QoS Adaptation in Service-Oriented Grids

Rashid Al-Ali[1], Abdelhakim Hafid[2], Omer F. Rana[1] and David W. Walker[1]
[1] *Department of Computer Science*
Cardiff University, UK
{Rashid,O.F.Rana,David.W.Walker}@cs.cardiff.ac.uk
[2] *Telcordia Technologies, Inc.*
Red Bank, NJ, USA
hakim@research.telcordia.com

Abstract

Some applications utilizing Grid computing infrastructure require the simultaneous allocation of resources, such as compute servers, networks, memory, disk storage and other specialized resources. Collaborative working and visualization is one example of such applications. In this context, Quality of Service (QoS) is related to Grid services, and not just to the network connecting these services. With the emerging interest in service-oriented Grids, resources may be advertised and traded as services based on a Service Level Agreement (SLA). Such a SLA must include both general and technical specifications, including pricing policy and properties of the resources required to execute the service – to ensure QoS requirements are satisfied. A QoS adaptation algorithm is presented to enable the dynamic adjustment of behavior of an application based on changes in the pre-defined SLA. The approach is particularly useful if workload or network traffic changes in unpredictable ways during an active session. The proposed QoS adaptation scheme is used to compensate for QoS degradation and optimize resource utilization, by increasing the number of requests managed over a particular time.

1 Introduction and Related Works

Quality of service management has been explored in various contexts, particularly for computer networks [17] and multimedia applications [4]; network QoS and end-system QoS, such as memory and CPU. QoS management has also been explored in the context of Grid computing. Foster et al. [9] propose a framework for QoS in Grid computing, called the Globus Architecture for Reservation and Allocation (GARA), which enables programmers and users to specify and manage end-to-end QoS for Grid-based applications. It also provides a uniform mechanism for making QoS reservations for different types of Grid resources, such as processors, networks and storage devices. QoS management covers a range of different activities, from resource selection and allocation through to resource release. Regardless of the context, a QoS management system should address the following needs:

- Specifying QoS requirements.
- Mapping QoS requirements to resource capabilities.
- Negotiating QoS with resource owners – where a requirement cannot be exactly met.
- Establishing SLAs with clients.
- Reserving and allocating resources.
- Monitoring parameters associated with a QoS session.
- Adapting to varying resource quality characteristics.
- Terminating QoS sessions.

The G-QoSM framework [2] builds on the Open Grid Service Architecture (OGSA) [10], and aims to address the above requirements. The term 'service' in 'Grid service' refers to a software entity that offers a particular capability and is network addressable. In the G-QoSM context, QoS can be viewed as providing assurance on a set of quantitative characteristics – such as packet loss, and qualitative characteristics – such as reliability, that are necessary to execute a Grid service. The QoS adaptation mechanism outlined here provides middleware that communicates with the Globus toolkit [16] and with network managers. The middleware enables service providers to adjust their service delivery properties based on changes in the network – for example, new services added or removed, and for clients to identify their service demand constraints. An underlying assumption is that a Grid environment contains users with different service requirements – i.e. users who are willing to pay different amounts to access Grid services. Similarly, resource providers must be able to distinguish between the different classes of such users, and be able to alter their provision costs.

QoS adaptation techniques have been successfully used in multimedia applications over public and private networks, as discussed in [12] and [17]. We extend these approaches to support service-oriented Grids – requiring more generic techniques than those available for multimedia-based applications. For example, although issues such as frame-rate or packet-jitter (within a multimedia application) may be easily quantified, it is more difficult to do so in the context of Grid-based applications. There is thus a need to annotate Grid services with QoS related data, and to subsequently monitor conformance to these metrics. Generally QoS adaptation for applications, executing over different resource types, needs a complex approach to maintain an adequate coordination between such diverse resources. Optimization heuristics and an adaptation algorithm are proposed to achieve this.

1.1 Related Works

QoS adaptation can be defined as *'the alteration of an application's behavior or interface in response to arbitrary context changes'* [14]. It has been explored in various contexts, such as communication networks, distributed multimedia applications, real-time systems and Web interfaces (browsers). For example, Mobiware – developed at Columbia University [17], is a toolkit that supports adaptation at the network level. Mobiware provides programmable network objects that can be manipulated to provide applications with their desired QoS. Applications must state their QoS requirements using an Application Program Interface (API), in the form of a utility function and an adaptation policy. The utility function expresses the desired application requirements with different levels of network bandwidth, while the adaptation policy determines how the applications' bandwidth allocation should vary as resource availability changes. This work primarily focuses on network QoS.

Hafid et al. [13] designed and implemented a QoS manager responsible for undertaking negotiation and adaptation in the context of distributed multimedia applications. Based on a user profile, the QoS manager considers possible system configurations, calls system offers, and selects an optimal one – called a user offer. During playback of a multimedia document, if the network or the server becomes congested, thereby lowering presentation quality, the QoS manager dynamically considers another system configuration from the list of system offers. If an alternate system offer is selected and the required resources reserved, the QoS manager then automatically changes to the new system offer – demonstrating adaptive behavior. This work is conceptually similar to that presented here, with one exception. In [13] the list of system offers is generated by the QoS manager based on the user profile; in G-QoSM the client explicitly states the range of acceptable qualities, and the system automatically selects a different quality when the best one cannot be supported.

