Fair Execution Time Estimation Scheduling in Computational Grids

Eleni Dafouli, Panagiotis Kokkinos, Emmanouel A. Varvarigos

Abstract We propose a fair scheduling algorithm for Computational Grids, called Fair Execution Time Estimation (FETE) algorithm. FETE assigns a task to the computation resource that minimizes what we call its fair execution time estimation. The fair execution time of a task on a certain resource is an estimation of the time by which a task will be executed on the resource, assuming it gets a fair share of the resource's computational power. Though space-shared scheduling is used in practice, the estimates of the fair execution times are obtained assuming that a time-sharing discipline is used. We experimentally evaluate the proposed algorithm and observe that it outperforms other known scheduling algorithms. We also propose a version of FETE, called Simple FETE (SFETE), which requires no a-priori knowledge of the tasks workload and in most cases has similar performance to that of FETE.

Key words: grids, scheduling, fairness, task workload

Eleni Dafouli
Department of Computer Engineering and Informatics, University of Patras,
e-mail: dafouli@ceid.upatras.gr

Panagiotis Kokkinos, Emmanouel A. Varvarigos
Research Academic Computer Technology Institute, Patras, Greece,
e-mail: kokkinop, manos @ceid.upatras.gr

1 Introduction

Grids consist of geographically distributed and heterogeneous communication, computation and storage resources that may belong to different administrative domains, but can be shared among users. Since the sharing of resources is the "raison d' etre" of Grids, fairness is a concept that is inherent in Grid scheduling, and has been previously ignored. Fairness can be defined in a number of different ways, but an intuitive notion of fairness is that a task submitted to the Grid, is entitled to as much use of the resources as any other task. When the Grid serves different classes of users (e.g., users willing to pay different prices for the service they receive) the notion of fairness depends on the class of the user, with users belonging to the same class having "equal" access to the resources.

In this work we propose a fair scheduling algorithm for Computational Grids, which we call the Fair Execution Time Estimation (FETE) algorithm. FETE assigns a task to the computation resource that minimizes what we call its fair execution time estimation. This estimation is obtained assuming that the task gets a fair share of the resource's computational power. Though space-shared scheduling is used in the actual resource, the estimates of the fair execution times are found assuming time-sharing is used. We also propose a version of FETE, called Simple FETE (SFETE), which is a good approximation of FETE, and does not require a-priori knowledge (or estimates) of the task workloads. FETE and SFETE can be implemented both in a centralized and in distributed way. We perform an extensive set of experiments using the GridSim [7] simulator and show that FETE outperforms other known scheduling algorithms with respect to performance and fairness related metrics. The improvements obtained by using FETE are particularly important when the load in the Grid, in terms of tasks submitted, increases. Finally, it is observed that the FETE and the SFETE algorithms give similar results, and so in almost every case the latter version is preferable, since it has no need for the a-priori knowledge of the task workloads. These results strengthen our belief that SFETE can in fact be incorporated in a production Grid Middleware.

The remainder of the paper is organized as follows. In Section 2 we report on previous work. In Section 3 we describe the Grid environment used. In Section 4 we present the Fair Execution Time Estimation (FETE) and in Section 5 the Simple FETE (SFETE) scheduling algorithms. Performance results are presented in Section 6. Finally, conclusions and directions for future work are presented in Section 7.

2 Previous Work

A number of scheduling algorithms have been proposed so far, both for single- and for multi-processor systems, some of which have also been adapted for use in the Grid environment. Lately a number of scheduling schemes that are specific to Grids have also been proposed. [1][2][3][4][11] present centralized, hierarchical and distributed scheduling schemes for Grids. Most of the scheduling algorithms proposed so far try to minimize the total average task delay [3] and maximize resource utilization, while several other performance metrics are used. In [8] and in [9] scheduling algorithms that support deadline and budget constraints are proposed and implemented.

The fair scheduling of packets in Data networks is a concept quite well studied [5][6]. On the other hand fair scheduling algorithms for Grids have received relatively little attention until now. In [10] a fair packet-by-packet algorithm for the joint allocation of processing and bandwidth resources is proposed. In [12] game theory is used to prove that a strong community control is required to achieve acceptable performance in Grids, by comparing centralized and distributed fair scheduling algorithms. In [13] the authors propose a resource allocation scheme based on fair resource sharing in hierarchical Virtual Organizations (VOs). Simulation results show that the proposed scheme provides greater fairness than other schemes, as well as better performance. In [14] three different fair scheduling algorithms are proposed and evaluated in a centralized scheduling environment.

