Future Generation Computer Systems 25 (2009) 674–686

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A framework for providing hard delay guarantees and user fairness in Grid computing

Panagiotis Kokkinos *, Emmanouel A. Varvarigos

Department of Computer Engineering and Informatics, University of Patras, Greece
Research Academic Computer Technology Institute, Patras, Greece

ARTICLE INFO

Article history:
Received 7 February 2008
Received in revised form
17 January 2009
Accepted 27 January 2009
Available online 7 February 2009

Keywords:
Quality of Service
Delay guarantees
User fairness
Scheduling
Grid computing

ABSTRACT

We present and theoretically and experimentally analyze a Quality of Service (QoS) framework for Grids that provides (i) deterministic delay bounds to Guaranteed Service (GS) users and (ii) fair sharing of resources to Best Effort (BE) users. The framework adopts concepts from Data Networks and applies them in the Grid environment. We initially describe the proposed framework assuming that task computational workloads are known (or can be estimated), and then provide extensions for the more realistic case where we have no a-priori knowledge of the task workloads. Task migration across multiple resources is also examined in this context. We also look at a specific implementation of the proposed QoS scheme, where we distinguish computational resources, based on the type of users (GS or BE) they serve and the priority they give to each type. We validate experimentally the proposed QoS framework for Grids, verifying that it satisfies the delay guarantees promised to GS users and provides fairness among BE users, while simultaneously improving performance in terms of deadlines missed and resource utilization. In our simulations, data from a real Grid Network are used.

1. Introduction

The continuing deployment of high speed optical networks is making the vision of Grid Networks a reality. Grids consist of geographically distributed and heterogeneous computational and storage resources that may belong to different administrative domains, but are shared among users by establishing a global resource management architecture. A number of applications in science, engineering and commerce can benefit from the use of Grid Networks. An important issue in determining a Grid's performance is the management of the resources and the scheduling of the tasks to the available resources. Grids are quite dynamic, with resource availability and load varying rapidly with time, while user tasks have very different characteristics and requirements. The extent to which resource management and scheduling are able to cope with this unpredictable environment 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 Data Networks, QoS mainly refers to packet delay, delay jitter, bandwidth and packet-loss rate. In Grid Networks, QoS mainly refers to the total time it takes for a user task to be completed. It can also refer to the time period over which a number of computational resources (space-sharing), or a percentage of one such resource (time-sharing), are reserved by a user. In the case of Storage Area Networks (SAN), QoS can refer to the bandwidth used for data transfers and to the storage capacity (size) available for a task. In order for a network to provide QoS guarantees to a user, a three-step procedure is generally followed. At first the user informs the network of the exact QoS parameters requested (delay, required resources, etc). Then the network, through a procedure called admission control, checks whether it can satisfy the user's request for guaranteed service, without violating the guarantees given previously to other users. If this is possible, then various mechanisms (resource reservation, scheduling, flow control) are employed to ensure that the agreed-upon QoS level is provided to the user.

Today's Grids provide only a best effort service to the users and their tasks. This is inadequate if the Grid Network is to be used for real world commercial applications and time-critical scientific computations. Best-effort service also limits the economic importance of Grids, since users will be reluctant to pay, directly or indirectly (e.g., by contributing resources to the Grid), for the service they receive, if they are not given performance guarantees. As a result, there is a growing need for Grid scheduling and resource management algorithms to be able to provide QoS to the users. Under these thoughts we believe that future Grids will serve two types of user. Some users will be relatively insensitive to the performance they receive from the Grid and will be happy to

* Corresponding author at: Department of Computer Engineering and Informatics, University of Patras, Greece.
E-mail address: kokkinop@ceid.upatras.gr (P. Kokkinos).

0167-739X/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2009.01.003

accept whatever performance they are given. Even though these Best Effort (BE) users do not require performance bounds, it is desirable for the Grid Network to allocate resources to them in a fair way. In addition to BE users, we also expect the Grid Network to serve users that do require a guaranteed QoS. These users will be referred to as Guaranteed Service (GS) users. By the term "user" we do not necessarily mean an individual user, but also a Virtual Organization (VO), or a single application, using the Grid infrastructure.

In this work we present a QoS framework that adopts concepts from Data Networks and applies them to Grids, for handling GS and BE users. For GS users, the framework guarantees an upper bound on the delay of the submitted tasks. A task's delay is defined as the time between the task's creation and the time the results of its execution return to the user. The delay guarantees imply that a GS user can choose a resource to execute a task before its deadline expires, with absolute certainty. In order to achieve this goal, the GS users are leaky-bucket constrained, so as to follow a self constrained task generation pattern, which is agreed separately with each resource during a registration phase. For BE users, the proposed framework describes a fair scheduling procedure that provides fairness *among users* instead of fairness *among tasks*. This notion of user fairness is more appropriate for Grids, since the main entities in Grids are not the tasks but the users creating them (a "user" may also refer to a Virtual Organization, or VO, using the Grid infrastructure). We initially describe the proposed framework assuming that the task computational workloads are known (or can be estimated accurately) and then provide extensions for the case where we have no a priori knowledge of the task workloads. Both single and multi-CPU resources are examined. Task migration across multiple resources is also considered in the context of the proposed framework.

We discuss a specific implementation of the proposed QoS framework, where we distinguish between four types of resource that serve either GS, or BE, or both types of user, with varying priorities. GS users are registered to the resources either statically or dynamically. Distributed, centralized and hybrid implementations of the QoS framework are analyzed. In addition to the theoretical analysis, we also implemented the proposed QoS scheme in the GridSim environment [1] and evaluated its performance. Our results indicate that as long as GS users respect their constraints, even with small deviations, the framework succeeds in providing hard delay guarantees to them. Also, we observe that the use of resources that handle both GS and BE users with varying priorities produces better results, in terms of deadlines missed and resource utilization, than the use of resources that are specifically assigned to each class (GS or BE) of user. The benefits obtained are similar to those obtained through statistical multiplexing in Data Networks. Our results also indicate that providing delay guarantees without exact a priori knowledge of task workloads is possible, if the proposed extensions are used. Furthermore, the user fair scheduling algorithm proposed does not only provide fairness among users, but also improves performance in terms of average task delay and probability to miss a deadline. Finally, 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 on previous work. In Section 3 we describe the proposed QoS framework for Grids. In Section 4 and 5 we propose extensions to this framework. In Section 6 we present the simulation environment, the parameters used, and the results of our simulations. Finally, conclusions are presented in Section 7.

