On Quality of Service Support for Grid Computing

D. Colling[3], T. Ferrari[1], Y. Hassoun[3], Chenxi Huang[4], C. Kotsokalis[2], A. S. McGough[3], Y. Patel[3], E. Ronchieri[1], P. Tsanakas[2]

[1] National Institute of Nuclear Physics, CNAF, Italy
{ferrari,ronchieri}@cnaf.infn.it
[2] National Technical University of Athens, Greece
ckotso@cslab.ece.ntua.gr, panag@cs.ntua.gr
[3] Imperial College London, UK
{d.colling, y.hassoun, asm, yash.patel}@imperial.ac.uk
[4] Brunel University, UK
chenxi.huang@brunel.ac.uk

Summary. Computing Grids are hardware and software infrastructures that support secure sharing and concurrent access to distributed services by a large number of competing users from different Virtual Organizations. Concurrency can easily lead to overload and resource shortcomings in large-scale Grid infrastructures, as today's Grids do not offer differentiated services. In addition, just a minor set of Grid sessions can be subject to load-balancing policies, as this would require the widespread usage of scheduling services. We believe that edge applications with specific Quality of Service requirements can benefit from mechanisms for the support of both *strict* and *loose* guarantees. We propose a framework for supporting Quality of Service guarantees via both reservation and discovery of best-effort services based on the matchmaking of application requirements and Quality of Service performance profiles of the candidate services. We illustrate the middleware components needed to support both strict and loose guarantees and the performance assessment techniques for the discovery of suitable services.[5]

Key words: Grid Computing, Quality of Service, Performance Measurements, Monitoring

1 Introduction

Quality of Service (QoS) is a broad term that is used, in this context, to denote the level of performance that a given client experiences when invoking an operation on a remote server. QoS support refers to the possibility of a given

[5] Contact authors: T. Ferrari, C. Kotsokalis

service instance to offer a performance level that satisfies the requirements of a given client.

QoS support is of paramount importance given the inherent shared nature of Grid services, and the limited capabilities of hardware and software resources available to satisfy client requests. In this environment, an aggressive best-effort usage of services can result in denial of service problems, as the non-policed use of services is against their effective provision. The more recent efforts for integration of scientific instrumentation and the related requirements for high availability and responsiveness, urge for the support of differentiated services.

QoS requirements can be of many different kinds, taking under consideration the heterogeneous nature of Grid resources which the applications utilize. For instance, applications running on computing Grids, can be sensitive to the time needed to complete individual tasks and may impose time execution boundaries to computing resources. Similarly, e-Science applications are often CPU demanding and can benefit from the availability of reserved RAM space on a given set of worker nodes.

Conversely, Instrument Grids, supporting the remote control and monitoring of distributed scientific and general-purpose instrumentation, need to allow exclusive access to instruments by users or groups of users, and to guarantee low delay and high responsiveness, given the interactive nature of many operations exposed by instruments.

The support of QoS requirements needs the adoption of specific enforcement techniques in the service backend. Exclusive access typically requires resource locking techniques, while other types of QoS parameters, such as responsiveness, are dependent on a number of different factors - both static and dynamic - such as the client and server physical location, the status of the communication infrastructure, the client/server implementations, the type of operation invoked, the number of concurrent clients served, etc.

We propose the support of two complementary types of guarantees: *strict* and *loose*.

Strict guarantees: the service provider that is in charge of delivering guarantees, offers a certain specific QoS profile for a given time duration on a contractual basis to the service consumers involved in the corresponding agreement. A strict guarantee requires the establishment of such a formal agreement via signaling and negotiation. This process involves both service consumers and service providers.
Strict guarantees require the availability of fabric-level mechanisms at the service provider end for the enforcement of the contract. These mechanisms are service-specific and depend on the type of agreement established.

Loose guarantees: the provisioning of loose guarantees consists of the capability of clients to select service providers that, on a statistical basis, exibit a best-effort QoS profile which meets the client's requirements. In this case,

service discovery relies on the monitoring of critical performance parameters over time. Gathered results combined with appropriate statistical estimation methods can assist clients and schedulers during the selection phase.
Loose guarantees are delivered on a best-effort basis, and for this reason, they do not require the prior establishment of a *Service Level Agreement* (SLA) and the consequent negotiation of a contract. If consumers experience a service level that contradicts the past QoS profile exhibited by the service providers, compensation, adaptation and/or other self-healing operations are needed to cope with requirements that are not met in practice. Adaptation has an inherent cost, which is however compensated by the possibility to avoid the overhead of the SLA negotiation process.

We propose a framework of Grid services for the support of both types of guarantees, which relies on a performance monitoring infrastructure for loose QoS. Firstly, we address the problem of establishing agreements for strict guarantees in Section 2 and of the delivery of QoS guarantees for atomic tasks that need to be executed instantly in Section 3. Then we analyze a list of performance metrics for loose QoS in Section 4, defined on the basis of the requirements of interactive applications, such as the ones requiring Instrument Grids. Then we elaborate on our methodology for measurements in Section 4.3. We continue to provide a framework for publishing and archiving relevant information in Section 6. Finally, we present a preliminary prototype in Section 7 and related work in Section 8. We conclude the paper and elaborate on future work in Section 9.