3 Grid Environment

We consider a Grid environment consisting of a number of users and a number of computation resources. By the term user we do not necessarily mean an individual user, but also (and probably more appropriately) a Virtual Organization (VO), or a single application, using the Grid infrastructure. Also a computation resource can be a cluster, a parallel computer or a Grid site.

Users generate atomic (undivisible and non-preemptable) tasks and every task i has workload w_i and non-critical deadline D_i. By the term "non-critical" we mean that if the deadline expires, the corresponding task remains in the system until completion, but it is recorded as a deadline miss. Upon creating a new task, the user sends the task characteristics to the central scheduler, in the form of a task request. The central scheduler works "offline" or "online". In the former case the central scheduler receives task requests by several users and stores them in a local queue. Periodically the scheduler orders the queued task requests (using an ordering policy) and assigns them to resources (using an assignment policy). In the "online" mode the central scheduler assigns tasks to resources immediately after the arrival of the corresponding task requests. Each resource j contains a number CPUs, of total computational capacity equal to C_j and uses a space-sharing policy. Tasks are served by the CPUs of a resource based on the order they arrive to it. At any time t there are $N_j(t)$ tasks in resource's j local queue or under execution in its CPUs.

The FETE and Simple FETE algorithms can work both in "offline" and "online" mode. However, these algorithms are presented, in this paper, in their "offline" mode and evaluated along with other "offline" algorithms. Finally, the FETE and Simple FETE algorithms do not use the task deadlines in their operation.

4 Fair Execution Time Estimation Algorithm

The Fair Execution Time Estimation (FETE) scheduling algorithm assigns task i to resource j that provides the minimum fair execution time X_{ij}. The fair execution time X_{ij} is an estimation of the time required for task i to be executed on resource j, assuming it gets a fair share of the resource's computational power. By fair share we mean that each time t the task gets a portion:

$$\frac{1}{N_j(t)+1},$$

of resource's j computational capacity C_j. That is, the estimates of the fair execution times are obtained assuming a time-sharing discipline, though space-shared scheduling is used in the actual resource. The parameter $N_j(t)$ is the total number of tasks already assigned (queued or executed) to resource j at the time t the assignment decision is made. The fair share of the resource's capacity each task gets changes with time, since $N_j(t)$ also changes with time, increasing by 1 every time a new task is assigned to resource j and decreasing by 1 each time a task completes service at resource j. For this reason, during the calculation of the fair execution time X_{ij} of task i on resource j, the fair execution time estimations of the tasks already assigned to resource j should also be taken into consideration.

In the example of Figure 1, we present two resources A and B that have the same computation capacity. At time 0 both resources have the same number of tasks N assigned to them, however the tasks assigned to resource B have smaller workloads. The first task completes its execution in resource A at time t_1^A, while in resource B at t_1^B, and $t_1^A > t_1^B$. Similarly, for the second task we have $t_2^A > t_2^B$, and so on. During the time periods $[0, t^A]$ and $[0, t^B]$, we assume that there are no new arrivals of tasks, so the last task, in both resources, utilizes the whole computational capacity of the corresponding resource. The last task in resource A finishes its execution at time t^A, while in resource B at time t^B, where $t^A > t^B$. These times, t^A and t^B, are also the fair execution time estimations of the corresponding tasks. We see that the fair execution time of a task depends not only on its workload, resource capacity and number of tasks assigned to resource, but also on the workloads of the other tasks.

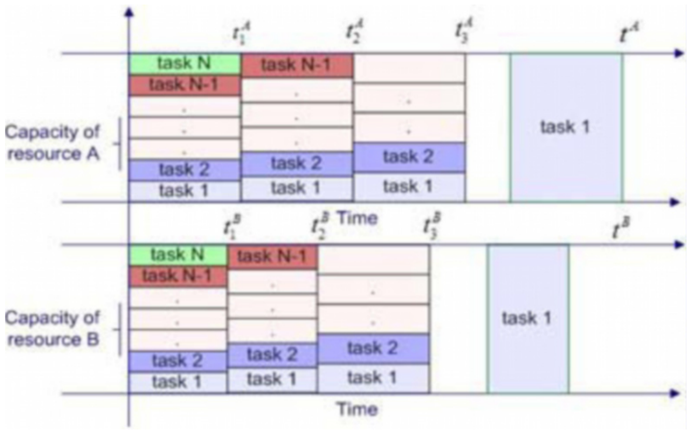


Fig. 1 Fair execution time estimation example.

