RESOURCES CONFIGURATIONS FOR PROVIDING QUALITY OF SERVICE IN GRID COMPUTING

Panagiotis C. Kokkinos and Emmanouel A. Varvarigos
Department of Computer Engineering and Informatics, University of Patras and Research Academic Computer Technology Institute, Patras, Greece
kokkinop@ceid.upatras.gr
manos@ceid.upatras.gr

Abstract Future Grid Networks should be able to provide Quality of Service (QoS) guarantees to their users. In this work we examine the way Grid resources should be configured so as to provide deterministic delay guarantees to Guaranteed Service (GS) users and fairness to Best Effort (BE) users. The resources are partitioned in groups that serve GS users only, or BE users only, or both types of users with different priorities. Furthermore, the GS users are registered to the resources either statically or dynamically, while both single and multi-Cpu resources are examined. Finally the proposed resource configurations for providing QoS are implemented in the GridSim environment and a number simulations are executed. Our results indicate that the allocation of resources to both types of users, with different priorities, results in fewer deadlines missed and better resources utilization. Finally benefits can be derived from the dynamic registration of GS users to the resources.

Keywords: computational resources configurations, QoS scheduling, dynamic and static registration, single and multi-Cpu resources

1. Introduction

Today's Grids provide only a best effort service to the users and their tasks, which is insufficient if a Grid is to be used for real world commercial applications. Users can be categorized in two types. Some users are relatively insensitive to the performance they receive from the Grid. Even though these Best Effort (BE) users do not require performance bounds, it is desirable for the Grid to allocate resources to them in a fair way. Besides BE users, Grids also 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 examine the way grid resources should be configured so as to provide deterministic delay guarantees to GS users and fairness to BE users. For the GS users, the objective is to guarantee an upper bound on the maximum delay of the submitted tasks. For BE users, we aim at attaining a fair scheduling procedure. The resources are configured to serve GS users only, or BE users only, or both types of users with different priorities. Furthermore the GS users are registered either statically or dynamically to the resources, while both single and multi-Cpu resources are examined. Finally the proposed resources configurations are implemented in the GridSim environment and a number simulations are executed. Our results indicate that the allocation of resources to both types of users, but with different priority, results in fewer deadlines missed and better resource utilization. Also benefits can be obtained by dynamically varying the configurations of the resources.

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. Furthermore lately, a number of scheduling schemes that are specific to Grids have also been proposed [4], [5] and [3].

QoS in Data Networks has been extensively studied. The Internet Engineering Task Force (IETF) has proposed the Integrated Services (IntServ) [7] and the Differentiated Services (DiffServ) architectures [8] 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) [9] and the Grid QoS Management (G-QoSM) architecture [11]. These works propose QoS schemes for Grids that take into account the network, computational and storage resources. 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.

In this work we build on the QoS framework for Grid computing first presented in [13]. This framework provides hard QoS, in terms of delay bounds guarantees given to each user, 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 [10]). Under the QoS framework presented in [13] we propose and evaluate the categorization of computational resources so as to serve either GS, or BE, or both types of users and examine static and dynamic registration of GS users to the resources.

The remainder of the paper is organized as follows. In section 2 we report previous work. In section 3 we describe how the resources can be configured in order to provide QoS guarantees. In section 4 we present the simulation environment, the parameters and the results of our simulations. Finally, conclusions are presented in section 5.

2. Grid QoS Framework Description

We consider a Grid consisting of a number of users and a number of 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 the priority they give to each type. Our framework gives delay bound guarantees to GS users and fairness to BE users. We assume that a task executing at a resource is non-divisible and non-interruptible (non-preemptable) and thath the local scheduler of every resource applies Weighted Fair Queuing (WFQ) to the queued GS tasks. We initially describe our framework assuming that every machine has one CPU, and later extend it to the multi-CPU machine case.

The GS users are leaky bucket constrained, and so they follow a (ρ, σ) constrained task generation pattern. A GS user must first register to a resource, before they can actually use it, while he can register to a number of resources. During this registration the specific values of his task generation pattern is agreed (Figure 1).

Based on [13] in order for a GS user i to register to a resource r the conditions (1) and (2) must be met:

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

and

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

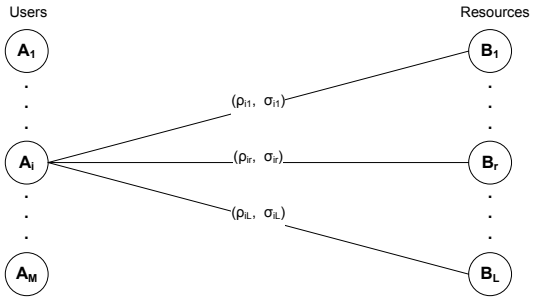

Figure 1. The (ρ, σ) constrained GS users in the Grid.