2 Strict Guarantees

The application of a specific QoS level to a Grid task requires the negotiation and establishment of a contract, named Service Level Agreement, between the customer or proxy, and the service provider. A SLA quantitatively defines the performance level for the service requested and the obligations for the parties involved in the contract. The SLA is typically specified through a template containing both quantitative and qualitative information. In particular, a SLA specification supplies administrative information (e.g., the identity of the entities involved in the contract, and the penalties applicable to the parties when the SLA guarantees are not honored), whereas the *Service Level Specification* (SLS) is a set of attributes and values describing the profile of the requested performance level [5].

The Grid Resource Allocation and Agreement Protocol Working Group (GRAAP-WG) of the Global Grid Forum (GGF) proposes an architecture for Grid SLA signalling comprising two functional levels: the *agreement layer* and the *service provider layer* [9]. The agreement layer implements the communication protocol used to exchange information about SLAs between the customer and the service provider, and it is responsible for ensuring that the SLA guarantees are enforced by a suitable service provider. In addition, the agreement layer exposes information about types of service and the related SLA templates offered, processes agreement creation requests to check the compliance of the so-called agreement offers (i.e. the SLA requests), and handles well-formed requests to the service provider layer. An agreement is successfully created if the corresponding guarantees can be enforced by one or more service providers, which are also responsible for supervising the status of their resource pools.

Reservation and allocation in Grids can be assisted by existing resource management capabilities. In this paper, we address the problem of the support of resource-independent agreement signalling by Grid Workload and Resource Management services (WMS), and we describe a prototype implementation developed in the framework of the GRIDCC and EGEE projects [2]. The WMS comprises a set of middleware components responsible for the distribution and management of tasks across Grid resources and services[6].

Scheduling policies define the internal strategy which aims at maximizing the individual task execution efficiency, and, at the same time, at optimizing resource and service usage. Workload Management greatly simplifies the usage of Grids by handling execution errors, by supporting the discovery of the services satisfying the user or application requirements, by supporting re-submission in case of failure and by providing logging and bookkeeping facilities which can increase the robustness of the entire Grid infrastructure [15, 1].

2.1 SLA Signalling

We propose an approach to SLA signalling based on the instrumentation of the WMS and on the availability of a new collective service, which is called the *Agreement Service* (AS). Resource broker instrumentation is needed in order to provide the capability of handling the allocation and reservation of Grid resources. The WMS supports SLA signalling phase, the discovery of service instances supporting the user's requirements, and the monitoring.

We borrow the notations of *Agreement Initiator*, *Agreement Offer* and *Service Provider* from the WS-Agreement proposed specification of the GRAAP-WG of the GGF [9]. In particular, as we focus on reservation and allocation, in this context the provider is called *Reservation and Allocation Service Provider* (RASP). The proposed architecture is illustrated in Figure 1, which details the interactions between the above components. Four types of resources are shown: the Computing Element (CE), the Storage Element (SE), the network (Network Service Access Point [6]) and the Instrument Element (IE) [10].

[6] In this context the term *resource* refers to fabric-layer facilities, e.g. computing farms, data storage systems and networking infrastructures. Conversely, we use the term *service* to refer to a general service instance regardless of the provided functionality level with respect to the Grid service architecture.

The agreement initiator is the entity which triggers the SLA signalling process. In our scenario, users, applications or Grid middleware components can be initiators provided that the identity of the resources to be involved in the SLA signalling is known. If available, this identity is defined in the SLA and the allocation and reservation task can be submitted directly to the AS. While direct negotiation with AS is possible, nevertheless, very frequently users have no a-priori knowledge of the Grid infrastructure. When this applies, we propose the allocation and reservation task to be submitted to the WMS in order to take advantage of its discovery capabilities. After the submission the WMS determines both the RASPs which can satisfy the task requirements, and the AS's which are capable of supporting the corresponding RASP interfaces. In this scenario, the WMS has the agreement initiator role and acts on behalf of the user or application.

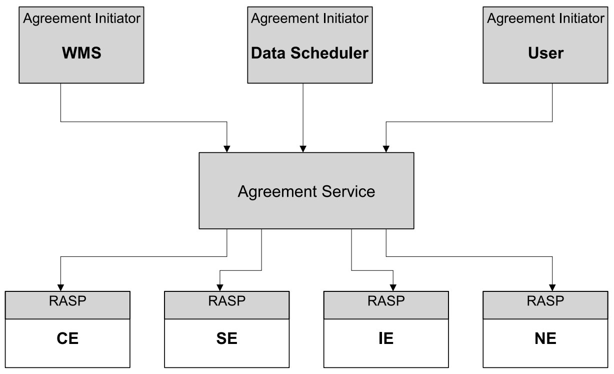

Fig. 1. Agreement Service and the other components of the proposed architecture

When the AS receives an offer, it initially checks SLA compliance to the appropriate corresponding template, it extracts attributes from the SLA request, and, finally, it dispatches them to a number of RASPs by invoking the appropriate interface.

2.2 Reservation and Allocation Service Provider

RASPs are fabric-layer resource-specific services that are responsible for the management of the availability of the local resource over time, for admission control, for the control of the satisfiability of each authorized incoming requests and, finally, for the enforcement of the guarantees. We propose each reservation-capable resource to be associated to an individual RASP instance. A computing farm, a data storage system or a network domain are examples of resources supervised by a RASP.

Admission control is about checking if the requested service can be actually guaranteed to the user, and applying the configuration techniques needed to enforce the guarantees when agreed reservations start. It depends on the user's identity, on the policies applicable to her/him and to the availability of a sufficient amount of resources. Enforcement of service guarantees requires actions on the physical resources. An example is the configuration of network devices in case of bandwidth reservation.