For the calculation of the fair execution time X_{ij} of task i on resource j, we consider the fair execution time estimations of the tasks already assigned to resource j. However, in this calculation it is not possible to also consider tasks that may be assigned to the resource in the future, which would change the fair share of existing tasks. he fair execution time estimations of the tasks are calculated only once and are not re-estimated when new tasks arrive at the resource. So, the calculation of the fair execution time of task i on resource j is just an estimate and not the actual time that the task would complete its execution, even if it were executed using an ideal time-sharing (processor sharing) scheme.

The pseudocode of the centralized and "offline" implementation of FETE is presented in Algorithm 1. We assume that at the time a task arrives at the resource, there are N tasks assigned to it, having fair execution times t_n, $n = 1, \cdots, N$. Without loss of generality we also assume that $t_{n-1} > t_n$ for all n. A task i has workload equal to w_i and its remaining workload (defined in Algorithm 1) is denoted by $\widehat{w}_i$. In the end FETE algorithm assigns task i to resource j that provides the minimum fair execution time estimation X_{ij}.

5 Simple Fair Execution Time Estimation Algorithm

The FETE algorithm requires the a-priori knowledge of the task workloads for obtaining the fair execution time estimations. However, the task workloads, in practice, are often not known and may be hard to estimate. For this reason

Algorithm 1 Fair Execution Time Estimation - FETE

```
1: for each task $i$ queued in the scheduler's ordered list do
2:   for each resource j in the Grid do
3:     Set $\widehat{w}_i = w_i$
4:     Estimate the fair execution time $X_{ij}$ :
5:     Set $t = 0$, $n = N$ and $X_{ij} = 0$
6:     Estimate task's $i$ temp fair execution time $X_{ij}$ assuming it gets computational capacity $\frac{C_j}{N_j(t)+1}$ on resource $j$: $X_{ij} = \widehat{w}_i \cdot \frac{N_j(t)+1}{C_j}$
7:     if $X_{ij} < t_n$ then
8:       $X_{ij} = X_{ij} + t$
9:     else
10:      $\widehat{w}_i = \widehat{w}_i - \frac{C_j}{N_j(t)+1} \cdot (t_n - t)$
11:      $X_{ij} = X_{ij} + t_n$
12:      Set $t = t_n$ and $n = n - 1$
13:      Goto(6)
14:    end if
15:  end for
16:  Assign task $i$ to resource $j$ that gives the minimum fair execution time $X_{ij}$
17:  $N_j(t) = N_j(t) + 1$
18:  Send the scheduling decision to the user of task $i$
19: end for
```

we propose a version of FETE, called Simple FETE (SFETE), which requires no a-priori knowledge of the tasks workload.

The SFETE assigns task i to resource j that provides the minimum simple fair execution time $\widehat{X}_{ij}$. The simple fair execution time $\widehat{X}_{ij}$ is an estimation of the time by which task i will be executed on resource j, assuming it gets a fair share of the resource's computational power, without taking into account the fair execution times of the other tasks already assigned to the resource (Figure 2). So, when the SFETE is employed in the example of Figure 1, then the simple fair execution time estimations of the tasks are equal ($t^A = t^B$).

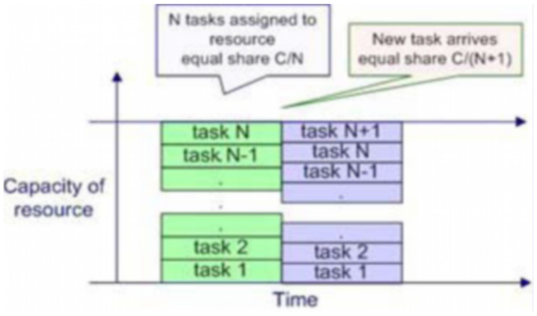


Fig. 2 Simple execution time estimation example.

The simple fair execution time $\widehat{X}_{ij}$ of task i on resource j, is defined as

$$\widehat{X}_{ij} = \frac{N_j+1}{C_j},$$

where C_j is the computational capacity of resource j and N_j is the number of tasks in the resource's queue, including the one being processed. It is once again important to note that the calculation of the simple fair execution time $\widehat{X}_{ij}$ of task i on resource j is only an estimate. New tasks may be sent to resource j, or existing tasks may complete their execution. This way the fair share of the computation capacity of the tasks already assigned to the resource changes, however their simple fair execution time estimations are not re-estimated.

