A Model for Quality-of-Service Provision in Service Oriented Architectures

Rashid Al-Ali[1], Omer Rana[1], Gregor von Laszewski[2],
Abdelhakim Hafid[3], Kaizar Amin[2] and David Walker[1].

[1]Cardiff University, UK
[2]Argonne National Laboratory, Argonne IL, US
[3]Telcordia Technologies Inc. Red Bank NJ, US
E-mail: Rashid, O.F.Rana@cs.cardiff.ac.uk

Abstract

Clients of service-oriented Grids are often concerned with service "quality" and the computational/economic costs for using such a service. A *Quality-of-Service* (QoS) mechanism should identify the resources needed to execute a service at a specified quality level. It is also important that the selection of such resources is undertaken subject to other constraints, such as increasing/decreasing the costs associated with service execution on a particular set of resources. The QoS models presented here is based on distinguishing between a service provider, a service and a resource. Resources are entities that can be reserved, and a *Service-Level-Agreement* (SLA) presents a contract that needs to be agreed between a client and a service provider prior to resource allocation. The problem is to determine, given multiple types of QoS requests from clients, the optimal resource allocation to maximize utilization and maintain requested quality levels. A model for such QoS provision is presented, along with a description of optimization heuristics. A QoS system which implements the model, and also involves support for resource reservations in advance is proposed. The implementation extends the functionality of the CoG Core toolkit.

Keywords: Quality of Service (QoS), Grid Computing, Resource Management, Service-Oriented Grids.

1. INTRODUCTION

Grid computing [1] so far has primarily focused on the sharing of distributed computational and data resources, making these resources appear as a single large-scale environment to execute applications. To achieve such an integrated view, Grid architectures need to support a variety of different network environments, with widely varying resource and security characteristics into one or more Virtual Organizations (VOs). A key theme of many current Grid middleware is to provide and sustain such a VO. The Grid computing concept may therefore be summarised as the "*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations"* [1].

Establishing a VO is therefore also equivalent to providing soft Quality of Service (QoS) assurances, i.e. once a VO has been established, this is equivalent to suggesting that resource providers within that VO have agreed to make available their resources for use by the authority who established the VO. It is important to identify under what circumstances (and by whom) the VO is established – i.e. whether by a user wanting to run an application over resources within the VO, or by a collection of resource owners who wish to group their resources (and therefore increase their utilisation or revenue). The motivation for establishing such groupings has been discussed elsewhere [6]. To ensure that resources will be accessible by users of the VO, reliable and high-speed connectivity is necessary between resources via dedicated networks. However, in a Grid environment provision of such a dedicated network is not feasible (as the network bandwidth must be shared between many users). Therefore, although some preliminary level of QoS is provided by VO participants, execution and performance guarantees cannot generally be provided. Nevertheless, the complexities involved in several critical Grid applications make it imperative to provide more stringent QoS assurances beyond those provided by the basic Grid infrastructure. Considering the increasing sophistication of Grid applications, and new hardware under development [2], such provisions become an inherent requirement within the Grid architecture. This implies a need for a QoS management entity that allows for the use of a negotiation mechanism, where clients can negotiate for appropriate resources during the establishment of the VO (based on particular QoS constraints that match the needs of the applications they are managing).

Motivated by the need to overlay an advanced QoS framework on existing Grid architectures, allowing them to support complex QoS requirements, an abstract model of QoS management in service-oriented Grids is proposed. Supporting recent standardization efforts of the Global Grid Forum [3], the QoS model presented is compatible with the latest Open Grid Services Architecture (OGSA) and the Web Services Resource Framework (WSRF) specification. We therefore make a distinction between a "service" and a "resource" (in accordance with WSRF). This work utilises the notion of a Service Level Agreement (SLA) – which forms the most essential mechanism to establish and sustain a VO.

The model presented in this paper has a number of important features: 1) an agreement-driven QoS model, i.e. Service Level Agreement (SLA) based, 2) a 'utility model' for the monetary revenue from resource and service utilization, 3) introduction of a 'service profile', including Grid service properties for suggested QoS levels, and 4) a generic resource 'reservation manager', with the following features:

- support for advance and immediate reservation,
- support for single and collective resource reservations (co-reservations),
- accommodation of arbitrary resource types, for example, compute, network, memory and disk, and
- a dynamic object-oriented approach that uses underlying resource characteristics at run-time.

In Section 2 of the paper the QoS model is presented along with a derivation of optimization heuristics. Section 3 discusses QoS management, and provides a definition of 'resource reservation'. In Section 4 a QoS system, based on the abstract model, is introduced and a discussion on its main components provided. An overview of related research in the area of resource reservation to support QoS needs is given in Section 5. In Section 6 the implementation status is discussed, and prototype implementation screen shots are shown. The paper is concluded in Section 7.

2. QUALITY OF SERVICE MODEL

Resource management in distributed systems, in its simplest form, consists of:

a) Selecting a set of resources for execution of tasks generated by an application,
b) Mapping tasks to computational resources,
c) Feeding data to these computations, and
d) Ensuring that task and data dependencies between executing tasks are maintained [4].

