COMPARISON OF CENTRALIZED AND DECENTRALIZED SCHEDULING ALGORITHMS USING GSSIM SIMULATION ENVIRONMENT

Marcin Krystek, Krzysztof Kurowski, Ariel Oleksiak
Poznan Supercomputing and Networking Center
Noskowskiego 10
61-704 Poznan, Poland
mkrystek,krzysztof.kurowski,ariel@man.poznan.pl

Krzysztof Rzadca
LIG, Grenoble University
51, avenue Jean Kuntzmann
38330 Montbonnot Saint Martin, France
and Polish-Japanese Institute of Information Technology
Koszykowa 86
02-008 Warsaw, Poland
rzadca@imag.fr

Abstract Various models and architectures for scheduling in grids may be found both in the literature and in practical applications. They differ in the number of scheduling components, their autonomy, general strategies, and the level of decentralization. The major aim of our research is to study impact of these differences on the overall performance of a Grid. To this end, in the paper we compare performance of two specific Grid models: one centralized and one distributed. We use GSSIM simulator to perform accurate empirical tests of algorithms. This paper is a starting point of an experimental study of centralized and decentralized approaches to Grid scheduling within the scope of the CoreGrid Resource Management and Scheduling Institute.

Keywords: Decentralized scheduling, scheduling architecture, scheduling algorithms, grid, GSSIM simulator

1. Introduction

Decentralization is a key feature of any architectural part of the grid, a system that is crossing organizational boundaries [7]. Nevertheless, standard approaches to scheduling, both theoretical and practical, concern mainly centralized algorithms. In large-scale grids, the centralized approach is clearly unfeasible. Firstly, centralized scheduling requires accurate, centralized information about the state of the whole system. Secondly, sites forming the grid maintain some level of autonomy, yet classic algorithms implicitly assume a complete control over individual resources.

We model the grid as an agreement to share resources between independent organizations. An organization is an entity that groups a computational resource (a cluster) and a group of users that submit jobs. Each organization, by granting access to its resource, in return expects that its jobs will be treated fairly in the system.

In the paper, we compare two classes of scheduling algorithms, centralized and decentralized. In centralized scheduling, one grid scheduler maintains a complete control over the clusters. All the jobs are submitted through the grid scheduler. In contrast, in decentralized scheduling, organizations maintain (limited) control over their schedules. Jobs are submitted locally, but they can be migrated to another cluster, if the local cluster is overloaded. The possibilities of migration are, however, limited, so that migrated jobs do not overload the host system.

The aim of this paper is to compare performance centralized and decentralized scheduling algorithms. Using GSSIM simulation environment, we perfom realistic simulation of example scheduling algorithms that use both approaches, and compute various performance measures of jobs.

In literature, decentralization has two distinct meanings in grid systems, composed of resources under different administrative domains [7]: the decentralization of the algorithm or the decentralization of the optimization goals. A *decentralized algorithm* does not need complete, accurate information about the system. An algorithm with *decentralized goals* optimizes many performance measures of different stakeholders of the system. Classic scheduling algorithm are centralized algorithms that optimize a centralized goal, such as the average completion time of jobs or the makespan. Such scheduling problems have been thoroughly studied, both theoretically [2]and empirically [6]. Decentralized algorithms optimizing decentralized goals include mainly economic approaches [3]. Decentralized algorithms optimizing system-level goal include e.g. [11], that proposes a load-balancing algorithm for divisible task model. Fi-

nally, in optimization of decentralized goals with a centralized algorithm, multi-objective algorithms are usually used [13, 10].

The paper is organized as follows. The architecture of the scheduling system and algorithms for centralized and decentralized implementations are proposed in Section 2. Section 3 contains a description of GSSIM, the simulation environment. Section 4 presents results of experiments.

2. Scheduling Algorithms

In this section we present the model of the grid and scheduling algorithms. In the first case, we assume that there is a single grid scheduler while local schedulers are not autonomous, i.e. they must accept decisions of a grid scheduler. In the second approach, there is no central grid scheduler and local schedulers are autonomous. However, they must obey certain rules agreed between organizations. In both cases, jobs come from users from considered organizations, i.e. there are no external jobs. The next sections contains used notation and details of algorithms.

2.1 Notation and the Model of the Grid

By $\mathcal{O} = \{O_1, \ldots, O_N\}$ we denote the set of independent organizations forming the grid. Each organization O_k owns a cluster M_k. By $\mathcal{M}$ we denote the set of all clusters. Each cluster M_k has m_k identical processors. Cluster have different processing rates. The inverse of M_k processing speed is denoted by s_k.