The ρ_{ir} is the long term task generation rate of the GS user. The 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. Furthermore C_r is the computing capacity of resource r, $N_r(t)$ 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. Also J_{ir}^{max} is that the maximum task length the GS user i will ever send to the resource r, while J_r^{max} is the resource's r maximum acceptable task length. Finally the condition (1) must hold both for a new and for the already registered, at a resource, GS users. 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.

When a GS user i creates a task j, he searches for the best suitable resource r, to which it has already registered. Specifically he first checks whether the agreed (ρ_{ir}, σ_{ir}) constraints are satisfied. Specifically we denote by $J_{ir}(t), i = 1, 2, \cdots, N$ the total tasks workload 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:

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

If one or more new GS user's tasks invalidate (3), then the GS user must locally withhold these tasks for a time period T_{ij} until (3) becomes valid again. This condition ensures that the user remains leaky bucket constrained. Furthermore the user checks if the task's j length I_i^j is not exceed the one agreed:

$$I_i^j \leq J_{ir}^{max}. \qquad (4)$$

After these checks in order for the task j to be submitted for execution to the resource r the task must not miss its deadline D_i^j . In [13] we showed that if the conditions (1) and (3) hold then the delay a task will incur from the time it

is submitted by the user, until it finishes its execution at a selected resource is at most:

$$\frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_{r}^{max}}{C_r}, \quad (5)$$

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} for transferring the task to the selected resource and the total time T_{ir} the GS user withholds the task in its local queue (Figure **??**). So the delay bound D_i^j equals to:

$$D_i^j \leq T_{ir} + d_{ir} + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_{r}^{max}}{C_r}. \quad (6)$$

Finally when the GS user hasn't any more tasks to send to the Grid, he can either do nothing or he can unregister from his registered resources. In the latter, dynamic, case the other GS users are infromed for the user's unregistration and they can try to register to these resources.

On the other hand for BE users, we aim at attaining a fair scheduling procedure. Based on [2]and [12] we proposed in [13] the Fair User and Task Scheduling (FUTS) algorithm, which tries to be fair both for the users and for the tasks.

2.1 Resources Configurations

We distinguishes 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_EQ. 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 Eq. (1) holds for all the registered users. 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_PR. resources again handle tasks originating from both GS and BE users, but tasks are not handled 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 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. A GS_BE_PR.

resource is characterized as preemptive if upon the arrival of a GS task, a BS task currently under execution is paused and replaced by the new GS task; otherwise the GS_BE_PR. resource is characterized as non-preemptive. Also, a BE task is scheduled to a GS_BE_EQ. or GS_BE_PR. resource only when its size is smaller than the resource's maximum acceptable task size. Finally when a GS_BE_PR. non-preemptive resource is used, the delay bound for GS tasks of Eq. (6), becomes:

$$D_i^j \leq T_{ir} + d_{ir} + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r} + R_r, \quad (7)$$

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

$$R_r \leq \frac{J_r^{max}}{C_r}. \quad (8)$$

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

Furthermore 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:

$$C'_r = \sum_{j=1}^{M} C_{rj}, \quad (9)$$

where C_{rj} is the computational capacity of machine j, and M is the total number of machines (CPUs) in the resource. However, in the multi-machine resources case the C_r used in Eqs. (1) and (6) is not always equal with the 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 the resource, C_r in Eq. (1) has to be replaced by C'_r, yielding

$$\rho_{ir} \leq g_{ir}(t) = \frac{w_{ir} \cdot \sum_{j=1}^{M} C_{rj}}{\sum_{k=1}^{N_r(t)+1} w_{kr}} \quad (10)$$

Since tasks are non-divisible, the resource cannot use its total computational capacity to process a job. 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 III has to be replaced by C_r^{min}. For example, Eq. (6) becomes:

$$D_i^j \leq T_{ir} + d_{ir} + \frac{\sigma_{ir}}{g_{ir}} + \frac{J_{ir}^{max}}{g_{ir}} + \frac{J_r^{max}}{C_r^{min}}. \quad (11)$$

3. Simulation Results

In our simulations we used realistic parameters based on [14]. Specifically in [14] a thorough analysis of the tasks inter-arrival times, the waiting times at the queues, 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. So based on these results in our experiment we used 3 GS users and 2 BE. The GS users had the characteristics presented in Table 1 and the BE users the characteristics presented in Table 2.