With QoS enhancement these tasks are slightly modified:

1. A client/application submits a service request with QoS requirements.
2. The QoS system selects a service that best matches the specified QoS constraints.
3. A Service Level Agreement (SLA), specifying the negotiated service and qualities is issued.
4. Resources required to execute the agreed-on service are reserved, for further allocation during the session when execution is taking place (during this period, QoS properties are also monitored).
5. SLA compliance is ensured, by periodically monitoring the quality of allocated resources, and adaptation techniques that need to be applied if there are quality degradations.

Figure 1 shows the model, architecture and its components.

2.1. Service Request

A client submits a *service request* (S_R) to the QoS management entity, specifying a requested service, optional resource quality levels, budget constraints and time interval required for the service. Multiple service requests: (S_{R1}, S_{R2}, ..., S_{Rn}) from n clients may be concurrently submitted. Each request, S_{Rj} undergoes a negotiation process where the system considers candidate services and available resources, and selects those most suitable for satisfying the request. Often a client is interested in making use of services with a specific level of "quality" and particular requirements, such as cost, reliability, availability and response time. Some of these requirements are difficult to measure (and therefore difficult to capture with monitoring tools such as Netlogger), and it is necessary to obtain this information from other sources. The concept of the *service profile* as a property of a service is therefore introduced, with service levels obtained from the service provider, a third-party reputation service or statistics based on client feedback. These levels are dynamically updated and stored in the service profile. The service profile is intended for use by the QoS management entity where a client i either specifically requests services with the default profile or does not have details of the resource quality required. In such cases, a service profile contains a suggested set of quality levels for each q_{ik} in SLA_i.

A key component in this model is an SLA, which may be defined as: $SLA = \{R, (A, value)\}$, where R is set of relations, and A is a set of attributes, each of which has a particular *value*. Hence, $R=\{r_1,...,r_m\}$ and $A=\{a_1,...,a_n\}$. Each element of R specifies a relationship that must be verified over a set of attributes. Elements of A consist of attributes that are provided as part of the service profile, and can take on numeric or string values. Set A represents the complete set of attributes that may be a part of such a profile. Hence, an example of an SLA may be:

$SLA_1 = \{ (greater\_than), (memory, 24) \}$
where *greater_than* is the relation, *memory* is the attribute and *24* is the value. This indicates that for this SLA to be valid, the provider should have greater than 24 units of memory for the client application.

SLA_2 = {(AND,(((greater_than), (memory,24)), ((equal_to), (bandwidth, 10)))))}
This indicates that for this SLA to be valid, the provider should have greater than 24 units of memory and a bandwidth equal to 10 units for the client application.

Attributes that are present within an SLA_i must be measurable and quantifiable during service execution. Some of these attributes will be quantified through the use of monitoring tools, whilst others need to be specified beforehand (such as owner credentials, security preferences and roles, etc). The SLA_i has the following properties:

1. SLA_i is *atomic* – all *relationships* must be verified to determine status. The complexity of these relationships can vary – from simple Boolean to user-defined.
2. SLA_i is *satisfiable* – it must evaluate to either *true* or *false.* That is relationships within it must evaluates to either *true* or *false*. An evaluation of *false* during a service session implies an SLA violation. Only attributes which are dynamically monitored can become false during the execution of a service.
3. SLA_i is *consistent* – attributes and relationships must not contradict each other. That is, two relationships over the same set of attributes must not hold both *true* and *false* at a particular time *t*.
4. At present all attributes are treated with an equal priority/importance. To give greater significance to particular attributes over others, we can modify the SLA definition so that each *relationship* also has a *value* associated with it. A modified definition of such an SLA would be: *SLA = {(R, value), (A, value)}.* SLA validation will now involve sorting the elements of the SLA based on the value associated with R, and requiring only relationships whose value exceed a threshold to not be violated. If this value is set to 1, then this collapses to the existing version of the SLA.

The SLA, by definition, must be held between two or more collaborating entities. In the simplest case, we may consider these to be a client and a provider. To support this notion, the QoS model consists of three key abstractions: (1) a service provider (*P*), (2) a service (*S*), and (3) a resource (*R*). This abstraction allows us to decouple a service provider from the resources it uses to execute a service (i.e. these resources may be owned by others). A service provider can offer one or more services, and must support a hosting environment (such as Apache Axis for Web services, for example). A service may use one or more resources to execute, and a resource may be stateful (i.e. can keep a track of all previous allocations made on it by the same client). We may express these relationships as:

$$P \Delta \{S_1, \ldots, S_n\} \ \Delta \{R_1, \ldots, R_m\}$$

A given Service (S_i) may be mapped to a group of resources ($R_1,\ldots, R_j$). This represents the simplest case where there is a single service provider offering a collection of services. The group of available resources may be further divided into their types – these include network, disk, CPU, and memory. The division of resources into types also indicates that the attributes available in an SLA must be divisible into these resource sets. Hence, network attributes, such as bandwidth, latency, jitter and delay must be associated with R^{net}. Similarly, attributes such as seek time and I/O throughput must be associated with R^{disk}, etc. The set of available resources is a union of resource sets based on their types:

$$\{R\} = \{R^{net}\} \cup \{R^{disk}\} \cup \{R^{CPU}\} \cup \{R^{memory}\}$$
$$and \ \{R^{net}\} = \{(R_1^{net}, \ldots, R_k^{net})\}$$

each type of resource – such as R^{net} – may have multiple instances (the above indicates that there are *k* instances of R^{net}). We may associate a *quality* measure with each instance of a resource, within a resource set of a given type.