2. Previous work

In the Grid related literature a large number of scheduling algorithms has been proposed [2,3]. A taxonomy of the various scheduling algorithms is presented in [4]. Relatively recently Quality of Service (QoS) in Grids started gaining attention, since it has been realized that the success and the economic impact of the Grid largely depends on its ability to guarantee QoS to the users. As mentioned, the term QoS is used differently based on the types of resources considered (communication, computational, storage). Most of the scheduling algorithms that have been proposed so far for Grid computing provide a best effort service to the submitted tasks, while fewer works provide hard QoS guarantees [4]. Another categorization of the various QoS algorithms is whether they handle one-dimension (e.g., only delay) [5] or multi-dimension (e.g., delay and bandwidth) QoS issues [6,7]. In the multi-dimension QoS studies the various parameters are considered jointly [6] or separately [7], with different weighting factors attached to the various parameters, reflecting the user preferences.

Best effort scheduling algorithms [3,8] in Grids try to optimize a metric of interest (for example task delay, resource utilization). The scheduling algorithm proposed in [8] is a best-effort algorithm where a metric (called response ratio) is used to define the QoS provided to an individual task. Soft QoS scheduling algorithms [5, 9,10] also exist that take into account the users' QoS requirements (delay, required number of CPUs, etc) and try to serve them in the best possible way. In [5] the task's QoS information is embedded into the scheduling algorithm in order to find a better matching between the tasks and the resources. In [9] the deadline and budget QoS constraints of a task are taken into account during the task scheduling process. In the hard QoS case, the requirements of the users are guaranteed by the scheduling algorithm. The main mechanism used for providing hard QoS guarantees to the Grid users is the reservation of the resources (communication, computational or storage). The reservation can be immediate, undertaken in advance [7] or flexible (malleable) [11]. The Globus Architecture for Reservation and Allocation (GARA) [12] is a framework for advance reservations that treats in a uniform way various types of resource such as communication, computation, and storage. Although GARA has gained popularity in the Grid community, its limitations in coping with current application requirements and technologies led to the proposal of the Grid Quality of Service Management (G-QoSm) framework [13]. Since their introduction, reservations have been studied in numerous works [14–17]. The reservation of resources generally tends to reduce resource utilization efficiency, since in many cases resources are underused. In this context a number of works that perform advance reservations, while achieving high utilization efficiency have been proposed [15,17]. The reservation of networking resource for Grid applications has been the focus of a number of works [11,18,19]. In [7], a multi-cost scheme is presented for jointly reserving computational and communication resources. In particular the proposed scheme selects the computation resource to execute the task and determines the path to route the input data, performing reservations in advance. Furthermore, a number of other QoS and Grid related issues is investigated. Users requiring QoS guarantees may negotiate a Service Level Agreement (SLA) with the Grid, to enforce desired level of service [14,20]. In [14] a QoS scheduler is presented that uses SLAs to efficiently schedule flexible advance reservations for computation services. Recently QoS and workflow issues were, also, investigated [6].

In the present work we propose a QoS framework, which is based on the theorem of the Parekh and Gallager [21] where instead of communication resources we have computational resources and instead of packet size we have task workload. In [21] it was proven that if a user's flow is leaky-bucket constrained and

Weighted Fair Queuing (WFQ) scheduling is applied at each router between the source (the user) and the destination, then hard delay bounds can be guaranteed for the flow. There have been hundreds of papers written on WFQ and its variations. Moreover, the Internet Engineering Task Force (IETF) has proposed the Integrated Services (IntServ) [22] and the Differentiated Services (DiffServ) QoS architectures [23] to support QoS differentiation with respect to bandwidth, latency and other data transfer parameters. The IntServ architecture can provide hard bounds on end-to-end packet delays, using the Parekh–Gallager theorem [21]. However implementing Intserv and its corresponding Resource ReSerVation Protocol (RSVP) in practice was proven to be quite difficult. The problem is that all intermediate routers must be RSVP capable, and also each router must maintain the state for each flow passing through the router.

Our framework uses "rules" similar to the ones provided by the Parekh and Gallager theorem [21], in order to guarantee a bound on the time (single dimension) by which a task will finish its execution on a resource. This bound is provided without using hard resource reservations, in contrast to the majority of Grid QoS related works [12,13]. A user and a resource simply agree upon the task load the former will generate and the latter will serve. We call it a *framework* because it gives conditions that if satisfied can provide delay guarantees for the completion of a task on a given resource, but leaves a great deal of flexibility in terms of the specific scheduling algorithms to be used. In the GARA [12] and G-QoSM [13] frameworks reserve computational resources explicitly, 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 [24]). Moreover, many of the works that use advance reservations either assume that the task workloads are known a priori [7] or a prediction mechanism is used [5]. We initially describe the proposed framework assuming that task computational workloads are known (or can be estimated), and then propose a number of novel techniques, which are not based on prediction mechanisms, to account for the more realistic case where we have no a priori knowledge of the task workloads.

In our work we concentrated on computational tasks and presented a way for guaranteeing their delay, assuming that no data dependencies exist. Trying to provide a unified framework that would also include guarantees for the possible data transfers required for executing a task, would complicate our analysis. Moreover, in the existing Grid literature there are solutions, based on the reservations of the networking paths that provide time guarantees on a task's related data transfers [11,18,19]. Using these solutions and our proposed framework the task's delay time, including the time required for the data transfers and its actual execution time, can be bounded. Also, in our framework we do not discuss in detail cost/pricing or implementation considerations. We decided to concentrate more on the theoretical basis of our framework and in the proof of its validity. However, we are already in the process of implementing the proposed framework in the gLite middleware [25]. In particular, we have evaluated gLite's existing scheduling mechanisms and succeed in implementing our own simple scheduling mechanism, by changing the Workload Management System (WMS). For the implementation of our proposed framework in gLite more changes are needed in various components, e.g., the User Interface (UI), the Information SuperMarket (ISM) and other. We hope that in the near future we will succeed this goal and present our work to the community.

Finally, fairness is not a new concept for scheduling in general, and especially for scheduling in Data Networks. Generalized Processor Sharing (GPS) [21] has been proposed for the fair sharing of capacity on communications links. Since GPS is difficult to implement, its approximation Weighted Fair Queuing (WFQ) [26] is instead often used in Data Networks. Fairness in Grid Network has been investigated in a number works [27,28]. In this work we also propose a fair scheduling algorithm for Grids that provides fairness among users instead of fairness among tasks [28]. In the literature a number of works have supported user fairness in Data [29] or in Grid Networks [30], instead of packet, flow or task fairness.

