A Framework for Providing Hard Delay Guarantees in Grid Computing

Panagiotis Kokkinos, Emmanouel A. Varvarigos and Nikolaos D. Doulamis

Research Academic Computer Technology Institute, Patras, Greece.

Abstract

Future Grid Networks should be able to provide Quality of Service (QoS) guarantees to their users. In this work we propose a framework for Grid Networks that provides deterministic delay guarantees to its Guaranteed Service (GS) users and best effort service to its Best Effort (BE) users. The proposed framework is theoretically and experimentally analyzed. We also define four types of computational resources based on the type of users (GS, BE) these resources serve and the priority they give them. We implement the proposed QoS framework for Grids and verify that it not only satisfies the delay guarantees given to GS users, but also improves performance in terms of deadlines missed and resource use. In our simulations, data from a real Grid Network are used, validating in this way the appropriateness and usefulness of the proposed framework.

1 Introduction

The existence of high speed optical networks makes the vision of computational Grids [1] a reality. Grids consist of geographically distributed and heterogeneous computational and storage resources that may belong to different administrative domains, but can be shared among users by establishing a global resource management architecture. A number of applications in science, engineering and commerce, and especially those with small communication interdependencies but large computational or storage needs, can benefit from Grid Computing. An important issue in the performance of Grids is the management of the resources and the scheduling of the tasks to the available resources. The Grid environment is quite dynamic, with resource availability and load varying rapidly with time, and tasks having very different characteristics and requirements. Scheduling is key to the success of Grid Networks, since it determines the efficiency in the use of the resources and the Quality of Service (QoS) provided to the users.

In Grid Networks, QoS usually refers to the communication and computation time it takes for a task to be completed, or to the amount of resources allocated to a user. In order for a network to provide QoS guarantees to a user, a three step procedure is generally followed. First, the user informs the network of the exact QoS parameters requested (delay, number of resources, etc). Then the network, through a procedure called admission control, checks if it can satisfy the user's request for guaranteed service, without violating the guarantees given previously to other users. If the network can satisfy the QoS requirements posed by the user, then various mechanisms (resource reservation, scheduling, flow control) are employed to ensure that the agreed upon QoS level will be provided.

Today's Grids provide only a best effort service to the users and their tasks. However, the best effort approach is inappropriate if the Grid Network is to be used for real world commercial applications and complex scientific simulations and computations. Under these thoughts we believe that future Grids will serve two types of users. Some users, called Best Effort (BE) users, will be relatively insensitive to the performance they receive from the Grid. Besides BE users, we expect Grids to serve users that do require a guaranteed QoS. These users will be referred to as Guaranteed Service (GS) users. We must mention that by the term "user" we do not necessarily mean an individual user, but also (and probably more appropriately) a Virtual Organization (VO), or a single application, using the Grid Network's infrastructure.

In this work we propose a QoS framework for computational Grids, concentrating more to GS users than to BE users. Specifically for the GS users the framework guarantees a maximum delay on the execution of the tasks submitted by them. In order to achieve this, the GS users are leaky bucket constrained, so as to follow a (ρ, σ) constrained task generation pattern, which is agreed separately with each resource. We also consider four types of resources that serve either GS, or BE, or both types of users, with varying priorities. Finally, we implement our proposed QoS framework in the GridSim [17] environment and execute a number of simulations. Our results indicate that the proposed framework succeeds in providing QoS guarantees to the GS users, even when the BE users produce many tasks. In our simulations, data from a real Grid Network are used, validating in this way the appropriateness and usefulness of the proposed

framework.

The remainder of the paper is organized as follows. In section 2 we report previous work. In section 3 we describe the proposed QoS framework for Grids. In section 4 we propose extensions to this framework. In section 5 we present the simulation environment, the parameters used, and the results of our simulations. Finally, conclusions are presented in section 6.

2 Previous Work

A number of scheduling algorithms have been proposed so far, both for single- and for multi-processor systems, some of which have also been adapted for use in Grids. These algorithms often wait for a period of time so that several tasks accumulate at the scheduler, before making scheduling decisions, and they consist of two phases: the task-ordering phase and the task-to-resource assignment phase. Lately, a number of scheduling schemes that are specific to Grids have also been proposed. [5],[8] present centralized scheduling schemes, while hierarchical schemes are proposed in [9]. Distributed schemes are explored in [6] and [7]. Usually in distributed scheduling algorithms the two-phase procedure described above is not followed, but tasks are scheduled to resources immediately upon their creation.

