3,333,333	6,666,667	-	3,333	1,167
(1,3)	4,800,000	-	5,200,000	4,800	1,520
(2,3)	-	6,666,667	3,333,333	3,333	1,500
(1,2,3)	2,592,593	5,185,185	2,222,222	2,593	1,352

Table 3: Solutions to the optimization example of minimizing the total execution time.

Allocation	Cycle Allocation (NC_i)			T	C
	1	2	3	(sec)	($)
(1)	10,000,000	-	-	10,000	1,000
(2)	-	10,000,000	-	5,000	1,250
(3)	-	-	10,000,000	3,333	2,000
(1,2)	4,800,000	5,200,000	-	4,800	1,130
(1,3)	5,000,000	-	5,000,000	5,000	1,500
(2,3)	-	9,600,000	400,000	4,800	1,280
(1,2,3)	4,800,000	5,200,000	-	4,800	1,130

Table 4: Solutions to the optimization example of minimizing the total cost.

	Time optimization		Cost optimization	
Allocation	$\%T_{\max}$	$\%C_{\max}$	$\%T_{\max}$	$\%C_{\max}$
(1,2)	69	78	100	75
(2,3)	69	100	100	85
(1,2,3)	54	90	90	75

Table 5: Comparison between the cost and time optimization solutions.

achieve a global end-to-end QoS. Table 6 summarizes the main QoS mechanisms applicable at each layer of the grid architecture.

Layers	**Mechanisms**
Application	SLA specification, adaptation, reservation, admission control
Collective	Co-allocation and Co-reservation, resource brokering, admission control
Resource	local resource management, allocation and reservation, admission control
Connectivity	sign-on, authentication and delegation, DiffServ and Intserv
Fabric	scheduling, reservation, preemption, measurement, signaling

Table 6: A multi layer view of Grid QoS mechanisms

The design of applications requires a standard language to describe complex user-oriented SLAs, such as "What is the best insurance quote for me?", and it requires interfaces to find grid services to satisfy customer requests. For example, our RAM application must be able to translate a request such as "What is the best insurance quote for me?" into an SLA specification indicating that a delayed service is requested and that a set of services with some characteristics must be consumed. Moreover, the RAM application must also be able to adapt itself to the grid [28] status: the resources needed by the application could be modified dynamically by means of online control interfaces. Sensors allow the application to detect when adaptation is needed (e.g., when there will be more computation resources available to increase the degree of parallelism to solve a RAM instance). Decision procedures allow the application to express a rich set of resource management policies. The capability of the application layer to provide a detailed description of the SLAs allows a more fine grained resource selection on the grid, reducing latency and increasing availability and throughput.

Complex application-level SLAs must be mapped into resource-oriented SLA specifications at the Collective layer. For example, the customer request "What is the best quote for me?" requires that the RAM batch job spans through the grid infrastructure, and that all available trusted databases be queried to collect a rich set of input parameters for the RAM (the lower premium must be provided). Since delayed RAM requests have no stringent latency requirements, resource reservation is not required and the tasks can be scheduled as low priority jobs. However, the customer request "Give me an insurance premium quote in

real time" requires co-reservation and co-allocation of a set of resources and the use of the fastest service providers to access external data. Therefore, the resource broker must dynamically determine the set of resources that are able to satisfy the required SLA at minimum cost. This operation has an intrinsic latency (as outlined in section 5) but maximizes the IC throughput, reduces the IC costs, and reduces customer-perceived latency. Hence, co-allocation and co-reservation are the main QoS-related mechanisms at this level. Knowing in advance the availability of all the resources needed by an application allows the collective layer to apply global admission control mechanisms that prevent resources from being allocated if QoS goals cannot be achieved. This also helps to prevent overloading of resources and avoids congestion.

