

QoS Support for High-Performance Scientific Applications

Rashid Al-Ali,^{1,2} Kaizar Amin,^{1,3} Gregor von Laszewski,¹ Mihael Hategan,¹

Omer Rana,² David Walker² and Nestor Zaluzec¹

¹ Argonne National Laboratory, U.S.A.

² Cardiff University, UK.

³ University of North Texas, U.S.A.

Abstract—Grid computing provides the ability to access and utilize heterogeneous sets of resources, distributed over multiple domains, as part of virtual organizations. The emerging Grid computing infrastructure gives rise to a class of scientific applications that have collaborative and distributed resource-sharing requirements, such as teleimmersive, visualization and simulation services. This class of application has stringent real-time constraints and Quality of Service (QoS) requirements. Because this class of applications operates in a collaborative mode, data needs to be stored and delivered to execution services in a timely manner, and, similarly, high-performance computing resources need to be ready to handle the execution of data analysis and calculations to meet deadlines. A QoS management approach is required to orchestrate, and guarantee, the interaction between such applications entities. In this paper we design and implement a QoS system and show how Grid applications can easily become QoS compliant. We validate this hypothesis in a case study on a Grid application called ‘Nanoscale Structures’ at the Argonne National Laboratory (ANL). We then run a comparative job submission based on Globus toolkit 2, and the developed Open Grid Service Architecture (OGSA) based QoS approach. Results show the QoS approach is superior to other approaches, which implies a better performance result for applications with QoS provision.

I. INTRODUCTION

Grid Computing [?], [1] has traditionally focused on large-scale, and high-end, resource sharing, innovative applications and the achievement of high performance. Grid applications integrate diverse network environments with widely-varying resource and security characteristics, to cater for multiple usage scenarios. The emerging Grid computing infrastructure gives rise to a class of recent commercial applications, and traditional scientific applications, that have collaborative and distributed resource-sharing requirements, such as teleimmersive, visualization and nanoscale structures. This new class of applications requires high performance resources, the provision of which is a key theme in Grid infrastructures. These resources are expensive and scarce, and are generally not accessible to small and mid-range corporations, but Grid computing is attempting to make these high-end resources available to most computer developers.

Much research effort has concentrated on providing mechanisms to administer and manage such complex settings. The focus is mainly on dealing with the so-called ‘resource management and scheduling problem’, and basically providing algorithms to deal with multiple incoming jobs to be dispatched to a somewhat limited number of high-end resources.

This is achieved by means of sophisticated queuing systems, and dispatching jobs as resources become available.

There is a fundamental problem and, while very important from the client’s viewpoint, this has not had a great deal of research attention. This problem can be described in the form of asking questions - ‘Which resource can execute the job, during a specific time, with the most success?’ and ‘What Quality of Service (QoS) guarantees can the client be offered?’ This ‘QoS Problem’ usually arises in a shared resource environment. As more and more requests are initiated for the same set of shared resources, eventually the new requests have to be queued, as resources become busy or congested. There are two solutions for this problem: i) either increase the scarce resources, which is very expensive, or ii) introduce QoS mechanisms that provide guarantees, fairness and efficiently utilizes the underlying resource.

Consider the following scenario; a group of scientists would like to conduct a collaborative simulation experiment for, and during, a specific period of time, using Grid high performance resources and infrastructure. The experiment will run at site ‘A’ on a Grid resource with fine-grained computation and storage requirements. The database, with the required data for the simulation, is located at site ‘B’. A second group of scientists participating in the simulation experiment are located at site ‘C’. Moreover, the experiment requires a specific network bandwidth to connect site B and site A throughout the simulation, and sufficient network bandwidth to deliver the simulation data input, from the database to the processing Grid node, during the experiment. It is clear that this typical scenario has a number of requirements that a normal resource management or scheduling system cannot fulfill. To make this experiment work we need to equip the Grid infrastructure with QoS management to support, not only network resources, but also for storage, compute, and possibly for specialized instruments.

Motivated by similar scenarios, we propose a comprehensive QoS architecture in service-oriented Grids, called G-QoS. The G-QoS addresses solutions to the concerns discussed earlier. We validate our proposed architecture with a ‘proof of concept’ prototype implementation, and show its effectiveness in a Grid system by conducting a case study in the Argonne National Laboratory, with a class of scientific application with QoS guarantee requirements from the ANL advanced analytical electron microscope, called ‘Nanoscale Structures’ Application.