Chu et al. [6] designed and implemented a Soft Real-time (SRT) system for multimedia applications. SRT supports multiple CPU service classes for real-time processes based on the usage pattern of these processes. They use the notion of 'contracts' to specify the CPU service class together with a parameter used to reserve CPU time. As the processing time per frame changes dynamically for some processes, the contract parameters are adjusted accordingly to reflect the change in processor usage pattern. SRT provides a *system-initiated adaptation* that can adjust contract parameters for the real-time processes based on their actual processor usage. One noticeable feature of this adaptation is the ability to reserve just enough CPU time to execute the required processes. This adaptation technique is limited to real-time processes whilst the approach presented in this paper is more generic, and may be applied to various types of resources.

In the context of Grid computing, Foster et al. [11] designed and implemented a prototype adaptive control system based on: (i) *actuators* that permit online control, (ii) *sensors* that permit monitoring of resource allocation and (iii) *a decision procedure* that allows entities to respond to sensor information by invoking actuators. The prototype was implemented with a particular emphasis on network resource usage. For example, a loss rate sensor might acquire information from a network edge router. The decision procedure then obtains information from the loss rate sensor and adapts the network reservation using the GARA *Create/Modify* reservation request via a reservation actuator. This work is similar to that presented in this paper in the sense that both use GARA as the underlying resource manager to create and modify reservations. However the decision procedure used in [11] is different to the adaptive system presented here, and their work is only concerned with network resources.

In the context of resource management adaptation, Cardei et al. [5] presented a Real-Time Adaptive Resource Manager (RTARM), developed at the Honeywell Technology Center. RTARM is a general middleware architecture/framework for adaptive management of Integrated Services, and is targeted at real-time mission-

critical distributed applications. RTARM recognizes three situations where the QoS for an application may change: (i) QoS reduction when a new application begins, (ii) QoS expansion/improvement when an application terminates and releases resources, and (iii) feedback adaptation. Situations (i) and (ii) impose contract changes due to adaptation, and are similar to the re-negotiation ideas presented here. Feedback adaptation, conversely, does not impose contract changes but operates as a closed-loop control system, monitoring the delivered QoS and using the difference between delivered and desired QoS parameters to adapt application behavior. The feedback adaptation aims to utilize 'just enough' resources, even though the contract specifies more resources or the application uses fewer resources. The adaptive approach presented in this paper aims to allocate resources based on an SLA specification, and under-utilized adaptation is not supported. Another difference is that in a contract change, or QoS renegotiation during a QoS session, the pricing component – responsible for implementing a cost model to price resources – plays a major role in proposing new QoS offers, as in the G-QoSM framework where services are traded against cost.

2 G-QoSM Background

The Grid-QoS management framework (G-QoSM) [2] provides three main functions: 1) support for resource and service discovery, based on QoS properties; 2) provision for supporting QoS guarantees at application, middleware and network level, and management of SLAs to enforce these QoS parameters; and 3) provision of QoS management of allocated resources. G-QoSM delivers three QoS levels: 'guaranteed' QoS, 'controlled load' QoS and 'best effort' QoS (see section 5.1).

2.1 G-QoSM System Architecture

As illustrated in figure 1, G-QoSM consists of three main components: 1) an Application QoS broker/manager (AQoS) – for each deployed application; 2) a middleware Resource Manager (RM); and 3) a Network Resource Manager (NRM). AQoS is the main focus of the system presented here, and is required to interact with clients, RMs, NRMs and neighboring AQoSs. The AQoS also negotiates SLAs with clients and communicates parameters associated with an SLA to the corresponding resource manager. The AQoS is responsible for ensuring SLA conformance to allocated resources, and provides support for parameter adaptation when a

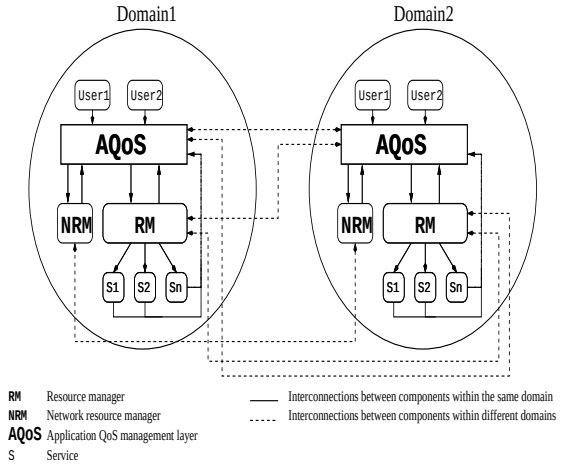

Figure 1: The G-QoSM Architecture

SLA violation is detected. The middleware resource manager (RM) exists within a given administrative domain. A domain can be defined via an IP mask or as an administrative domain in Globus, for instance, and contains a set of services over which the RM has administrative and configuration control. A RM, in this context, is considered as a combination of the Globus Resource Allocation Manager (GRAM) [8] and a Universal Description and Discovery Integration (UDDI) registry [1]. Globus is used to manage service execution, and UDDI to provide a registry and discovery system – to enable discovery of services based on their capability and QoS attributes. To support discovery of services based on their properties, the UDDI registry has been extended as UDDIe [19] – service users can now also specify particular service properties, such as QoS parameters, with which services are registered, and based on which services can subsequently be discovered. The Network Resource Manager (NRM) is conceptually a Bandwidth Broker (BB) (a concept described in [21]), and manages QoS parameters within a given domain based on the SLAs agreed to in that domain. The NRM is also responsible for managing inter-domain communication with NRMs in neighboring domains, in order 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.