Table 1. GS user task lengths and inter-arrival times used in our simulations

User	*Characteristic*	*Value*
U1	Task Length	10000 MI
U2	Task Length	1700 MI
U3	Task Length	10 MI
U1	Task Inter-arrival	10 secs
U2	Task Inter-arrival	60 secs
U3	Task Inter-arrival	110 secs
U1	ρ	$10000/10 = 1000$ MIPS
U2	ρ	$1700/60 \approx 30$ MIPS
U3	ρ	$10/110 \approx 0.1$, we used 1 MIPS
U1	σ	50000 MI
U2	σ	8500 MI
U3	σ	50 MI

Table 2. BE users characteristics

User	*Characteristic*	*Value*
U4	Task Length	10000 MI
U5	Task Length	10000 MI
U4	Task Inter-arrival	100, 80, 60, 40, 20, 10, 1 secs/task
	or	or
	Task Generation Rate	$0.01, 0.0125, 0.017,$ $0.025, 0.05, 0.01, 1$ tasks/sec
U5	Task Inter-arrival	100 tasks/sec
	or	or
	Task Generation Rate	0.01 tasks/sec

In our simulations we use 3 clusters/resources. For the single-CPU case the resource capacity of R1 is 1015 MIPS, of R2 is 680 MIPS and of R3 is 340

MIPS. For the multi-CPU case all the CPUs have capacity equal to 34 MIPS and R1 has 30 CPUs, R2 has 20 and R3 has 10. Furthermore in our simulations we used the resource type scenarios presented in Table 3. When the BE resources scenario is used then all the users (U1, U2 ,U3) are BE users with the exact same characteristics as the corresponding GS users (U1, U2 ,U3).

Table 3. Resources scenarios

Scenarios	*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, with a time window equal to 1 second. 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. Also in our simulations all the users, GS and BE, have equal deadlines. Specifically the deadline used in our simulations is the maximum deadline calculated for a GS user, based on (6) or (11) and adding a small overhead in order to take into account the T and d (communication) delays. Furthermore, in our simulations we assume that the resources, the users and the meta-scheduler communicate directly with links of equal bandwidth, while the propagation delay is equal to 0. Finally in the static case all the users stop producing new tasks at the same time, while in the dynamic case the U1 user stops producing new tasks first. This way in the dynamic case U1 unregisters from his registered resources and U2, U3 GS users can try to register to these. In our simulations we recorded the following performance metrics:

- the number of tasks that miss their non-critical deadlines.
- the resource's utilization, defined as the total time a resource is used for the execution of tasks over the total time of the simulation.
- the number of failed tasks.

3.1 Resources Configurations

A number of simulations where conducted to evaluate the different resources configurations. We used the configuration scenarios presented in Table 3. In our simulations we used 5 users (3 GS and 2 BE), 3 resources and a meta-scheduler. Finally using these parameters of Table 1 and Table 2 U1 registers to R1, while U2 and U3 register to R2.

First in Figure 2 we show that our framework succeeds in providing QoS to the GS users. Figure 2 presents the percentage of the per user number of tasks that miss their non-critical deadlines over the total number of tasks each user creates. This percentage is presented for the various resource scenarios and tasks generation rates of the BE user U4 (0.05, 0.1, and 1 tasks/sec). We observe that in all cases the GS users (U1, U2, U3) do not miss their deadlines. Only when the BE resource 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 3) fewer tasks miss their deadlines, however in the GBE resource scenario many tasks fail (Figure 3). In the GBE resource scenario when a BE task arrives at a GS_BE_EQ. 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 resource scenario seems the best in terms of the number of tasks sucessfully scheduled without missing their deadlines.

