

# QoS Support for High-Performance Scientific Grid Applications

Rashid Al-Ali,<sup>1,2</sup> Gregor von Laszewski,<sup>1</sup> Kaizar Amin,<sup>1,3</sup> Mihael Hategan,<sup>1</sup>  
Omer Rana,<sup>2</sup> David Walker,<sup>2</sup> and Nestor Zaluzec<sup>1</sup>

<sup>1</sup> Argonne National Laboratory, U.S.A.   <sup>2</sup> Cardiff University, UK.   <sup>3</sup> University of North Texas, U.S.A.

## Abstract

*The Grid approach provides the ability to access and use distributed resources as part of virtual organizations. The emerging Grid infrastructure gives rise to a class of scientific applications and services to support collaborative and distributed resource-sharing requirements as part of teleimmersion, visualization, and simulation services. Because such applications operate in a collaborative mode, data must be stored and delivered in timely manner to meet deadlines. Hence, this class of applications has stringent real-time constraints and quality-of-service (QoS) requirements. A QoS management approach is required to orchestrate and guarantee the interaction between such applications and services. In this paper we discuss the design and prototype implementation of a QoS system and show how we enable Grid applications to become QoS compliant. We validate this approach through a case study of nanomaterials. Our approach, enhances the current Open Grid Services Architecture. We demonstrate the usefulness of the approach on a nano materials application.*

## 1 Introduction

Advanced commercial and scientific applications often require high-performance, high-end resources that are both expensive and scarce. Based on the Grid approach [1] the emerging Grid infrastructure [2] is making these resources available to service providers and users, but effective use of such resources requires mechanisms to manage sophisticated Grid environments. Consequently, considerable effort has gone into security issues, resource scheduling, and complex execution frameworks. Although the Grid community has collectively made considerable progress, only recently has consideration been given to a fundamental problem prevailing in service-oriented architectures: providing deterministic quality-of-service (QoS) assurances to service consumers. Providing nontrivial QoS is one of

the primary goals of the Grid approach. Nevertheless, most Grid environments operate on a best-effort basis, sharing the Grid resources among its users with equal priority. A considerable degradation in performance and efficiency must be addressed when a large number of requests are issued for shared Grid resources.

To address this problem, we concentrate on ways to increase the collective efficiency a term referred to by von Laszewski as the efficiency of scientists collaboratively using these resources. One practical solution is to introduce QoS mechanisms that enable service providers to partition their services based on quality criteria such as priority, fairness, and economic gain. In other words, a QoS-aware Grid infrastructure can offer deterministic QoS assurances based on a particular criterion, rather than on a best-effort service.

Consider the following scenario [3]. A group of scientists want to conduct a collaborative simulation experiment for a specific period of time using high-end resources, such as an electron microscope, accessible through a Grid infrastructure. The experiment requires access to resources suitable for fine-grained, real-time computation and requires a specific network bandwidth to connect the sites during the experiment. After the experiment is concluded, the data must be moved to a large-capacity, shared data-store, allowing access to the data by groups of scientists not located at the electron microscope site. Clearly, this scenario has a number of requirements that a non-Grid-based resource management or scheduling system cannot fulfill.

Motivated by such a scenario, we propose a comprehensive QoS management architecture in service-oriented Grids, called Grid Quality of Service Management (G-QoS<sub>m</sub>). We validate our proposed architecture with a proof-of-concept prototype implementation and show its effectiveness in a scientific application involving nanomaterials.

The paper is structured as follows. In Section 2 we provide an overview of quality of service in networking and distributed computing. In Section 3 we

outline the general requirements of a Grid QoS management system and give an overview of existing Grid QoS systems. In Section 4 we present G-QoS and its major components. In Section 5 we discuss a typical high-performance Grid application and outline its QoS requirements. In Section 7 we discuss performance results based on executing applications with QoS support. We conclude the paper with a summary of future work.

## 2 Quality-of-Service: Background and Terminology

Quality of service has been explored in various contexts [4, 5]. Two types of QoS can be distinguished: quantitative and qualitative characteristics of the Grid infrastructure. Qualitative characteristics refer to elements such as service reliability and user satisfaction regarding service. Quantitative characteristics refer to elements such as networks, CPUs, or storage. For example, the following are quantitative parameters for network QoS: *Delay*, the time it takes a packet to travel from sender to receiver. *Delay jitter*, the variation in the delay of packets taking the same route. *Throughput*, the rate at which packets go through the network (i.e., bandwidth). *Packet-loss rate*, the rate at which packets are dropped, lost, or corrupted.

Together, these four parameters form the network QoS measurement matrix.

CPU, or compute, QoS can be divided into shared and exclusive categories [6]. In shared systems, in which more than one user-level application shares the CPU, the application can specify a percentage of the CPU. In exclusive systems, in which usually one user-level application has exclusive access to one or more CPUs, the application can specify the number of CPUs as the QoS parameter.

Storage QoS is related to device such as memory and disks. In this context, QoS is specified by bandwidth and space. Bandwidth is the rate of data transfer between the storage devices to the application. Space is the amount of storage space that the application can use for writing data.