The set of all the jobs *produced* by O_k is denoted by $\mathcal{I}_k$, with elements $\{J_{k,i}\}$. By $\mathcal{J}_k$ we denote the set of jobs *executed* on O_k's cluster M_k. If $J_{k,i} \in \mathcal{J}_k$, the job is executed *locally*, otherwise it is *migrated*. Job $J_{k,i}$ must be executed in parallel on $q_{k,i}$ processors of exactly one cluster M_l during $p_{k,i} \cdot s_l$ time units. It is not possible to divide a job between two, or more, clusters. The system works on-line. $J_{k,i}$ is not known until its *release date* $r_{k,i}$. Each job has a due date $d_{k,i}$.

In a schedule, by $C_{k,i}$ we denote the completion (finish) time of job $J_{k,i}$. Flow time of job $J_{k,i}$ is defined as the total time the job stays in the system, i.e. $f_{k,i} = C_{k,i} - r_{k,i}$. Tardiness $l_{k,i}$ of job $J_{k,i}$ is defined as the difference between job's completion time and its due date $l_{k,i} = C_{k,i} - d_{k,i}$, if $J_{k,i}$ is completed after its due date ($C_{k,i} > d_{k,i}$), or 0 otherwise.

Organization O_k, in order to measure the performance of its jobs $\mathcal{J}_k$, computes aggregated measures. In this work, we will consider *sum*-type aggregations, such as the sum of flow times $\Sigma_i f_{k,i}$, or the sum of tardiness $\Sigma_i l_{k,i}$, or the number of late jobs U_k.

The performance of the system is defined as a similar aggregation over all the jobs. For instance, system's sum of completion times is defined as $\Sigma_{k,i}C_{k,i}$.

2.2 Centralized Scheduling Algorithm

This algorithm assumes that all the jobs in the system are scheduled by a centralized Grid scheduler, that produces schedules for all clusters $\mathcal{M}$. Each organization must accept decisions of the Grid scheduler, which means that Grid scheduler is the single decision maker and enforcement point within the system.

The algorithm works in batches. This approach is motivated by a possibility of schedule optimization within a batch. In the worst case, online FCFS policy may result in linearly increased makespan (compared to the optimal). Moreover, FCFS prevents Grid scheduler from taking full advantage of available information. We applied the following approach for creating batches. We introduced two parameters batch size s and batch length l. Batch size is a number of jobs that form a batch. Batch length is an amount of time between start time of the last and first job in the batch. The current batch is scheduled if a threshold related to any of these two parameters is achieved or exceeded, i.e. if $s \geq S$ or $l \geq L$. The size limit threshold prevents batches from being to large while the length limit decrease delays of jobs waiting in the next batch. Use of batches has also a practical justification. It causes that local schedules grow more slowly which helps to react in case of failures or imprecise job execution times. Some experimental studies on impact of batch sizes on the performance can be found in [10].

The algorithm consist of two independent policies. The first policy defines the order of jobs in a batch while the second determines the way job is assigned to a given cluster M_k. As the first policy the earliest due date (EDD) has been used. This policy ensures that jobs within a batch are sorted by increasing deadline. Every job J_i must be at position k such as that $d_{j(k-1)} \leq d_i \leq d_{j(k+1)}$, where $j(k)$ denotes a number of a job at position k in a queue.

Jobs J_i from the queue are assigned to one of clusters $\mathcal{M}$ using a greedy list-scheduling algorithm based on [8, 5]. To this end, the Grid scheduler queries each organization O_k about a list of free slots $\psi_{ki} \in \Psi_k, \psi_{ki} = (t', t'', m_{ki}), i = 1..|\Psi|$. Parameters t' and t'' denote start and time of a slot, respectively. m_{ki} is a number of processors available within the slot i at organization O_k. Slots are time periods within which a number of available processors is constant. The Grid scheduler sorts collected slots by increasing start time. The schedule is constructed by

assigning jobs in the Grid scheduler's queue to processors in given slots in a greedy manner. For each slot ψ_{kI} (starting from the earliest one) the scheduler chooses from its queue the first job J_j requiring no more than m_{ki} processors in all subsequent slots $i \geq I$ such as $t''_{ki} \geq t'_{kI} + p_j$, which simply means that jobs' resource requirements must be met for the whole duration of a job. If such a job was found the scheduler schedules it to be started at t'_{kI}, and removes it from the queue. If there is no such a job, the scheduler applies the same procedure to the next free slot with a number of available processors larger than the current one.