The resource layer must provide, through the GIS, information to the resource broker at the collective layer and must provide interfaces and mechanisms to reserve, allocate, and use resources. A detailed resource description, and up to date information, allows the resource broker to perform a fine grain selection to find the local SLAs that meet the desired quality of service at low cost. For example, the resource broker must know exactly the amount of processing cycles and memory available at each node, the number of active processes, and the list of pending jobs in order to select the most appropriate computing resources to solve a "real time" RAM instance. Knowing the version of the OS installed can help optimize the execution of a RAM sub-task. Knowledge of network state information helps to select the nodes that can best establish low latency communication with databases. All this information, provided by the local resource manager and processed by the request broker, must be combined with resource usage costs to find, for example, a computing resource capable of executing a RAM task i in a given time E_i and capable of transferring the task's output in a time T_i such that $E_i + T_i \leq T_{SLA_i}$, at the lowest cost.

To guarantee resource availability, maximize throughput and also reduce latency, admission control mechanisms must be provided by the local resource managers to prevent either congestion at a pool of local resources or unavailability of popular functionalities.

The connectivity layer must provide mechanisms to satisfy SLAs in terms of security requirements, communications latency and network availability. The connectivity infrastructure must allow users to create services dynamically and must be capable of interacting in a secure fashion with other services. The authentication of resources belonging to different domains (e.g., the law enforcement database) is a key issue; VO participants must be able to join dynamically and trust other members [48]. Besides, it is fundamental to negotiate different resource accesses on the basis of different policies, to guarantee data integrity, privacy, and confidentiality (the IC accesses and processes confidential information about its users). Mechanisms to meet security SLAs are mandatory, even if they impact negatively on the performance of the grid. The goal to share the network among different priority flows and at the same time provide QoS may be achieved by coupling IntServ and DiffServ services in an architecture named "IntServ over DiffServ" [13].

At the fabric layer, vendors must implement scheduling mechanisms for resources (e.g., CPU, storage boxes, and network bandwidth) that take into account different classes of requests with different SLAs. For example, a workstation or mainframe shared between an immediate-RAM sub-task and a delayed-RAM sub-task must be able to schedule appropriately the higher priority process to satisfy the stringent SLA. Moreover, by providing reservation services, a resource will enable the resource level to reserve and then to allocate the resource. Grid nodes must also provide, to the local resource manager, a detailed description of their status (availability, load state, pending jobs, and scheduled jobs) in a standard format. Resources must also be able to issue signals when a critical state is entered to enable the implementation of exception handling and fault tolerant mechanisms that improve the service availability.

7 Concluding Remarks

Grid architectures provide an interesting service-oriented platform for scientific and commercial applications. These platforms provide dynamic resource sharing using open standards providing a quality of service that could not be achieved otherwise. This paper investigated some of the relevant issues that must be considered in designing grid applications that deliver appropriate QoS for commercial applications: definition of metrics, relationships between resource allocation and SLAs, and QoS-related mechanisms.

Capacity planning, performance modeling, and analysis are still open and interesting problems for grid architectures. The main problem lies in the fact that resources are shared in a dynamic way. Moreover, the allocation and scheduling mechanisms used at the global and local level play an important role in the performance of grid applications. A model that takes into account the priority of local resource owners over global computations in a heterogeneous networked environment was considered in "Performance Prediction of Parallel Applications on Networks of Workstations" [43].

Static and dynamic heuristic scheduling mechanisms for heterogeneous multiprocessors that process parallel jobs represented as arbitrary task graphs was addressed in "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures" [45]. Studies of that nature have the potential to be applied with extensions and new assumptions to grid environments.

It is also relevant to consider how local resources should be designed so that they can plug-and-play in a grid environment in a way that QoS mechanisms at the local and global level can interact seamlessly. The framewok for QoS-aware

software components developed in "A Framework for QoS-aware Software Components" [44] could be useful when designing software for the grid.

Acknowledgements

The authors would like to thank Jonathan Gladstone for his useful edits on this paper.

References

[1] Biomedical informatics research network. www.nbirn.net/.

[2] Enterprise grid alliance. www.gridalliance.org.

[3] European union data grid. www.eu-datagrid.org/.

[4] Global grid forum. www.gridforum.org.

[5] Griphyn, the grid physics network. www.griphyn.org/.

[6] International virtual data grid laboratory. www.ivdgl.org/.

[7] Network for earthquake engineering simulation. www.neesgrid.org/.