Usually, applications specify two QoS requirements: the characteristics of the resource and the period the resource is required. Reservation involves giving the application the confidence, or assurance, that the resource allocation will succeed with the required level of QoS when needed. The reservation can be immediate or in advance, and the duration of the reservation can be definite (for a defined period of time) or indefinite (for a specified start time and unlimited duration).

## 3 QoS in Grid Computing

The Grid approach can be seen as a global-scale distributed-computing infrastructure with coordinated resource sharing [1, 2]. The fundamental Grid problem that many researchers have been investigating is resource management, specifying how Grid middleware can provide resource coordination for client application transparently. One of the most successful middleware projects that provides such coordination is the Globus Alliance [7]. The availability of Grid middleware tools, such as the Globus Toolkit, facilitates persistent access to Grid services. The use of Grid middleware has expanded from scientific applications to business-oriented disciplines while envisioning a service-oriented architecture to build sophisticated Grid applications with complex Grid resources requirements.

In most Grid settings, Grid applications submit their requirements to Grid resource management services that schedule jobs as resources become available. Some classes of applications cannot wait for resources to become available, however. For these applications, it must be possible to reserve Grid resources and services at a particular time (in advance or on-demand). In addition, other features are highly desirable, indeed required, if the Grid resource management service is to be able to handle complex scientific and business applications. We review these requirements in the next subsection and then briefly discuss how well current QoS systems meet these requirements.

### 3.1 Requirements

A Grid resource management system should adhere to the following requirements that relate to QoS issues.

**Advance Resource Reservation.** The system should support mechanisms for advance, immediate, or ‘on-demand’ resource reservation. Advance reservation is particularly important when dealing with scarce resources, as is often the case with high-performance and high-end scientific applications in Grids.

**Reservation Policy.** The system should support a mechanism that facilitates Grid resource owners enforcing their policies governing when, how, and who can use their resource, while decoupling reservation and policy entities, in order to improve reservation flexibility [8].

**Agreement Protocol.** The system should assure the clients of their advance reservation status, and the resource quality they expect during the service session. Such assurance can be contained in an agreement protocol, such as Service Level Agreements (SLAs).

**Security.** The system should prevent malicious users penetrating, or altering, data repositories that hold information about reservations, policies and agreement protocols. In addition to a secure channel between the clients and applications and the Grid resources, a proper security infrastructure is required.

**Simplicity.** The QoS enhancement should have a reasonably simple design that requires minimal overheads in terms of computation, infrastructure, storage and message complexity.

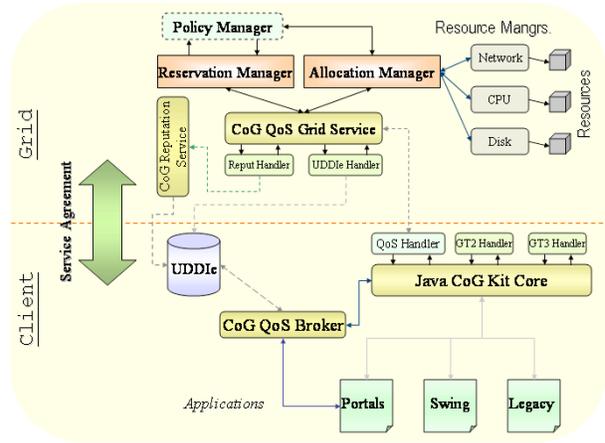
**Scalability.** The approach should be scalable to a large number of entities, since the Grid is a global-scale infrastructure, and there will be dynamic resources and users joining the Grid.

### 3.2 Current QoS Systems

QoS in Grids is actively being researched. However, the only substantial system developed within the Grid community (to our knowledge) is the General-purpose Architecture for Reservation and Allocation (GARA). GARA [9] provides programmers a convenient access to end-to-end QoS. It provides advance reservations, with uniform treatment to various types of resources such as network, compute, and disk. GARA’s reservation is a promise that the client or application initiating the reservation will receive a specific quality of service from the resource manager. GARA also provides an application programming interface to manipulate reservation requests, such as *create*, *modify*, *bind*, and *cancel*. GARA uses the Dynamic Soft Real-time (DSRT) scheduler [10] as the underlying compute resource manager, and it uses Cisco routers to deliver network QoS.

Although GARA has gained popularity in the Grid community, it has certain limitations in coping with current application requirements and technologies:

- Most current applications employ the emerging new technology of the Web services and the Open Grid Service Architecture (OGSA) [11]. Unfortunately, GARA is not OGSA-enabled, and OGSA-enabled applications cannot therefore leverage GARA services.
- Grid applications require the simultaneous allocation of various resources. This process is performed by the resource manager that contacts the required resources and possibly reserves the resources for future allocation. An agreement protocol should exist to inform the application about the resources negotiated for allocation and the level of quality the application expects. This information is usually encapsulated in a Service Level



**Figure 1. The G-QoS architecture with an OGSA-enabled QoS service.**

Agreement. GARA does not support the concept of an agreement protocol and cannot distinguish whether an application requires a CPU and a network resource. The application has to perform two separate calls to GARA and, on success, receives two different handlers. It is the application’s responsibility to manage these handlers when claiming the resources.