In this instance, *quality (q)* represents a requested measure on a particular attribute value. Examples of this include network bandwidth of 10Mbps, main memory of 512MB, etc. An SLA may be viewed essentially as a group of quality levels that must be validated (via a set of relationships – essentially formulated as a requested SLA). In order to execute a service at a particular quality level, it is therefore necessary to select resources from each of the four sets (network, memory, disk, CPU) – based on measurable attributes associated with these services. Resource selection is driven by the fact that different instances of resources provide different quality levels to an application.

On receiving the service request, S_{Rj} (which consists of the desired SLA) containing the required quality specifications, whether obtained from the client/application or from a previously recorded service profile, the QoS manager initiates a discovery process. The QoS manager then goes through a service selection process aiming to select the best match from the list of services returned.

2.2. Service Level Agreement Formation

After successfully negotiating resources for client *i*, the system presents a *Service Level Agreement* (SLA) contract for approval by the client. $SLA_i = \{Q(t)\};$ where $t \in [t_1, t_2]$ denotes the time over which the SLA can be activated, and $Q_i(t) = \{q_1(t), q_2(t), \ldots, q_n(t)\}$ includes agreed-on quality levels for the service requested, where each $q_j(t)$ denotes the quality levels of the j^{th} attribute, e.g. *20Mbps* or *30%* of CPU cycles. Each such level $q_j(t)$ is an integer number or a range – i.e. a pair of minimum and maximum acceptable qualities.

The negotiation process therefore involves a client initially proposing an SLA to the service provider – and the service

provider sending back a counter proposal if the initial SLA cannot be satisfied with the resources it currently has access to. The generation of the counter proposal is based on the types of relationships, and the attribute range specified in the initial client SLA.

2.3. Resources

To support requested services, the Quality of Service (QoS) management system needs to reserve, and subsequently allocate, sufficient resources – e.g. *20Mbps* bandwidth or *40%* of CPU cycles. Assume that client *i* requests a particular service, resulting in SLA_i with the following quality attributes $Q_i(t) = \{q_1(t), q_2(t), \ldots, q_n(t)\}$. Each resource that must support a particular quality level has a limited amount of capacity to support the same attribute – i.e. *655Mbps* or *90%* CPU cycles.

2.4. Utilization Model

We assume that a Grid service provider charges for use, with the *utility* value of a service being the monetary revenue from executing a particular service. Let *cost(q,R)* be a function that evaluates the cost of obtaining a particular quality from a single or collection of resources. This function could therefore help to calculate the cost of executing a job which makes use of 30% of CPU cycles and 100% of available bandwidth, for instance. This function may simply be based on a table lookup, or may be more dynamically calculated as a service executes. As a service needs to make use of a collection of resources to run, we may aggregate the cost of running a service based on the quality levels specified in the SLA, hence:

$$Service\ cost = \sum_i cost(q_i(t), r_i)$$

Depending on whether the cost for executing a service is being calculated by the client or the provider, we may optimise this cost from different perspectives. Given a particular quality level, we may be interested in identifying a set of resources that can offer this quality for a minimum cost. Evaluation of C1 may be viewed as a search to determine the best resource ensemble that can offer a particular quality at the minimum cost. In this case, the quality level is constant, and it is necessary to determine the resource ensemable:

$$C1 = min \sum_j cost(q(t), r_i)$$

Alternatively, we may be interested in maximising the revenue that could be obtained from all the available resources, regardless of the quality level that can be provided (*X* in C2 represents a "*don't care*"):

$$C2 = max \sum_j cost(X, r_i)$$

One can compute the Grid provider revenues for a given set of service requests from *n* clients, and corresponding resource allocations as follows:

$$\int_{t1}^{t2} dt \sum_{i=1}^{n} \sum_{k=1}^{l} cost_i(q(t), r_k)$$

2.5. Optimization Problem

When considering QoS issues for a particular service provider, we may now define this as: given (SLA_1, SLA_2,...,SLA_n), the QoS management system should allocate resources so that each resultant quality level satisfies the corresponding service request, with Grid revenue maximised. This problem therefore consists of two parts: (1) validation of the SLA, (2) an optimization on cost. The validation of the SLA may be binary, or consist of a threshold on a sorted list of relationships defined in the SLA. The optimization must:

1. Not exceed the resource capacity of the service provider; $\forall\, i, (1 \le i \le n), \sum_{k}^{m} r_{ik} \le K_i$; where K_i is the maximum capacity of the resource type R^k that corresponds to the i^{th} quality attribute.
2. Maximize the grid revenue.

$$Maximize \int_{t1}^{t2} dt \sum_{i=1}^{n} \sum_{k=1}^{l} cost_i(q_{ik}(t), r_k)$$

2.6. Service Level Agreement Compliance

Having allocated resources for the specified quality levels, it is important to ensure that during the active QoS session SLA compliance is maintained, and all quality attributes of the SLA are satisfied. One approach is through a monitoring service *M* that periodically monitors the status of the allocated resources, and an adaptation service *V* that compares the agreed-on qualities with those obtained from *M*. The adaptation service *V* applies adaptive techniques to compensate for quality degradation, if possible, such as the techniques described in [5]. If such compensation is not possible then a report is made to the QoS manager about the SLA violation. The adaptation process therefore attempts to update particular attribute within the constraints of the relationships agreed upon in the SLA.

3. QoS MANAGEMENT FUNCTIONS

QoS management includes a range of activities, including resource specification, selection, reservation, allocation and resource release, and identifying which activities take place in the course of a QoS session. A QoS session includes three main phases, *establishment*, *active*, and *clearing* [7], which include a number of QoS functions. See Figure 2. In a QoS-oriented architecture, during the establishment phase, a client's application specifies the desired service and quality. The QoS manager undertakes a service discovery, based on the specified QoS properties, and negotiates an agreement offer for the client's application, finally reserving the

resources as per the agreement. During the active phase, additional activities including resource allocation, QoS monitoring, adaptation, accounting, and possibly re-negotiation, may take place. The clearing phase terminates the QoS session, either through resource reservation expiration, agreement violation or service completion, whereafter resources are freed for use by other clients.

To realize some of these QoS management functions in the model, a mechanism for advance resource reservation is presented. These reservation mechanisms are mainly intended to increase system flexibility and to maximize resource utilization.

3.1. Advance Resource Reservation

A reservation model for reserving a collection of Grid resources is defined, to increase the flexibility of the admission control. The reservation is for a collection of resources – and not a single resource as most applications can only run successfully if only a collection of resources are available. The fundamental problem with advance reservation, as discussed in [8], is that when an advance reservation is granted, to utilize or grant, reservations during hold-back time (time from when the reservation is submitted until the start time) is a complex situation. The problem arises when clients request immediate reservation for an indefinite period, which may, obviously, overlap a previously-granted advance reservation. A number of solutions are proposed to solve this problem; for example, all reservations, including immediate reservation, must be specified within a time frame, i.e. indefinite reservation is not supported; another solution proposes to partition resources for immediate reservation, and allow advance reservation with specified durations. In this model the first proposal is utilized; all reservations must be accompanied by duration specifications. This is considered a valid assumption where one is dealing with high performance (or high demand) resources; with application domains, like scientific experiments or simulations, meaning there is prior knowledge of the need for such resources, and there are no ad-hoc requests for simple resources.

Reservation *Res* is formally defined in terms of the following (5) parameters:
t_s: reservation start time
t_e: reservation end time
cl: reservation class of service (these include: *Guaranteed, Controlled Load,* and *Best Effort* – see Section 4)
r_i: each resource *i* has a resource type (as discussed previously).
$c(r,t)$: a function that returns the capacity of resource *r* at time *t*

Based on this notation one can express a reservation request $Res(t_s,t_e,cl,r_1,c(r_1)),\ldots,(r_n,c(r_n)))$, which essentially represents a co-reservation for *n* resources, with start time t_s and end time t_e, using QoS reservation class *cl* on r_i with the associated capacities $c(r_i,\Delta t)$. This definition also introduces the concept of pre-emption priority, which allows higher priority jobs to reduce the priority of already running services [8]. The pre-emption priority determines (when the reservation is not in effect – either before or after the reservation period) that the service that makes use of the reserved resource is not refused or eliminated, but is rather assigned a low priority value, which means switching its status from 'guaranteed' to a 'best effort' type of service. In practice to support this concept the underlying resource manager should be a priority-based system, such as the Dynamic Soft Real Time (DSRT) scheduler [9]. This feature is very useful in protecting applications when reservations expire.

3.1.1 - Admission Control

Admission control is the process of granting/denying reservation requests based on factors such as the actual load on the specified resource, and the policy that governs who, how and when reservation for a resource should be granted. This is achieved via an admission control mechanism – formally described as a Boolean function that returns *true* or *false* for a reservation request *Res* at time *t*, with *true* meaning the reservation can be granted for the given time *t* with the resource specifications. To further define the admission control function algorithm, the notion of resource load *L* at time *t* is first defined:

$$L(r_j,t) = \sum_{i=1}^{g(t)} c(rj,t)$$

where $g(t)$ is the number of granted reservations for time *t* and $c(r_j,t)$ is the amount of capacity reserved on the resource type *j* at time *t*. $max(r_i)$ is the maximum capacity that the resource *i* can provide. With the above basic primitives, one can define the algorithm for the admission control function.