An operation scenario of G-QoSM is illustrated in figure 2, outlining interactions between the various system components. All interactions are encoded as XML messages. A client contacts the AQoS broker with its service information and QoS requirements, such as reservation time and budget constraints. The AQoS queries the UDDIe registry for services with the specified QoS capabilities. The UDDIe registry sends a list of match-

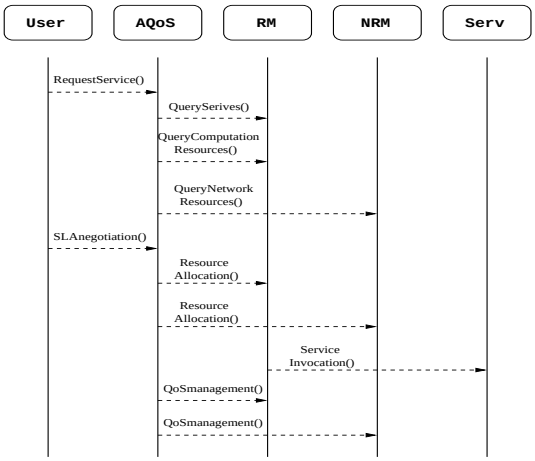

Figure 2: A Sequence Diagram Showing the Interaction between Various G-QoSM Components

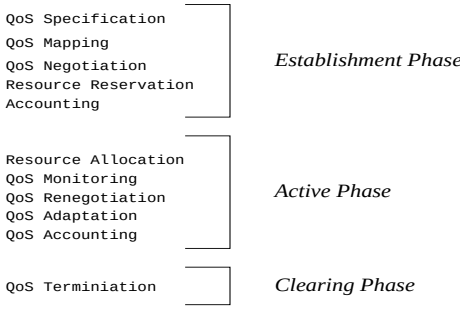

Figure 3: QoS Management Functions

ing services (if any) to the AQoS. The AQoS then contacts the corresponding resource managers, namely the NRM and Globus GRAM, to verify resource availability with the required QoS levels. This concludes the discovery phase. The AQoS and the client subsequently enter a negotiation phase aimed at reaching mutual agreement on resource QoS levels and establishing a Service Level Agreement (SLA). Once the SLA is established, its parameters are relayed to the corresponding resource managers, namely Globus GRAM and NRM, to facilitate resource allocation. Then Globus GRAM invokes the service for execution – concluding the second phase, namely resource QoS specification and SLA establishment. Once resources have been allocated and the service invoked for execution, the QoS management phase is initiated. In this phase, the AQoS queries and monitors status information on allocated resources to ensure SLA conformance, and utilises adaptation techniques to prevent SLA violation. These techniques are only applicable for 'guaranteed' QoS and 'controlled load' QoS levels, and result in some of the constraints being re-evaluated or an alert being sent to the client service.

3 QoS Management

A QoS session consists of three main phases: i) the Establishment phase, ii) the Active phase and iii) the Clearing phase [12]. Each of these phases have QoS functions as depicted in figure 3. In G-QoSM, during the Establishment phase, a client states the QoS specification and the AQoS broker undertakes the service and resource discovery, based on these QoS properties, in negotiation with the client [3]. During the Active phase, additional activities such as QoS monitoring, adaptation and possibly re-negotiation make take place. The Clearing phase is when the QoS session is terminated – due to resource reservation expiration, SLA violation or a Grid service completion, and resources are freed for use by other clients.

3.1 Resource Reservation and Allocation

Resources are temporarily reserved during the discovery phase until the client and the AQoS conclude a SLA. Once the proposed SLA is approved by the client/application, the AQoS establishes a final SLA document and saves it in the SLA repository for subsequent reference. The SLA portion that describes the resources is relayed to the corresponding resource managers (RM for computation resources and NRM for network resources) and the resource reservation status is changed from 'temporarily reserved' to that in the SLA. Table 1 shows a sample SLA portion relayed to the resource managers.

```
<Service-Specific>
        ...
   <CPU-QoS>4 CPU</CPU-QoS>
   <Memory-QoS>64MB</Memory-QoS>
   <Network_QoS>
      <Source_IP> 192.200.168.33 </Source_IP>
      <Dest_IP> 135.200.50.101 </Dest_IP>
      <Bandwidth> 10 Mbps </Bandwidth>
      <Packet_Loss> LessThan 10% </Packet_Loss>
   </Network_QoS>
</Service-Specific>
```

Table 1: Sample SLA Specification

The Allocation manager *(Alloc-M)* within the AQoS also receives its copy of the resource configuration. This triggers resource managers to identify if resource reservations can be made based on the status described in the SLA. A Reservation System *(RS)* has been designed and implemented that takes requests for resources, with specified start and end times, from the AQoS along with resource specific parameters. An example of a RM, in

the case of computational (CPU) resources is the GARA library [9][18], which is an application level interface to underlying resource managers, such as the Dynamic Soft Real-Time scheduler (DSRT) [6]. Table 2 shows sample primitives from the GARA API.

```
globus_gara_reservation_create(gatekeeper,req_rsl,
          &reserve_handle)
globus_gara_reservation_bind(reserve_handle,
          &bind_param)
globus_gara_reservation_unbind(reserve_handle)
globus_gara_reservation_cancel(reserve_handle)
```

Table 2: Sample Primitives Provided by the GARA API

In the context of GARA, resource specifications are described in Globus Resource Specification Language (RSL) [16] and used as the input parameters for reservation purposes. A successful reservation returns a reference called a *Reservation Handler*. Subsequently reservations need to be claimed before they can be used. For example, when a Grid service is launched, its process binds to a previously-made reservation using GARA primitive *globus-gara-reservationbind(...)*. This primitive binds a process to individual reservations by providing the reservation reference and the parameters needed to claim the reservation; in the case of computational resources, the process ID of the launched process is the only parameter required. Based on the primitives provided by GARA API and GARA reservation concepts, the Reservation System (RS) within the AQoS broker implements reservation as follows:

- During the discovery phase, resources are reserved on a temporary basis until the proposed SLA is approved by the client/application.
- The RS generates the appropriate resource specification RSL string, which describes the resources, and submits it to GARA for reservation.
- If reservation succeeds, a reservation reference is sent to the AQoS broker.
- RS waits for a pre-defined period of time for the corresponding reservation confirmation from the AQoS.
- If the RS does not receive such confirmation within the pre-defined period of time, it instructs GARA to cancel the reservation. Otherwise, the resources are committed.
- When the Grid service is ready to use the reservation, it must claim the reservation, and initiates a bind call to GARA with its (process) ID; this call will associate the previously made reservation with the reserved resources.