The RASP is invoked by the authorized AS's that support both the service type requested by the user specified in the agreement offer, and the appropriate RASP interface. A given agreement can be successfully created only if one or more RASPs (depending on the nature of the agreement) accepted the responsibility of providing the guarantees associated with it.

We assume the RASP to be associated with an endpoint address, which unambiguously identifies it. In addition, the RASP is characterized by the type of interface exposed, its capabilities and the resources it controls. This type of information is necessary to assist the WMS during the resource discovery phase, and it is supplied by the resource information schema available from the Information System.

2.3 Agreement Service

The AS is the novel service component that provides SLA management capabilities to the WMS. We propose the AS to be responsible of interacting with the agreement initiator (as a server) and with the RASPs (as a client) which can satisfy the initiator's request. In particular, the AS accepts agreement offers from initiators and is responsible for compliance checking as well as for the interaction with one or more RASPs in case of success.

The AS exposes RASP capabilities through agreement templates [9]. The template is the SLA skeleton and includes a list of creation constraints applicable to the SLS. The templates advertised by one AS explicitly define the types of service the agreement service can handle, and consequently depend on the service providers the AS interacts with. The AS we propose can offer different templates for the same type of service, in order to offer various types of agreements to different user groups according to their Virtual Organization (VO) membership. Membership, role and capability information is provided by the VO Membership Services (VOMS) [11].

The AS can perform negotiation in order to assist initiators during the establishment of SLS whose characteristics depend on the current status of the RASP. For every well-formed offer, SLS attributes are translated to low-level RASP-specific terms according to the RASP interface.

Finally, the AS supplies information about the status of agreements under negotiation and the attributes of the agreements successfully established.

2.4 Agreement Initiator

The initiator is responsible for triggering the negotiation of a new agreement by interacting with the AS, as defined in [9]. Initiators can be the reservation consumer itself or a proxy. The former case is applicable when the consumer has a-priori knowledge of the resources to be reserved. Alternatively, the consumer can decide to submit the request to the WMS, which performs discovery and provides information to the AS about the candidate RASPs. We expect that the agreement initiator will provide information to the AS about the candidate RASPs, as this approach gives the opportunity to use existing WMS capabilities and it reduces the overall AS complexity.

Requests are originated by the user or application and submitted to the WMS. We propose them to be modelled following the agreement offer structure defined in [9]. Resource requirements and preferences are part of the SLS and they consequently need to be specified as a list of *Service Description* and *Guarantee Terms*. We identify four term categories: *general* attributes (type of task requested and corresponding functionality), *time* attributes (information about the start time, end time and duration of the reservation), *functionality* attributes (reservation-specific information) and *guarantee* attributes (definition of the QoS profile).

3 Loose Guarantees

As explained in Section 1, the support of strict guarantees is highly dependent on the availability of mechanisms for SLA enforcement and it is subject to SLA signalling overhead. Consequently, the requirements of lightweight applications may more easily addressed in a best-effort fashion via discovery and prediction of performance in the future.

The performance experienced when invoking an operation on a remote service instance, can largely benefit from the availability of loose guarantees, as it depends on a large number of factors. For example, asynchronous operations typically just exchange one-way messages from the client to the server, while synchronous operations imply the additional issuing of a reply message back to the client. Thus, the client/server interaction profile typically depends on the network path connecting the client and the server, and the processing overhead during the entire messaging process at both ends, this including request/reply message composition/decomposition and processing.

We propose a novel architecture and methodology for the delivery of loose guarantees, which relies on three functional components.

Monitoring: the availability of distributed sensors for the monitoring of the QoS performance profile of various Grid infrastructure components, both at the fabric and resource layer;

Publishing: data gathered from the sensors must be made available to distributed consumers;

Discovery: the availability of service discovery engines is required to allow clients to select the most appropriate service instance(s), by comparing the QoS profiles of the services of interest to the client requirements.

Discovery and scheduling services used when selecting adequate service providers and agreement responders in case of strict guarantees, can be easily extended to support this.

The provisioning of loose guarantees thus requires the complementing of the Grid middleware described in Section 2 via additional functional components. Consequently, the sections which follow, will focus on the problem of information gathering and publishing.

We have identified a number of different stages in the process of completing a typical Web services-based client-server interaction, as shown in Figure 2. REQ refers to the request message exchanged during a session, RES identifies the corresponding response message, C represents the client side and S represents the server side of the interaction.