Many of the scheduling algorithms proposed so far try to minimize the total average task delay [7] and maximize resource usage. Other works incorporate economic models in Grid scheduling. In [18] and in [19] scheduling algorithms that support deadline and budget constraints are proposed and implemented. Our work differs from previous works in that it gives to the users hard delay guarantees, provided that certain constraints are enforced in the task submission process.

QoS in Data Networks has been extensively studied. The Internet Engineering Task Force (IETF) has proposed the Integrated Services (IntServ) [11] and the Differentiated Services (DiffServ) architectures [12] to support QoS in networks, and differentiate traffic in terms of bandwidth, latency and other data transfer parameters. Relatively recently QoS in Grids has also started gaining attention. Two important efforts addressing this issue were the General-purpose Architecture for Reservation and Allocation (GARA) [13] and the Grid QoS Management (G-QoSM) architecture [16]. These works propose QoS schemes for Grids that take into account the network, computational and storage resources. Various other works have concentrated on specific aspects of QoS in Grids, such as network QoS for Grid applications [14], and admission control [3]. GARA is the oldest framework for supporting QoS in Grids. This framework provides guarantees to an application requesting specific end-to-end QoS characteristics. G-QoSM is a newer QoS framework for Grids, which is more actively developed, following the recent trends in Grid Networks. So G-QoSM is Open Grid Service Architecture (OGSA) enabled, and incorporates various useful features.

In the present work we propose a QoS framework for Grid computing, which provides hard delay guarantees to GS users. We show both theoretically and experimentally that hard QoS, in terms of delay bounds guarantees given to each user, can in fact be achieved without using resource reservations. The users and the resources, simply, agree upon the task load the former will generate and the latter will serve. On the other hand the GARA and G-QoSM frameworks reserve computational resources quantitatively, either by reserving a number of CPUs in a resource or by reserving a percentage of a CPU's capacity (Dynamic Soft Real-time scheduler - DSRT [15]). Furthermore, in our QoS framework we also propose and evaluate the categorization of computational resources so as to serve either GS, or BE, or both types of users, with varying priorities.

3 Description of the Framework

3.1 General

We consider a Grid Network consisting of a number of users and resources. There are two kind of users: Guaranteed Service (GS) and Best Effort (BE) users. The tasks originating from these users are of GS or BE type, respectively. Also there are various types of resources based on the types of tasks they serve (GS or BE or both) and on the priority they give to each type. Our framework gives service guarantees to GS users. In order to achieve this the GS users are leaky bucket constrained, so as to follow a (ρ, σ) constrained task generation pattern, which is agreed separately with each resource. On the resources, the arriving tasks are first queued in a Weighted Fair Queuing (WFQ) scheduler [10]. This way guaranteed task service rates (e.g., measured in Million Instructions Per Second - MIPS) can be given to each GS user, in the same way that WFQ provides guaranteed bandwidth services in Data Networks.

Our proposed framework describes the distributed mechanisms used to provide service guarantees to GS users. We assume that a task executing at a resource is non-divisible and non-interruptible (non-preemptable). We initially describe our framework assuming that every machine has one CPU, and later extend it to the multi-CPU machine case.

3.2 Guaranteed Service (GS) users

Based on our framework a GS user must first register to a resource, before it can actually use it. During the registration phase the GS user and the resource agree upon the characteristics of the traffic the GS user will send to that

resource, that is, the leaky bucket's constraints characteristics. A GS user can register himself to a number of resources. Next, when a GS user creates a task, he chooses for its execution one of his registered resources, based on various criteria, such as performance (e.g., delay), fairness (e.g., uniform usage of the registered resources) and others.