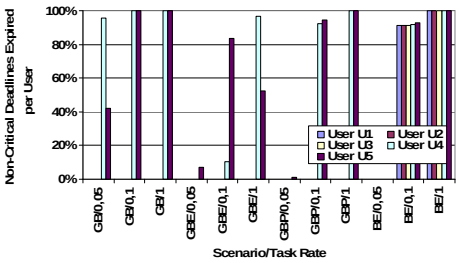

Figure 2. Non-critical deadlines expired per user

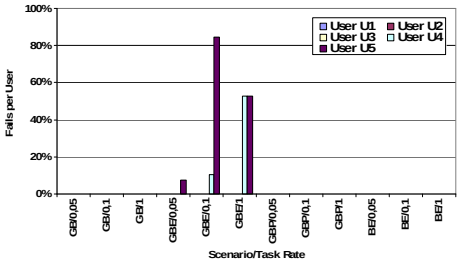

Figure 3. Failed tasks per user

In Figure 4 the utilization per resource is presented again for the same scenarios as before. The resources are utilized more in the GB resource scenario and this is due to resource R3 that handles exclusively BE tasks. In the other resource scenarios, all resources can serve tasks from all users and as a result the resources utilization is smaller. The smallest utilization occurs in the GBE scenario, but this is the result of the failed tasks, so the GBP scenario seems the best in terms of resources utilization (Figure 5). Finally, in Figure 6 the standard deviations of the resources utilization are presented. The standard deviation is high in the GB scenario, where the R3 resource is more utilized than the R1 and R2 resources, while it is very small for the GBP scenario.

3.2 Static against Dynamic Registration

We conducted a number of experiments to evaluate the benefits of the static against the dynamic registration of the GS users to the resources. In our ex-

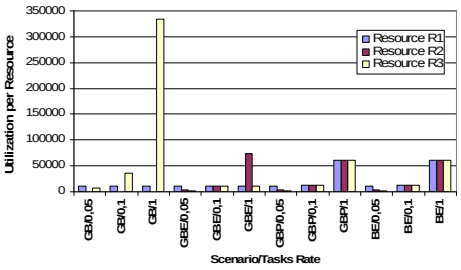

Figure 4. Utilization per resource

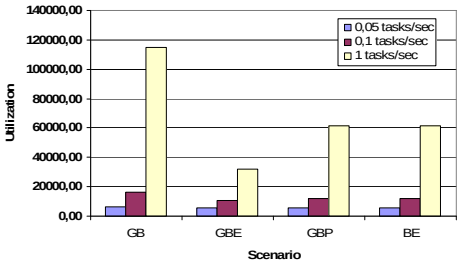

Figure 5. Utilization

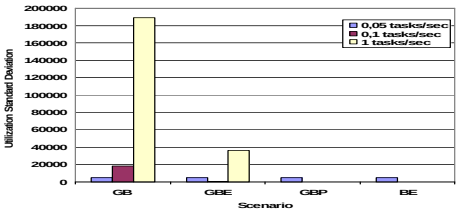

Figure 6. Standard deviation of resource utilization

periments we decreased the number of tasks the U1 generates, this way at a point at time, during an experiment, U1 stops producing new tasks and so R1 becomes available for use from the other GS users. When static registration is used, U2 and U3 users do not take advantage of resource R1, while when dynamic registration is used U1 unregisters from R1 and U2, U3 register to it.

In our experiments we observe the benefits of the dynamic registration mainly in the GBE resource scenario. Then fewer tasks miss their non-critical deadline (Figure 8) and fewer tasks fail (Figure 10), than when static registration is used (Figure 7, Figure 9). This happens because in the dynamic registration case the BE tasks of user U5 can also use the R1 resource, when the U1 unregisters from it. Furthermore in the GBE scenario R1's utilization increases because the BE tasks of U5 are executed there, however the utilization increases and for R2,R3 because of fewer failed tasks. Finally in the GBP scenario the benefits of the dynamic registration are very few, both in terms of the number of tasks missing their deadlines and in terms of the resources utilizations.

3.3 Multi-Cpu Resources

We conducted a number of experiments using the multi-Cpu resources. Two sets of experiments were conducted. In the first the delay bounds given to the GS users and the deadlines of all the users tasks are calculated based on (6),