- QoS monitoring and adaptation during the active QoS session is one of the most important and successful mechanisms to provide a quality guarantee. GARA is not tooled with adaptive functions [12].

The most significant limitation, however, is that GARA is at this time no longer actively being developed.

## 4 Grid QoS Management

Grid Quality of Service Management (G-QoS) is an evolving approach to support QoS management in computational Grids in the context of the Open Grid Service Architecture (OGSA) [13, 14]. G-QoS consists of three main operational phases: establishment, activity, and termination. During the *establishment* phase, a client’s application states the desired service and QoS specification. G-QoS then undertakes a service discovery, based on the specified QoS properties, and negotiates an agreement offer for the client’s application. During the *activity* phase, additional activities, including QoS monitoring, adaptation, accounting and possibly renegotiation, may take place. During the *termination* phase, the QoS session is ended,

through resource reservation expiration, agreement violation, or service completion; resources are then freed for use by other clients. Our framework supports these three phases through interaction among components, as depicted in Figure 1. In the next section we describe these interactions and highlight service provision for Grid applications.

#### 4.1 QoS Grid Service

The basic building block of our architecture is the QoS Grid service (QGS), an OGSA-enabled Grid service providing QoS functionalities such as negotiation and reservation. Each QoS-enabled resource is accessed through a QGS. Since QGS is a Grid service, it publishes itself to a QoS registry service. In addition to the QoS functionalities, it supports two types of resource allocation strategy:

- resource domain, which provides compute, network, and disk QoS with fine-grained specifications, and

- time domain, where the whole Grid node, in which the QGS resides, can be reserved for a defined period of time.

This functionality is enabled by ensuring that all requests must be issued through the QGS. Further, the QGS interacts with a number of modules to deliver QoS guarantees. These modules are the QoS Handler, reservation manager, allocation manager, and the QoS registry service (see Figure 1). Currently, G-QoS supports compute resource-based QoS; we have begun integrating network and disk support.

**QoS Handler** Access to Grid services is enabled through the Java CoG Kit, which provides a convenient abstraction to Grid services for Globus Toolkit versions 2 and 3. The Java CoG Kit introduced the concept of task handlers, which allow us to build QoS-enhanced abstractions for computational tasks (see Section 4.2). Hence, we have introduced a QoS Handler as a link with the Java CoG Kit Core. The QoS Handler implements the required QoS action, encapsulated in the Java CoG Kit Task object, such as QoS negotiation request or QoS job submission. We have enhanced the Task object with QoS-related parameters dependent on the Task action required; for example, in the case of a negotiation request, parameters include start time, end time, resource type, and specifications. Once the Task object has been specified, the QoS Handler is, for example, delegated, on behalf of the client or application, to negotiate QoS requests. In this case the QoS Handler is seen as the client, from the QGS point of view.

This is a useful approach especially when the application requires more than one Grid resource. All the application needs is to instantiate the required number of QoS Handler objects, submit the Task object to the handlers, and let the handlers negotiate QoS requests with the QGS to return an agreement. Furthermore, in a QoS-enabled job submission through the interactive mode, the QoS Handler listens for notifications of job status, with the notification implemented by the QGS as an OGSA notification. We plan to align ourselves with the GGF Grid Resource Agreement and Allocation Protocol (GRAAP) Working Group [15] that is defining a WS-Agreement protocol meant to address machine-to-machine negotiations.

**Reservation Manager.** The reservation manager is based around a data structure that supports reservations for quantifiable resources; resources associated with defined capacities. The reservation manager is decoupled from the underlying resources and does not have direct interaction with them. However, it obtains resource characteristics, and policies governing resource usage, from the policy manager. The policy manager, on the other hand, is responsible for validating reservation requests by applying domain-specific rules, established by the resource owners, on when, how, and by whom the resource can be used. In brief, when the reservation manager receives a reservation request from the QGS, it contacts the policy manager for validation and then performs admission control to check the availability of the requested resource. Upon success, it returns a positive reply to the QGS, which allows the QGS to propose a negotiable service agreement offer.

**Allocation Manager.** The Allocation Manager's primary role is to interact with underlying resource managers for resource allocation and deallocation and to inquire about the status of the resources. It has interfaces with various resource managers, namely, the Dynamic Soft Real Time Scheduler (DSRT) [10] and Network Resource Manager (NRM); we are also investigating *Nest* as the disk storage resource manager [16]. When the allocation manager receives resource allocation request from the QGS, it forwards the request to the designated underlying resource manager. The Allocation Manager interacts with adaptive services to enforce adaptation strategies; for details, see [12].