3.2 Resource Monitoring

The QoS monitoring system keeps track of Grid resources and provides information on resources, such as resource availability and utilization, to be used for adaptation purposes. QoS monitoring is an essential requirement for SLA conformance and verification. In the AQoS broker, the verification can be accomplished by a SLA conformance test on an explicit request by the client/application. The SLA verification *(SLA-Verif)* component – part of the AQoS, sends a request for QoS levels to the various resource managers. The reply is sent back to the client/application and is used to compare the actual measured QoS levels to the previously agreed QoS (in the SLA). Table 3 shows an example reply (encoded in XML) in response to a network QoS parameters request. The AQoS does not constantly monitor the QoS levels of the allocated resources; rather it relies on the *SLA-Verif* component. The *SLA-Verif* obtains QoS levels from both the NRM, for network resources, and the Globus information service (MDS) [7] for CPU QoS. The *SLA-Verif* also generates a notification of any QoS degradation of an agreed on QoS. In the case of QoS degradation the underlying resource manager attempts to rectify the problem by applying adaptation techniques at the resource management level, as outlined in [6]. If these adaptation techniques do not eliminate QoS degradation, then the AQoS applies adaptation techniques (see section 5) at the AQoS level to compensate if possible. The *SLA-Verif* uses the Java CoG Kit [22] MDS APIs to periodically retrieve QoS data. When the network QoS degrades, the Network Resource Manager (NRM) notifies the *SLA-Verif* system of such degradation.

```
<QoS_Levels>
  <SLA-ID> 1055 </SLA-ID>
  <Measured_Network_QoS>
      <Source_IP>192.200.168.33</Source_IP>
      <Dest_IP>135.200.50.101</Dest_IP>
      <Bandwidth>9.5 Mbps</Bandwidth>
      <Packet_Loss>LessThan 10%</Packet_Loss>
      <Delay>10ms</Delay>
   </Measured_Network_QoS>
         ... ....
</QoS_Levels>
```

Table 3: An XML message after a SLA conformance test showing measured network QoS levels.

4 Adaptation Scenarios

QoS adaptation is a key function of QoS management during the Active phase of a session. The response of the AQoS should result in either (a) restoring the agreed on

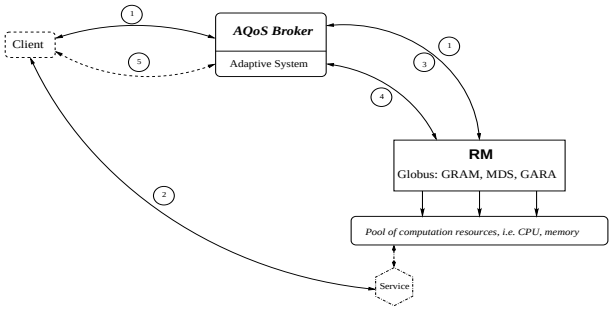

Figure 4: An interaction between Client, AQoS Broker, RM and the Grid Service through the following QoS management phases: 1) QoS negotiation and SLA establishment, 2) resource allocation, 3) resource monitoring, 4) QoS adaptation, and 5) QoS re-negotiation.

QoS (in SLA); (b) re-negotiating QoS as per the SLA; or (c) terminating the service being delivered due to a major QoS degradation. Figure 4 shows the interactions between a client, AQoS, RM and Grid Services through different phases of QoS management including QoS adaptation. Three scenarios in the context of the G-QoSM framework, where adaptation is required are described.

Scenario 1: New Service Request : In this scenario a new service request is received but there are insufficient resources to accommodate the request. Adaptation can be used to free resources to accommodate the new request by adjusting resource allocations of active services while still satisfying their SLAs. The adaptation function queries the AQoS broker about the list of currently active services. The list is filtered to include only those services whose SLAs indicate willingness to accept a degraded QoS and/or termination of service to support compensation.

Scenario 2: Service Termination : In this scenario a service completes successfully, and its resources are released. Adaptation can be used to increase resources allocation for a selected number of existing services while still satisfying their SLAs; the objective is to use the released resources and thus increase the profits of the service provider. This can be realized by (a) upgrading the QoS of existing services that had their QoS reduced; or (b) upgrading the QoS of existing services that are not currently receiving the 'best' QoS, as defined in their SLAs; or (c) presenting promotion offers to existing services for upgrading their QoS to attract additional resource requests.

Scenario 3: QoS Degradation : This scenario is the classical QoS adaptation situation where QoS falls below the specified QoS level (i.e. minimum acceptable QoS) in the SLA. The QoS degradation is detected either by the resource monitoring system or by an explicit notification from the underlying resource manager. Adaptation is used, if possible, to restore the degraded QoS to an acceptable QoS as defined in the SLA.