Admission Control Function

Input: *reservation* $Res(t_s,t_e,cl,(r_1,c(r_1)),..,(r_n,c(r_n)))$
Output: *boolean*

1. for $i=1$ to n
2. for $t=t_s$ to t_e
3. if $c(r_i,t) > (max(r_i) - L(r_i, t))$ then
4. return *false*
5. end if
6. end for
7. end for
8. return *true*

4. AN OGSA-BASED QoS MANAGEMENT SYSTEM

The Grid Quality of Service (QoS) Management (G-QoSm) [10] framework is an architecture based on the QoS abstract model described above. G-QoSm provides three main functions: 1) support for resource and service discovery

based on QoS properties, 2) support for providing QoS guarantees at middleware and network levels, and establishing Service Level Agreements (SLAs) to enforce these guarantees, and 3) providing QoS adaptation for the allocated resources. The G-QoSm delivers three types of QoS levels: *Guaranteed*, *Controlled Load* and *Best Effort* QoS. At the guaranteed level, constraints related to the QoS attributes specified by the client need to exactly match those that are being offered by a service provider. Hence, if a client is requesting 24GB of disk, exactly this amount of disk should be made available by the service provider. Controlled load is similar to the guaranteed level, with the exception that less stringent parameter constraints are defined, and the notion of range-based QoS attributes is used along with range-based SLAs. In this case, a client is expected to specify a range of acceptable attributes – disk space between 20GB and 24GB, rather than an absolute value. Similarly, the relationships in the SLA need not evaluate to true/false, but must have a *degree* of satisfiability. At the best effort QoS level the resource manager has full control in choosing the QoS level without constraints, corresponding to the default case when no QoS requirements are specified.

G-QoSm is an ongoing project, previously investigated and implemented in the context of Globus toolkit (GT 2.x), [10, 11] using a GARA framework to provide QoS support for 'compute' resources [12]. However, with the emergence of Service-Oriented Grids, and Open Grid Service Architecture (OGSA) [13], and in particular WSRF, it is necessary to introduce new features to G-QoSm to make it OGSA-enabled and GT3.x compliant. In this new G-QoSm architecture GARA is not utilized, and is replaced by a new reservation manager, policy manager, allocation manager and a newly-developed Java API for a Dynamic Soft Real Time (DSRT) scheduler [9]. The new features in the OGSA-enabled G-QoSm are:

- Introduction of a 'QoS Grid service'.
- Introduction of a new generic resource 'reservation manager'.
- Introduction of a new `policy manger'.
- Introduction of a QoS framework that is OGSA-enabled and can be initiated in the context of GT3.x.

Figure 3 shows the new G-QoSm OGSA-enabled architecture. The main components are: QoS Grid service, an extended version of the Universal Description Discovery and Integration (UDDIe) [14], resource reservation manager, resource allocation manager and policy manager.

4.1. QoS Grid Service

The basic building block of the architecture is the QoS Grid service (QGS), an OGSA-compliant Grid service providing QoS functionalities such as negotiation, reservation and resource allocation with certain quality levels. Each QoS-enabled resource is accessed through a QGS. The QGS also publishes its properties as a services within a registry, so clients and QoS brokers are able to make use of it. Further, the QGS interacts with the client's application, the QoS selection service, the reservation manager and the policy manager to support the following:

4.1.1 - Interaction with Client's Application

To capture the service request with QoS constraints, and to negotiate a QoS agreement, SLA interaction with a client's application is needed. This negotiation can be summarized as attempting to find the 'best match' service, based on given properties and priority levels; for example, one might request that cost have a higher priority than service reliability, and the matching process should comply with such a requirement. Once the best service match is found, and corresponding resources are reserved, an agreement offer is proposed to the client's application. If the proposed agreement is approved, it becomes a commitment, and the QGS regards this agreement as a fixed guarantee. Otherwise resources are released and no agreement takes place. More details of this negotiation process can be found in [19].

4.1.2 - Interaction with the QoS Selection Service

The main function of the Selection Service is to provide information and the decision for selecting the best a service that matches a particular QoS profile. Normally, the selection service (e.g. service registry) replies with a list of service matches which necessitates the QGS selecting one of the returned services. To facilitate this selection process, the client's application should associate an importance or priority level value to each required QoS attribute in the initial service request. A selection algorithm based on a priority value (weighted average (WA)) concept is adapted, taking into account the proportional value of each QoS attribute. The algorithm computes the WA for every returned service and selects the service with the highest WA. Performing the weighted average can lead to a loss of information, as it is not possible to determine which attribute are of greater importance than others. However, the WA approach provides a simple way to differentiate between a list of returned services which are found to be suitable in other ways.