**QoS Registry Service.** Since the framework operates in the OGSA architecture, the QGS and other Grid services in the OGS container should be published in some registry service so they can be known

by others. Service publishing, in this discussion, does not mean simply publishing a service name, URL, and basic description. For example, for QGS, it includes information on what QoS-enabled service it offers, what allocation strategies it employs, and what classes of network QoS it offers (e.g., best effort, controlled load, or guaranteed). For other Grid services, service publishing includes information about QoS properties such as performance characteristics and service execution requirements. To date, we have used an extended version of the Universal Description Discovery and Integration as part of our QoS registry service. The UDDIe [17] is a Web services registry system, which provides service providers a means to publish their services with QoS properties and, hence, to search for these services based on the QoS properties.

## 4.2 Java CoG Kit Core

The Java CoG Kit Core (cog-core) [18] component offers a technology- and architecture-independent abstraction layer that provides true interoperability across multiple Grid implementations. Cog-core provides convenient APIs for Grid applications to access the underlying Grid technology. Furthermore, cog-core offers several useful abstractions reusable as part of a Quality of Service framework including Grid tasks and their corresponding handlers.

**Task.** The Java CoG Kit defines a *Task* as an atomic unit of execution. A task is not limited to executing a job. Instead, it abstracts the generic Grid functionality, including authentication, remote job execution, file transfer request, or information query. It has a unique identity, a security context, a specification, and a service contact.

The task identity helps in uniquely representing the task across the Grid. The security context represents the abstract security credentials of the task. This abstraction makes it possible to integrate a variety of different own security context as part of the Grid infrastructure. Hence, the security context in cog-core offers a common construct that can be extended by the different implementations to satisfy the corresponding back-end requirement. The specification represents the actual attributes or parameters required for the execution of the Grid-centric task. The generalized specification can be extended for common Grid tasks such as remote job execution, file transfer, and information query. The service contact associated with a task symbolizes the Grid resource required to execute it.

**Handlers.** The *Task Handler* provides a simple interface to handle interaction with a generic Grid task. It categorizes the tasks and providing the appropriate functionality. For example, the task handler will handle a remote job execution task differently from a file transfer request task. This approach does not impose any restrictions on the implementation of the task handler as long as its working is transparent to the end user. This module is back-end-specific and has a separate implementation for each abstraction it supports.

## 4.3 Application Integration

We have prototyped an implementation to demonstrate the ease of use and effectiveness of a Grid application using QoS functions. To enable other Grid applications to use the QoS-enabled framework, one needs to conduct the following simple steps: (a) create a task object based on cog-core, (b) depending on the type of the required QoS function, set up the necessary objects for security, job specification, and service contact, (c) instantiate a QoS Handler, and (d) Associate the created task with the QoS Handler, and submit the task.

Figure 2 shows a Java code fragment demonstrating how applications can generate QoS negotiation request, QoS job submission, and task submission to a QoS handler, respectively.

The concept of abstracting the QoS services and interacting with the QGS by creating a task (i.e., QoS function) and submitting it to a QoS Handler has a great advantage when dealing with multiple distributed Grid resources: it makes the design and specification of abstract QoS-based brokers easier.

## 5 Application Case Study: Nanoscale Structures

To validate our QoS approach, we applied our reference implementation and prototyped a Grid computing environment for the analysis of a nanoscale structures application. This application involves a new experimental technique, position-resolved diffraction, being developed as part of Argonne National Laboratory's advanced analytical electron microscope program [19]. With this technique, a focused electron probe is sequentially scanned across a two-dimensional field of view of a thin specimen. At each point on the specimen a two-dimensional electron diffraction pattern is acquired and stored.

Analysis of the spatial variation in the electron diffraction pattern of each measured point allows the

```

/** QoS: Prepare Negotiation Task */
private void prepareQosNegotiationTask() {
    // create a QoS service, and setup QoS attributes
    Task task =
        new QosTaskImpl("myTask", QoS.NEGOTIATION);
    this.task.setAttribute("startTime", startTime);
    this.task.setAttribute("endTime", endTime);
    this.task.setAttribute("allocStrategy", strategy);
    this.task.setAttribute("cpu_capacity", cpuCapacity);

    // create a Globus version of the security context
    SecurityContextImpl securityContext =
        new GlobusSecurityContextImpl();
    // selects the default credentials
    securityContext.setCredential(null);
    // associate the security context with the task
    task.setSecurityContext(securityContext);

    // create a contact for the Grid resource
    Contact contact = new Contact("myGridNode");

    // create a service contact
    ServiceContact service =
        new ServiceContactImpl(qosServiceURL);
    // associate the service contact with the contact
    contact.setServiceContact("QGSurl", service);

    // associate the contact with the task
    task.setContact(contact);
}

/** QoS: Prepare Job Submission Task */
private void prepareQosJobSubmissionTask() {
    // create a QoS JobSubmission Task
    Task task =
        new TaskImpl("myTask", QoS.JOBSUBMISSION);
    this.task.setAttribute("agreementToken", token);

    // create a remote job specification
    JobSpecification spec = new JobSpecificationImpl();

    // set all the job related parameters
    spec.setExecutable("/bin/myExecutable");
    spec.setRedirected(false);
    spec.setStdOutput("QosOutput");

    // associate the specification with the task
    task.setSpecification(spec);

    // create a Globus version of the security context
    SecurityContextImpl securityContext =
        new GlobusSecurityContextImpl();
    securityContext.setCredential(null);
    task.setSecurityContext(securityContext);

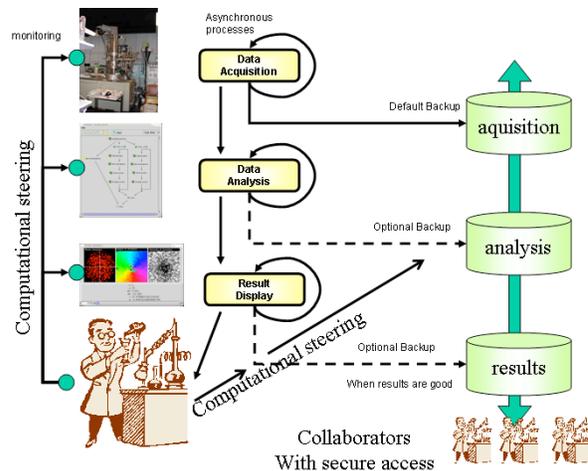
    Contact contact = new Contact("myQoScontact");

    ServiceContact service =
        new ServiceContactImpl(qosServiceURL);
    contact.setServiceContact("QGSurl", service);
    task.setContact(contact);
}

/** QoS: Task Submission to QoS Handler */
private void QosTaskSubmission(Task task) {
    TaskHandler handler = new QosTaskHandlerImpl();
    // submit the task to the handler
    handler.submit(task);
}