We provide a discussion on QoS background, in Section II. In Section III we outline the general requirements of the Grid QoS management systems and present the current Grid QoS systems. In Section IV our Grid QoS management (G-QoS) is presented and the major components are outlined. In Section V we discuss an example of a typical high performance Grid application, called nanoscale application and its QoS requirements are outlined. In Section VII we discuss some performance results, based on executing applications with QoS support, and prototype implementation screen shots are given. A summary of future work concludes our paper.

II. QUALITY OF SERVICE BACKGROUND

Quality of Service (QoS) has been explored in various contexts, particularly for computer networks [2] and multimedia applications [3]. Network QoS allows the application to describe its network link requirement so as to function efficiently. Multimedia QoS on the other hand, deals with network and server resource requirements to ‘play’ a ‘multimedia document’ in an acceptable mode. Recently, the Grid community expressed interest in QoS, and as part of the Globus project the General-purpose Architecture for Reservation and Allocation (GARA) was designed and implemented [4]. Moreover, quantitative characteristics are providing QoS for quantifiable elements, such as networks and CPUs. Qualitative characteristics are providing QoS for qualitative elements, such as service reliability and user satisfaction on service performance perception.

For example, in a network community these quantitative characteristics are described in terms of network QoS matrix to describe data transmission requirements, including:

- 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 become corrupted.

These four parameters of network QoS form the network QoS measurement matrix.

CPU or ‘compute’ QoS can be divided into shared and exclusive categories [5]. In shared CPU, where more than one user-level application shares the CPU, the application can specify a percentage of the CPU. In the case of exclusive in large multiprocessors systems, where usually one user-level application has exclusive access to one or more CPUs, the application can specify number of CPUs as the QoS parameter. Another dimension of QoS is related to storage device, namely, memory and disks. In this context, QoS is specified in two parameters: bandwidth and space. Bandwidth is the rate of data transfer between the storage devices to the application. Space parameter is the amount of storage space that the application can use for writing data. When applications specify QoS requirements, the characteristics of the resource, and the period the resource is required, is usually specified. Reservation can be seen as giving the application the confidence, or assurance,

that the resource allocation will succeed with the required level of QoS when needed. Moreover, 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.

III. QoS IN GRID COMPUTING

The Grid can be seen as a global-scale distributed-computing infrastructure with coordinated resource sharing [1]. The fundamental Grid problem that many researcher have been investigating is “resource management”, specifying how can the Grid middleware provide resource coordination for client/application transparently in such a complex infrastructure with diverse resource characteristics? One of the most successful middleware tools that provides such coordination is the Globus Alliance Project [6]. The availability of Grid middleware tools, such as the Globus Toolkit, where facilitate persistent access to Grid services, motivated the application developers, first in scientific and more recently business-oriented disciplines, to build sophisticated Grid applications with complex Grid resources requirements. In most Grid settings, Grid applications submits their requirements to Grid resource management entity that schedules the jobs as resources become available. However, there exists a class of applications that cannot wait for resources to be available or must be executed at Grid resources at a particular time. Therefore, we believe that the Grid resource management entity should be equipped to handle such sophisticated situations. To this end, the following section we outline the basic requirements for Grid QoS management.

A. Requirements

Grid resource management system should adhere to important requirements that relate to QoS issues:

a) Resource Advance 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.

b) Reservation Policy: The system should support a mechanism which 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 [7].

c) 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).

d) 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 client/application and the Grid resource(s), a proper security infrastructure is required, such as Public Key Infrastructure (PKI).

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

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

B. Current QoS Systems

QoS in Grids is relatively recent, and, as a result, the only substantial system developed, within the Grid community to our knowledge, is the General-purpose Architecture for Reservation and Allocation (GARA). GARA is a QoS framework that 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/application who initiated the reservation will receive a specific level of service quality from the resource manager. GARA also provides Application Program Interface (API) to manipulate reservation requests, such as, *create*, *modify*, *bind* and *cancel*. GARA utilizes a Dynamic Soft Real-time (DSRT) [8] scheduler as the underlying 'compute' resource manager. Moreover, it also utilizes Cisco routers to deliver network QoS.