5 QoS Adaptation Scheme

The QoS adaptation scheme to realize the scenarios described in section 4 are outlined. Section 5.1 describes the QoS classes supported by the scheme; these classes correspond to the ones defined in the G-QoSM framework [2]. Section 5.2 discusses SLA and how it is used by the adaptation scheme. Section 5.3 introduces an optimization heuristic used by the adaptation scheme to adjust resource allocation to optimize resource utilization. Section 5.4 presents the adaptation algorithm; based on reserving extra resources for guaranteed services. Finally, an example of the operation of the adaptation scheme is presented in Section 5.6.

5.1 QoS Classes

The G-QoSM framework adopts a service model which classifies the service delivery into 3 distinct classes: (1) 'guaranteed' service [20]; (2) 'controlled load' service [15]; and (3) 'best effort' service. The 'guaranteed' service provides QoS based on pre-defined constraints identified by the user, and agreed on by the provider within a SLA. These constants are specified using pre-agreed parameters, and must be supported by the Grid service provider. In this type of service, QoS parameters are enforced and monitored; the service provider is committed to deliver the service with the exact QoS specification described in the SLA. In the 'controlled load' service, users state their QoS requirements based on parameter ranges; the service provider must now be able to offer QoS within the specified range. In the 'best effort' service, there is no SLA associated with the service request – which corresponds to the default case where no QoS provision is taking place. In this class, any suitable resources found are returned to the user.

5.2 SLA and QoS Adaptation

Choosing the appropriate adaptation strategy and its constituent parameters relies on terms that have been agreed on, in advance, during SLA establishment. These involve, for example, acceptable levels of resource quality, inter-dependencies between resources and SLA violation penalties.

There are 2 essential SLA elements that must be agreed on during QoS negotiation, and which impact on adaptation decisions: (1) based on the selected class of service, a level of total acceptable quality must be established. For example, in the case of 'controlled load' class the user would specify the range within which an acceptable QoS level must fall. (2) Only in the 'controlled load' class is there an optional element related to 'promotion offers' during service execution. The QoS negotiation phase, when a SLA is established, plays a major role in constraining the adaptation strategy, having control of the parameters that execute the adaptive functions. Table 4 is an example of a SLA generated from a negotiation process.

```
<Service_SLA>
     ...........
   <QoS_Specification>
     ...........
   </QoS_Specification>
   <QoS_Class> Controlled-load </QoS_Class>
   <Adaptation_Options>
      <Alternative_QoS>
          <CPU> 55 nodes on Linux OS </CPU>
          <Memory> 48 MB </Memory>
          <Bandwidth> 45 Mbps </Bandwidth>
      </Alternative_QoS>
      <Promotion_Offer>Accept</Promotion_Offer>
   </Adaptation_Options>
</Service_SLA>
```

Table 4: A sample negotiated SLA document encoded as an XML message highlighting the adaptation strategy.

5.3 Resource Allocation Optimization

Within the G-QoSM framework there can be a number of different users, each requesting a particular QoS. This quality level must be agreed on in the negotiated SLA – consisting of the quality parameters required by a user, along with other service management parameters, such as service name, service class and duration. If all the parameters associated with QoS are extracted and expressed as set QoS, then $QoS = \{a_1, a_2, ..., a_n\}$, where each a_i represents a different parameter of interest (e.g. cache, primary memory, CPU capability and bandwidth). One is now able to compare two different QoS sets, by comparing each element of the set; hence if $QoS_x = \{a_1^x, ..., a_n^x\}$ and $QoS_y = \{a_1^y, ..., a_n^y\}$, then one can compare a_i^x with a_i^y. Furthermore, QoS parameter values a_i may be recorded in the SLA in two forms: (1) based on a parameter range; such that: $a_y \leq a_i \leq a_x$; where a_x is a better quality than a_y; implying that the user requires a minimum of a_y level of quality, but it would be better, from the user's point of view, to receive an a_x level of quality, and (2) based on a list – where the user states distinct values for a particular QoS parameter, for example: $a_i = \{x, y, z\}$; where x, y and z are integer numbers representing the acceptable values for QoS parameter a_i. Each QoS a_i has a corresponding cost c_i[1]; where c_i is a constant, related to the pricing formula for the class of service assigned to this user. The monetary cost for a particular QoS parameter may be calculated as $Cost(a_i) = c_i * a_i$, and, subsequently, the monetary cost of the QoS set for a particular service may be calculated as:

$$Service\_Cost(QoS) = \sum_{i=1}^{n} (c_i * a_i)$$

Given the above assumptions, the optimization problem can be defined as:

$$Total\ Cost = max \sum_{i=1}^{n} (Service\_Cost(QoS_i))$$

where n represents the total number of active services. The AQoS implements this optimization by varying the resource quality selection, based on supplied levels of quality in the SLA, which aims to maximize overall monetary profit, while maintaining the user's acceptable quality.

5.4 Adaptation Algorithm

Unlike the optimization heuristic, this adaptation algorithm only operates on the 'guaranteed' and 'best effort' classes. As the 'guaranteed' class of user receives the highest level of attention, it is important to provide them with extra assurances through adaptation approaches. The algorithm requires the system administrator to specify the total resource capacity specified for the 'guaranteed' and 'best effort' users. The term 'resource capacity' encompasses CPU, network and storage resources. The algorithm reserves an 'adaptive ca-

[1] Although specified as a "cost", these weighting parameters may also have other semantic interpretations, such as priority or user preference

pacity', based on the specified rate of resource failure or congestion provided by the system administrator. The algorithm also considers a minimum capacity for 'best effort' clients, as determined by the system administrator. These capacity allocations are dynamic in that if the adaptive and/or guaranteed capacities are not used, then the 'best effort' capacity compensates and utilizes free resources, provided they are not currently allocated. The Algorithm starts execution by invoking: (a) the *Allocate_Guaranteed_Resource* or (b) the *Allocate_Best_Effort_Resource* function, as outlined in Algorithm 1.