```

**Figure 2. A sample code fragment for QoS negotiation, and task submission to QoS handler**



**Figure 3. Asynchronous processes define a workflow steered by the scientist to support the problem-solving process with the help of abstract Grid tasks.**

researcher to study subtle changes resulting from microstructural differences, such as ferro- and electromagnetic domain formation and motion, at unprecedented spatial scales. As much as one terabyte of data can be taken during such an experiment. The analysis of this data requires a resource-rich Grid infrastructure to accommodate real-time constraints. Results need to be archived, remote compute resources need to be reserved and made available during an experiment, and the data needs to be moved to the compute resources where they will be analyzed. Moreover, results need to be gathered and presented in a form that is meaningful to the scientist.

The need for a flexible infrastructure is demonstrated through a simple flow diagram depicted in Figure 3. The elementary logic of the instrument control can be expressed as a sequence of processes that depend on each other: (a) Data acquisition: gathers time-delayed images from the electron microscope. (b) Backup: backs up the incoming data. (c) Data analysis: performs scientific calculations on the time delayed images. (d) Result display: gathers the results from the data analysis, in a form easy to interpret, to enable further judgments for steering the experiment.

The nanostructures application presents one of many scientific use patterns that occur in high-end instrument scenarios. The pattern includes a high volume of interaction during an experiment that must be dealt with in an adaptive and flexible way. Unexpected and unpredicted experiment conditions must be considered, and the instrument operator's interface to the

Grid must be as simple as possible while at the same time providing needed flexibility to interactively modify the experiment setup.

The Java CoG Kit provides a convenient abstraction for formulating these tasks while reusing the patterns for file transfer, job execution, and job management. At the same time it hides much of the complexity, which the Grid application developer may not want to see. To provide the necessary flexibility, we plan to develop graphical components for the Java CoG Kit and integrate them in a problem-solving environment that targets the use of a scientific instrument. Through this interface, the scientist will be able to interact easily with the experiment resources and decide when, what, and where data gathered during the course of the experiment is backed up. Image filters and monitors, plugged dynamically into the workflow for image analysis, help to validate the correctness and usefulness of the running experiment. Since the sample in the instrument may require specialized and individual filters, the experiment operator must be given a methodology that allows their easy creation and adaptation. Because of the focus on the experiment itself, the use of the Grid should be through abstractions as much as possible. Based on the application description, we derive the following requirements for QoS: (a) Network requirements to transfer the time-delayed images from the electron microscope as part of the data acquisition process. (b) Disk storage to cache quickly incoming data during the acquisition process and the availability of large storage for a backup process. (c) Computation power to process the scientific calculations on the time-delayed images in real time, as new images become available in the data analysis process. (d) Collect results produced by the data analysis process and transfer them to a display, where the scientist can interpret outcomes and further steer the experiment.

## 6 Case Scenarios and Requirements

In this section, we use case scenarios to illustrate how the QoS framework can also benefit scientists in other disciplines.

**Collaborative Real-Time Experiments.** A group of scientists located in different domains are collaborating on a nanoscale structure experiment. Each scientist participates in the experiment by providing local data *augmentation* and then transferring that data to a high-performance computing resource for collaborative data analysis. The scientists at corresponding domains establish a guaranteed network bandwidth to conduct

data transfer; similarly, the scientists at the data analysis location establish resource guarantees, not only for the data transfer but also for computing power with adequate resources to perform the data analysis and produce results in a specific time, when all scientist are online to interact with or steer the experiment.