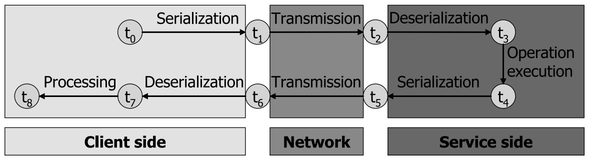

Fig. 2. Decomposition of a synchronous operation

1. $\overline{t_0t_1}$ *(Message Generation Delay, mgd)*: serialization of the request message REQ is taking place on the client side. For example, serialization time can be measured by using software hooks provided by different SOAP stacks (such as Axis handlers, gSOAP I/O handlers, etc).
2. $\overline{t_1t_2}$ *(Message Transport Delay, mtd)*: REQ is being dispatched by the communication infrastructure connecting C to S. This corresponds to the message one-way delay metric introduced in Section 4.1.
3. $\overline{t_2t_3}$ *(Message Processing Delay, mpd)*: REQ is de-serialized by S. This time interval can be estimated by following the same produres that can be adopted for message serialization.
4. $\overline{t_3t_4}$ *(Execution Time, et)*: the operation called is being executed. We assume this interval not to be subject to any specific measurement, as it can greatly vary depending on the input parameters conveyed by message REQ.
5. $\overline{t_4t_5}$ *(Response Message Generation Delay, rmgd)*: at this stage serialization of the response message RES takes place. This time interval can be estimated by using the same techniques adopted for $\overline{t_0t_1}$ and $\overline{t_2t_3}$.
6. $\overline{t_5t_6}$ *(Response Message Transport Delay, rmtd)*: it is the one-way delay experienced by the RES message after being issued by S in order to be transmitted by the network to C.
7. $\overline{t_6t_7}$ *(Response Message Processing Delay, rmpd)*: RES is de-serialized by C.

8. $\overline{t_7 t_8}$ *(Response Client-side Processing Time, rcpt)*: C processes the response message and acts accordingly. This component is not taken into account as with $\overline{t_3 t_4}$.

Different Grid infrastructure components contribute in the various stages identified above, for this reason we propose the estimation of end-to-end performance via the individual monitoring of each part (network, clients and servers), and the composition of the measurements gathered. The metrics of relevance and the composition methodology are described in what follows.

4 End-to-end Performance Estimation

In our framework, the computation of the expected QoS service profiles is required in order to support the selection the Grid services with good chances to meet the client QoS requirements. The estimation of a QoS profile is based on the constant monitoring of three fundamental aspects. Firstly, network performance needs to be probed by specific network sensors that are distributed over the network infrastructure of interest. Secondly, we propose the monitoring of performance at the client side, as this component affects the time needed to construct a request message in the first place and, if applicable, to process a reply messages from invoked services. Finally, performance needs to be analyzed also at the server end, being the server a component that contributes to determine both the time needed to process a given request messages from remote clients and the time needed to construct reply messages. In this framework, as explained later, the time needed to process a specific operation is not considered, being it highly specific to the type of operation and the input parameters supplied to it.

4.1 Performance Metrics

The main components contributing to end-to-end performance, are network paths, clients and servers. In this framework, the actual metrics that need to be subject to monitoring for loose guarantee support, necessarily depend on the application requirements considered. Starting from a set of requirements from applications for Computing and Instrument Grids, we have worked out a list of relevant metrics, and defined different measurement approaches that are applicable to different metric cathegories.

Network performance metrics

Several network performance metrics can be relevant to the estimation of some Grid service performance parameters. For example, all client-server sessions are inherently affected by network latencies (namely, *one-way delay* in case of asynchronous operations, and *round-trip delay* for synchronous operations).

In addition, interactive and collaborative applications can be also affected by latency variations, and operations generating the exchange of large data amounts, are also dependent on the amount of *available* and *achievable bandwidth* on the network path connecting the two end-points under consideration. Available bandwidth highly depends on network capacity and on the instantaneous amount of background traffic – which is typically unpredictable over short time spans, and we consider it out of scope in this paper.

Message one-way delay (RFC2679) from the node hosting client C to the node hosting server S is of paramount importance, as it affects the completion and response time of a given operation. It depends on the network distance, the message size, the capacity of the network links on the path from C to S, the status and type of the networking devices on the path (determining the queueing delays that are experienced by traffic), and the type of network protocol stacks supported at the sending and receiving end. Measurement of one-way delay requires accurate clock synchronization in the client C and in the server S.

Round-trip Delay also known as Round Trip Time (RTT) (RFC2681), complements one-way delay and corresponds to the time span from the instant when a given message is sent to the time when that message is received.

Throughput is particularly relevant in scenarios where applications require the exchange of large amounts of data. Throughput is typically a function of the transport protocol stack in use and the packet loss behaviour of the network under analysis. *Achievable throughput* is the average volume of application data that can be exchanged over time from C to S. The availability of achievable throughput [4] estimations, in combination with information about the file size to be exchanged, can help with the computation of the time needed to transfer the file from C to S.

Finally, *IP Packet Delay Variation* (IPDV), defined in (RFC3393), quantifies the variation of one-way delay for two subsequent IP packets. IPDV is critical for interactive and video/audio streaming applications, whose performance is optimized in case of low IPDV, which allows the receiving node to maintain a short backlog and consequently apply limited playback delays.

Service metrics

The performance of a stand-alone service can be quantified via a number of metrics, all contributing in a complementary fashion to the estimation of the performance level which can be expected by a client when interacting with a given service. Several of these metrics strictly depend on several factors, such as message size and its level of complexity, the type of operation invoked, etc. In the list of metrics which follows, S denotes a generic server, while C the client.

Availability: we define it to be the parameter which expresses the probability that S can be effectively invoked by C at a given time. High availability is an idication of service robustness. *Time To Repair* provides complementary information and estimates the average time needed to bring S back to operation.

Accessibility: it denotes the probability that S can receive a request and return a response message. Essentially, this shows how well a service scales with an increasing number of consumers C. Accessibility can be quantified by computing the average throughput, i.e. the average number of operations successfully completed by S over time. S can be available (serving some clients) but not accessible (rejecting or failing to serve additional requests).