Our framework is implemented in a distributed way, and as a result scheduling logic exists in the GS user and in the resource (local scheduler). During the registration phase a GS user i and a resource r agree upon the (ρ_{ir}, σ_{ir}) constraints (Figure 1) of the user. The parameter ρ_{ir} is the long term task generation rate, measured in computation units per second (e.g. MIPS), that the GS user requests. The parameter σ_{ir} is the maximum size of tasks (burstiness) that the GS user will ever send, in a very short time interval, to the specific resource. The parameter σ_{ir} is measured in computation units (e.g., Million Instructions - MI). If the resource r agrees that it will satisfy these constraints, then the GS user is registered to the resource. From then on, the GS user becomes responsible for the observance of these constraints and the resource for the satisfaction of the QoS guarantees given to the user, as explained below. Alternatively, other approaches can be used (such as the centralized and the hybrid approaches described in Section 4.1), where a meta-scheduler is used as an intermediary for the monitoring of the observation of the (ρ, σ) constraints.

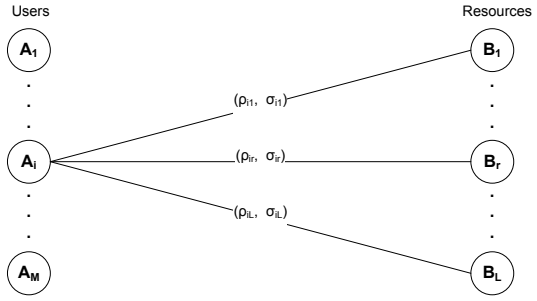

Figure 1. The (ρ, σ) constrained GS users in the Grid Network.

In order for a resource r to accept the registration of GS user i a number of requirements must be met. First, the resource checks whether it can serve the GS user with the requested task generation rate ρ_{ir} without violating the task generation rates agreed with the already registered GS users. The local scheduler of every resource applies Weighted Fair Queuing (WFQ) to the queued tasks, so the following condition must hold for new and old GS users:

$$\rho_{ir} \leq g_{ir}(t) = \frac{C_r \cdot w_{ir}}{\sum_{k=1}^{N_r(t)+1} w_{kr}}, \quad (1)$$

where C_r is the computing capacity of resource r measured in computation units per second (e.g. MIPS), $N_r(t)$ is the number of GS users already registered to the resource r at time t, and w_{ir} is the weight of the GS user i for using the resource r. This weight can depend on various parameters, such as the price the GS user has paid or its other contributions to the Grid. Condition (1) ensures that the resource can satisfy the task generation rates of the new and the old GS users. Furthermore, one more condition for the successful registration is that the maximum task length (workload) the GS user will ever send to the specific resource J_{ir}^{max} will not be larger than the resource's maximum acceptable task length J_r^{max}:

$$J_{ir}^{max} \leq J_r^{max}. \quad (2)$$

If both (1) and (2) hold then the GS user can register to the resource; otherwise, the registration fails and the GS user must search for another resource. The GS user can repeat the same procedure so as to register to multiple resources. Also a user can cancel its registration whenever he wants to and for whatever reason. Finally, every user can repeat periodically the registration phase, in order to negotiate the registration to new resources or to resources from which other users have canceled their registration.

A GS user is equipped with a queue to temporarily withhold tasks that, if submitted to a resource r, would invalidate the agreed (ρ_{ir}, σ_{ir}) constraints. Specifically, we denote by $J_{ir}(t), i = 1, 2, \cdots, N$ the total tasks length (measured, e.g., in MI) submitted by GS user i to resource r in the interval $[0, t]$. We will say that a GS user i is (ρ_{ir}, σ_{ir}) controlled with respect to resource r, if the following condition is valid:

$$J_{ir}(t) < \sigma_{ir} + \rho_{ir} \cdot t, \forall t > 0. \quad (3)$$

So the total tasks length (workload) a user i can send, over a time period, for execution to a resource r is restricted by the (ρ_{ir}, σ_{ir}) constraints. If a GS user i would invalidate (3) by submitting a task j, then the GS user must locally withhold this task for a time period T_{ir}^j until (3) becomes valid again (Figure 2). So our framework includes in every GS user an admission control (leaky bucket) mechanism to ensure that the user's (ρ, σ) constraints are always satisfied.