### **Ad Hoc Real-Time Experiments Needing Computing Power.**

Several scientists decide to conduct an experiment to verify certain findings. The decision is made on an ad hoc basis, that is, without prior arrangement. The experiment must be conducted in a Grid infrastructure, with enough computing resources to perform the desired experiment in a reasonable time and fulfill the scientists ad hoc requirements. Here, the scientists require some commitment from the Grid middleware that the resources needed for the experiment are indeed available at this time. The scientists therefore submit a QoS negotiation request to a QoS manager. The QoS manager gives such a commitment if the resources are available at the specific time; if the resources are not available, the QoS manager proposes a new available time, which the scientist may accept or reject.

**Experiments with Deadline Constraints.** A team of scientists has a deadline for delivering experiment results. The scientists therefore contact the QoS manager in advance to negotiate a QoS agreement to guarantee resource availability during the experiment.

These three scenarios have the following common elements: (a) The need for Grid resources with particular capabilities (b) The need for resources to be available for a predefined period of time (c) The need for an agreement to indicate the commitment level of resource availability

With these elements in mind, we have engineered the G-QoS framework to fulfill resource requests with QoS specifications, perform advance reservations of resources, generate QoS agreements, and execute services based on prenegotiated QoS agreements. In the rest of the paper, we focus on the third element the commitment level of resource availability as we discuss the implementation of G-QoS and provide initial experience results.

## 7 Implementation and Results

Our testbed contained Grid computing resources including two Linux-based computers with a 1.8 GHz Pentium processor and 256 MB of memory for the service consumer, and a Pentium processor 1.2 GHz and 512 MB of memory for the service provider. Deployed

on these machines were the Globus Toolkit version 3 OGSi service container, the Globus Toolkit version 2, and the Java CoG Kit. We experimented with the nanoscale application using two different approaches: (1) with a QoS handler through the Java CoG Kit and (2) with a GT2 handler through the Java CoG Kit.

### 7.1 Time-Domain Example

In this section we show results for a nanostructures image analysis based on a sample electron diffraction using up to 900 input images. We used a time-domain strategy for resource allocation. In other words, the whole computer was reserved for the application; multiple jobs were submitted to the reserved node but only one of them was executed.

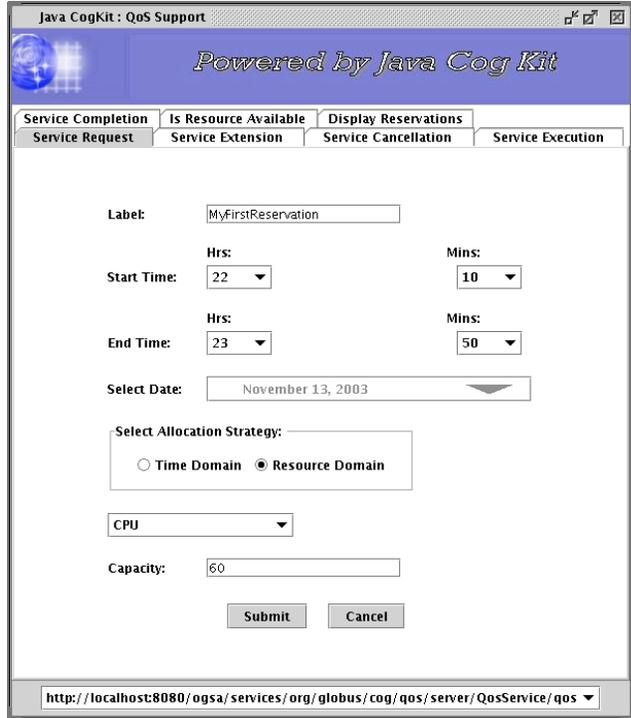
**Table 1. Number of images and time taken to process the images under QoS service- and GT2-based job submission**

Time taken to process images using QoS & GT2				
Number – of – Images	25	50	75	90
Dataset1 : QoS	4:40	9:20	13:55	16:55
Dataset2 : GT2	5:20	10:35	15:42	18:25

We conducted two sets of runs: (1) job submission based on QoS and (2) standard job submission based on GT2 GRAM. Each set consisted of four runs to analyze 25 images, 50 images, 75 images, and 90 images. Table 1 shows the performance results, with the number of images and the time taken to process that number of images.

Dataset 1, QoS, shows that the time taken to process the images is less than that for the GT2 approach. This result is expected because the reservation mechanism employed in this time-domain strategy is to reserve the full processing power of the Grid node for the QoS-based application and thus prevents other processes from using the processing power as long as the reservation holds.

Dataset 2, GT2, shows that the time taken to process the images is more than that for the QoS approach. The reason is that multiple processing loads were applied to simulate a shared multiuser environment. Since the GT2 technology does not employ a reservation mechanism, different processes were able to use the processing power while the submitted job was running.