Completion Time: given t – the time at which the composition (for example, its serialization if the Web services technology is used) of a message starts in C, and T – the instant when the execution of the operation O triggered by the message, finishes at the server end, we define Completion Time $CT(C, S, O, t)$ the time interval (Tt). In other terms, considering the operation decomposition introduced in Section 3, $CT(C, S, O, t) = \overline{t_0 t_4}$ Measurement of CT requires clock synchronization at both the client and the server end.

Completion Time Variation: given two subsequent invocations of operation O by C, at time t_1 and t_2, Completion Time Variation $\Delta CT(C, S, O)$ is the difference $|CT(C, S, O, t2) - CT(C, S, O, t1)|$. $\Delta CT(C, S, O)$ dynamically varies, given the shared nature of the software and hardware of Grid infrastructures.

Response Time: given t - the time at which the composition (for example, its serialization if the Web services technology is used) of a message starts in C; and T - the time when the processing of the corresponding response message is terminated by C, Response Time $RT(C, S, O, t)$ is the time interval (Tt), or equivalently $RT(C, S, O, t) = \overline{t_0 t_8}$.
RT is only applicable to synchronous operations, as RT takes into account the network latencies in both directions (C to S and S to C), while the Completion Time is only dependent on one-way delay and processing time at S).

4.2 Network Metrics Measurements Methodology

Network monitoring frameworks generally rely on a set of distributed sensors for the gathering of raw data, which is accomplished by end-to-end and/or edge-to-edge probing sessions. End-to-end measurements require the hosting of the sensors in customer networks (i.e. in leaf sites, not offering network transport services), whereas edge-to-edge measurements refer to one or more network transit domains, and require the installation of sensors in network Points of Presence at the edge of provider's domains, giving the possibility to monitor the various intermediate network segments that compose an end-to-end path. The metrics that can be gathered in the two scenarios are complementary. Edge-to-edge monitoring is out of scope in this document.

The problem of client-server performance monitoring is addressed by adopting the data gathered by nearby network sensors as reference values. The amount of active test traffic is controlled by enabling few network sensors in the customer networks. Every sensor needs to represent a set of nodes characterized by a homogeneous network connectivity QoS profile.

In this scenario, the gathering of network performance data for a given couple (Client, Server), develops through several interactions with the Grid information service:

1. the discovery of the domains C and S pertain to: $(Domain_1, Domain_2)$;
2. the discovery o sensors available in the two domains: $(Sensor_1, Sensor_2)$;
3. the collection of monitoring data gathered via monitoring sessions involving $Sensor_1$ and $Sensor_2$.

4.3 Service Metrics Measurements Methodology

The performance metrics described in Section 4.1 can be appropriately estimated via various methodologies at different levels of complexity.

Availability: When estimating availability, service performance has to take into account the quality of network connectivity experienced by the service. Monitoring of connectivity for every client-server pair is infeasible for scalability reasons, hence, we propose availability to be directly measured and logged by the service's clients. Note that availability can not be monitored at the service end, as the service has no a-priori knowledge of both its clients and of the level of reliability offered by the connecting network paths. For every client-server communication succesfully established on a socket-layer level, we propose the event to be logged and published according to the type of information service in use.

Accessibility: If a service is accessible, then operations can complete successfully. This event can be monitored differently for synchronous and asynchronous operations. In the former case, a response message is returned, and the successfull completion can be logged and published, if the client receives a well formed response message directly or indirectly indicating success, whereas completion of an aynchronous operation for a given client C, needs to be recorded and notified at the server end.

Completion Time and CT Variation: Measuring interval $\overline{t_0 t_4}$, needed to estimate CT, is intrinsically difficult due to the requirement for clock synchronization between clients and services. Furthermore, operations experience different CT values, depending on the input and the type of operation. Because of this, we request a server to expose a "dummy" (empty) method for every time-critical operation, whose request processing overhead is null. For empty operations CT can be accuratly approximated to the time interval $\overline{t_0 t_3}$, and the response processing time $\overline{t_3 t_4}$ can be neglected, as CT is reduced to the time for serializing, transferring and de-serializing the request message.

CT is then calculated by adding $\overline{t_0t_1}$ (as computed by the client), to the network one-way delay $\overline{t_1t_2}$ (where relevant information is gathered according to the network monitoring framework in use), and, finally, to $\overline{t_2t_3}$ (as computed by the server):

$$CT = \overline{t_0t_1} + \overline{t_1t_2} + \overline{t_2t_3}$$

CT can be computed by any Grid software component by getting information about the single time intervals either from a central Grid infomation repository or via notification.

In order to compute ΔCT, subsequent CT probes need to be identified and compared by subtraction. Unfortunately, correlation of related probes for intervals $\overline{t_0t_1}$, $\overline{t_1t_2}$ and $\overline{t_2t_3}$ is impossible by an external observer, due to the lack of a global clock source in distributed systems. For this reason, we also require a different CT estimation method to be adopted by every time-critical service S in order to allow external Grid components to compute CT variation. We propose every CT probe from a client to carry along the serialization start time T_1 as measured by the client. Similarly, on the server side, the end of the deserialization time T_2 is recorded and the time interval $T_2 - T_1$ is published. The approximation of $CT = T_2 - T_1$ is inherently inaccurate without clock synchronization at C and S, however subtraction can compensate clock offset: $\Delta CT = |CT_2 - CT_1|$, and consequently lead to a correct computation by every external estimator. Of course, calculations should be based on CT_1 and CT_2 values referring to similar input data sets.