2.3 Distributed Scheduling Algorithm

The proposed algorithm consists of two parts. Most of the local jobs are scheduled on the local machine with a list-scheduling algorithm working in batches (Section 2.3.1). Moreover, the scheduler attempts to migrate jobs which would miss their due dates when executed locally. Section 2.3.2 shows an algorithm for handling such migration requests from the receiver's point of view. Although migration improves the performance of the originator, migrated jobs can possibly delay local jobs, and, consequently, worsen the local criterion. We solve this dilemma by introducing limits on the maximum total size of jobs an organization must accept (as a result of the grid agreement), and, at the same time, is able to migrate. These limits are proportional to the length of the current batch multiplied by a *cooperation coefficient* c_k, controlled by each organization O_k.

2.3.1 Scheduling local jobs. Let us assume that the scheduling algorithm was run at time t_0 and returned a schedule which ends at t_1. For each job $J_{k,i}$ released between t_0 and the current makespan $t_{\max}$, the algorithm tries to schedule $J_{k,i}$ so that no scheduled job is delayed and $t_{\max}$ is not increased (conservative backfilling). If it is not possible, $J_{k,i}$ is deferred to the next batch, scheduled at $t_{\max}$. However, if it caused $J_{k,i}$ to miss its due date (i.e. $t_{\max} + p_{k,i}s_k > d_{k,i}$), the scheduler tries to migrate the job to other clusters, by sending migration requests (Section 2.3.2). If cluster M_l can accept $J_{k,i}$ before its due date, the job is removed from the local queue and migrated. If there is more than one cluster ready to accept $J_{k,i}$, the earliest start time is chosen.

At $t_{\max}$, a list scheduling algorithm schedules all the deferred jobs. Jobs are sorted by increasing due dates (EDD). Then, jobs are scheduled with a greedy list-scheduling algorithm [8, 5]. The schedule is constructed by assigning jobs to processors in a greedy manner. Let us assume that at time t, m' processors are free in the schedule under construction. The scheduler chooses from the list the first job $J_{k,i}$ requiring

no more than m' processors, schedules it to be started at t, and removes it from the list. If there is no such job, the scheduler advances to the earliest time t' when one of the scheduled jobs finishes. At t', the scheduler checks if there is any unscheduled job $J_{k,i}$ that missed its due date (i.e. $t + p_{k,i}s_k < d_{k,i}$, but $t' + p_{k,i} > d_{k,i}$). For each such job $J_{k,i}$, scheduler tries to migrate it, using the same algorithm as described in the previous paragraph. The rest of the delayed jobs are scheduled locally.

After all the jobs are scheduled, O_l broadcasts the resulting makespan.

2.3.2 Handling migration requests. Acceptance of a migration request depends on the total surface of migrated jobs already accepted by the host in the current batch, on the total surface of jobs already migrated by the owner and on the impact of the request on the local schedule. Moreover, each organization can control these parameters by means of *cooperation coefficient* c_k ($c_k \geq 0$).

Assuming that the current batch on M_k started at t_0 and will finish at t_1, until the current batch ends, O_k is obliged to accept foreign jobs of total surface of at most $L_k = c_k \cdot (t_1 - t_0) \cdot m_k$. Moreover, O_k can reject a foreign job, if the makespan of the current batch increases by more than $c_k \cdot (t_1 - t_0)$. In order to motivate O_k to declare $c_k > 0$, any other organization O_l can reject O_k's migration requests, if the total surface of jobs exported by O_k in the current batch exceeds L_k.

O_k is obliged to answer to foreign migration requests on-line. Let us assume that, at time t' ($t_0 < t' < t_{\max}$), O_k receives a migration request for job $J_{l,i}$. O_k can reject $J_{l,i}$ straight away in two cases. Firstly, when the total surface of the jobs migrated by the sender O_l exceeds its current limit L_l. Secondly, when O_k has already accepted enough foreign jobs of surface of at least L_k.

Otherwise, O_k's local scheduler finds the earliest strip of free processors of height of at least $q_{l,i}$ and of width of at least $p_{l,i}s_k$ (an incoming, foreign job never delays a scheduled job). If such a strip exists only at the end of the schedule, O_k can reject $J_{l,i}$, if the makespan is increased by more than $c_k \cdot (t_1 - t_0)$. Otherwise, a positive response is returned to the sender O_l. If O_l finally decides to migrate $J_{l,i}$, this decision is broadcasted so that all other organizations can update the surface of O_l's migrated jobs.