3. Description of the framework

3.1. General

We consider a Grid Network consisting of a number of users and resources. There are two kinds of user: Guaranteed Service (GS) and Best Effort (BE) users, who generate tasks 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. The objectives that we set for the proposed QoS framework are: (a) to provide service guarantees to GS users and (b) to ensure fair sharing of the resources among BE users. In order to achieve the first objective, the GS users are leaky-bucket constrained, so as to follow a (ρ, σ) constrained task generation pattern that is agreed separately with each resource. That is a user can, in a very short time interval, submit to a resource σ tasks of unitary length, even if this way she violates the ρ average task submission rate constraint; however, in the long term the user cannot exceed this average rate. Moreover, the ρ and σ parameters are measured in different units, for example tasks of unitary length (or Million Instructions) per second and number of tasks of unitary length (or Million Instructions), correspondingly. To determine the appropriate values for the (ρ, σ) parameters between a user and a resource, each user has to estimate his average long term service requirements (ρ) and its desired burstiness (σ), based on past statistics and approximate assumptions (see also Section 4.1). On the resources, the arriving tasks are queued in a Weighted Fair Queuing (WFQ) scheduler [26]. This way guaranteed task service rates (e.g., measured in Millions of Instructions Per Second) and guaranteed task delays can be given to each GS user, in the same way WFQ provides guaranteed bandwidth and packet delay services in Data Networks. BE users, on the other hand, are handled by our framework with fairness as the main goal. However, in contrast to most other fairness related works [28], we aim at providing fairness among users and not among individual tasks.

We assume, unless otherwise stated, that a task executing at a resource is non-divisible and non-interruptible (non-preemptable). We initially describe our framework assuming that each machine has a single CPU, and later extend it to the multi-CPU machine case.

3.2. Guaranteed Service (GS) users

In the proposed QoS framework, a GS user must first register to a resource, before it can actually use it. During the registration phase, the GS user (or Virtual Organization — VO) and the resource agree upon the characteristics of the computational workload the GS user will send to that resource, that is, the leaky-bucket's parameters. A GS user can register to a number of resources. Next, when a GS user creates a task, one of his registered resources is chosen for its execution, based on various criteria, such as performance (e.g., delay), fairness among resources (e.g., uniform utilization of the registered resources), etc.

Our framework is implemented in a distributed way, and, as a result, scheduling logic exists at the GS user site and at the resource site (local scheduler). During the registration phase, a GS user i and a resource r agree upon the (ρ_{ir}, σ_{ir}) constraints (Fig. 1) of the user. The parameter ρ_{ir} is the long term workload

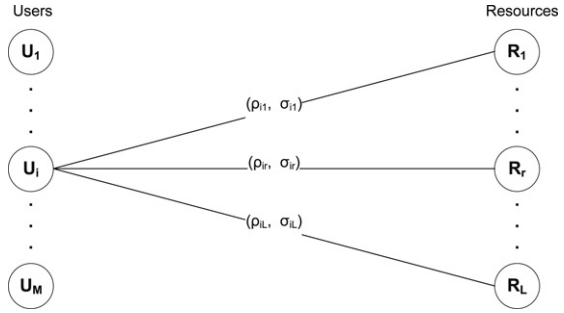

Fig. 1. The (ρ, σ) constrained GS users in the Grid Network.

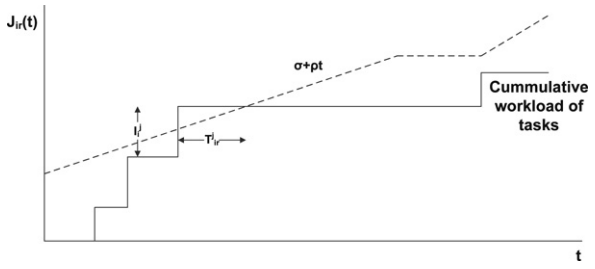

Fig. 2. The GS user is responsible for the observance of his (ρ_{ir}, σ_{ir}) constraints.

generation rate, measured in computation units per second (e.g., Million Instructions Per Second — MIPS), that GS user i will submit to resource r. The parameter σ_{ir} is the maximum size of tasks (burstiness) that GS user i will ever send, in a very short time interval, to resource r, and is measured in computation units (e.g., Million Instructions — MI). If resource r can accept this average load and burstiness, 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 5), where a meta-scheduler is used as an intermediary for the monitoring of the observation of the (ρ, σ) constraints.

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 computational workload generation rate ρ_{ir} without violating the workload 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:

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

where C_r is the computing capacity of resource r, $N_r(t)$ is the number of GS users already registered to resource r at time t, and w_{ir} is the weight of GS user i in using resource r. The weights w_{ir} can depend on various parameters, such as the prices the GS users are willing to pay, or their other contributions to the Grid (as in [28]). Condition (1) ensures that resource r can satisfy the task generation rates of both the new and old GS users.

An additional condition that is agreed during the registration of GS user i to resource r is that the maximum task workload $I_{ir}^{\max}$ user i will ever send to resource r will not exceed the resource's maximum acceptable task workload $I_r^{\max}$:

$$I_{ir}^{\max} \leq I_r^{\max}. \tag{2}$$

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

Each GS user i is equipped with an input 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 $J_{ir}(t), i = 1, 2, \ldots, N$, the total computational workload (measured, e.g., in MI) submitted by GS user i to resource r in the interval $[0, t]$. We will say that GS user i is (ρ_{ir}, σ_{ir}) controlled with respect to resource r, if the following condition is valid:

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

If a GS task j invalidates (3), then the GS user must locally withhold this task for a time period, denoted by T_{ir}^j, until (3) becomes valid again (Fig. 2). So our framework includes in every GS user an admission control (leaky-bucket) mechanism, to make sure a task reaches a resource only when some specific constraints are valid.