The proposed adaptation algorithm has the following advantages: (a) Resources are never under-utilized due to the dynamic property of the algorithm. The extra reserved capacity is used by 'best effort' users as long as it is not needed by 'guaranteed' users; and (b) a minimum resource capacity is allocated for 'best effort' users, therefore users with no SLAs can always make use of the 'best effort' resources.

5.5 Adaptation Strategies of Grid Services

The adaptation scheme is based on the above algorithm, and the resource allocation optimization described in section 5.4 and section 5.3 respectively. The optimization heuristic is executed periodically by the AQoS broker; if there is a considerable gain in terms of benefits to the Grid Service provider, resources allocation is accordingly modified. On receipt of a request from a 'guaranteed' client, the adaptation algorithm (section 5.4) is applied; if the request cannot be accommodated, the optimization heuristic is executed.

5.6 Example

An example to illustrate the operation of the proposed adaptation scheme is presented here, with the emphasis on computation resources, such as CPUs. Assume a group of scientists are about to conduct a simulation experiment using Grid services and infrastructure. The experiment will run at site A on an SGI multiprocessor machine with 64 CPU/processor nodes and 10 GB of memory. The database, in which the required data for the simulation resides, is located at site B. A second group of scientists participating in the simulation experiment are located at site C. The resources required for this experiment are:

Algorithm 1 *QoS adaptation*

C: the total resource capacity
C_G: the 'guaranteed QoS' capacity
C_A: the adaptive capacity
C_B: the 'best-effort QoS' capacity
Then $C = C_G + C_A + C_B$

U: set of ALL user $U = \{u_1, ..., u_n\}$
G: set of users of class 'guaranteed' $G = \{v_1, ..., v_n\}$
B: set of users of class 'best effort' $B = \{w_1, ..., w_n\}$
$c(u,t)$ = capacity required at time t by user $u \in G$
$b(u,t)$ = capacity required at time t by user $u \in B$
$g(u)$ be the guaranteed capacity with a SLA for user $u \in G$

Available_Guaranteed_Resource(g(u))
if $\sum_{u \in G} g(u) \leq C_G$ **then**
  SLA guarantees to g(u) can be honored
end if

Adapt()
Net capacity $N_G(t) = C_G(t) - \sum_{u \in G} g(u)$
if $N_G(t) < 0$, (guarantees cannot be honored at time 't') **then**
  $ADD(\sum_{u \in G} g(u) - C_G(t))$ from A to G
  $ADD(C_A(t) - [\sum_{u \in G} g(u) - C_G(t)])$ from A to B
end if

Allocate_Guaranteed_Resource(c(u,t), g(u))
if $\{c(u,t) \leq g(u)$ **then**
  $c(u,t)$ capacity must be given
else if NOT Available_Guaranteed_Resource(g(u)) **then**
  Adapt; allocate c(u,t) capacity
else if $\{c(u,t) > g(u)$ **then**
  only g(u) capacity is given
  $c_{new(u,t)} \leftarrow g(u)$
  Allocate_Guaranteed_Resource$(c_{new(u,t)}, g(u))$
end if

Allocate_Best_Effort_Resource(b(u,t))
if $b(u,t) \leq N_B(t)$; $(N_B(t) = C_B(t))$ **then**
  allocate $b(u,t)$
else
  cannot allocate the required capacity
end if

- 622 Mbps communication link to connect site B and site A.
- 45 Mbps communication link to connect site C and site A.
- 10 processor nodes, 2 GB of memory and 15 GB of disk space at site A.

The resources must be allocated over the duration of the experiment – $(t_5\ to\ t_9)$. The SGI machine is configured to provide 26 processor nodes to all Grid users, with the rest dedicated for local processing. The Grid system administrator partitions the 26 processor nodes as:

$C_G = 15$, $C_B = 6$ and $C_A = 5$ processor nodes
$C = C_G + C_B + C_A$ = 15 + 6 + 5 = 26 processor nodes

A composite SLA was negotiated with the AQoS based on 3 sub-SLAs over the period $t_5\ to\ t_9$:

SLA_1: network bandwidth of 622 Mbps from site B to site A
SLA_2: network bandwidth of 45 Mbps from site C to site A
SLA_3: 10 processor nodes, 2 GB of memory and 15 GB of disk space on the SGI machine at site A

The following measurements are recorded during the period t_0 through t_9. Note the subscripts 'a' and 'u' correspond to $available$ and $used$ resource CPU nodes respectively.