4.1.3 - Interaction with Reservation Manager

After selecting a Grid service the functional requirements, required in support of the reservation, are extracted and formulated as resource specifications. These resource specifications are then submitted to the reservation manager for resource reservation, and a reservation 'handle' is returned in the case of a successful reservation. This reservation handle can later be used to claim, or manipulate (e.g. modify), the reservation.

4.1.4 - Interaction with Policy Manager

Interaction with the policy manager enables the QGS to capture policy information necessary to validate the service request. For example, to discover if there are any limitation on resource utilization per service, or the class of service requested. The QGS validates the service request by applying the rules obtained from the policy manager.

4.1.5 - Interaction with Allocation Manager

Interaction with the allocation manager is primarily for passing resource specifications and giving instructions to the underlying resources managers to allocate, or de-allocate, resources. The allocation manager has the appropriate interfaces to the underlying resource managers, (e.g. network and compute).

4.2. QoS Service Registry

UDDIe [14] forms the service registry system in the context of the G-QoSm framework; Figure 3 shows how the UDDIe integrates with the rest of the components in the framework. It is primarily used as a registry to publish services with QoS attributes, and, subsequently, to search for services based on QoS attributes. The publication process, as per the Web service concept and, requires that service providers supply two separate Web Services Description Language (WSDL) documents, namely service interface and implementation documents that describe the syntax and semantics for accessing the service.

Every service in the framework has two interfaces: functional and management interfaces. The functional interface describes how the given service can be accessed, whereas the management interface describes attributes related to QoS and performance characteristics associated with the service. The QoS properties, such as resources needed to execute the service and service cost, are incorporated into the service management interface. With this extension to the WSDL document, the service provider is able to describe their service with QoS properties. For this extension to be recognized by a registry system, such as UDDI, and hence be searched, the UDDI registry needs to be extended to recognize these additional attributes. Three search capabilities are implemented by extending UDDI: 1) service properties, 2) service leasing and 3) range-based search. These properties can together be used to search for stored services based on their properties rather than their keys or TModels as undertaken in a standard UDDI implementation. Documents stored in the registry have a lease associated with them, and an event manager, to support these leases, is implemented in UDDIe. In this way, the authors are able to search for documents which match a given range of QoS attributes for supporting various classes of QoS. More detailed discussion on the UDDIe enhancement can be found in a previous work [14].

4.3. QoS Allocation Manager

The Allocation Manager's primary role is to interact with underlying resource managers for resource allocation and de-allocation, and to enquire about the status of resources. It has interfaces with various resource managers employed in this framework, namely the Dynamic Soft Real Time Scheduler (DSRT) [9] and a Network Resource Manager (NRM), and associates the execution of grid services with a previously-negotiated SLA agreement; which process, of associating grid services with SLAs, is beyond the scope of this paper. The allocation manager further interacts with adaptive services to enforce adaptation strategies, with more details on adaptation to be found in a previous work [11].

4.3.1 - DSRT Resource Manager

The DSRT [9] is a user-level soft real-time scheduler, based on the changing priority mechanism supported by Unix and Linux operating systems. The highest fixed priority is reserved for the DSRT and the real-time process admitted by the DSRT can then run under the DSRT scheduling mechanism. The real-time process can thus be scheduled to utilize a specific CPU percentage. Therefore, the compute QoS supported by the DSRT can be specified in terms of CPU percentage; for example, a real-time process might request the allocation of 40% of the CPU. Figure 4 shows the DSRT resource manager integrated with the new G-QoSm.

4.3.2 - Network Resource Manager

The Network Resource Manager (NRM) is conceptually a Differentiated Services (Diffserv) Bandwidth Broker (BB) (a concept described in [15]), and manages network QoS parameters within a given domain, based on agreed SLAs. The NRM is also responsible for managing inter-domain communication, with NRMs in neighbouring domains, to coordinate SLAs across domain boundaries. The NRM may communicate with local monitoring tools to determine the state of the network and its current configuration. Figure 5 depicts a NRM managed Diffserv domain. We do not currently make use of the network manager in G-QoSm – although integration of a BB is expected.

4.4. QoS Policy Manager

The policy manager aims to provide dynamic information about the domain-specific resources characteristics and the domain's policy concerning when, what and who is authorized to use resources. This policy service relies heavily on the existence of a policy repository – a storage space for the policies. Resource owners include domain-specific rules in the policy repository; for example, resource capacity allowed to be utilized with user authentication, time of the day and class of service. These rules are utilized by the policy manager to provide information on resource characteristics and domain policies. Having a separate policy manager allows the following advantages:

- The ability for resource owners to update their policy repository without interfering with the QGS.
- The resource owner may delegate a remote 'super' policy manager to act as policy controller of their resources. Similarly, a policy manager might control more than a single grid resource.
- Decoupling the policy repository from other QGS, allows the ability to dynamically change resource usage policy and system scalability.

5. RELATED WORK

Immediate and advance reservation is considered in a wide variety of systems mostly in networking, communication and distributed applications, including Distributed Multi-Media applications (DMM). Hence it is of considerable interest to the Grid community.