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

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

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 [21], that the delay a task will incur from the time it reaches resource r until it finishes its execution is at most

$$\frac{\sigma_{ir}}{g_{ir}} + \frac{I_{ir}^{\max}}{g_{ir}} + \frac{I_r^{\max}}{C_r}, \tag{5}$$

where g_{ir} is the minimum value of $g_{ir}(t)$ that does not invalidate (1) for any registered user. This delay includes the maximum time needed to execute a task of another user that may occupy the single-CPU resource when the task of user i arrives, $\frac{I_r^{\max}}{C_r}$, the maximum time needed for the previously submitted tasks of user i to execute in the resource, $\frac{\sigma_{ir}}{g_{ir}}$, and the maximum time needed to execute the task itself, $\frac{I_{ir}^{\max}}{g_{ir}}$. To this delay we must add the total communication delay, denoted by d_{ir}^j, required for transferring the task to the selected resource, and the time T_{ir}^j the GS user withholds the task in its local queue (Fig. 2). So the delay bound B_{ir}^j resource r guarantees to user i satisfies

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir}}{g_{ir}} + \frac{I_{ir}^{\max}}{g_{ir}} + \frac{I_r^{\max}}{C_r}. \tag{6}$$

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

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir} + I_{ir}^{\max}}{g_{ir}} + \frac{I_r^{\max}}{C_r}, \quad \text{or}$$

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{(\sigma_{ir} + I_{ir}^{\max}) \cdot (N_r(t) + 1) + I_r^{\max}}{C_r}.$$

When the GS user does not have any more tasks to submit, he can either do nothing, or he can deregister from his registered

resources. In the latter, dynamic, case the other GS users are informed for the user's deregistration and they can try to register to these resources.

Furthermore, we can pipeline the communication delay d_{ir}^j and the input queuing delay T_{ir}^j to obtain:

$$B_{ir}^j \leq \max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir} + I_{ir}^{\max}) \cdot (N_r(t) + 1) + I_r^{\max}}{C_r}.$$

By pipelining, we mean that if d_{ir}^j is larger than T_{ir}^j, then the user i sends 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. In both cases time savings are achieved.

In order for a task j of 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:

$$B_{ir}^j \leq D_i^j. \tag{7}$$

If more than one resource fulfills the conditions of Eqs. (4) and (7), the GS user can choose one based on any other desired optimization criterion. If no resource fulfills these conditions, the GS user drops the task or schedules it like a BE task. Also, from (6) we conclude that it may be beneficial to partition the resources in groups offering different maximum delay guarantees. More specifically, the a priori determination of a resource's computational capacity C, maximum task length I, maximum burstiness σ and maximum number of GS users allowed N, provides a guaranteed maximum delay for the tasks sent to that resource:

$$B(C, I, N, \sigma) \leq \max(T, d) + \frac{(\sigma + I) \cdot N + I}{C}, \tag{8}$$

where T and d do not depend on the resource but on the user side. If σ is expressed as a multiple of I, $\sigma = m \cdot I$ (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 (8) can be rewritten as:

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

3.3. Best Effort (BE) users

Most Grid scheduling algorithms proposed to date schedule BE tasks based on various task characteristics, such as its deadline, workload, or price the user is willing to pay for its execution. Fairness, however, is another very important criterion that should be taken into account in Grid scheduling. The number of different resource types comprising a Grid Network (computation, communication, storage) makes the application of fairness in Grids a more complex issue than, for example, in Data Networks. In our view, a fair Grid scheduling algorithm should have the following characteristics:

- User fairness: Fairness should not be enforced on a per task basis, but on a per user basis [29,30], where the term "user" may also refer to a Virtual Organization (VO). For example, it is not fair for a task belonging to a BE user who creates only this task, to be handled equally with the possibly thousands of tasks created by some other BE user. Different weights can also be given to users (or VOs), in which case we talk about weighted user fairness.
- Joint fairness: Grids consist of computational, communicational and storage resources. As a result, fairness must be achieved jointly for all these types of resources [31].
- Task fairness: Fair scheduling only on a per user basis may not be the best policy, because of different task characteristics and requirements (e.g., deadline, task workload, completion time etc), which should also be incorporated in a fair scheduling algorithm.
- Resource uniformity: Future Grids will consist of consumers (users willing to pay for the use of the Grid resources) and providers (users offering their resources). In such an environment it is desirable that the resources are fairly used, even when there is a plethora of resources available for the execution of the tasks.

In this section we concentrate on user fairness and propose a centralized user fair scheduling algorithm for Grids, called WFQ/EST. Other fair algorithms proposed in the literature [28] can also be used instead of WFQ/EST in the context of the proposed framework for serving BE tasks. The WFQ/EST algorithm consists of two phases, task-ordering and task-to-resource assignment, and it is executed at periodic intervals. During a period, tasks belonging to different users arrive at the central scheduler and are handled by a Weighted Fair Queuing (WFQ) scheduler [26] (Fig. 3). The WFQ scheduler places these tasks in different queues based on their originating users, and has similar functionality to that of the local WFQ scheduler used at the resources for serving GS tasks. When a period expires, the task-ordering phase is executed, during which the tasks are dequeued from the WFQ scheduler and proceed to the task-to-resource assignment phase. In the task-to-resource assignment phase the Earliest Starting Time (EST) algorithm is used to assign tasks to resources.

Note that WFQ/EST schedules BE tasks in a centralized way, while the proposed framework schedules GS tasks in a distributed way. However, as mentioned earlier, WFQ/EST is only an example of a user fair scheduling algorithm and any other fair (or not) scheduling algorithm centralized (or not) can also be used in its place. Furthermore, in the following sections we also describe a centralized implementation of the proposed scheme for GS users.

3.4. Resource categorization

To obtain a specific implementation of the proposed QoS framework, we distinguish four types of resources, to be referred to as GS, BE, GS_BE_EQUAL and GS_BE_PRIORITY resources. 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 in the resource becomes free, 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 Serve (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 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 Eq. (1) holds for all the registered users (the weight in Eq. (1) for any BE user, equals 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 decreased by one. If (1) is violated 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 both GS and BE tasks, but not in the same way. GS tasks are handled by the local WFQ scheduler, while BE tasks are placed in a FCFS queue. When a machine becomes free, the tasks in the local WFQ scheduler are handled first. If there are no such tasks, the BE tasks from the FCFS queue are served. A GS_BE_PRIORITY resource is characterized as preemptive if upon the arrival of a GS

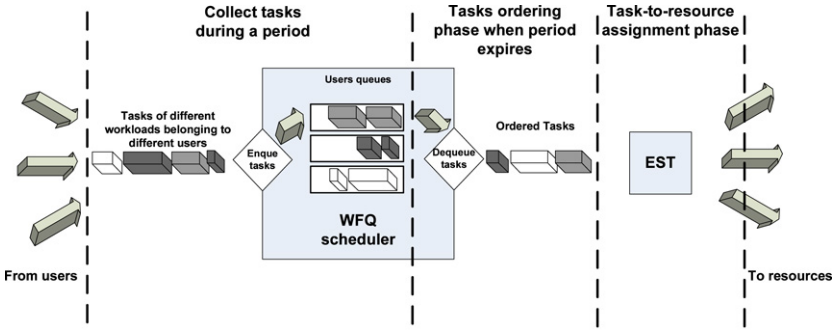

Fig. 3. User fair scheduling using a WFQ scheduler.