2.3.3 Setting the Cooperation Coefficient. Each organization O_k periodically broadcasts its cooperation coefficient c_k, that specifies the organization's desire to balance its load with other organizations. In general, larger values of c_k mean that more migration requests must be locally accepted, but also more local jobs can be migrated. Conse-

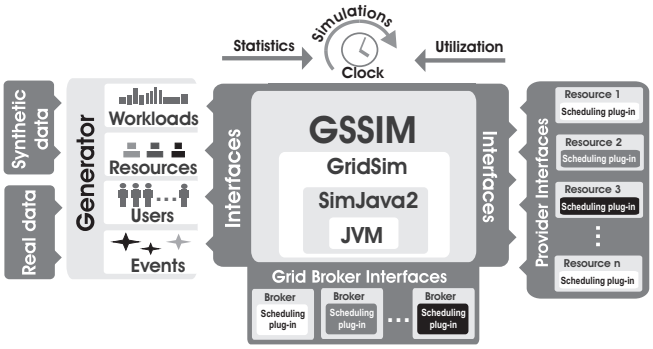

Figure 1. GSSIM architecture

quently, c_k value should depend on the local load and on the observed load (by means of migration requests) of other clusters. An algorithm for setting c_k is beyond the scope of this paper.

In order to make the system more stable, the delay T between two broadcasts of c_k value for an organization must be strongly greater than the average batch size.

3. GSSIM Simulation Environment

To perform experimental studies of models and algorithms presented above we used the Grid Scheduling SIMulator (GSSIM) [9]. GSSIM has been designed as a simulation framework which enables easy-to-use experimental studies of various scheduling algorithms. It provides flexible and easy way to describe, generate and share input data to experiments. GSSIM's architecture and a generic model enables building multilevel environments and using various scheduling strategies with diverse granularity and scope. In particular, researchers are able to build architectures consisting of two tiers in order to insert scheduling algorithms both to local schedulers and grid schedulers. To enable sharing of the workloads, algorithms and results, we have also proposed a GSSIM portal [1], where researchers may download various synthetic workloads, resource descriptions, scheduling plugins, and results.

The GSSIM framework is based on GridSim [4]and SimJava2 packages. However, it provides a layer added on top of the GridSim adding capabilities to enable easy and flexible modeling of Grid scheduling components. GSSIM also provides an advanced generator module using real and synthetic workloads. The overall architecture of GSSIM is presented in Figure 1.

In the centralized case, the main scheduling algorithm is included in a single Grid scheduler plugin (Figure 1, grid broker interfaces). This plugin include implementation of the algorithm presented in Section 2.2. The input of this plugin consists of information about a queue of jobs in a current batch and available resources. Local schedulers (resource providers in the right part of Figure 1) can be queried about a list of time slots. The output of the Grid scheduling plugin is a schedule for all clusters for the batch of jobs.

The implementation of the decentralized version of scheduling algorithm is included exclusively within the local scheduling plugins. Plugins receive as an input a queue of jobs, description of resources and the requests from other local schedulers. Each plugin produces a schedule for its resource.

4. Experiments

This section contains a description of experiment defined using GSSIM. The following subsections contain information about applied workload, how scheduling components were modeled, and results

4.1 Settings: Workload and Metrics

In our experiments, we decided to use synthetic workloads, being more universal and offering more flexibility than the real workloads ([12, 9]).

In each experiment, $n = 500$ jobs are randomly generated. The job arrival ratio is modeled by the Poisson process with λ either the same, or different for each organization. A job is serial ($q_i = 1$) with probability 0.25, otherwise q_i is generated by a uniform distribution from [2,64]. Job length p_i is generated using normal distribution with a mean value $\mu = 20$ and standard deviation $\sigma = 20$. Job's due date d_i set to be d'_i time units its release date. d'_i is generated by a uniform distribution from $[p_i * s_S, 180]$ ($p_i * s_S$ is the job's processing time at the slowest resource).

We used $N = 3$ organizations in the experiment. Each organization owns a single homogeneous cluster. Clusters have 32, 64, and 128. For the sake of simplicity we assumed that their relative processing speed identical.