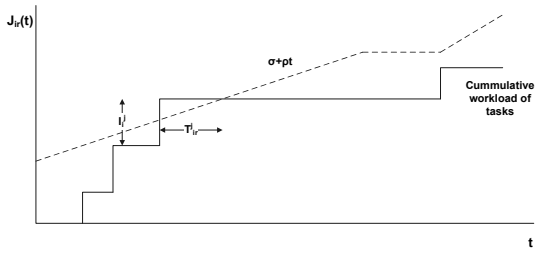

Figure 2. The GS user is responsible for the observance of his (ρ_{ir}, σ_{ir}) constraints.

When a task j is created, the GS user searches for the best suitable resource to which it has already registered. We assume that task j is characterized by its deadline D_i^j and its length I_i^j (measured, e.g., in MI). In order for task j to be sent to the resource r again two conditions must hold. First, the task's length must not exceed the one agreed,

$$I_i^j \leq J_{ir}^{max}, \tag{4}$$

and, second, the task must not miss its deadline. One of the benefits of (ρ, σ) constrained GS users and of the registration phase is that the maximum delay until a task is completed on a resource can be bounded. If conditions (1) and (3) hold and WFQ is used, then it can be proved, by arguing as in [2], that the delay a task will incur from the time it reaches resource r until it finishes its execution at a selected resource is at most:

$$\frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r},$$

where g_{ir} is the minimum value of $g_{ir}(t)$ that does not invalidates (1) for any registered user. To this delay we must add the total communication delay d_{ir}^j, required for transferring task j data to the selected resource r, and the total time T_{ir}^j the GS user i withholds the task j in its local queue (Figure 2). So the delay bound B_{ir}^j the resource r guarantees to the user i for task j is given by:

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r}. \tag{5}$$

Finally, based on (1) and assuming that $w_{ir} = 1$, for all i, r, we have:

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{(\sigma_{ir} + J_{ir}^{max}) \cdot (Nr(t)+1) + \cdot J_r^{max}}{C_r}.$$

So in order for a task j of a GS user i to be scheduled to a resource r, its deadline D_i^j must be larger (or equal) than the resource's delay bound B_{ir}^j:

$$B_{ir}^j \leq D_i^j. \tag{6}$$

Furthermore, we can pipeline the T_{ir}^j and d_{ir}^j delays:

$$B_{ir}^j \leq max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir} + J_{ir}^{max}) \cdot (Nr(t)+1) + \cdot J_r^{max}}{C_r}.$$

By pipelining, we mean that if d_{ir}^j is larger than T_{ir}^j, then the user i sends the task j to the selected resource immediately, without waiting for the T_{ir}^j time period to expire, while if T_{ir}^j is larger than d_{ir}^j then the user sends the task to the resource after $T_{ir}^j - d_{ir}^j$ time units. In both cases time savings are achieved.

If more than one resources fulfill the conditions of Equations (4) and (6) then the GS user can choose one based on any other optimization criterion. If no resource fulfills these conditions then the GS user drops the task or schedules it like a BE task. Also from (5) we conclude that it may be beneficial to distinguish resources in groups of resources offering different maximum delay guarantees. More specifically, the a priori knowledge or determination of a resource's computational capacity C, maximum task length J, maximum allowed burstiness σ and maximum number of GS users N allowed, provides a guaranteed maximum delay for the tasks sent to that resource:

$$B(C, J, N, \sigma) \leq max(T, d) + \frac{(\sigma + J) \cdot (N + 1) + J}{C}, \tag{7}$$

where T and d do not depend only on the resource but also on the user side. If σ is expressed as a multiple of J, $\sigma = m \cdot J$ (that is, the user is allowed to send up to m maximum-sized tasks in a very short interval if he has not sent any other tasks recently), then (7) can also be written as:

$$B(C, J, N, m) \leq max(T, d) + \frac{((m+1) \cdot N + 1) \cdot J}{C}.$$

3.3 Resources

In the context of our framework we also propose to distinguish four types of resources: GS, BE, GS_BE_EQUAL and GS_BE_PRIORITY. GS resources handle only tasks originating from GS users. When a GS task arrives at a GS resource, it is queued at the local WFQ scheduler. When a machine is freed, the local WFQ scheduler selects the next GS task for execution. BE resources handle tasks originating only from BE users. The arriving tasks are placed in a queue and served following a First Come First Served (FCFS) policy to the first available machine. GS_BE_EQUAL resources handle tasks originating from both GS and BE users. GS tasks are served using a local WFQ scheduler as in the GS resources. Each arriving BE task is considered as belonging to a new user, who wants to register to the resource. So a BE task is queued in the local WFQ scheduler only if the condition of Equation (1) holds for all the registered users (the weight of Equation (1) for any BE user, equals to the smallest weight assigned to any GS user). In this case the number of registered users is increased by one and when the BE task finishes execution it is correspondingly decreased by one. If (1) does not hold for at least one registered user then the task is rejected and a failure notice is returned to the originating user. GS_BE_PRIORITY resources handle tasks originating from both GS and BE users, but their tasks are not handled in the same queue. GS tasks are handled by the local WFQ scheduler, while BE tasks are placed in a FCFS queue. When a machine is freed the tasks in the local WFQ scheduler are handled first. If there are no such tasks then the BE tasks from the FCFS queue are served.

Table 1. Delay bounds given to GS users with respect to the resource type.

Resource	Delay Bound for GS users
GS	$max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+J_{ir}^{max})\cdot(Nr(t)+1)+J_r^{max}}{C_r}$
BE	-
GS_BE_EQUAL	$max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+J_{ir}^{max})\cdot(Nr(t)+1)+J_r^{max}}{C_r}$
GS_BE_PRIORITY preemptive	$max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+J_{ir}^{max})\cdot(Nr(t)+1)+J_r^{max}}{C_r}$
GS_BE_PRIORITY non-preemptive	$max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+J_{ir}^{max})\cdot(Nr(t)+1)+2\cdot J_r^{max}}{C_r}$

A GS_BE_PRIORITY resource is characterized as preemptive if upon the arrival of a GS task, a BE task currently under execution is paused and replaced by the new GS task; otherwise, the GS_BE_PRIORITY resource is characterized as non-preemptive. Finally, a BE task is scheduled to a GS_BE_EQUAL or GS_BE_PRIORITY resource only when its size is smaller than the resource's maximum acceptable task size.

When a GS_BE_PRIORITY non-preemptive resource is used, the delay bound for GS tasks of Equation (5), becomes:

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r} + R_r,$$

where R_r is the residual time for the BE task found at the resource (if any) to complete execution. Thus,

$$R_r \leq \frac{J_r^{max}}{C_r}.$$

In the other resource types R_r equals to 0.

Using the proposed framework delay bounds can be provided to GS users. Table 1 summarizes the delay bounds that the various resource types provide the GS and BE users.

4 Extensions of the Proposed Framework

4.1 Distributed, centralized and hybrid implementations

In Section 3 we assumed a distributed implementation of our proposed QoS framework, where registration is done by each user (or VO) by communicating directly with the resource and negotiating its (ρ, σ) constraints. However, other approaches can also be used. In the centralized approach the registration of the GS users to the resources is handled by a meta-scheduler. The meta-scheduler accepts, from the GS users, registration requests containing their (ρ, σ) constraints. Then the meta-scheduler searches for resources r that can satisfy these constraints. The GS users submit tasks to the meta-scheduler, which schedules them to one of their registered resources. Finally the meta-scheduler is responsible for enforcing the (ρ, σ) constraints to the GS users. A hybrid approach is also possible, where again a meta-scheduler is responsible for the registration of the GS users to the resources, but following the registration, the users submit their tasks directly to one of their registered resources, and are themselves responsible for the observation of their (ρ, σ) constraints.

4.2 Multi-machine resources

The proposed framework can easily be extended to the case of resources that consist of many machines-CPUs, provided that some of the definitions and conditions given earlier are appropriately modified. The total computational capacity C'_r of a multi-machine resource's r is expressed as:

$$C'_r = \sum_{j=1}^{M_r} C_{rj},$$

where C_{rj} is the computational capacity of machine j, and M_r is the total number of machines (CPUs) in the resource r. We also assume that the local scheduler assigns tasks to the first available machine-CPU, in a round-robin manner.