The pseudocode of the centralized and "offline" implementation of the SFETE algorithm is presented in Algorithm 2.

Algorithm 2 Simple Fair Execution Time Estimation - SFETE

```
1: for each task $i$ queued in the scheduler's ordered list do
2:    for each resource j in the Grid do
3:       Estimate the fair execution time: $\widehat{X}_{ij} = \frac{N_j+1}{C_j}$
4:    end for
5:    Assign task $i$ to resource $j$ that gives the minimum simple fair execution time $\widehat{X}_{ij}$
6:    $N_j = N_j + 1$
7:    Send the scheduling decision to the user of task $i$
8: end for
```

6 Performance Results

6.1 Simulation Environment

The proposed FETE and SFETE scheduling algorithms, along with other algorithms used for comparison (Table 1) were implemented and evaluated in the GridSim [7] simulator. The scheduling algorithms were implemented in a centralized and "offline" manner. FETE and Simple FETE algorithms were compared against some well-known algorithms presented in . In the Earliest Deadline First (EDF) ordering policy the task with the most imminent deadline is scheduled first, while in the Least Length First (LLF) ordering policy, the task with the smallest workload is given priority. The Earliest Completion Time (ECT) assignment policy, assigns a task to the resource where the task will finish its execution earlier. Also, the FETE and the simple FETE algorithms use the First Come First Serve (FCFS) ordering policy, where tasks are processed (assigned to resources) in the order they arrive to the scheduler.

All the scheduling algorithms were evaluated in a Uniform resource scenario, in which all resources have the same characteristics (number of CPU

Table 1 The scheduling algorithms compared with the FETE and the SFETE.

Algorithm	Ordering policy	Assignment policy
FCFS/ECT	First Come First Served (FCFS)	Earliest Completion Time (ECT)
EDF/ECT	Earliest Deadline First (EDF)	Earliest Completion Time (ECT)
LLF/ECT	Least Length First (LLF)	Earliest Completion Time (ECT)

and capacity) and in a non-Uniform resource scenario, in which the resources have different characteristics. The total computational capacity of the resources in both scenarios was the same. The tasks characteristics are defined probabilistically and the users task submission rate follows an exponential distribution, whose mean takes the following values: 12, 20, 25, 33, 40, 50, 55, 60, 65, 70 tasks/sec. Finally, in our simulations we assume that the communication delays are negligible compared to the execution time of the tasks, which is the case in Computational Grids.

6.2 Simulation Metrics

The algorithms are evaluated using the following metrics:

- Average Task Delay: The average of the delays of the tasks (task Delay = task Finish Time - task Creation Time).
- Task Delay Standard Deviation: The standard deviation of the task delays.
- Average Excess Time: The average time by which a task misses its non-critical deadline (task Excess Time = task Finish Time - task Deadline Expiration).
- Excess Time Standard Deviation: The standard deviation of the time by which the tasks miss their non-critical deadlines.
- Deadlines Missed: The number of tasks that miss their non-critical deadlines.

6.3 Simulation Results

For all the scheduling algorithms examined the average task delay increases as a function of the task submission rate (Figure 3). Specifically, for light load all the algorithms have similar behavior, however, when the task submission rate increases the FETE algorithms (FETE and SFETE) achieve smaller average task delay. This happens because the proposed algorithms treat the tasks and utilize the resources in a more fair manner, something that becomes more evident as the task load increases. We also observe that the FETE algorithms result in smaller task delay standard deviation than the other algorithms (Table 1). These results where confirmed both for the Uniform and for the non-Uniform resource scenario.