In order to measure the performance of the system experienced by the users, we used metrics such as the mean job flow time $\bar{F} = (\Sigma_{k,i} f_{k,i})/n$, and the mean job tardiness $\bar{L} = (\Sigma_{k,i} l_{k,i})/n$. The tightness of a schedule for a cluster is expressed by resource utilization $\bar{R}$, defined as $\frac{\Sigma_k \Sigma_{i \in J_k} p_{k,i} q_{k,i}}{\Sigma_k m_k (t_k^{end} - t_k^{start})}$, where $t_k^{end} = \max_i C_{k,i}$, and $t_k^{start} = \min_i r_{k,i}$. Results may be often distorted by first and last jobs that are executed when load of clusters is

relatively low. In order to avoid this inconvenience we did not consider within our results first and last 10% of jobs.

In order to check how equitable are obtained results for all organizations we used the corresponding performance metrics for every single organization O_k. Then we tested how disperse are these values by calculating the standard deviation and comparing extreme values. We also compared the performance metric of each organization to the performance achieved by *local* scheduling algorithm, scheduling jobs on clusters where they are produced (i.e. it is the same as *decentralized*, but with no migration). If there is no cooperation, *local* scheduling is the performance each organization can achieve.

4.2 Results

We performed two series of tests, comparing centralized, decentralized and local algorithm. Firstly, we measured system-level performance in order to check how the decentralization of the algorithm infuences the whole system. In this series of tests, we assumed that all the organizations have similar job streams. Secondly, we compared the fairness proposed by algorithms when organizations' loads differ. Jobs incoming from overloaded clusters cannot lower too much the performance achieved by underloaded organizations.

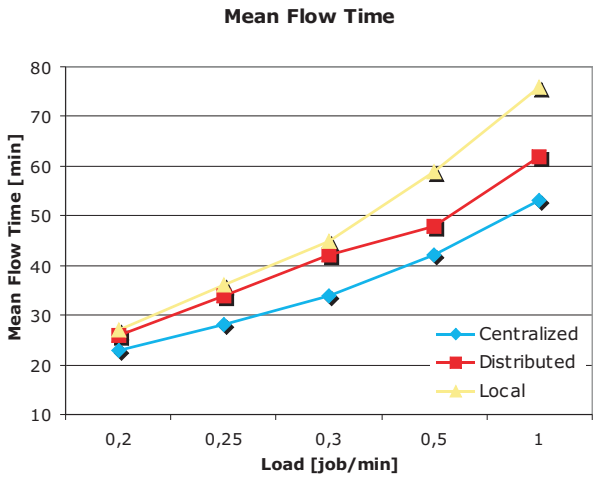

Figure 2. Mean Flow Time for three strategies: centralized, distributed, and local

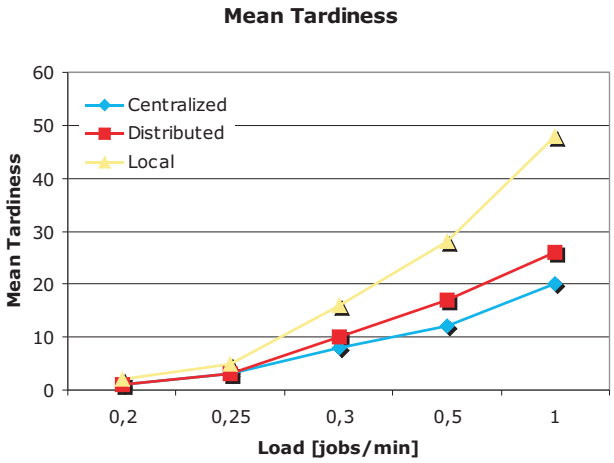

Figure 3. Mean Tardiness for three strategies: centralized, distributed, and local

Generally, the decentralized algorithm schedules slightly worse than the centralized algorithm. However, migration considerably improved the results comparing to the local algorithm. This observation is valid especially for tardiness as this criterion is used by a distributed algorithm to decide about job migration.Typical results of experiments comparing system-level performance are presented in Figure 2 and 3 for mean flow time and tardiness, respectively. In Figure 2 it is easy to notice that mean flow time of the distributed algorithm is comparable to local strategy for low load. These poor results are caused by low number of migrations since majority of jobs can be executed without exceeding their due dates. This situation changes for higher loads when number of migrations is increased and the distributed algorithm outperforms the local one. We achieved similar results for different performance metrics and different sets of parameters.