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

Resource	Delay bound for GS users	Fair/s for BE users
GS	$\max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+I_{ir}^{\max})\cdot(N_r(t)+1)+I_r^{\max}}{C_r}$	–
BE	–	Yes
GS_BE_EQUAL	$\max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+I_{ir}^{\max})\cdot(N_r(t)+1)+I_r^{\max}}{C_r}$	Yes
GS_BE_PRIORITY pr.	$\max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+I_{ir}^{\max})\cdot(N_r(t)+1)+I_r^{\max}}{C_r}$	Yes
GS_BE_PRIORITY non-pr.	$\max(T_{ir}^j, d_{ir}^j) + \frac{(\sigma_{ir}+I_{ir}^{\max})\cdot(N_r(t)+1)+2\cdot I_r^{\max}}{C_r}$	Yes

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 size.

When a GS_BE_PRIORITY non-preemptive resource is used, the delay bound for GS tasks of Eq. (6) becomes

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

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

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

In all other resource types (namely, GS, GS_BE_PRIORITY preemptive) R_r equals to 0.

Therefore, delay bounds are provided to GS tasks submitted to GS, GS_BE_EQUAL or GS_BE_PRIORITY resources, while fairness is also provided among BE users for tasks submitted to BE, GS_BE_EQUAL or GS_BE_PRIORITY resources. Table 1 presents the user/resource combinations that provide either delay bounds or fairness. In case delay bounds are provided, Table 1 gives their exact values.

4. Realizing the framework

In order to realize the QoS framework in an actual Grid system we consider two main topics. First we propose policies based on which the user selects his (ρ, σ) parameters. Next, we adapt our framework in order to operate without the a priori knowledge of task workloads, which is usually the case.

4.1. (ρ, σ) constraints selection policy

To determine the appropriate values for the (ρ, σ) parameters between a user (or VO) and a resource, each user has to estimate his average long term service requirements, and its desired burstiness (or the delay he can afford for "smoothing" the traffic to fit a given σ). A user may be based on past statistics and approximate assumptions about his task generation rate and workload. It is natural to assume that the price a user pays for the use of the Grid is an increasing function of ρ and σ. If the user chooses ρ too close to his average long term needs and/or chooses a small σ, then an arriving task may have to suffer a long delay T waiting for (3) to become valid. If the user can afford to pay a higher price, it may be beneficial to overestimate ρ or σ so as to reduce this delay. The (ρ, σ) parameters requested by the user may be too large for the meta-scheduler to accept. During the registration phase the meta-scheduler will determine the exact values of these parameters dynamically, based on the computational power of the resource, the distance from the user, the resource's delay bound, the number of users already registered, etc. For example, the meta-scheduler may choose to accept the requested (ρ, σ) values if the resource has significant computational power, or it may negotiate with the user smaller values if the resource is less powerful.

4.2. Scheduling without a priori knowledge of task workloads

The proposed QoS framework offers hard delay guarantees to GS users that respect their negotiated (ρ, σ) constraints. The observance of a GS user's (ρ, σ) constraints requires the a priori knowledge or accurate estimation of the task workloads, which is not always possible. Even though it is clearly possible for the user to measure and control dynamically the rate at which he/she submits tasks to the Grid, it may not be easy to measure and control their workload.

In what follows we examine how our framework can be extended to operate without a priori knowledge of the task workloads. Specifically, we propose the following methods:

- Conservative Task Submission: The meta-scheduler assumes that each new task has a workload equal to the user's maximum task workload, which is the maximum of $I_{ir}^{\max}$ for all the user's i registered resources r, and updates the variable $J_{ir}(t)$ based on this assumption. In case condition (3) is violated, the task is backlogged.
- n-Window Aggressive Task Submission: The meta-scheduler schedules up to n consecutive new tasks of GS user i to a resource r, assuming their workload is equal to zero. Any new task j destined for resource r that arrives after the n consecutive tasks is backlogged, until a workload feedback message for any of these n tasks arrives from resource r. Specifically, when a task completes execution on resource r, the resource informs the meta-scheduler of the task's actual workload, and the variable $J_{ir}(t)$ is updated. When this method is used, the delay bound resource r guarantees to user i for a task j is increased to:

$$T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir} + n \cdot I_{ir}^{\max}}{g_{ir}} + \frac{I_{ir}^{\max}}{g_{ir}} + \frac{I_r^{\max}}{C_r}. \quad (9)$$

When no limit is placed on the number n of consecutive tasks that can be sent without a priori knowledge of their workload, then no deterministic delay bound can be given. We refer to this case as Full Aggressive Task Submission method.

- Conservative Task Submission with Feedback: The above two methods can be combined. The meta-scheduler schedules new tasks of GS user i to resource r, assuming that their workload is equal to the user's maximum task workload, and updates $J_{ir}(t)$ based on this assumption. When a task completes execution, a workload update message is sent back to the meta-scheduler, which corrects its previous assumption on the task workload, and updates $J_{ir}(t)$ accordingly. In case condition (3) is violated, the corresponding task is backlogged.

5. Extensions of the proposed framework

5.1. Distributed, centralized and hybrid implementations

In Section 3 we assumed a distributed implementation of our proposed QoS framework, where registration is performed by each user (or VO) by communicating directly with the resource and negotiating its (ρ, σ) constraints. However, other approaches can be used.

In the centralized approach the registration of the GS users to the resources is handled by a central meta-scheduler. The meta-scheduler accepts, from the GS users, registration requests containing their requested (ρ, σ) parameters, and searches for resources that can satisfy these constraints. The meta-scheduler is responsible for enforcing the (ρ, σ) constraints of the GS users. This centralized approach has various advantages. First, many of the heavy operations performed by the GS users are transferred to a central, possibly more powerful, machine. Second, it is possible to use more than one central meta-scheduler in order to balance the load and the traffic in the Grid Network. On the other hand, the use of a single central meta-scheduler increases the risk of a failure in the Grid Network. Also GS task average total delay increases, because of the delay induced by the communication between the GS users and the central meta-scheduler.