- In the context of Grid computing, General-purpose Architecture for Reservation and Allocation (GARA) [12] is a QoS framework that provides programmers a convenient access to end-to-end QoS. It provides advance reservations with uniform treatment of various types of resources such as network, compute and disk. GARA's reservation mechanism provides a guarantee that the client/application which initiated the reservation will receive a specific level of service quality from the resource manager. GARA also provides a reservation application program interface (API) to manipulate reservation requests, such as *create*, *modify*, *bind* and *cancel*.

- NAFUR [16] describes the design and implementation of a QoS negotiation system with advance reservation support in the context of DMM applications. NAFUR aims to compute the QoS that can be supported at the time the service request is made, and at certain carefully-chosen, later times. For example, if the requested multimedia service with the desired QoS cannot be supported at the time the service request is made, the proposed approach allows the computation of the earliest time the user can start the multimedia service with the desired QoS.

- In [17] a resource broker (RB) model, in the context of middleware for DMM application, is proposed. The proposed RB has the following design goals: 1) advance and immediate reservation, 2) a new admission control scheme based on using a timely adaptive state tree (TAST) and 3) the RB processes brokerage requests for reservation, modifications, allocation and release. The new admission control based on TAST is used to make advance reservation decisions.

- In [8] advance reservation is formalized in the context of networking systems and the fundamental problem of admission control associated with resource reservation is introduced.

Based on these systems, it is concluded that none of the previous approaches is sufficiently flexible to cover needs of users making use of Grid services and resources. The proposed solution to this fundamental problem is to separate the issue into a technical and a policy part, supported by specifying a generic reservation service description and a corresponding policy layer. This combination improves the flexibility of resource advance reservation compared to the other approaches.

None of the research efforts mentioned above address advance reservation in the context of service-oriented architecture, as in our approach. In general, resource reservation is not widely explored in service-oriented Grids. Nevertheless, the Global Grid Forum Grid Resource Agreement and Allocation Protocol (GRAAP) Working Group has produced a 'state of the art' document, which lays down properties for resource reservation in Grids [18]. It is envisaged that the reservation model in this paper, can be used to support the reservation properties outlined by the GRAAP-WG.

The features that distinguish this project from existing QoS management approaches are that:

- the generic QoS management service is not coupled to any specific resource type, or even limited to resource quantity;
- the separation between a (1) a service provider, (2) a service, and (3) a resource. A service instance may execute on one or more resource instances. We believe that the "*treat everything as a service*" viewpoint can be restricted from a QoS perspective, as particular properties of resources cannot be easily exposed.
- the object-oriented design and the abstraction approach gives the proposed service the ability to integrate with any brokerage system that supports web service interaction;
- dynamic information gathering and management, such as resource characteristics and policy information, improves scalability; and
- usage policy frameworks, for resource providers/administrators and users, enable a fine-grained request specification.

6. IMPLEMENTATION SCENARIO

As proof of concept, a reference implementation for the proposed G-QoSm architecture is designed and implemented. The implementation highlights the negotiation process and demonstrates SLA creation and reservation capability. The implementation test-bed is based on the

following technologies: Globus Toolkit 3.0, Linux Red-Hat 9.0, JSDK 1.4.2, Dynamic Soft Real Time (DSRT) schedule, UDDIe and Tomcat 4.1 application server. Figure 6 shows the implementation architecture.

In this scenario a user utilizes GUI component tools to invoke the QoS Grid Service (QGS) to negotiate a Service Level Agreement (SLA), in order to execute the desired Grid service. The SLA is generated through a negotiation process that takes place during the *Establishment Phase*. A number of components are further involved in the negotiation, and Figure 7 is a UML interaction sequence diagram showing the interaction between the various components.

Figure 8 is a screen shot of the user application GUI with the required parameters, in the upper half of the screenshot, to start the negotiation. The service request contains QoS constraints, stated by the user application, such as the maximum budget, the lowest service reputation, networking requirements and the associated importance level, or priority level, for each QoS attribute. Once the request is submitted to the QGS, the QGS carries out the discovery process by contacting the UDDIe servlet for similar services with the QoS specified. The QGS then implements a selection algorithm to select the best match based on the user's quality preference (importance/priority) stated in the service request. After the selection process, the QGS contacts the policy service to validate the request. When the request is validated, a reservation manager is contacted to co-reserve the required resources, with the reservation manager contacting the policy service to obtain resource configurations and reservation policies. Having selected a service and reserved the required resources, the QGS proposes a SLA offer for the user application and expects an agree/reject reply within a pre-defined time frame.

Figure 8, the lower half of the screenshot, shows the proposed SLA offer. If the user application accepts the offer then the SLA is stored in the SLA repository and becomes a commitment, and, subsequently, the user application can claim this Grid service, with the reservation parameters stated in the SLA, by contacting an execution service with the given SLA-ID; which execution service is beyond the scope of this paper. Otherwise, if the user application rejects the offer, or allocated time elapses without reply from the user, the SLA offer, and the associated resource reservation, is cancelled. The user application can repeat this process a number of times, with altering QoS parameters to demonstrate negotiation process.