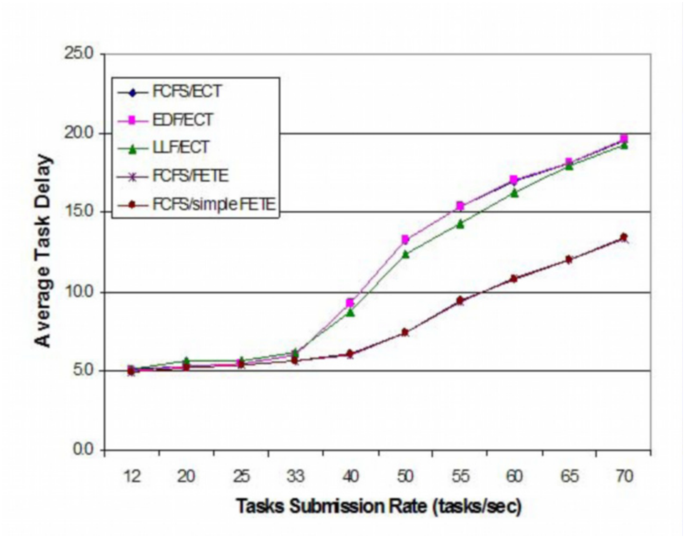
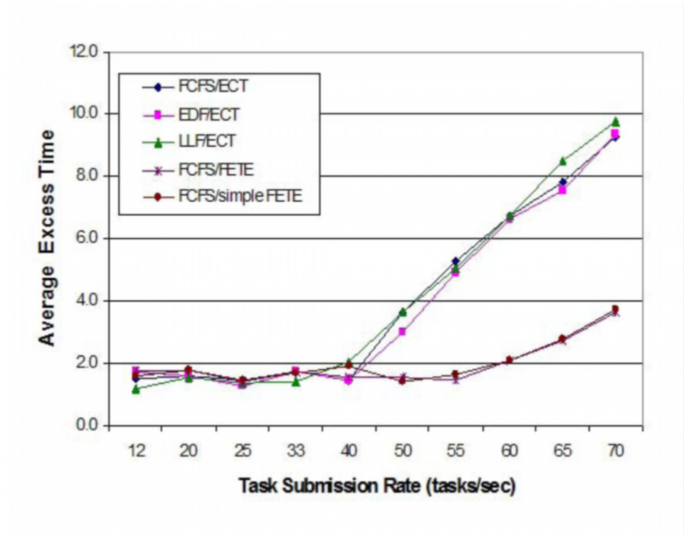


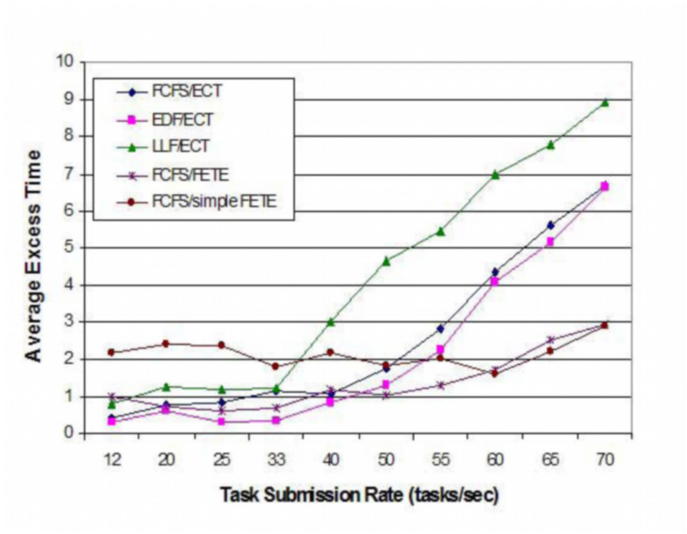
Fig. 3 Average task delay versus task submission rate in the Uniform resource scenario.

Figure 4 illustrates that the average excess time increases as a function of the task submission rate, for both resource scenarios. In the Uniform resource scenario (Figure 4.a) the increase is smaller when the FETE algorithms (FETE and SFETE) are used, meaning that the times by which the tasks miss their deadlines are also smaller. In the non-Uniform resource scenario (Figure 4.b) and for small task submission rates, the SFETE algorithm's performance is worse than that of the FETE and of the other algorithms examined. Next, as the task submission rate increases SFETE overpowers the other algorithms, whose performance deteriorates, while SFETE's remains almost constant.

SFETE does not have any knowledge of the task workloads and indirectly assumes a constant value for all the queued tasks fair execution times. On the other hand the FETE algorithm estimates more accurately the queued tasks fair execution times, whose values, however, are quite different due to the non-uniformity of the resources. When the submission rate increases, the number of tasks in the Grid also increases and the queued tasks fair execution times (estimated by FETE) approach on average a constant value. This is in accordance with the Law of Large Numbers and it is confirmed by Figure 4.b. Specifically, in Figure 4.b the average excess time achieved by the SFETE is almost constant and only increases in the very end when the Grid environment is almost saturated by the large number of tasks. On the other hand the average excess time of the FETE increases reaching that



(a)



(b)

Fig. 4 Average excess time versus task submission rate, (a) in the Uniform resource scenario, (b) in the Non-Uniform resource scenario.

of the SFETE. Similar results were observed for the excess time standard deviation, for both resource scenarios.