In (1), $g_{ir}(t)$ is the average service rate the resource r guarantees to provide to user i. Since C'_r is the total service rate the user has access to from the resource, C_r in Equation (1) has to be replaced by C'_r, yielding

$$\rho_{ir} \leq g_{ir}(t) = \frac{w_{ir}\cdot\sum_{j=1}^{M} C_{rj}}{\sum_{k=1}^{Nr(t)+1} w_{kr}}.$$

Since tasks are non-divisible, the resource cannot use its total computational capacity to process a task. The worst case is obtained when a task is assigned to the machine (CPU) with the lowest computational capacity $C_r^{min} = \min_j C_{rj}$. Therefore, C_r in Equation (5) and in all the other delay bounds given in Section III has to be replaced by C_r^{min}. For example, Equation (5) becomes:

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r^{min}}. \quad (8)$$

5 Simulation Results

We implemented the centralized version of the proposed framework by extending the GridSim [17] simulator.

5.1 Parameters and scenarios

In our simulations we used realistic parameters based on [4]. In [4] a thorough analysis of the task inter-arrival times, the execution times, and the data sizes exchanged at the kallisto.hellasgrid.gr cluster, which is part of the EGEE Grid infrastructure, were presented and analytic models were proposed. Based on these results and numeric data we decided to simulate three GS users, corresponding to three of the five VOs presented in [4] (namely, the Atlas, Magic and Dteam VOs). So these VOs represent three different classes of users, based on their task inter-arrival process and execution times. Also, using the VO's average tasks execution times we calculated their average task lengths, measured in MI. From [4] we got our initial data, and in our simulations we used the corresponding normalized and rounded values (Table 2).

Table 2. GS users task lengths and generation rates.

User	Characteristic	Distribution
Atlas/U1	Task Length	Fixed: 10000 MI
Magic/U2	Task Length	Fixed: 1700 MI
Dteam/U3	Task Length	Fixed: 10 MI
Atlas/U1	Task G/tion Rate	Fixed: $\frac{1}{10}$ tasks/secs
Magic/U2	Task G/tion Rate	Fixed: $\frac{1}{60}$ tasks/secs
Dteam/U3	Task G/tion Rate	Fixed: $\frac{1}{110}$ tasks/secs

Based on these data, the (ρ, σ) constraints of each user were statically calculated. Specifically the σ parameter of each user is selected $m = 5$ times bigger than the corresponding GS user's average task length. The ρ parameter of each GS user is calculated by dividing its average task length by its average task inter-arrival time (Table 3).

Table 3. GS users (ρ, σ) constraints.

User	Characteristic	Distribution
Atlas/U1	ρ	Fixed: 1000 MIPS
Magic/U2	ρ	Fixed: 30 MIPS
Dteam/U3	ρ	Fixed: 1 MIPS
Atlas/U1	σ	Fixed: 50000 MI
Magic/U2	σ	Fixed: 8500 MI
Dteam/U3	σ	Fixed: 50 MI

In our simulations we also used two BE users, named U4 and U5. U4's task inter-arrival time changes in every simulation, while U5's remain the same (Table 4). The lengths of the tasks submitted by these users were fixed and equal to the U1 GS user's task length (namely, 10000 MI).

In our simulations we choose to use 3 computational resources, named R1, R2, R3, each consisting of one machine with one CPU and computational capacity 1015 MIPS, 680 MIPS and 340 MIPS, respectively. Furthermore, in our simulations we used the resources scenarios presented in Table 5. When the BE resources scenario is used then the U1, U2 ,U3 users are BE users with the exact same characteristics as the corresponding GS users.

Table 4. BE users task lengths and generation rates.

User	Characteristic	Distribution
U4	Task Length	Fixed: 10000 MI
U5	Task Length	Fixed: 10000 MI
U4	Task G/tion Rate	Fixed: $0.05, 0.1, 1$ tasks/sec
U5	Task G/tion Rate	Fixed: 0.01 tasks/sec

Table 5. Scenarios of resources configurations.