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

- Most current applications employ the emerging new technology, the 'Web Services' and the Open Grid Service Architecture (OGSA) [9]. Unfortunately GARA is not OGSA-enabled, and OGSA-enabled applications cannot therefore leverage GARA services.
- Grid applications require the simultaneous allocation of various resources, once the resource manager establishes the required resources and possibly reserves the resources for future allocation. An agreement protocol should then be in place to inform the application about the resources negotiated for allocation, and what level of quality the application should expect. This information is usually encapsulated in a Service Level Agreement (SLA). GARA does not support the concept of an agreement protocol, and if 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, and it is the application's responsibility to manage these handles when claiming the resources. We view this as a limitation.
- QoS monitoring and adaptation during the active QoS session is one of the most important and successful mechanisms to provide a quality guarantee. Most QoS management systems are toolled with adaptive functions [], unlike GARA.

IV. THE GRID QoS MANAGEMENT

Grid Quality of Service Management (G-QoS_m) is a new approach to supporting Quality of Service (QoS) management in computational Grids, in the context of Open Grid Service Architecture (OGSA). The G-QoS_m is an ongoing project and

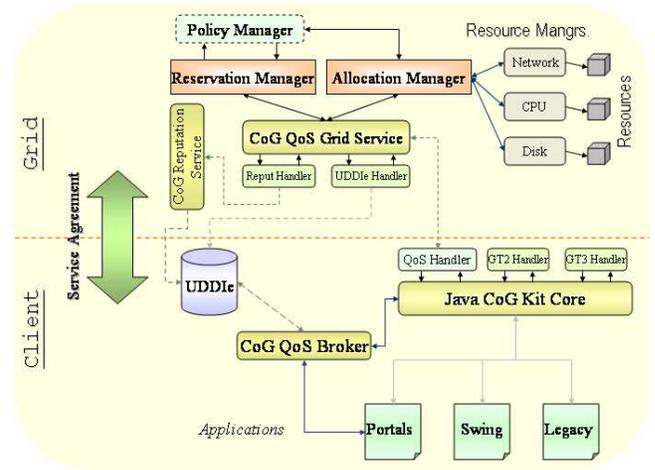


Fig. 1. The G-QoS_m Architecture with an OGSA-enabled QoS service.

more details can be found in [10], [11]. The QoS management consists of several of operational phases include a number of QoS functions; for example, in QoS-oriented architectures, during the 'establishment' phase, a client's application states the desired service and QoS specification. The QoS management system then undertakes a service discovery, based on the specified QoS properties, and negotiates an agreement offer for the client's application. During the 'active' phase, additional activities, including QoS monitoring, adaptation, accounting and, possibly, re-negotiation, may take place. The 'clearing' phase is responsible for terminating the QoS session, either through resource reservation expiration, agreement violation or service completion, and resources are then freed for use by other clients. Our framework supports these 3 phases and can be realized through the interaction between the various framework components. Figure 1 depicts the G-QoS_m architecture. In the following subsections we provide a brief discussion on the interaction between the framework components and highlight the Grid application's integration provision.

A. 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. It exists in every Grid node that can give resources for use by Grid users under QoS constraints. As QGS is a Grid service, it publishes itself to QoS registry service, such as the UDDIe [12] to be known for others. In addition to the QoS functionalities, it supports two types of resource allocation strategies; 1) resource-domain, that is to provide compute, network and disk QoS with fine-grained specifications and 2) time-domain, where the whole Grid node, in which the QGS resides, can be reserved for defined period of time. This is handled by ensuring that all executables on this particular Grid node should pass through the QGS. Further, the QGS interacts with a number of modules to deliver QoS guarantees, these modules are, as depicted in Figure 1, QoS Handler, reservation manager, allocation manager, and QoS registry service. The

underlying QoS that QGS offers are ‘compute’, and currently we are in the process of integrating network and disk support.