**Figure 4. User Interface showing parameters for the QoS Negotiation Task**

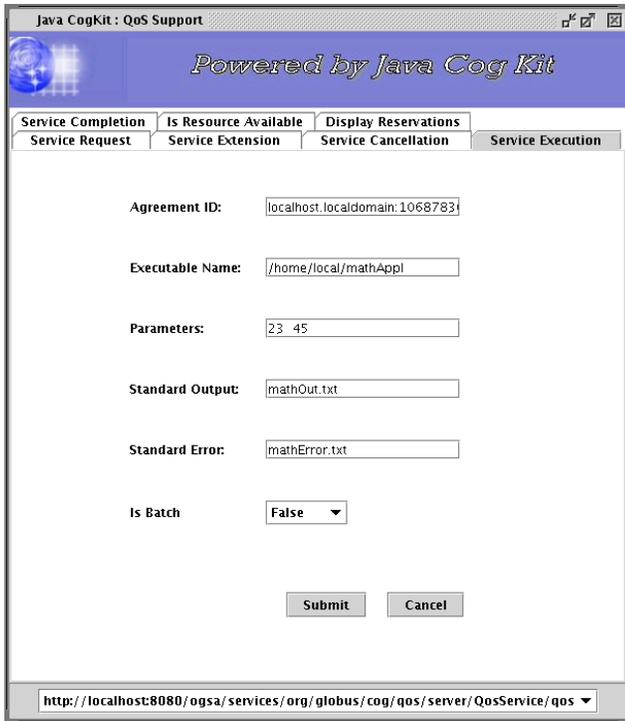
### 7.2 Resource-Domain Example

In this section we show results when the G-QoS framework was used to allocate CPU resources with a QoS specification using a resource-domain allocation strategy. With this strategy, a slot of the CPU power is reserved, and the client or application can submit jobs to be executed under fractional reservation constraints. The process is implemented by using the Java CoG Kit API to create a task object and then submitting the created task to the QoS Handler to negotiate the required resources or services. Upon success, a Service Level Agreement is returned for use when claiming a reserved resource in the future.

We prototyped an easy-to-use set of graphical components designed to make access to QoS services transparent for the nontechnical user. Figure 4 shows a screenshot of the form used to specify the parameters of the task to be submitted to the QoS Handler.

Figure 5 shows a screenshot of the details of a QoS job submission object specifying the executable ‘math-App1’ and a reserved CPU power of 60%.

We conducted a simple feasibility study evaluating the behavior of our system under heavy load. Two computation-intensive, competing processes were



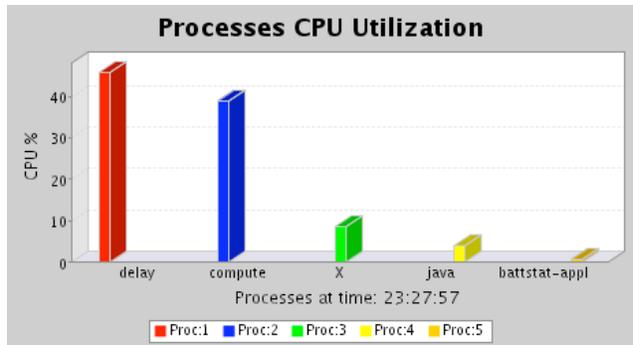
**Figure 5. User Interface showing parameters for the QoS job submission task**

started before we ingested into our systems the guaranteed process ‘mathAppl’. We prototyped a simple CPU monitor that allows us to study the behavior of the system during runtime. Example screenshots of this monitor are depicted in Figure 6 and 6. We show snapshots during two different times. In Figure 6 the six most CPU-intensive processes are shown before the guaranteed process ‘mathAppl’ is submitted. Figure 7 is a screenshot showing CPU utilization of the six most CPU-intensive processes after the guaranteed process has been started. The figure also shows ‘mathAppl’, as a ‘Guaranteed’ process, colored red, and using 60% of the CPU power of this Grid node, while the competing processes use the remainder of the CPU power.

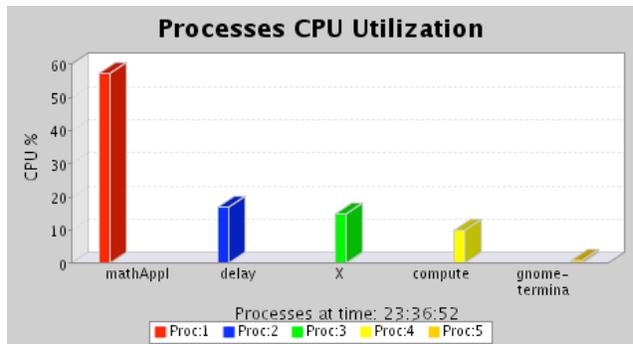
## 8 Conclusion and Future Work

In this paper, we have discussed QoS in several disciplines, including networking, distributed multimedia, and Grid computing. We defined a QoS matrix for networking, computation, and storage media. We also outlined general requirements for QoS management in the context of service Grids.