Response Time: Response Time RT, corresponding to the time interval $\overline{t_0t_8}$, is approximated by means of empty operations as described for parameter CT:

$$RT = \overline{t_0t_3} + \overline{t_4t_7}$$

where $\overline{t_0t_3}$ is the Completion time and $\overline{t_0t_3}$ is the Completion Time referring to the response message.

5 Metric Composition

While availability and accessibility can be measured relatively easily, accuracy of temporal metrics estimation is in general more difficult due to the reasons elaborated earlier, relevant to lack of clock synchronization. Unfortunately, a number of additional constraints come into play when estimating some of the metrics, such as CT and RT. Firstly, the computation of one-way delay for every (C, S) pair via active measurements, is infeasible for scalability reasons, and this requires the approximation of actual one-way delay to the value measured by nearby sensors. More important, both CT and RT vary according to the input sets considered, and forecasting is an additional source of inaccuracy. We propose CT and RT to be estimated via the composition of various estimated time intervals, as explained in detail below. In particular, time intervals mentioned in Section 3 are individually measured and then combined together. The accuracy achieved by this metric composition, is currently under testing. We are considering a technique to model temporal overheads based on the Monte Carlo approach and application of Central Limit Theorem [7, 20]. There is also another approach using Chebyshev inequality [7, 20]. Chebyshev inequality states that for any distribution, $1 - 1/k^2$ values lie within k standard deviations of the mean of the probability distribution. However, Chebyshev inequality tends to give loose bounds on the confidence intervals. Thus, to tackle that and provide tighter confidence interval bounds, we make use of the Monte Carlo approach along with Central Limit Theorem. The Central Limit Theorem states that the mean of any set of random variables with arbitrary distributions, having a finite mean and variance and defined on the same probability space, tends to the normal distribution. Moreover, if the *Probability Density Function (PDF)* is known, then finite integration could be performed to obtain tighter bounds. In the following subsection we will model certain temporal overheads which relate to the invocation of a remote operation using this principle.

5.1 Modelling temporal overheads

Distribution parameters

The parameters of the different distributions which we will use as input, are the times and delays that were defined in Section 3 as the ones constructing the total time of a remote operation invocation. We assume that we know the distribution of these stages by archiving historical performance data. Denoting the mean and variance of the distributions by μ and σ^2 and their subscripts by the corresponding stage, we define the completion time and the response time. We use the Monte Carlo approach to perform M experiments, each consisting of N samples. These N samples are randomly picked from the delay distributions and the required metric is computed for each sample within an experiment. Consequently, we get M random variables, each defined on the same probability space, having a finite mean and variance. If the number of scenarios (sampling size or N) and the number of experiments (M) are sufficiently large, then the convolution of the M random variables is approximately normal, according to the Central Limit Theorem. Thus, finally we can model the metric using the mean and variance of the approximated normal distribution.

Completion Time

Completion time is a summation of the distributions of message generation delay, message transfer delay, message processing delay and operation execution as these have been defined in Section 3 (we do not take the later into consideration as mentioned earlier). Denoting the mean and variance of the distribution by $\mu_{mgd+mtd+mpd}$ and $\sigma^2_{mgd+mtd+mpd}$, we can model CT by the following equation, where k is a constant which depends on the required QoS guarantee:

$$CT = \mu_{mgd+mtd+mpd} + k\sigma_{mgd+mtd+mpd} \tag{1}$$

Response Time

The response time distribution is a summation of various delay distributions. Denoting the mean and variance of the response time distribution by μ_{all} and σ_{all}, we model the response time by the following equation, where $all = mgd+mtd+mpd+rmgd+rmtd+rmpd$ as these have been defined in Section 3:

$$RT = \mu_{all} + k\sigma_{all} \tag{2}$$

Since the convolution of the above sampling distributions is normal, according to the Central Limit Theorem as we indicated, the value of k is readily available, e.g., if 95% QoS guarantee is required, then k turns out to be nearly 1.96 and so forth.

6 Information publishing and archiving

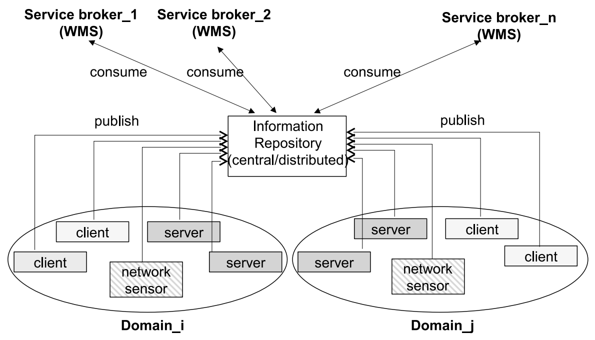

Fig. 3. QoS performance information publishing and consuming

As explained in Section 4.3, different types of metrics are probed and published by various sensors in the infrastructure. The clients measure availability, accessibility for synchronous operations, message generation delay and response message processing delay. The services do the same for message processing delay, response message generation delay, accessibility for asynchronous operations, and completion time variation. The network probes measure message transport delay (simulated by message one-way delay), response message transport delay (also simulated by message one-way delay), IPDV and achievable bandwidth. Monitoring data is published by these producers, namely the clients with QoS requirements, servers, and network sensors, as shown in Figure 3. Information is gathered in a repository, which can be central or distributed, depending on the approach adopted to information publishing/subscribing. Information consumers are the Grid middleware components that need to discover services matching the QoS requirements specified by client applications.