Scenario Name	R1	R2	R3
GB	GS	GS	BE
GBE	GS_BE_EQ.	GS_BE_EQ.	BE
GBP	GS_BE_PR. n.pr.	GS_BE_PR. n.pr.	BE
BE	BE	BE	BE

The meta-scheduler uses a two-phase scheduling procedure for BE users. More specifically, the Earliest Deadline First (EDF) algorithm is used for the queuing phase and the Earliest Start Time (EST) for the resource assignment phase. All users have non-critical deadlines equal to 110 seconds. Furthermore, in our simulations we assume a simple network topology, where the resources, the users and the meta-scheduler communicate directly with links of equal bandwidth (100 Mbps). The sizes of the data sent to a resource from a user before a task's execution, and the sizes of the data produced by a resource after a task's completion are the same for all users and equal to 1000 bytes. The maximum acceptable task length of the resources is equal to twice the larger task length produced by any user, that is equal to 20000 MI. The GS users maximum task lengths are equal to their corresponding task lengths presented in Table 2. Finally, in each simulation experiment every user produces 500 tasks.

5.2 Performance metrics

In our simulations we recorded the following performance metrics:

- the per user percentage of the number of tasks that miss their non-critical deadlines over the total number of tasks each user creates. In general if a non-critical deadline expires the task remains in the Grid.

- the resource use, defined as the total time a resource is used for the execution of tasks.

5.3 Results obtained

A number of simulations where conducted in order to validate that the proposed framework indeed guarantees QoS to the GS users. In our simulations we used 5 users (3 GS and 2 BE), 3 resources and a meta-scheduler. User task lengths and task generation rates followed a fixed distribution, using the values of Table 2 and Table 4.

5.3.1 Framework Validation

In Figure 3 we show that our framework succeeds in providing QoS to the GS users. Figure 3 presents the per user percentage of the number of tasks that miss their non-critical deadlines over the total number of tasks each user creates. This percentage is presented for all the combinations of the resources scenarios (GB, GBE, GBP, BE) and the BE user's U4 task generation rates (0.05, 0.1, and 1 tasks/sec): GB/0.05, GB/0.1, GB/1, GBE/0.05, GBE/0.1, GBE/1, GBP/0.05, GBP/0.1, GBP/1, BE/0.05, BE/0.1, BE/1. We observe that in all cases the GS users (U1, U2, U3) do not miss their deadlines. Only when the BE scenario is used, where the GS users are treated as BE users, then all the users miss many of their deadlines. In the GBE and the GBP scenarios (Table 5) fewer tasks miss their deadlines, but in the GBE scenario many tasks fail. In the GBE scenario when a BE task arrives at a GS_BE_EQUAL resource but cannot be scheduled, because the constraints of the already registered GS uses cannot be guaranteed, then the task is dropped (fails). So the GBP scenario seems the best in terms of the number of tasks successfully scheduled without missing their deadlines.

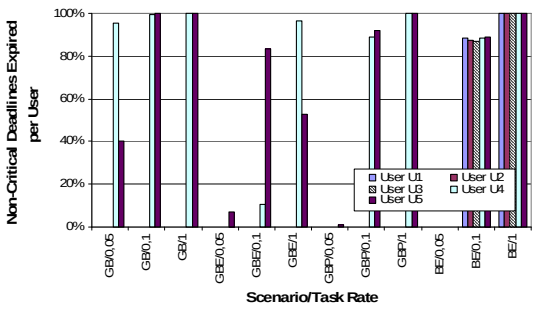

Figure 3. The per user percentage of the number of tasks that miss their non-critical deadlines, for different resource scenarios and U4 task generation rates.

In Figure 4 the total time each resource is used is presented for the same scenarios as before. Resource R3 is utilized more in the GB resource scenario, since it handles exclusively BE tasks. In the other resource scenarios, all resources can serve both GS and BE tasks and as a result the use of resource R3 is smaller. Finally, in Figure 5 the standard deviations of the resources' use are presented. The standard deviation is high in the GB scenario, where resource R3 is more utilized than resources R1 and R2, while it is very small for the GBP scenario. This indicates that the GBP scenario makes more efficient and uniform use of the available resources than the other scenarios.

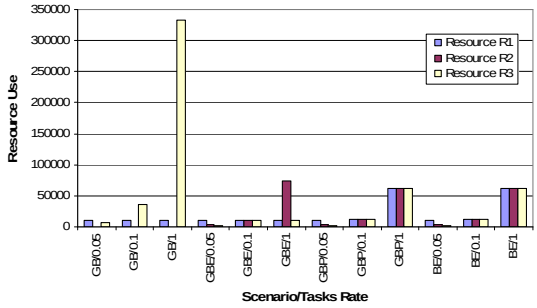

Figure 4. The resource use, for different resource scenarios and U4 task generation rates.