To meet these requirements, we have proposed a



**Figure 6. CPU Utilization of the six most CPU-intensive current processes, before starting the guaranteed process.**



**Figure 7. CPU Utilization of the six most CPU-intensive current processes, after starting the guaranteed process.**

Grid QoS resource management architecture, called G-QoS. This architecture overcomes some of the limitations of earlier efforts in the Grid community, such as GARA. The development of G-QoS benefits from our experience in designing the Java CoG Kit, which uses convenient abstractions to integrate QoS capabilities and is easily ported to the Globus Toolkit version 2 and 3.

We developed a prototype of G-QoS and validated it using a nanoscale structures experiment as a placeholder for a typical Grid application with data, compute, and interactive requirements. We showed the usefulness of part of our architecture based on a simple comparative use scenario under heavy load.

We note that the G-QoS architecture is suitable not only for nanoscale structures but also for many other applications with computation-intensive and networking requirements.

Our architecture currently includes a set of compo-

nents that abstract the use of QoS for the nonprogrammer. We emphasize that these components are critical if the Grid is to gain widespread acceptance in real applications. The current set of components must be augmented and their utility demonstrated to convince new users of the Grids usefulness.

We intend to continue our research in Grid resource management in accordance with the Global Grid Forum Grid Resource Agreement and Allocation Protocol working group WS-agreement standard. We believe, however, that a resource agreement and allocation protocol is just a small fraction of the work necessary to enable full QoS in Grids. Hence, we plan to investigate much more difficult areas, such as resource allocation strategies, capacity planning, and integrating network QoS support.

## Acknowledgment

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Alliance research and development. The Java CoG Kit Project is supported by DOE SciDAC and NSF Alliance.

We acknowledge Hema Arora and Karthika Arunachalam from the Illinois Institute of Technology for their contribution in implementing the visual interfaces.

## References

- [1] G. von Laszewski and P. Wagstrom, *Tools and Environments for Parallel and Distributed Computing*, ser. Series on Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp. 149–187. <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf>
- [2] I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” *International Journal of Supercomputing Applications*, vol. 15, no. 3, 2002. <http://www.globus.org/research/papers/anatomy.pdf>
- [3] G. von Laszewski, M.-H. Su, J. A. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Thiebaut, M. L. Rivers, S. Wang, B. Tieman, and I. McNulty, “Real-Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources,” in *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, TX, 22-24 Mar. 1999. <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--siamCmt99.pdf>
- [4] A. Oguz, A. T. Campbell, M. E. Kounavis, and R. F. Liao, “The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking,” *IEEE Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability*, vol. 5, no. 4, 1998.
- [5] G. Bochmann and A. Hafid, “Some Principles for Quality of Service Management,” Universite de Montreal, Tech. Rep., 1996.
- [6] A. Roy, “End-to-end quality of service for high-end applications,” Ph.D. dissertation, The University of Chicago, August 2001.
- [7] “The Globus Project,” Web Page. <http://www.globus.org>
- [8] M. Karsten, N. Berier, L. Wolf, and R. Steinmetz, “A policy-based service specification for resource reservation in advance,” in *International Conference on Computer Communications (ICCC’99)*, 1999.
- [9] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, “A distributed resource management architecture that supports advance reservation and co-allocation,” in *Proceedings of the International Workshop on Quality of Service*, 1999, pp. 27–36.
- [10] H. Chu and K. Nahrstedt, “A CPU Service Classes for Multimedia Applications,” in *IEEE Multimedia Systems ’99*, 1999.
- [11] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” Argonne National Laboratory, Tech. Rep., January 2002.
- [12] R. Al-Ali, A. Hafid, O. Rana, and D. Walker, “QoS Adaptation in Service-Oriented Grids,” in *Proceedings of the 1st International Workshop on Middleware for Grid Computing (MGC2003) at ACM/IFIP/USENIX Middleware 2003*, Rio de Janeiro, Brazil, 2003.
- [13] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail, “G-QoS: Grid Service Discovery using QoS Properties,” *Computing and Informatics Journal, Special Issue on Grid Computing*, vol. 21, no. 4, pp. 363–382, 2002.
- [14] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, and D. Walker, “An OGSA-Based Quality of Service Framework,” in *Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC2003)*, Shanghai, China, 2003.
- [15] J. MacLaren, “Advance reservations: State of the Art,” GGF GRAAP-WG, See Web Site at: <http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html>, Last visited: August 2003.
- [16] J. Bent, V. Venkataramani, N. LeRoy, A. Roy, J. Stanley, A. Arpacı, and H. Remzi, “Flexibility, Manageability, and Performance in a Grid Storage Appliance,” in *Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing*, Edinburgh, Scotland, 2002.
- [17] A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker, “UDDIE: An extended registry for web services,” in *Proceedings of Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT 2003*, IEEE CS Press, Orlando FL, USA, 2003, pp. 85–90.
- [18] K. Amin, M. Hategan, G. von Laszewski, and N. Zaluzec, “Abstracting the Grid,” in *Proceedings of the 12-th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP 2004)*, A Coruna, Spain, 2004.
- [19] N. Zaluzec, “Argonne National Laboratory TPM/AAEM Colaboratory,” See Web Site at: <http://tpm.amc.anl.gov/>.