7. CONCLUSION

An abstract model of Quality of Service (QoS) management in service-oriented architectures is presented. A mechanism for advance resource reservations is subsequently provided. Further, a QoS management system which makes use of the concepts discussed in the model is presented, comprising a QoS Grid service and a number of supporting modules, including policy manager, reservation manager, allocation manager and QoS selection service. Throughout this paper individual components of the framework are described, and their patterns of interaction outlined. An OGSA compliant prototype implementation for the G-QoSm architecture is also discussed, highlighting the negotiation process.

The key features of our approach are: the QoS manager is a Grid service and dynamically interacts with reservation and policy modules, which makes it possible for resource owners to update/modify their policies during run-time; the reservation is abstracted as a generic data structure for co-reservation support, which makes it very suitable for distributed computing, such as Grids. This abstraction allows the reservation service to operate with any underlying resources, without previous knowledge of the resource characteristics; with the association of resource characteristics taking place during run-time by querying the policy service. This novel feature demonstrates scalability – highly desirable in Grid infrastructures.

ACKNOWLEDGEMENT

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 are grateful to the participants at the “Grid and Cooperative Computing” (2003) workshop in Shanghai for their comments and suggestions for improving our contribution.

REFERENCES

1. I. Foster, C. Kesselman, and S. Tuecke, 2002. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” International Journal of Supercomputing Applications, Vol. 15, No.3.

2. “TeraGrid,” Web Page, 2001. http://www.teragrid.org/

3. “The Global Grid Forum Web Page,” May 2004 Web Page. http://www.gridforum.org

4. O. Rana, M. Winikoff, L. Padgham and J. Harland, 2002. “Applying Conflict Management Strategies in BDI Agents for Resource Management in Computational Grids.” Proceedings of the Fifth UK Workshop on Multi-Agent Systems, UKMAS 2002, Liverpool, 2002.

5. R. Al-Ali, A. Hafid, O. Rana, and D. Walker. An Approach for QoS Adaptation in Service-Oriented Grids. Concurrency and Computation: Practice and Experience Journal, 16(5): pp. 401-412, 2004.

6. Asif Akram and Omer F. Rana, "Organizing Service-Oriented Peer Collaborations", International Conference on Service-Oriented Computing, Italy 2003, pp 451-466. Springer Verlag.

7. A. Hafid and G.Bochmann, 1998. "Quality of Service Adaptation in Distributed Multimedia Applications," ACM Springer-Verlag Multimedia Systems Journal, Vol. 6, No. 5, pp. 299-315, 1998.

8. M. Karsten, N. Berier, L.Wolf, and R. Steinmetz, 1999. "A Policy-Based Service Specification for Resource Reservation in Advance," in International Conference on Computer Communications (ICCC'99), 1999.

9. H. Chu and K. Nahrstedt, 1999. "A CPU Service Classes for Multimedia Applications," in IEEE Multimedia Systems '99, 1999.

10. R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail, 2002. "G-QoSm: 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, A. Hafid, O. Rana, and D. Walker, 2003. "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.

12. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, 1999. "A Distributed Resource Management Architecture that Supports Advance Reservation and Co-allocation," in Proceedings of the International Workshop on Quality of Service, Vol. 13, No. 5, 1999, pp. 27–36.

13. I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, 2002. "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Argonne National Laboratory, Chicago, Tech. Rep., January 2002.

14. A. ShaikhAli, O. Rana, R. Al-Ali and D. Walker, 2003. "UDDIe: An Extended Registry for Web Services." Proceedings of Workshop on Service Oriented Computing: Models, Architectures and Applications, SAINT-2003 IEEE Computer Society Press, 2003.

15. B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, R. Narayan, and F. Reichmeyer, 1999. "Internet2 QBone: Building a Testbed for Differentiated Services," IEEE Networks, 1999.

16. A. Hafid, G. Bochmann, and R. Dssouli, 1998. "A Quality of Service Negotiation Approach with Future Reservation (NAFUR): A Detailed Study," Computer Networks and ISDN, Vol. 30, No. 8, 1998.

17. K. Kim and K. Nahrstedt, 2000. "A Resource Broker Model with Integrated Reservation Scheme," in IEEE International Conference on Multimedia and Expo (ICME2000).

18. J. MacLaren, "Advance reservations: State of the Art," GGF GRAAP-WG, See Web Site at: http://www.fz-uelich.de/zam/RD/coop/ggf/graap/graap-wg.html, Last visited: May 2004.

19. Rashid Al-Ali, Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Omer Rana, David Walker and Nester Zaluzec. "QoS Support for High-Performance Scientific Applications" IEEE/ACM 4th International Symposium on Cluster Computing and the Grid (CCGrid 2004) IEEE Computer Society Press. Chicago IL, USA, April 2004.

FIGURES