This centralized approach is more close to the actual Grid model, consisting of different administrative domains offering their resources to the users. That is, each user can negotiate with the central entity (e.g., the central scheduler) of each administrative domain in order to find the domain that can satisfy its (ρ, σ) constraints. The central entity that receives a user's requirements, negotiates with the domain's resources in order to register the user to one (or more) of them. Upon the arrival of a new task, the central entity submits the task to one of the users' registered resources, so as not to violate its constraints. The user can concurrently negotiate and use the resources of a number of different administration domains. Moreover, the exact (ρ, σ) constraints agreed between a domain and a user can be based on the domain the user itself belongs to. For example, better (ρ, σ) parameters can be agreed with a user belonging to a more powerful administration domain.

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 observance of their (ρ, σ) constraints. Using this hybrid approach the meta-scheduler is relieved from the burden of scheduling GS tasks. Furthermore, GS tasks do not experience the delay of communicating with the meta-scheduler, reducing in this way the total task delay.

5.2. Multi-machine resources

The proposed framework can easily be extended to the case of resources that consist of many machine-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 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 resource r. However, in the multi-machine resources case the term C_r used in Eqs. (1) and (6) is not always equal to C'_r. Furthermore, we 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 resource r, C_r in Eq. (1) has to be replaced by C'_r, yielding

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

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 Eq. (6) and in all the other delay bounds given in Section 3 has to be replaced by $C_r^{\min}$. For example, Eq. (6) becomes:

$$B_{ir}^j \leq T_{ir}^j + d_{ir}^j + \frac{\sigma_{ir}}{g_{ir}} + \frac{I_{ir}^{\max}}{g_{ir}} + \frac{I_r^{\max}}{C_r^{\min}}. \tag{10}$$

5.3. Framework's application to task migration

Up until now we have assumed the usual scenario where a task is created by a user, then executed on a resource and finally returned back to the user. It is possible, however, under certain conditions, to extend our framework to the case where tasks migrate among a number of resources, before returning the final results to the user. Various migration policies exist:

- The user explicitly asks for task migration.
- The resource decides to migrate an executing task to another resource.
- The task itself decides that it needs more or less computational power and decides to migrate to another resource.
- The user specifies the price he is willing to pay for task execution and when a cheaper resource becomes available the task migrates to it.
- The task migrates in response to network failures or DoS attacks, providing fault tolerance in this way.

All these migration policies are dynamic, in the sense that a task starts execution on some resource and may then migrate to an other resource if various runtime conditions hold.

In such cases, where the sequence of resources to be visited is not known a priori, the proposed QoS framework cannot be applied. It can be applied, however, when the meta-scheduler has a priori knowledge of the resources a task is going to be executed on (and the maximum task workload to be executed on each resource). This can be the case when data replication strategies are applied and a task needs for its execution various types of data

located at different predetermined resources. For example, assume that a task needs for its execution data of type A, B and C, which are stored at different corresponding resources. In this case the task can visit sequentially the three resources, containing the A, B and C type of data. In each resource the task is executed before migrating to the next one, carrying with it any intermediate results. This strategy may be more beneficial in terms of communication times and storage requirements, than transferring all the data (A, B and C) to one resource. So in case it is known a priori that a task will visit for its execution a static sequence of K resources, then Eq. (6) can be expressed as

$$B_{ir}^{j} \leq T_{i}^{j} + \sum_{r=1}^{K} d_{ir}^{j} + \frac{\sigma_{ir}}{g_{ir}} + \sum_{r=1}^{K} \frac{I_{ir}^{\max}}{g_{ir}} + \sum_{r=1}^{K} \frac{I_{r}^{\max}}{C_{r}}, \tag{11}$$

where T_i^j is the maximum input delay of task j for all the resources in the sequence.

6. Simulation results

6.1. Simulation environment and assumptions

We implemented the centralized version of the proposed QoS framework in the GridSim simulator [1]. For a GS user, his (ρ, σ) constraints are specified, along with the maximum workload I of his generated tasks. Each resource is of a specific type and has a maximum acceptable task workload. In our simulations, a GS user uses the same (ρ, σ) parameters for all the resources it registers to. The central meta-scheduler is responsible for the registration phase, the observance of the (ρ, σ) agreements between GS users and resources, and the assignment of tasks to resources. All users register to resources at the beginning of the simulation and remain registered for its entire duration. The local-scheduler of a resource is equipped with a FIFO queue and a Self-Clocked Fair Queueing (SCFQ) scheduler. SCFQ is a variation of Weighted Fair Queueing (WFQ) that is easier to implement than WFQ. Based on their type and the type of the resources, tasks are assigned either to the FIFO queue or to the SCFQ scheduler. We assume that the local-scheduler uses a space-sharing resource allocation policy. We also assume, unless stated otherwise, that each resource consists of a single-CPU machine and task workloads are known a priori.

6.2. Parameters and scenarios

In order to obtain realistic simulation parameters, we used the results of the Grid profiling study of [32], where numeric data (as well as analytic models) on the cumulative distribution functions, average values and standard deviations of the task inter-arrival times, queue waiting times, task execution times, and data sizes exchanged at the kallisto.hellasgrid.gr cluster (part of the EGEE infrastructure) were presented.

Based on these results and numeric data we decided to simulate three GS users, named U1, U2 and U3, corresponding to three of the five VOs presented in [32] (the Atlas, Magic and Dteam VOs). Using the VO's average task execution times and processor speed (estimated to be about 26000 MIPS for the processors in the kallisto.hellasgrid.gr cluster) we calculated their corresponding average task workloads, measured in Million Instructions (MI). Based on [32] we decided the following simulation parameters for modeling GS users:
Also, based on Table 2, the (ρ, σ) constraints of each GS user were calculated (Table 3). Specifically, the ρ parameter of each GS user is calculated by dividing its average task workload by its average task inter-arrival time, while the σ parameter is selected to be $m = 5$ times larger than the corresponding GS user's average task workload.

Table 2
GS users task workload and inter-arrival time.

User	Characteristic	Distribution
Atlas/U1	Task workload	Fixed: 419900000 MI
Magic/U2	Task workload	Fixed: 71383000 MI
Dteam/U3	Task workload	Fixed: 419900 MI
Atlas/U1	Task inter-arrival	Fixed: 546 secs/task (s/t)
Magic/U2	Task inter-arrival	Fixed: 3278 secs/task (s/t)
Dteam/U3	Task inter-arrival	Fixed: 6010 secs/task (s/t)

Table 3
GS users (ρ, σ) constraints.