- At t_0 to t_3 the processor nodes allocation is as follows:
  C_G: u = 10, a= 5
  C_B: u = 6, a= 0
  C_A: u = 0, a= 5; adaptive capacity from C_G point of view
  C_A: u = 4, a= 1; adaptive capacity from C_B point of view
- At t_4
  C_G: u = 4, a= 11
  C_B: u = 6, a= 0
  C_A: u = 0, a= 5; C_G point of view
  C_A: u = 3, a= 2; C_B point of view ('best effort' users use resources in an unpredicted pattern)
- At t_5: three processors from C_G resource pool become inaccessible, and therefore: $C_G = 12$ processor nodes. Also SLA_3 is due $\Rightarrow$ allocating $g(SLA_3) = c(SLA_3, t_5)$= 10 processors

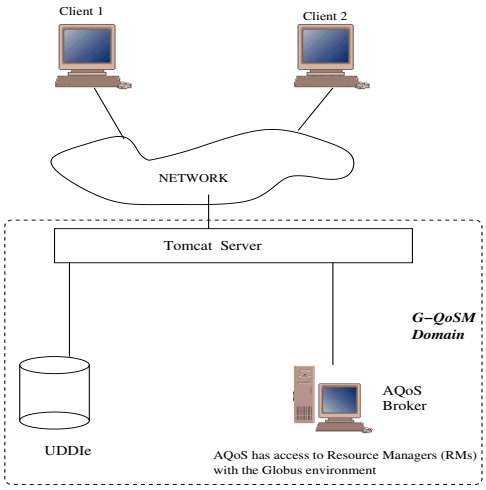

Figure 5: G-QoSM Test-bed Architecture: Clients send XML messages to the AQoS broker using SOAP over HTTP. The AQoS and the UDDIe are server processes running within a Tomcat application server. The AQoS has control over the resource managers.

  C_G: u = 14, a= (1); to be brought from C_A when is required.
  C_B: u = 6, a= 0
  C_A: u = 2, a= 3; C_G point of view
  C_A: u = 3, a= 0; C_B point of view
- At t_8: the three inaccessible processors become accessible, and now:
  C_G: u = 14, a= 1
  C_B: u = 6, a= 0
  C_A: u = 0, a= 5; C_G point of view
  C_A: u = 3, a= 2; C_B point of view
- At t_9: SLA_3 has completed its validity period:
  C_G: u = 4, a= 11
  C_B: u = 6, a= 0
  C_A: u = 0, a= 5; C_G point of view
  C_A: u = 3, a= 2; C_B point of view

6 Current Implementation Status

The G-QoSM framework is a three phase project: (1) investigation design and implementation of a discovery system with QoS support, (2) investigation design and implementation of a QoS broker, and (3) study of domain-specific QoS requirements for an application framework and integrating it with the G-QoSM. Currently phase 3 is being investigated.

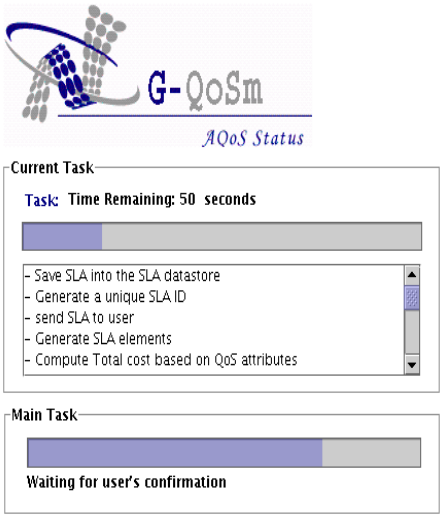

Figure 6: A screenshot, showing activities undertaken by the AQoS broker

The implementation test-bed is built on RedHat Linux 7.2 and the Globus toolkit v2.0. The programming tools are: Java2 SDK Version 1.4.0, Java CoG kit, UDDIe (an extended version of the UDDI) and the Tomcat application server. A QoS broker (AQoS) which supports the functions outlined in section 1 is implemented. The developed QoS broker is integrated with the Dynamic Soft Real-Time (DSRT) scheduler [6] as the computation (CPU) scheduler – which operates in a single processor and multiprocessor system. GARA's DSRT resource manager API is used to facilitate the interaction between the QoS broker and the DSRT scheduler.

The overall architecture is depicted in Figure 5; a client interface application starts at the client side; the client application communicates with the AQoS broker using SOAP messages over HTTP protocol. The AQoS and the UDDIe are server processes and reside in the Tomcat application server as servlets within a Globus-managed environment. The AQoS communicates with the DSRT scheduler through GARA's DSRT manager API for resource reservation and allocation.

Figures 6 and 7 are screen shots taken from the prototype implementation to demonstrate activities outlined in the sequence diagram in Figure 2. Figure 7 is a client interface screen – the client has to fill out the `service_request` message and send it to the AQoS 'servlet'; the lower half of this screenshot shows the response from the AQoS. In this case it is a service offer based on the supplied QoS criteria for the desired service. A client interface is used, primarily for demonstration purposes; however, in practice applications should

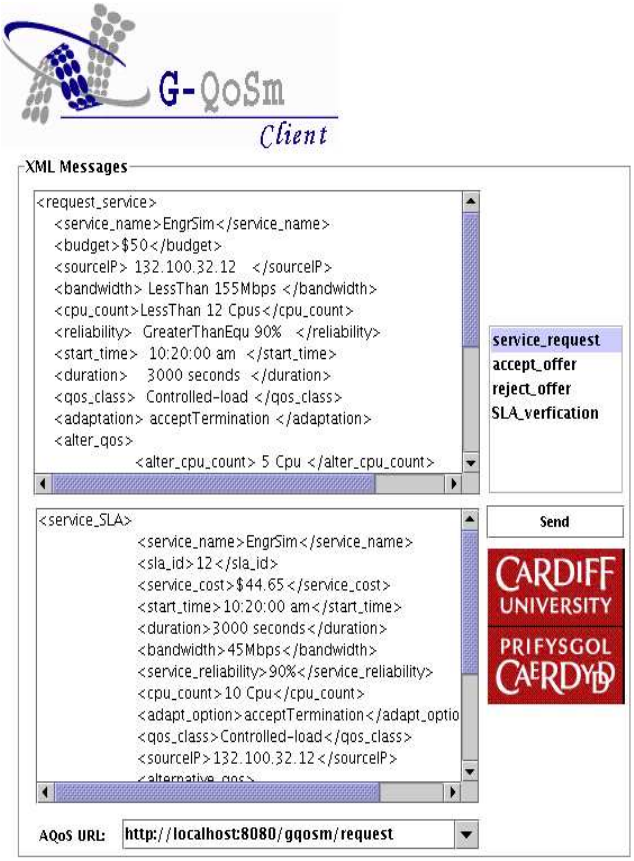

Figure 7: A client interface screenshot, showing the client entered a 'service_request' and the AQoS broker replied with a service offer