g) *QoS Handler*: The QoS Handler constitutes the link between the Java CoG Kit Core and the QGS, with more information on the Java CoG Kit Core given below. Further, it is an implementation of the Core ‘TaskHandler’ interface, which captures the required QoS action, encapsulated in the Core ‘Task’ object, such as QoS negotiation request or QoS job submission. The Task object also contains QoS related parameters depending on the Task action required; for example, in the case of a ‘negotiation request’, parameters provided include: start-time, end-time, resource type and specifications. Once the Task object has been formulated, then the QoS Handler is, for example, delegated, on behalf of the client/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 very useful approach, especially when the application requires more than one Grid node, and all it needs do 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 and return with an agreement, if any. It is worth mentioning that the GGF Grid Resource Agreement and Allocation Protocol (GRAAP) Working Group [13] is attempting to define a WS-Agreement protocol meant to address machine to machine negotiations. We therefore structure our approach to fulfill this particular requirement. Further, in a QoS job submission in an ‘interactive mode’, the QoS Handler listens for notifications of job status, with the notification implemented by the QGS as an OGSA notification.

h) *Reservation Manger*: The reservation manager is basically a data structure that supports reservations for quantifiable resources; resources associated with defined capacities. The reservation manager is de-coupled 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 who can use their resource. 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 on 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.

i) *Allocation Manger*: The Allocation Manager’s primary role is to interact with underlying resource managers for resource allocation and de-allocation, and to inquire about the status of the resources. It has interfaces with various resource managers employed in this framework, namely, the Dynamic Soft Real Time Scheduler (DSRT) [8], a Network Resource Manager (NRM), and currently investigating the *Nest* as the disk storage resource manager [14]. Furthermore, when the allocation manager receives resource allocation request, from the QGS, it forward the request to the designated underlying resource manager. Also the Allocation Manager interacts with adaptive services to enforce adaptation strategies, with more details on our adaptation strategies to be found in [15].

j) *QoS Registry Service*: As the framework operates in the OGSA architecture, then the QGS and other Grid services in the OGSI container should be published in some registry service so they can be known for others. The service publishing process, in this discussion, doesn’t mean simply publishing a service name, URL and simple description. However, services are published with the conventional web services information, in addition to QoS related information. For example, in the case of QGS, is publishes information on what computation QoS does it offer? and what allocation strategies does it employ? and similarly for networks what classes of network QoS does it offer, e.g. best effort, controlled-load or guaranteed. On the other hand, in the case of the other Grid services, the publishing process involves publishing QoS properties, which may include performance characteristics and service execution requirements. In this framework an extended version of the Universal Description Discovery and Integration is used as our QoS register service. The UDDIe [12] is web services registry system, which facilities to service providers a means to publish their services with QoS properties and, hence, be able to search for these services based on their QoS properties.

B. Java CoG Core

The Java CoG Kit Core provides a technology- and architecture-independent abstraction layer that provides true interoperability across multiple Grid implementations. The Java CoG Core provides convenient API for the Grid applications to access the underlying Grid technology. Further, the Java CoG Core has a number of components. Two of which are related to this discussion; i) Task and ii) Handlers. More detail on Java CoG Core can be found in [16].

k) *Task*: A *Task* is the atomic unit of execution in Java CoG Kit Core. It represents a generic Grid functionality including 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. It is apparent that every underlying Grid implementation enforces its own security requirements therefore making it necessary to abstract a generalized security context. Hence, the security context in Core offers a common construct that can be extended by the different implementations of Core 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.

l) *Handlers*: The *TaskHandler* is to process task objects. The task handler provides a simple interface to handle a generic Grid task submitted to it. It is capable of categorizing the tasks and providing the appropriate functionality for it. For example, the task handler will handle a remote job execution task differently than 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 Grid architecture it supports.

C. Application Integration

We have developed a prototype implementation to demonstrate the ease of use, and effectiveness, of a Grid application utilizing QoS functions. The implementation is publicly available from the CoG project web site <http://www.globus.org/cog/java>. In order for Grid applications to make use of this framework, they need to do the following (4) simple steps: a) create a Task object using Java Core, b) depending on the type of the required QoS function, setup 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 finally submit the Task. Table I is a java code segment showing three sample methods demonstrating how applications could generate QoS negotiation request, QoS job submission, and Task submission to the 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 this to a QoS Handler, has a great advantage, and flexibility and scalability, when dealing with multiple Grid nodes. For example, this approach forms the basic element for the design of a QoS broker, whereby the broker contacts the QoS registry service to discover available QGS(s). Then, if the application, for instance, requires ‘10’ Grid nodes, the broker creates 10 different Task objects and only ‘1’ QoS Handler and submits the Task to the corresponding QGS. This approach, a basis for a brokering system, obviously supports scalability; highly desirable and required in Grid computing.