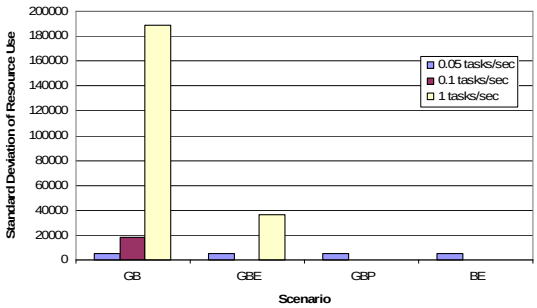

Figure 5. The standard deviation of the resource use, for different resource scenarios and U4 task generation rates.

In our simulations we also experimented with scenarios where the users do not fully respect their agreed (ρ, σ) constraints. Our simulations show that even when the GS users violate their agreed task submission constraints, with small deviation from the agreed values, then the framework is still able to preserve the delay bounds guarantees given.

6 Conclusions

In this work we proposed a QoS framework for computational Grid Networks, which provides deterministic delay guarantees to its Guaranteed Service (GS) users. Our simulations indicate that the framework indeed provides guarantee delay bounds to its GS users, even when a large number of tasks belonging to the Best Effort (BE) users request

service. We also examined several types of resources and showed that the use of resources that serve both GS and BE users, with varying priorities, results in fewer deadlines missed and better resource use.

References

[1] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, 1999.

[2] A. K. Parekh, R. G. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: the single-node case*, IEEE/ACM ToN, 1993.

[3] Y. Zhang, J. Cao, X. Chen, S. Lu, L. Xie, *Threshold-based admission control for a multimedia Grid: analysis and performance evaluation: Research Articles*, Concurrency and Computation: Practice & Experience, 2006.

[4] M. Oikonomakos, K. Christodoulopoulos, E. Varvarigos, *Profiling Computation Jobs in Grid Systems*, Proc. CCGrid, 2007.

[5] I. Ahmad, Y.K. Kwok, M.Y. Wu, K. Li, *Experimental Performance Evaluation of Job Scheduling and Processor Allocation Algorithms for Grid Computing on Metacomputers*, Proc. IPDPS, 2004.

[6] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, *Distributed job scheduling on computational grids using multiple simultaneous requests*, Proc. HPDC, 2002.

[7] Y. Cardinale, H. Casanova, *An evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms*, Proc. HPC&S, 2006.

[8] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, R.F. Freund, *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems*, JPDC, 2001.

[9] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov, A. Shokurov, *Comparison of Scheduling Heuristics for Grid Resource Broker*, Proc. Fifth Mexican Int. Conf. in Computer Science, 2004.

[10] A. Demers, S. Keshav, S. Shenker, *Analysis and simulation of a fair queuing algorithm*, Proc. SIGCOMM, 1989.

[11] R. Braden, D. Clark, S. Shenker, *Integrated services in the internet architecture: an overview*, RFC 1633, IETF, 1994.

[12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Service*, RFC 2475, IETF, 1998.

[13] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, *A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation*, Proc. IWQOS, 1999.

[14] S. N. Bhatti, S.A. Srensen, P. Clarke, J. Crowcroft, *Network QoS for Grid Systems*, Int. Journal of High Performance Computing Applications, 2003.

[15] H. Chu, *CPU Service Classes: a Soft Real Time Framework for Multimedia Applications*, UIUC, Technical Report, 1999.

[16] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, S. Sohail, *G- QoSM: Grid Service Discovery using QoS Properties*, Journal of Computing and Informatics, 2002.

[17] R. Buyya, M. Murshed, *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, Concurrency and Computation: Practice and Experience, 2002.

[18] R. Buyya, J. Giddy, D. Abramson, *An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications*, Proc. 2nd Workshop on Active Middleware Services, 2000.

[19] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, *Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimization Algorithm*, International Journal of Software: Practice and Experience (SPE), 2005.