[8] Nug30 project. www-unix.mcs.anl.gov/metaneos/nug30/.

[9] Particle physics data grid. www.ppdg.net/.

[10] Teragrid project. www.teragrid.org/.

[11] *The Open Group, Preliminary Specification: X/Open Single Sign-On Service (XSSO) - Pluggable Authentication*. 1997.

[12] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: a tool for performing parametrised simulations using distributed workstations. In *Fourth IEEE International Symposium on High Performance Distributed Computing - HPDC '95, Washington D.C.*, 1995.

[13] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. G-qosm: Grid service discovery using qos properties. *Computing and Informatics*, 21:363–382, 2002.

[14] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing Journal*, 28(5), May 2002.

[15] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. Gridftp protocol specification., 2002.

[16] J. Beiriger, H. Bivens, S. Humphreys, W. Johnson, and R. Rhea. Constructing the asci computational grid. In *Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, 2000.

[17] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level scheduling on distributed heterogeneous networks. 1996.

[18] CACTUS. Cactus. www.cactuscode.org.

[19] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. Technical Report CS-96-328, Knoxville, TN 37996, USA, 1996.

[20] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards and architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 2001.

[21] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi. Access grid: Immersive group-to-group collaborative visualization. In *4th International Immersive Projection Technology Workshop*, 2000.

[22] K. Czajkowski, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. 2001.

[23] Entropia. Fightaidsathome. Entropia Inc's, www.fightaidsathome.org/.

[24] I. Foster and C. Kesselman. *The grid: Blueprint for a new Computing infrastructure*. 1999.

[25] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6), 2002.

[26] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: an open grid services architecture for distributed systems integration, 2002.

[27] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[28] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *8th International Workshop on Quality of Service - IEEE*, 2000.

[29] I. T. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[30] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press*, 2001.

[31] M. Gasser and E. McDermott. An architecture for practical delegation in a distributed system. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Electrical and Electronics Engineering IEEE Inc.: Computer Society Press*, 1990.

[32] F. Gillett. Grids gets big, but the term is confusing. *Trends*, May 18, 2004.

[33] GlobusAlliance. The globus alliance. www.globus.org.

[34] GlobusAlliance. Globus toolkit. www-unix.globus.org/toolkit/.

[35] GPDK. Gpdk. http://doesciencegrid.org//projects/GPDK/.

[36] J. Howell and D. Kotz. End-to-End authorization. pages 151–164.

[37] IBM. "ibm grid toolbox". http://www-1.ibm.com/grid/developers.shtml.

[38] J. Leigh, A. E. Johnson, and T. A. DeFanti. Cavern: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. 2(2), 1997.

[39] D. Menascé. Mapping service level agreements in distributed applications. *IEEE Internet Computing*, 8(5), 2004.

[40] D. Menascé. Performance and availability of internet data centers. *IEEE Internet Computing*, 8(3), 2004.

[41] D. Menascé and E. Casalicchio. A framework for resource allocation in grid computing. In *Proc. 12th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2004.

[42] D. Menascé and E. Casalicchio. Qos in grid computing. *IEEE Internet Computing*, 8(4), July/August 2004.

[43] D. Menascé and A. Rao. Performance prediction of parallel applications on networks of workstations. In *Proceedings of the Computer Measurement Group Conference*, San Diego, CA, USA, 1996.

[44] D. Menascé, H. Ruan, and H. Gomaa. A framework for qos-aware software components. In *Proceedings of the 2004 ACM Workshop on Software and Performance*, San Francisco, CA, USA, 2004.

[45] D. Menascé, D. Saha, S. Porto, V. Almeida, and S. Tripathi. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. *J. Parallel and Distributed Computing*, 1(28), 1995.

[46] H. Nakada, M. Sato, and S. Sekiguchi. Design and implementations of ninf: towards a global computing infrastructure. 1999.

[47] U. of Southern California Information Science Institute. Generic authorization and access control interface. www.isi.edu/gost/info/gaaapi/.

[48] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. Security for grid services. In I. Press, editor, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June.

[49] WorkBench. http://workbench.sdsc.edu.