V. POSITION-RESOLVED DIFFRACTION FOR NANOSCALE STRUCTURES APPLICATION CASE STUDY

To validate our hypothesis, we integrate this application with our reference implementation and prototype a Grid computing environment for the analysis of nanoscale structures. As part of this structure a new experimental technique, named position-resolved diffraction, is being developed to study nanoscale structures, as part of Argonne National Laboratory’s advanced analytical electron microscope [17]. 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 (Figure 2).

Analysis of the spatial variation in the electron diffraction pattern of each measured point allows the researcher to study subtle changes, resulting from micro-structural 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. This analysis of the 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

```

/** 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(`allocationStrategy`, 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(`/rashid/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 QoSHandlerImpl();

    // submit the task to the handler
    handler.submit(task);
}

```

TABLE I

A SAMPLE CODE SEGMENT FOR QOS NEGOTIATION AND JOB SUBMISSION TASK, AND TASK SUBMISSION TO QOS HANDLER

produce result in a specific time, when all scientist are online to interact/steer the experiment.

n) *Ad-hoc Real-time Experiments needing Computing Power*: Sometimes scientists conduct experiments to find out, or verify, certain findings on an ad-hoc basis, i.e. without prior arrangement. The experiment needs to be conducted in a Grid infrastructure, with enough computing resources to perform the desired experiment in a reasonable time and fulfill the scientist ad-hoc requirements. Here, the scientist would need to generate the experiment work flow and will require some commitment from the Grid middleware, that the required resources for the experiment work flow is indeed available at this time. This requirement can be accomplished by submitting a QoS negotiation request to a QoS manager. The QoS manager can give such commitment if resources are available at that specific time, or can propose a new available time, and then it is up to the scientist to accept or reject it.

o) *Experiments with Deadline Constraints*: Here the scientist has deadline constraints on delivering the experiment results, and the Grid resources to undertake the experiment should be planned ahead to guarantee such requirements. The QoS manager must be contacted in advance to negotiate a QoS agreement to guarantee resource availability during the experiment.

The above ‘use cases’ have the following common elements:

- Need Grid resources with particular capabilities,
- The resources must be available for a pre-defined period of time, and
- An agreement to indicate the commitment level of resource availability.

With these elements in mind, the G-QoS framework is engineered to fulfill resource requests with QoS specifications, perform advance reservations of resources, generate QoS agreements and execute services, based on pre-negotiated QoS agreements. The G-QoS must be able to handle these scenarios. We conducted this experiment based on the third ‘use case’ scenario, at Argonne National Laboratory (ANL). In the following section we provide an experiment implementation discussion, and give some performance results.

VII. IMPLEMENTATION AND RESULTS

The implementation test bed was based on Intel Pentium processor 1.2 GHz and 512 MB of memory. The machine has Linux and Globus toolkit version 3 OGSi service container, as well as Globus toolkit version 2. As indicated earlier, the Java CoG is the API provider for the application. This gives a great advantage in conducting a comparative performance study, based on different underlying technologies using the same API interface. We experimented with the nanoscale application, using two different approaches: 1) Using a QoS handler and 2) Using a GT2 handler.

A. Time-Domain Example

In this section we show results based on using the framework for resource allocation in a time-domain strategy. Time-domain means the whole computer is reserved for a particular application and the application can thus submit multiple jobs