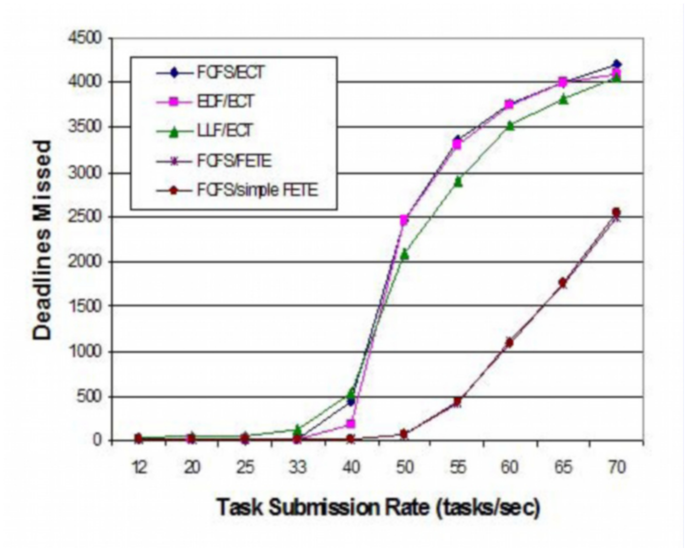


Fig. 5 Deadlines missed versus the task submission rate, in the Uniform resource scenario.

Finally, our performance results showed (Figure 5) that fewer tasks miss their deadlines when they are scheduled using the FETE algorithms than when they are scheduled with other algorithms. This is due to the fact that resources are utilized more uniformly, something that becomes more evident as the task load increases.

7 Conclusions

In this work we proposed two fair scheduling algorithms for Computational Grids, called Fair Execution Time Estimation (FETE) and Simple Fair Execution Time Estimation (SFETE). The FETE algorithms (FETE and SFETE) assign a task to the resource that minimizes what we call its fair execution time estimation. The fair execution time of a task on a certain resource is an estimation of the time by which a task will be executed on the resource, assuming it gets a fair share of the resource's computational power. The FETE algorithms where evaluated and compared against a number of known scheduling algorithms. The results indicate that in most cases and especially at large task submission rates, the FETE algorithms have similar performance and both outperform the other algorithms considered, with respect to performance and fairness related metrics. In addition, SFETE is more realistic, since it does not need the a-priori knowledge of task workload.

Based on these facts, we implemented SFETE in a production Grid Middleware and specifically in gLite [15]. Currently we are in the process of evaluating the efficiency and the scalability of our algorithm against the other, relative simple, scheduling algorithms implemented in gLite, by utilizing a real Grid Testbed.

References

1. I. Ahmad, Y.-K. Kwok, M.-Y. Wu, K. Li, Experimental Performance Evaluation of Job Scheduling and Processor Allocation Algorithms for Grid Computing on Metacomputers, IPDPS, 2004.
2. V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, HPDC, 2002.
3. Y. Cardinale, H. Casanova, An evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms, HPC&S, 2006.
4. T. Braun, et al., A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems, JPDC, 2001.
5. A. Parekh, R. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM ToN, 1993.
6. A. Demers, S. Keshav, S. Shenker, Analysis and simulation of a fair queuing algorithm, SIGCOMM, 1989.
7. R. Buyya, M. Murshed, GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing, Concurrency and Computation: Practice and Experience (CCPE), 2002.
8. R. Buyya, J. Giddy, D. Abramson, An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications, Active Middleware Services, 2000.
9. R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling Parameter Sweep Applications on Global Grids: A Deadline and Budget Constrained Cost-Time Optimization Algorithm, Journal of SPE, 2005.
10. Y. Zhou, H. Sethu, On Achieving Fairness in the Joint Allocation of Processing and Bandwidth Resources, IWQoS, 2003.
11. S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov, A. Shokurov, Comparison of Scheduling Heuristics for Grid Resource Broker, ENC, 2004.
12. K. Rzadca, D. Trystram, A. Wierzbicki, Fair Game-Theoretic Resource Management in Dedicated Grids, CCGrid, 2007.
13. K. H. Kim, R. Buyya, Fair Resource Sharing in Hierarchical Virtual Organizations for Global Grids, Grid Computing, 2007.
14. N. Doulamis, E. Varvarigos, T. Varvarigou, Fair Scheduling Algorithms in Grids, IEEE TPDS, 2007.
15. http://glite.web.cern.ch/glite/