Cooperation in Multi-organization Scheduling*

Fanny Pascual[1,2], Krzysztof Rzadca[2,3], and Denis Trystram[2]

[1] INRIA Rhône-Alpes, France
[2] LIG Grenoble University, France
[3] Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract. The distributed nature of the grid results in the problem of scheduling parallel jobs produced by several independent organizations that have partial control over the system. We consider systems composed of n identical clusters of m processors. We show that it is always possible to produce a collaborative solution that respects participant's selfish goals, at the same time improving the global performance of the system. We propose algorithms with a guaranteed worst-case performance ratio on the global makespan: a 3-approximation algorithm if the last completed job requires at most $m/2$ processors, and a 4-approximation algorithm in the general case.

1 Introduction

The grid computing paradigm [1] introduces new and difficult problems in scheduling and resource management. A grid can be viewed as an agreement to share resources between a number of independent organizations (such as laboratories, or universities), with little, or no, central, administrative control [2], forcing them to interact. An organization is an administrative entity grouping users and computational resources. Organizations are free to join or to leave the system, if the gain experienced is lower than the cost of participation. Therefore, in order to sustain the grid, the resource management system must achieve an acceptable performance not only at the level of the community of users (as in classic, monocriterion scheduling), but also on the between-organizations level. Some globally-optimal approaches may be unacceptable because they implicitly favor jobs produced by one organization, therefore reducing the performance experienced by the others.

In this paper, we study the problem of scheduling parallel jobs [3] produced by several *organizations*. Each organization owns and controls a cluster, that together form a computational grid. The global goal is to minimize the makespan [3], the time moment when all the jobs are finished. However, each organization is only concerned with the makespan of its own jobs. An organization can always quit the grid and compute all its jobs on its local cluster. Therefore, a solution which extends the makespan of an organization in comparison with such a local solution is not *feasible*, even if it leads to a better global makespan. Such an organization would

* This research was partly supported by the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). Krzysztof Rzadca is partly supported by the French Government Grant number 20045874.

A.-M. Kermarrec, L. Bougé, and T. Priol (Eds.): Euro-Par 2007, LNCS 4641, pp. 224–233, 2007.

prefer to quit the grid, to compute all its jobs locally and not to accept any other jobs on its cluster. The considered scheduling problem is therefore an extension of the the classic, parallel job scheduling [3] by a series of constraints stating that that no organization's makespan can be increased.

The main contribution of the paper is the demonstration that several independent organizations have always interest to collaborate in a load-balancing grid system. We propose an algorithm producing solutions that guarantee that no organization's makespan is increased, at the same time having guaranteed approximation ratio (worst-case performance) regarding the globally-optimal solution. Assuming that each cluster has m of processors, the proposed algorithm is a 3-approximation if the last finished job is *low* (requires at most half of the available processors), and a 4-approximation in the general case.

This paper is organized as follows. Section 2 introduces some notations, formally defines the model and the problem and presents some motivating examples. Section 3 considers a problem of scheduling local and foreign jobs on a single multiprocessor cluster with guaranteed performance for local jobs. Section 4 presents the algorithms for n multiprocessor clusters and proves the approximation ratios. Related work is discussed in Section 5. Section 6 discusses the results obtained and concludes the paper.

2 Preliminaries

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. Each cluster M_k has m identical processors. By $\mathcal{M}$ we denote the set of all clusters.

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 during $p_{k,i}$ time units. It is not possible to divide a job between two, or more, clusters. We denote by $p_{\max} = \max p_{k,i}$ the maximum length of job. $J_{k,i}$ is *low* if it needs no more than a half of cluster's processors ($q_{k,i} \leq \frac{m}{2}$), otherwise it is *high*.

By $C_{k,i}$ we denote the completion (finish) time of job $J_{k,i}$. For an organization O_k, we may compute the maximum completion time (makespan) as $C_{\max}(O_k) = \max_{k,i}\{C_{k,i} : J_{k,i} \in \mathcal{I}_k\}$. The global makespan $C_{\max}$ is the maximum makespan of organizations, $C_{\max} = \max_k C_{\max}(O_k)$.

For cluster M_k, a *schedule* is a mapping of jobs $\mathcal{J}_k$ to processors and start times in such a way that, at each time, no processor is assigned to more than one job. We can define the *makespan* $C_{\max}(M_k)$ *of cluster* M_k as the maximum completion time of jobs $\mathcal{J}_k$ assigned to that cluster, $C_{\max}(M_k) = \max_i\{C_{j,i} : J_{j,i} \in \mathcal{J}_k\}$. At any time t, *utilization* $U_k(t)$ of M_k is the ratio of the number of assigned processors to the total number of processors m. A *scheduler* is an application which produces schedules, given the sets of jobs produced by each organization.

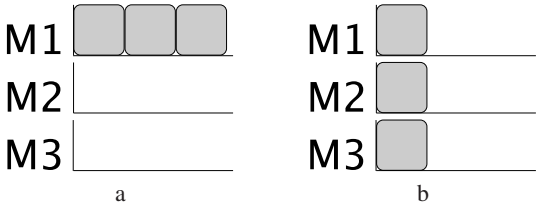

Fig. 1. Executing all the jobs locally (a) may lead to n approximation ratio regarding the globally-optimal solution (b). All the jobs were produced by organization O_1, the owner of M_1.