There could be thousands of widely distributed sensors and monitoring data will be frequently updated by them. This data is persistent so that it can be queried at any time. There are two solutions for the information repository: centralized and decentralized. A traditional relational database is a good example of the former. It still remains open whether a centralized repository can meet the scaling requirements which the Grid poses, or if a distributed repository would perform better to a degree which would compensate for the complexity of implementing such a solution. The implementation of centralized approaches, based on the principle that information is made available by few dedicated Information Service instances is one option. Centralized repositories are suitable to gather information about attributes that are not subject to frequent value updates, and which are of interest to a large number of clients. On the other hand, distributed approaches where relevant information is just either directly or indirectly forwarded to the information consumers interested in it, typically provide better scalability and are more suitable to distribute data frequently changing and relevant to a relatively small number of data consumers

Publishing of information can be accomplished either with a push model, or with a pull model, or with a combination of the two. The model to use is typically associated with the archiving model that is being chosen (centralized, virtual centralized with distributed repositories, or fully distributed). The most common approaches are that of a central repository to which sensors push monitoring data, and that of a publish/subscribe mechanism where each client is subscribing for notifications from the services which are of interest, with each such service pushing data to the client whenever there is such a need.

The characteristics of monitoring a large number of distributed components under the Grid environment has been discussed in [8] with the conclusion that the complicated requirements are unable to be met by using a traditional information management system, such as a relational database, because these systems are not designed for the purpose. A consumer/producer Grid Monitoring Architecture (GMA) was therefore proposed to solve the issues of performance, scalability and interoperability. The GMA supports three interactions for transferring data between producers and consumers:

publish/subscribe, query/response and notification. The GMA has already certain implementations, with the Relational Grid Monitoring Architecture (R-GMA) [3] being the most complete one.

7 Prototype

A prototype is being implemented to demonstrate the feasibility of the described framework. To this moment, it supports only strict guarantees, with loose guarantees support being under development. It contains two components: the WMS and an Agreement Service with storage and instrument reservation templates exposed. This AS is the client of a *Storage Element* (SE) exposing a *Storage Resource Management* (SRM) 2.1 [12] interface, as well as an *Instrument Element* (IE) using a custom, novel interface for instrument reservations. The implementation of the web-service part of the AS was based on gSOAP v.2.7 [21]. The resulting AS (both the client and the server side) are entirely based on the WS-Agreement XML schema definitions. The operations exposed by the AS are similar to the Agreement Factory ones defined in [9], however the prototype WSDL exposed by the AS does not fully comply with the standard, as it is not currently based on the WS-Resource Framework [14]. The AS prototype implementation will be revised to fully comply with the new WS-Agreement specification as soon as it is released.

The RASP offering space reservation support to the prototype AS, is based on the StoRM [17] Service Provider exposing a SRM 2.1 interface which supports immediate space reservation, file copy and file creation. StoRM enforces space reservation by allocating space directly onto the file system, using the GPFS [19] pre-allocation capabilities. Similarly, the RASP offering instrument reservation support is an implementation of the custom interface definition mentioned above, with the reservations taking place within a database included in the Instrument Element.

8 Related work

Relevant work in the field of QoS for Web services and Grid Computing has taken place, by exploring different approaches.

In [16], Al-Ali et. al propose a framework that allows QoS criteria to be specified as part of a service interface, thus enabling service selection based on QoS profiles of candidate services. On an architectural level, the framework defines a Resource Manager which is responsible for resource allocation, a Network Resource Manager which effectively manages networking resources through monitoring and brokering, an Application QoS (AQoS) Manager, which performs service discovery based on QoS constraints and then monitors service execution performance, and the Service, which can belong to one of three classes (Guaranteed, Controlled-load, Best-effort). The framework defines three QoS layers with metrics similar to ours: the application layer (availability, accessibility, security etc), the middleware layer (resource requirements) and the network layer (networking QoS). All monitoring is performed by the AQoSM on every service execution invocation, where this includes the creation of sandboxes, transfer of data and execution startup. Thus, client-related delays are not included in the aggregate values that are archived and used to extract the QoS profile of a service. It is interesting to note that, again, execution time of the job is not measured (similarly to our framework not measuring operation execution time). Furthermore, performance may be classified per domain, or per service, without any specific restrictions on the model to be used. Finally, the framework presents a series of extentions to WSDL [22] and UDDI [13] so that they support the necessary constructs for QoS-based service discovery.

In [18], Ran proposes a service discovery model which also takes QoS (non-functional) requirements as input, similarly to what has been mentioned up to this point. More specificially, the proposed framework includes roles for service providers, service consumers and QoS certifiers, as well as UDDI extentions to hold relevant information. The basic idea is that when a service is registered into the UDDI registry and claims a specific QoS level for itself, then the QoS certifier actually verifies these claims for correctness. The paper continues to define the new structures required and finishes with a list of QoS attributes, similar to the ones defined in our work. The four major QoS metrics categories which are identified in this framework, are runtime related, transaction support related, configuration management and cost related, and security related. The list is quite extensice, but there is no implementation work to suggest how to measure these parameters.

In both cases, extensions to WSDL and UDDI are proposed. This is different from our work where such extensions are not suggested for reasons of compliance to current standards.