User	Characteristic	Distribution
Atlas/U1	ρ	Fixed: 768473 MIPS
Magic/U2	ρ	Fixed: 21773 MIPS
Dteam/U3	ρ	Fixed: 699 MIPS
Atlas/U1	σ	Fixed: 2099500000 MI
Magic/U2	σ	Fixed: 356915000 MI
Dteam/U3	σ	Fixed: 2099500 MI

Table 4
BE users task workloads and inter-arrival time.

User	Characteristic	Distribution
U4	Task workload	Fixed: 419900000 MI
U5	Task workload	Fixed: 419900000 MI
U4	Task inter-arrival	Fixed: 5464, 4371, 3278, 2186, 1366, 1093, 820, 546, 55 secs/task (s/t)
U5	Task inter-arrival	Fixed: 5464 secs/task (s/t)

Table 5
Resource's computational capacities.

Resource	Simulated CPUs	Total computational capacity (MIPS)
R1	30	$30 \times 26000 = 780000$
R2	20	$20 \times 26000 = 520000$
R3	10	$10 \times 26000 = 260000$

Table 6
Resource scenarios.

Scenario	R1	R2	R3
GB	GS	GS	BE
GBE	GS_BE_EQUAL	GS_BE_EQUAL	BE
GBP	GS_BE_PRIORITY n.pr.	GS_BE_PRIORITY n.pr.	BE
BE	BE	BE	BE

In our simulations we also included two BE users, named U4 and U5. U4's average task inter-arrival times change in every simulation, while U5's remain the same (Table 4). The task workloads submitted by these users were equal to the Atlas/U1 average task workload (namely, 419900000 MI).

At the kallisto.hellasgrid.gr cluster presented in [32], 60 working nodes (Intel Xeon processors) are used, with a total capacity of 60×26000 MIPS. In our simulations, we used 3 clusters (resources) each having one machine with one CPU of computational capacity equal to a multiple of 26000 MIPS (Table 5). The resource type scenarios examined are presented in Table 6. For example, in the GB scenario of Table 6, resource R1 and R2 are allocated for serving GS users, while resource R3 serves BE users. When the BE resource scenario is used then all the GS users (U1, U2, U3) are treated as BE users with the same characteristics as before.

The meta-scheduler uses a two-phase procedure for scheduling BE users, with task collection period equal to 1 s. Unless stated otherwise, the two-phase scheduling procedure uses the Earliest Deadline First (EDF) algorithm (the task with the most imminent deadline is scheduled first) for the ordering phase and the Earliest Start Time (EST) algorithm (the task is assigned to the resource where it will start execution sooner) for the assignment phase. All the users have non-critical deadlines, with values equal to 110 s.

In general, if a critical deadline expires, the corresponding task is removed from the Grid, while if a non-critical deadline expires the task remains in the Grid, but is recorded as a deadline miss. The deadline's value was selected based on (5) by adding a small time overhead to account for the input queuing (T) and communication (d) delays. Furthermore, in our simulations we assume that the created tasks have very small data dependencies; for this reason we use a simple grid infrastructure where the resources, the users and the central meta-scheduler communicate directly with each other over links of equal bandwidth and zero propagation delay. In particular, the sizes of the data sent to a resource before task execution and the sizes of the data produced by a resource when the task is completed are the same for all tasks and equal to 1000 bytes. The resources use a space-sharing policy, and their maximum acceptable task workload is taken to be larger than the task workload produced by any user. The GS users maximum task workload is equal to their corresponding fixed task workload (Table 2).

6.3. Performance metrics

In our simulations we recorded the following metrics:

- the per user percentage of the number of tasks that miss their non-critical deadlines over the total number of tasks the user creates
- the resource use, defined as the total time duration a resource is used for the execution of tasks
- the per user percentage of the number of failed BE tasks over the total number of tasks the user creates. A BE task fails (and is dropped) when it arrives at a GS_BE_EQUAL resource and finds that it cannot be scheduled without violating the delay guarantees given to the already registered GS users
- the percentage of GS tasks that have to wait at the input (backlogged), so that the GS user remains (ρ, σ) constrained, over the total number of GS tasks the user creates
- the average delay of a computation unit (taken to be 1 MI) of a task's workload. A computation unit's delay is defined as the time between its creation as part of a task, and the time this task's results return to the user
- the standard deviation of the computation unit delay
- the deadline fairness metric, defined as the average relative deviation of the demanded task deadlines from the actual task delays:

$$\frac{1}{N}\sum_{i,j}\frac{|D_i^j - \max(Y_i^j, D_i^j)|}{D_i^j},$$

where D_i^j is the deadline and Y_i^j is the actual delay of the jth task of user i. A task's delay is defined as the time that elapses between its creation and the time its execution results return to the user. N is the total number of tasks created by all users. Tasks whose delays are smaller than their respective deadlines do not contribute to this metric.

6.4. Results obtained

A number of simulations were conducted to validate that the proposed framework indeed provides hard delay guarantees to GS users and fairness to BE users. In our simulations we used 3 GS and 2 BE users, 3 single-machine and single-CPU resources, and a meta-scheduler. We examine all the resource scenarios presented in Table 6. The task workloads and inter-arrival times follow a fixed (deterministic) distribution with the values of Tables 2 and 4, respectively. The task inter-arrival times (task generation rates) of user U4 change in many simulation scenarios. Using the previous parameters in our simulations, we observe that the GS users register to one resource; specifically, U1 registers to R1, and U2 and U3 to R2.

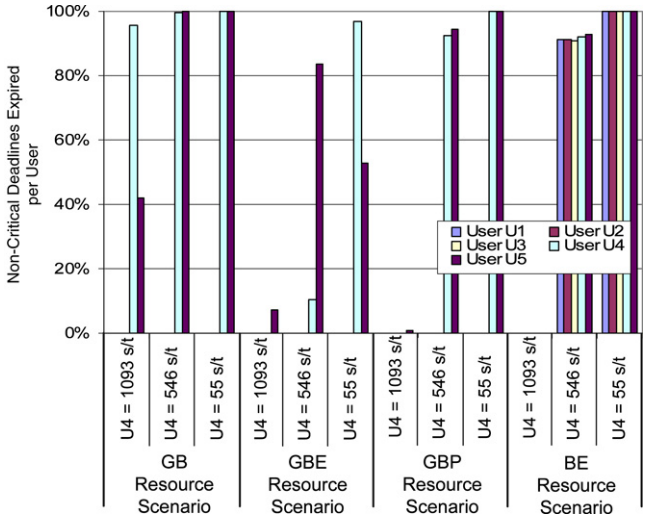

Fig. 4. The per user percentage of the number of tasks that miss their non-critical deadlines, for various resource scenarios and task inter-arrival times (in secs/task) of BE user U4.