When clusters' loads differ, the decentralized algorithm was able to increase the fairness of results, by limiting the number of jobs that less-loaded clusters must accept. Figure 4 presents typical results. In this case, performance measures depend strongly on the collaboration factor c_k of less-loaded clusters. Strategies 'distr1', 'distr2', and 'distr3' denotes distributed approach with cooperation factors for organizations $O1$, $O2$, and $O3$ equal to (0.25, 0.25, 0.25), (0.2, 0.2, 0.3), and (0.1, 0.1, 0.5), respectively. When their c_k is too low the system as a whole starts to

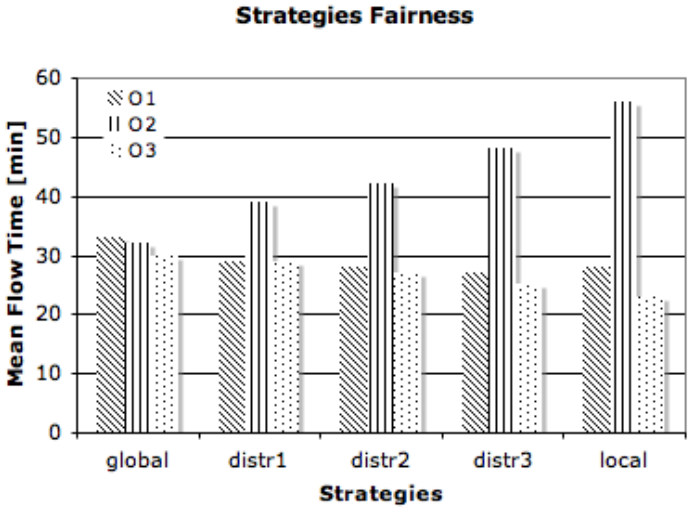

Figure 4. Influence of strategies on fairness of results

be inefficient, although the performance of the less-loaded clusters is not affected. As it is illustrated in Figure 4 the total performance can be improved for certain values of c_k, while mean flow time of less-loaded clusters does not differ dramatically from the local ("selfish") approach. Consequently, we consider that there must be some minimal value of c_k that results from a grid aggreement. As in real systems the job stream changes, this minimal c_k can be also interpreted as an "insurance" to balance the load.

5. Conclusion and future work

In this paper we proposed an experiment to compare centralized and decentralized approaches to scheduling in grid systems. We have chosen two extreme architectures. In centralized scheduling, the grid scheduler had total control over resources. In decentralized scheduling, local schedulers maintained almost complete control over their resources, as each scheduler sets a limit on the maximum migrated workload that it would have to accept. We have proposed to compare these approaches using GSSIM, a realistic simulation environment for grid scheduling. The main conclusion from experiments is that the decentralized approach, although slightly worse for the system-level performance, provides schedules that are much more fair, especially for less-loaded grid participants.

This work is a start of a longer term collaboration in which we plan to study decentralized scheduling algorithms in context of organizationally-distributed grids. We plan to extend the algorithms to support features or constraints present in current grid scheduling software, such as reservations, preemption or limitation to FCFS scheduling. We also plan to validate experimentally by realistic simulation our previous theoretical work on this subject.

References

[1] The gssim portal. `http://www.gssim.org`, 2007.

[2] J. Blazewicz. *Scheduling in Computer and Manufacturing Systems*. Springer, 1996.

[3] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. In *Special Issue on Grid Computing*, volume 93, pages 698–714. IEEE Press, 2005.

[4] GridSim Buyya R., Murshed M. A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[5] L. Eyraud-Dubois, G. Mounie, and D. Trystram. Analysis of scheduling algorithms with reservations. In *Proceedings of IPDPS*. IEEE Computer Society, 2007.

[6] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling a status report. In *Proceedings of JSSPP 2004*, volume 3277 of *LNCS*, pages 1–16. Springer, 2005.

[7] I. Foster. What is the grid. `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`, 2002.

[8] R.L. Graham. Bounds on multiprocessor timing anomalies. *SIAM J. Appl. Math*, 17(2), 1969.

[9] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Grid scheduling simulations with gssim. In *Proceedings of ICPADS'07*. IEEE Computer Society, 2007.

[10] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Multicriteria approach to two-level hierarchy scheduling in grids. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid resource management: state of the art and future trends*, pages 271–293. Kluwer, Norwell, MA, USA, 2007.

[11] J. Liu, X. Jin, and Y. Wang. Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization. *IEEE TPDS*, 16(7):586–598, 2005.

[12] U. Lublin and D. Feitelson. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 11(63):11051122, 2003.

[13] F. Pascual, K. Rzadca, and D. Trystram. Cooperation in multi-organization scheduling. In *Proceedings of the Euro-Par 2007*, volume 4641 of *LNCS*. Springer, 2007.