In [23], Liu et. al propose a dynamic QoS registry and computation model for Web services selection. The measurements are taken, similarly to our work, as a mixture of information sampled by either the service consumer or the service producer, depending on the metric which is being measured. In certain cases, it is the end-user herself that may provide the QoS feedback. The authors do not include availability or accessibility metrics, but rather cost, execution time, reputation (reliability), transaction support, compensation rate and penalty rate. It is however supported that the model is easily extensible to other QoS parameters, either generic or in the domain space. A central registry is holding QoS parameters measurements for all service providers, and can be queried for service selection based on such non-functional criteria. The work of Liu et. al focuses more on the qualitative QoS metrics while we also put emphasis on the temporal ones. Furthermore, we extend the QoS registry from centralized to distributed, with accompanying publishing mechanisms.

9 Conclusion and Future Work

We presented a QoS framework for Computing Grids, with an emphasis on Web services-based middleware implementations, which aims at supporting Grid sessions with various types of specific QoS requirements. The proposed framework relies on two complementary approaches. The former is designed to provide QoS guarantees on a deterministic basis, and relies on a novel Grid service, the *Agreement Service*, for SLA signalling. Conversely, the latter is a lightweight approach delivering probabilistic QoS guarantees for those use cases which can not rely on deterministic QoS enforcing techniques. Loose QoS guarantees are based on archived performance information and statistical models. Therefore, we propose an architecture for publishing of monitoring data, which is integrated with a QoS-aware Grid service for workload management, namely the Workload Management System. Several QoS performance metrics for network and Grid service profile characterization are defined starting from a number of application use cases for computing and instrument Grids, and a metric composition approach is presented for the estimation of some of the temporal QoS metrics of interest in the paper. Finally, preliminary implementation results are presented.

Future directions include extensive experimentation with the framework described. The purpose of this is manifold. The viability of the resource-independent Grid services for SLA signalling and monitoring needs to be proved in large scale and heterogeneous Grid infrastructures and, secondly, the distributions which stem from performance measurements and monitoring data, need to be understood in depth. We plan to analyze the accuracy of the composition approach proposed. Finally, effective publishing mechanisms will be sought for, and will be tested in an integrated framework supporting single and composite services.

Acknowledgements

This work has been partially funded by the GRIDCC (Grid-enabled Remote Instrumentation with Distributed Control and Computation) EU project under contract number 511382 and is presented on behalf of the GRIDCC consortium. It was also supported by the EU project EGEE sponsored by the European Union under contract number INFSO 508833, by the INFN project INFN-GRID, and by the Greek Secretariat for Research and Technology.

The authors would like to thank Peter Hobson and Paul Kyberd (Brunel University, UK), Panagiotis Louridas (Greek Research and Technology Network, Greece), Gaetano Maron and Francesco Lelli (National Institute of Nuclear Physics, Italy), Luke Dickens, Marko Krznaric and Janusz Martyniak (Imperial College London, UK) for their work and feedback on the paper.

References

1. EGEE Middleware Architecture Aug 2004. http://edms.cern.ch/document/594698.
2. The EGEE project home page. http://www.eu-egee.org.
3. A. W. Cooke et al. The Relational Grid Monitoring Architecture: Mediating Information about the Grid. *J.Grid Computing*, 2:323–339, 2004.
4. B. Lowekamp et al. A Hierarchy of Network Performance Characteristics for Grid Applications and Services, 2004.
5. EGEE. Institution of SLAs and appropriate policies Deliverable DSA2.2. http://edms.cern.ch/document/565447, 2005.
6. EGEE. ISpecification of interfaces for Bandwidth Reservation Deliverable DJRA4.1. https://edms.cern.ch/document/501154, 2005.
7. William Feller. *Introduction to Probablity Theory and its Applications, Vol II.* John Wley and Sons, 1966.
8. GGF Performance Working Group. A Grid Monitoring Architecture. 2002.
9. Global Grid Forum / A. Andrieux et al. Web Services Agreement Specification (WS-Agreement). https://forge.gridforum.org/projects/graap-wg.
10. GRIDCC. GRIDCC Architecture - Deliverable 1.2. http://www.gridcc.org/getfile.php?id=1560, 2005.
11. INFN. Virtual Organization Membership Service.
12. J. Gu et al. The Storage Resource Manager Interface Specification. http://sdm.lbl.gov/srm-wg/.
13. OASIS. Universal Description, Discovery and Integration. http://www.uddi.org/.
14. OASIS. Web Services Resource Framework (WSRF). http://www.oasis-open.org/.
15. P. Andreetto et al. Practical Approaches to Grid Workload and Resource Management in the EGEE Project. In *in Proc. of the Conference on Computing in High Energy and Nuclear Physics (CHE0P 2004)*, pages 899–902, Interlaken, CH, 2004.
16. R. Al-Ali et al. G-QoSm: Grid Service Discovery Using QoS Properties. *Computing and Informatics Journal*, 21(4), 2002.
17. R. Zappi et al. StoRM: grid middleware for disk resource management. In *in Proc. of the Conference on Computing in High Energy and Nuclear Physics (CHE0P 2004)*, volume 2, pages 1238–1242, Interlaken, CH, 2004.
18. Shuping Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.
19. Frank Schmuck and Roger Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *First USENIX Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, 2002.
20. Henk Tijms. *Understanding Probability: Chance Rules in Everyday Life.* Cambridge University Press, 2004.
21. Robert van Engelen. gSOAP toolkit. http://gsoap2.sourceforge.net/.
22. W3C. Web Services Description Language. http://www.w3.org/TR/wsdl.
23. Y. Liu et al. QoS computation and policing in dynamic web service selection. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 66–73, New York, NY, USA, 2004. ACM Press.