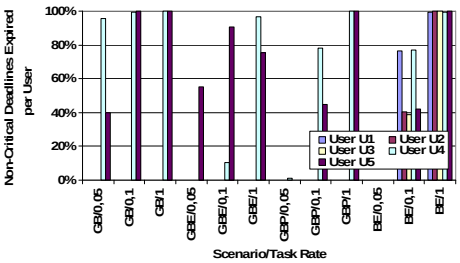

Figure 7. Non-critical deadlines expired per user in the static case

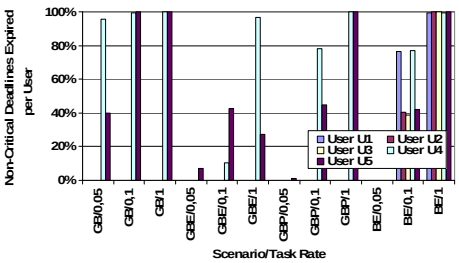

Figure 8. Non-critical deadlines expired per user in the dynamic case

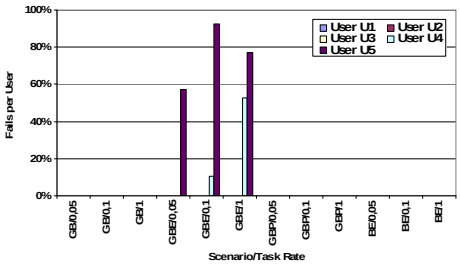

Figure 9. Failed tasks per user in the static case

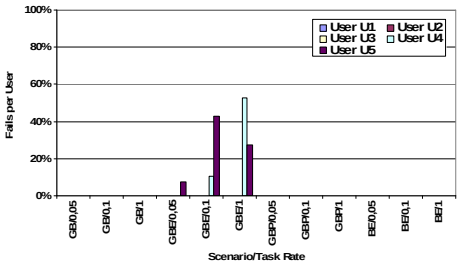

Figure 10. Failed tasks per user in the dynamic case

while in the second set based on (11). In Figure 11 and Figure 12 we show that in case condition (6) is used then most of the tasks the users create lose their deadlines. So in the multi-Cpu case the (11) is the correct formulation of the delay bound given to the GS users.

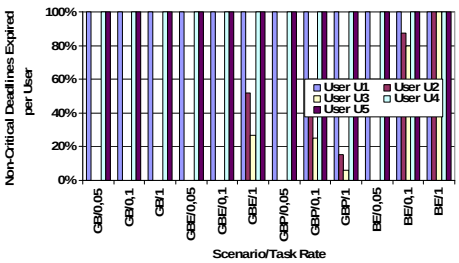

Figure 11. Non-critical deadlines expired per user using condition (6)

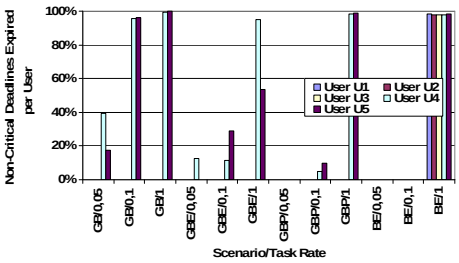

Figure 12. Non-critical deadlines expired per user using condition (11)

4. Conclusions

In this work we examine the way Grid resources should be configured so as to provide deterministic delay guarantees to Guaranteed Service (GS) users and fairness to Best Effort (BE) users. A number of simulations are conducted, which indicate that the resources which server both types of users provide better results, in terms of deadlines missed and resources utilizations. Finally we show that benefits exist from the dynamic registration of the GS users to the resources.

References

[1] A. Parekh, R. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: the single-node case*, IEEE/ACM TON, 1993, Vol.1, No.3, pp. 344-357

[2] B. Briscoe, *Flow Rate Fairness: Dismantling a Religion*, In SIGCOMM Comput. Commun. Rev., 2007, Vol. 37, No. 2, pp. 63-74.

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

[4] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, R. Freund, *A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems.*, Journal of Parallel and Distributed Computing, 2001, Vol. 61, No. 6, pp. 810-837.

[5] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov and A. and Shokurov, *Comparison of Scheduling Heuristics for Grid Resource Broker*, Proc. of the 5th Mexican Int. Conf. in Computer Science (Enc'04), 2004, Vol. 00, pp. 388-392.

[6] A. Demers, S. Keshav and S. Shenker, *Analysis and simulation of a fair queuing algorithm*, Proc. of SIGCOMM, 1989, pp. 1-12.

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

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

[9] I. Foster and C. Kesselman and C. Lee and R. Lindell and K. Nahrstedt and A. Roy, *A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation*, Proc. of IWQOS, 1999, pp. 27-36.

[10] H. Chu, *CPU Service Classes: a Soft Real Time Framework for Multimedia Applications*, University of Illinois at Urbana-Champaign, Technical Report, 1999.

[11] R. Al-Ali, O. Rana, D. Walker, S. Jha and S. Sohail, *G- QoSM: Grid Service Discovery using QoS Properties*, Journal of Computing and Informatics, 2002, Vol. 21, No. 4, pp. 363-382.

[12] N. Doulamis, A. Doulamis, E. Varvarigos, and T. Varvarigou, *Fair QoS Resource Management in Grids*, to appear in IEEE Transactions on Parallel and Distributed Systems.

[13] P. Kokkinos, E. Varvarigos, *A Quality of Service Framework for Grid Computing*, submitted to 8th IEEE/ACM Int. Conf. on Grid Computing.

[14] M. Oikonomakos, K. Christodoulopoulos, E. Varvarigos *Profiling Computation Jobs in Grid Systems*, to be appear on CCGrid, Brazil, 2007.