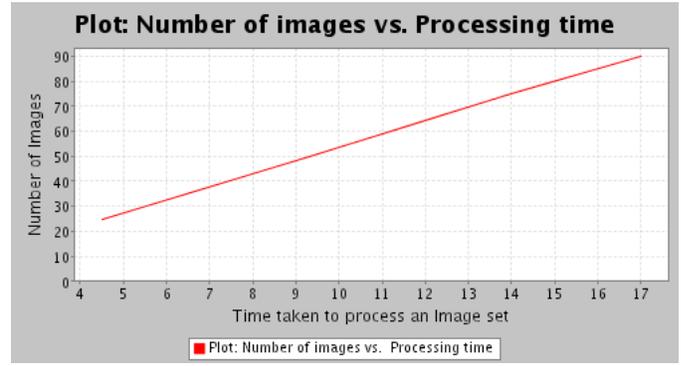


Fig. 4. Plot of Performance Data based on QoS Service Job Submission

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

TABLE II

NUMBER OF IMAGES AND TIME TAKEN TO PROCESS THE IMAGES UNDER QoS SERVICE AND GT2 BASED JOB SUBMISSION

to the reserved node. We tested this methodology with the nanoscale application, which is workflow driven, with the actual computation image analysis based on a given sample electron diffraction. The sample consists of a large number of images, for example, 900 images, depending on the resolution of the microscope. As the 900 images are from the same electron diffraction sample, the size, and image pattern, difference between images is minimal. This implies that the processing requirement for individual image analysis, from the same sample, is almost constant.

Our hypothesis is that job submission based on QoS constraints, in shared resource environments, such as Grids, perform better than other methods, such as using a standard GT2 job submission.

To prove our hypothesis we conducted two sets of runs; 1) job submission based on QoS and 2) a standard job submission based on GT2 Gram. Each set consists of four runs; a) 25 images, b) 50 images, c) 75 images and d) 90 images. Table II shows the generated performance result with the number of images and the time taken to process that number of images.

Figure 4 plots the results from dataset (1), a job submission based on QoS. The plot is linear, as expected, because images within a sample are almost identical in size, and very similar in pattern, and the processing time per image is therefore almost constant within any one sample. Another observation drawn from this result is that the linear plot, increasing as the image number increases, proves the QoS service manages to maintain reserved resources, throughout the reserved duration of the experiment. The dataset (2), standard job based on GT2 Gram, would also produce a linear plot, and shows that it took more time to process the images compared to the QoS option. This is because a constant load is applied to simulate a shared multi-user environment. It is important to note that, in a typical environment, the plot would not be linear, as the expected

Fig. 5. User Interface showing Parameters for the QoS Negotiation Task

Fig. 6. User Interface showing Parameters for the QoS Job Submission Task

loading of the system will keep changing dynamically as user processes start and finish, but in this test we simulate a multi-user shared environment in a worst case scenario.

From these data our hypothesis is thus proven to be valid.

B. Resource-Domain Example

In this section we show results when the framework used to allocate CPU resources with QoS specification, using a resource-domain allocation strategy. The resource domain implies that a slot of the CPU power is reserved and the client/application can submit jobs to be executed under such constraints. The process can be done 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. Upon success a service level agreement (SLA) is returned for use when claiming reserved resource in the future. As indicated in our architecture Figure 1, the API can be utilized through a variety of types of application, such as Portals, Swing and Legacy applications.

For demonstration purposes, we employ, in this example, a GUI Swing technology to access the API. Figure 5 is a screenshot showing parameters required to create a Negotiation Task to be submitted to the QoS Handler. Figure 6 is a screenshot showing a QoS Job Submission Task object being created to be submitted for execution. Note the executable program is 'mathAppl' and needs to be executed within 60% of the CPU power. Two, computation-intensive, competing processes were started (process: compute & delay) before the guaranteed process 'mathAppl' starts, in Figure 7 the 6 most CPU intensive processes are shown before the guaranteed process

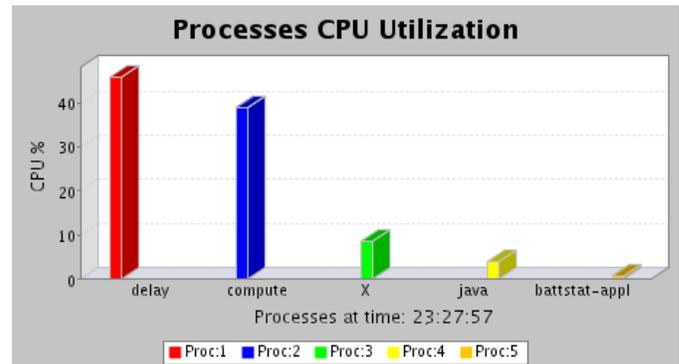


Fig. 7. CPU Utilization of the 6 most CPU-Intensive Current Processes, before starting the guaranteed process.