themselves generate the `service_request` messages and contact a SOAP server to transmit messages. The client interface screen has four options (on the right-hand side): (a) requesting a service with QoS properties, (b) accepting SLA offers, (c) rejecting SLA offers, or (d) requesting an explicit SLA verification test. Figure 6 provides a screenshot of activities undertaken by the QoS broker to accomplish the specified request, such as, contacting the UDDIe registry, reserving resources and computing the total service QoS cost. The system administrator may also use this interface to see service offers from the AQoS, and subsequent client approval or rejection of the offer.

7 Conclusions

QoS management and QoS adaptation is defined in the context of the G-QoSM framework. A generic adaptation model is outlined based on reserving extra re-

source capacity to guarantee resources for the 'guaranteed' class of users if there is resource failure or congestion. The dynamic nature of this model allows unused resources to be more effectively utilized. The implementation of the resource reservation and monitoring features, as the underlying tools for the adaptation functions, is also described. An optimization heuristic to optimize resource utilization is proposed, which allows the system to maximize monetary benefits to the Grid service provider; with the basic concept being to enable better resource allocation while satisfying pre-agreed SLAs. The adaptation scheme aims to provide the best possible resource quality within a dynamically changing environment. As a future topic it is planned to evaluate this adaptation technique in the context of a particular Grid application.

Acknowledgement

We would like to acknowledge the work of Ali ShaikhAli, of Cardiff University, for his implementation of UDDIe – and the initial idea for this system from Vijay Dialani of Southampton University, UK. We also appreciate contributions from Jonathan Giddy of the Welsh eScience Centre, especially for his comments on utilizing Globus in the G-QoSM framework.

References

[1] Universal description, discovery and integration of business for the web. See Web site at: http://www.uddi.org.

[2] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail. G-QoSM: Grid service discovery using QoS properties. *Computing and Informatics Journal, Special Issue on Grid Computing*, 21(4):363–382, 2002.

[3] R. Al-Ali, A. ShaikhAli, O. Rana, and D. Walker. Supporting QoS-based discovery in service-oriented grids. In *Proceedings of IEEE Heterogeneous Computing Workshop (HCW'03)*, Nice, France, 2003.

[4] G. Bochmann and A. Hafid. Some principles for quality of service management. Technical report, Universite de Montreal, 1996.

[5] I. Cardei, R. Jha, M. Cardei, and A. Pavan. Hierarchical architecture for real-time adaptive resource management. In *IFIP/ACM International Conference on Distributed Systems Platforms*, pages 415–434, 2000.

[6] H. Chu and K Nahrstedt. A cpu service classes for multimedia applications. In *IEEE Multimedia Systems '99*, 1999.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid infromation services for distributed resource sharing. In *Proc. of the 10th IEEE High Performance Distributed Computing*, pages 181–184, 2001.

[8] K. Czajkowski, I. Foster, C. Kesselman, et al. A resource management architecture for metacomputing systems. In *Proc. IPPS/SPDP 98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[9] I. Foster, C. Kesselman, et al. A distributed resource management architecture that supports advance reservation and co-allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36, 1999.

[10] I. Foster, C. Kesselman, et al. The physiology of the grid:an open grid services architecture for distributed systems integration. Technical report, Argonne National Laboratory, Chicago, January 2002.

[11] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the 8th International Workshop on Quality of Service (IWQOS)*, pages 181–188, Pittsburgh, PA, June 2000.

[12] A. Hafid and G.Bochmann. Quality of service adaptation in distributed multimedia applications. *ACM Springer-Verlag Multimedia Systems Journal*, 6(5):299–315, 1998.

[13] A. Hafid, G.Bochmann, and B. Kerherve. A quality of service negotiation procedure for distributed multimedia presentational applications. In *HPDC '96*, pages 330–339, 1996.

[14] K. Henricksen and J. Indulska. Adapting the web interface: An adaptive web browser. In *Second Australasian User Interface Conference (AUIC'01)*, 2001.

[15] J.Wroclawski. Specification of the controlled-load network element service. Internet Engineering Task Force, RFC 2211, 1997.

[16] Argonne National Laboratory. The globus project. See Web Site at: http://www.globus.org/, Last visited: February 2003.

[17] A. Oguz et al. The mobiware toolkit: Programmable support for adaptive mobile networking. *IEEE Pesronal Communications Magazine, Special Issue on Adapting to Network and Client Variability*, 5(4), 1998.

[18] A. Roy. *End-to-End Quality of Service for High-End Applications*. PhD thesis, The University of Chicago, August 2001.

[19] 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*, pages 85–90, Orlando FL, USA, 2003.

[20] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. Internet Engineering Task Force, RFC 2212, 1997.

[21] B. Teitelbaum, S. Hares, L. Dunn, R. Neilson, R. Vishy Narayan, and F. Reichmeyer. Internet2 qbone: Building a testbed for differentiated services. *IEEE Network*, 13(5):8–17, September 1999.

[22] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001.