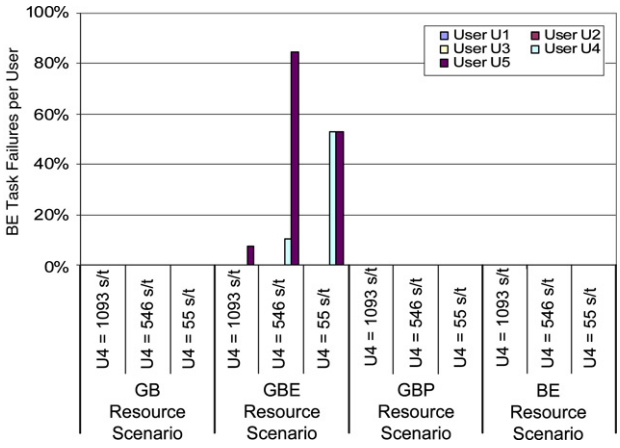

Fig. 5. The per user percentage of the number of failed tasks, for various resource scenarios and task inter-arrival times (in secs/task) of BE user U4.

6.4.1. Guaranteed delay bounds for GS users

Fig. 4 shows that our scheme succeeds in providing hard delay guarantees to the GS users. Fig. 4 presents the per user percentage of tasks that miss their non-critical deadlines, for all resource scenarios (GB, GBE, GBP, BE), and for different values of the task inter-arrival times of BE user U4 (1093, 546, 55 secs/task). First, and most importantly, we observe that in all cases users U1, U2 and U3 do not miss any of their deadlines, verifying that our framework succeeds in providing hard delay guarantees to GS users. Only when the BE resource scenario is used, where GS users are treated as BE users, do users start missing many of their deadlines.

In the GBE and the GBP scenarios (Table 6) fewer BE tasks miss their deadlines; however, in the GBE resource scenario many BE tasks fail (Fig. 5). So the GBP resource scenario, where resource R1 and R2 are used by both GS and BE users but with different priorities, seems to be the best in terms of the number of tasks successfully scheduled without missing their deadlines. This is because in this resource scenario better use of the available resources is achieved, by multiplexing GS and BE users (with different priorities) on resources R1 and R2.

In Fig. 6 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

Fig. 6. The resource use, for various resource scenarios and task inter-arrival times of BE user U4.

Fig. 7. The standard deviation of the resource use, for various resource scenarios and task inter-arrival times of BE user U4.

tasks and as a result the use of resource R3 is smaller. Finally, in Fig. 7 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 in the GBP scenario. This indicates that the GBP scenario makes more efficient and uniform use of the available resources than the other scenarios.

We also observed that the total task delays of the GS users have very small deviations, compared to the total task delay deviations of the BE users. The tasks of GS user U1 have a smaller total task delay than the equally-sized tasks of BE users U4 and U5. Also, the total task delays of users U4 and U5 are larger in the GS scenario than in the other scenarios, because in the first case only one resource is available for BE users. The total task delays and the corresponding standard deviations for all the users increase, as expected, when the task inter-arrival rate of user U4 increases. Also, because the generated GS user tasks conform to the agreed (ρ, σ) constraints, none of their tasks is ever backlogged at the input of the Grid.

Finally, we conducted a number of experiments using resources with multiple CPUs and measured the per user percentage of tasks that miss their non-critical deadlines. The results were obtained for the case where resources R1, R2 and R3 have 30, 20 and 10 CPUs, respectively, each of computational capacity equal to 26000 MIPS. In the experiments conducted the delay bounds given to the GS users and the deadlines of all users tasks are calculated based on (10). We observe that in all cases the GS users (U1, U2, U3) do not miss their deadlines. So even when multi-CPU resources are used, our framework again succeeds in providing hard delay guarantees to GS users.

Fig. 8. The per user percentage of the number of tasks that miss their non-critical deadlines under the GBP resource scenario, for various task inter-arrival times (secs/task) of GS users U1, U2, U3 (Fixed, Un.5, Un.20, Un.50, Mixed) and of BE user U4.

6.4.2. (ρ, σ) constraints violation

Next, we look at the behavior of the proposed schemes when the GS users violate their (ρ, σ) agreements with the Grid Network. In theory when a GS user starts violating his (ρ, σ) constraints, then either his GS tasks are backlogged until condition (3) becomes valid again, or some of his GS tasks are handled as BE tasks because there is no GS resource that can complete them before their deadline expires. In the first case we expect the total task delay of GS tasks to increase. In the second case we expect many GS tasks to miss their deadlines.

In addition to the fixed (deterministic) distribution, we also considered in our simulations the following distributions for the task inter-arrival times of GS users U1, U2 and U3:

- uniform distribution with minimum value 5%, 20%, or 50% smaller than the corresponding fixed value of Table 2 and maximum value 5%, 20%, or 50% larger than the corresponding fixed value of Table 2, to be referred as Un.5, Un.20, Un.50 distributions, respectively.
- uniform distribution with minimum value 50% smaller than the corresponding fixed value of Table 2 and maximum value 20% larger than the corresponding fixed value of Table 2, to be referred as Mixed distribution.

Fig. 8 shows the per user percentage of tasks that miss their non-critical deadline in the GBP resource scenario (Table 6), under various distribution scenarios (Fixed, Un.5, Un.20, Un.50, Mixed), and for mean inter-arrival times of 1093, 546 and 55 secs/task for user U4. We observe that in the Un.5 and Un.20 distribution scenarios the number of non-critical deadlines expired for all the users is almost the same as that of the original fixed distribution scenario. However, we observe a small rise in the number of backlogged GS tasks and a corresponding increase in their average delay. On the other hand, in the Un.50 and the Mixed distribution scenarios, the number of non-critical deadlines expired increases, and, more importantly, the GS users miss many of their non-critical deadlines. This happens because these GS tasks cannot be served by any GS capable resource before their deadline expires, and are consequently handled as BE tasks, without delay guarantees. From these results we conclude that as long as the GS users respect their (ρ, σ) constraints, even with small deviations, our QoS framework succeeds in providing them with hard delay guarantees.

Fig. 9 shows the per user percentage of tasks that miss their non-critical deadline for the GBP and BE resource scenarios, under the Mixed distribution scenario. We observe that even though GS users miss many of their non-critical deadlines, the use of the proposed framework (under the GBP resource scenario) reduces