'mathAppl' gets submitted. Figure 8 is a screenshot showing CPU utilization of the 6 most CPU intensive processes after the guaranteed process has been started. The figure also shows the process 'mathAppl', as a 'Guaranteed' process, colored red, and utilizing exactly 60% of the CPU power of this Grid node, while the competing processes utilizing the remainder of the CPU power.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we present a discussion on QoS in different disciplines, including the networking community, distributed multimedia and Grid computing. We define the QoS matrix for networking, computation and storage media. General requirements for QoS management, in the context of service Grids, are outlined. GARA as an example of a popular QoS

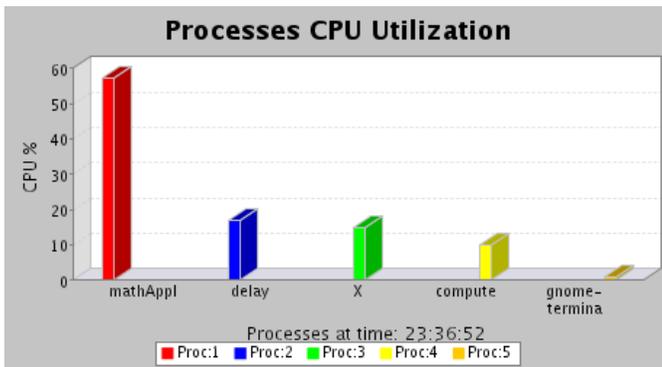


Fig. 8. CPU Utilization of the 6 most CPU-Intensive Current Processes, after starting the guaranteed process.

framework in the Grid community is discussed, and some of its shortcomings are highlighted. Then we describe our G-QoS architecture, emphasizing the application integration and describe how practical Grid applications can utilize our system through the Java CoG Kit API.

We chose the ‘Nanoscale Structures’ application as a typical Grid high performance computation-intensive application and demonstrate the usefulness of the G-QoS architecture for such class of applications. We also present a number of ‘use case’ scenarios where our framework would be of great benefit. It is important to note that the proposed architecture is not only suitable for Nanoscale Structures, but this application has many similarities to many applications with computation intensive and networking requirements. We finally show performance results based on a comparative performance of GT2, and QoS based job submissions. Based on our performance results the QoS approach stands out amongst other non-QoS approaches, which validates our hypothesis. Another important result from this reference implementation is the practicality of how a non-QoS-aware legacy application can, with simple API, become a QoS-aware application and can run on guaranteed resources.

As future work, although in our framework we employ a simple agreement protocol, we nevertheless intend to investigate this line of research in accordance with the GGF GRAAP working group WS-agreement standard. We are also in the process of further investigating resource allocation strategies and capacity planning.

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 Project research and development. The Java CoG Kit Project is supported by DOE SciDAC and NSF Alliance.

We would like to acknowledge Hema Arora and Karthika Arunachalam from Illinois Institute of Technology for their contribution in the implementation work.

REFERENCES

- [1] 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. [Online]. Available: <http://www.globus.org/research/papers/anatomy.pdf>
- [2] A. Oguz *et al.*, “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.
- [3] G. Bochmann and A. Hafid, “Some principles for quality of service management,” Universite de Montreal, Tech. Rep., 1996.
- [4] 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.
- [5] A. Roy, “End-to-end quality of service for high-end applications,” Ph.D. dissertation, The University of Chicago, August 2001.
- [6] A. N. Laboratory, “The globus project,” See Web Site at: <http://www.globus.org/>, Last visited: February 2003.
- [7] 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.
- [8] H. Chu and K. Nahrstedt, “A cpu service classes for multimedia applications,” in *IEEE Multimedia Systems ’99*, 1999.
- [9] I. Foster, C. Kesselman, *et al.*, “The physiology of the grid: an open grid services architecture for distributed systems integration,” Argonne National Laboratory, Chicago, Tech. Rep., January 2002.
- [10] 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.
- [11] R. Al-Ali, K. Amin, G. von Laszeswski, 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) LNCS*, Shanghai, China, 2003.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] K. Amin, M. Hategan, G. von Laszeswski, 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.
- [17] N. Zaluzec, “Anl tpm/aaem collaboratory,” See Web Site at: <http://tpm.amc.anl.gov/>.