2.2 Problem Statement

We consider off-line, clairvoyant scheduling with no preemption on time-sharing processors. Those assumptions are fairly realistic in most of the existing scheduling systems, which use batches [4] and which require the user to define the run-time of the posted jobs. Each organization O_k wants to minimize the date $C_{\max}(O_k)$ at which all the locally produced jobs $\mathcal{I}_k$ are finished. Organization O_k does not care about the performance of other organizations, nor about the actual makespan $C_{\max}(M_k)$ on local cluster M_k, if the last job to be executed is not owned by O_k. However, $C_{\max}(O_k)$ takes into account jobs owned by O_k and executed on non-local clusters, if there are any.

The *Multi-Organization Scheduling Problem* (MOSP) is the minimization of the makespan of all the jobs (the moment when the last job finishes) with an additional constraint that no makespan is increased compared to a preliminary schedule in which all the clusters compute only locally produced jobs. More formally, let us denote $C^{loc}_{\max}(O_k)$ as a makespan of O_k when $\mathcal{J}_k$, the set of jobs executed by M_k is equal to the set of locally produced jobs, i.e. $\mathcal{J}_k = \mathcal{I}_k$. MOSP can be defined as:

$$\min C_{\max} \text{ such that } \forall_k \; C_{\max}(O_k) \leq C^{loc}_{\max}(O_k). \tag{1}$$

By restricting the number of organizations to $n = 1$, the size of the cluster to $m = 2$ and the jobs to sequential ones ($q_{k,i} = 1$), we obtain the classic, NP-hard problem of scheduling sequential jobs on two processors $2|p_j|C_{max}$ [5]. Therefore, MOSP is also NP-hard.

2.3 Motivation

A number of instances motivate organizations to cooperate and accept non-local jobs, even taking into account the fact that the resulting configuration is not necessary globally optimal. A non-cooperative solution (without the grid) is that all the organizations compute their jobs on their local clusters. However, such a solution can be as far as n times worse than the optimal one (see Figure 1). Note also that careful scheduling offers more than simple load balancing of the previous example. By matching certain types of jobs, bilaterally profitable solutions are also possible (see Figure 2). Nevertheless, a certain price must be paid in order to produce solutions in which all the organizations have incentive to participate. Figure 3 presents an instance in which the globally-optimal solution extends the makespan of one of the organizations. Consequently, all

Fig. 2. By matching certain types of jobs, cooperative solution (b) delivers better makespans for both organizations than a solution scheduling all the jobs locally (a). The light gray jobs were produced by organization O_1, the dark gray ones by O_2.

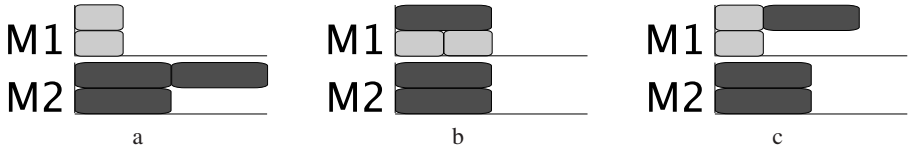

Fig. 3. Globally-optimal solution (b) is inadmissible, as it extends the makespan of organization O_1 (the producer of light gray jobs) in comparison with the local solution (a). The best solution not extending O_1's makespan (c) is $\frac{3}{2}$ from the global optimum.

the algorithms that meet the constraint have at least $\frac{3}{2}$ approximation ratio regarding the globally-optimal solution.

3 Scheduling on One Cluster

Let us first focus on the simple case of scheduling rigid parallel jobs on one cluster consisting of m identical processors. Note that as in this section there is no reason to distinguish between a cluster and an organization, we will simply use $C_{\max}$ to denote the makespan and omit the index of the organization in other notations (e.g. $q_{k,i}$ becomes q_i). We will use here the classic list scheduling algorithm, which has an approximation ratio equal to $2 - \frac{1}{m}$. We show that if the jobs are ordered according to decreasing number of required processors, the resulting schedule achieves fairly homogeneous utilization. Preliminary results established in this section will be later used to solve the general problem of multi-organization scheduling in Section 4.

3.1 List Scheduling

List scheduling [6] is a class of heuristics which work in two phases. In the first phase, jobs are ordered into a list. In the second phase, 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_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.

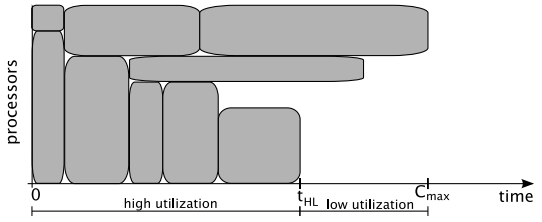

Fig. 4. When jobs are presorted according to number of required processors, the schedule can be divided into two regions with utilization $U(t) > \frac{1}{2}$ (up to t_{HL}) and $U(t) \leq \frac{1}{2}$ (after that moment)

Although straightforward, list scheduling of rigid parallel jobs is an approximation algorithm with guaranteed worst case performance of $2 - \frac{1}{m}$ [7], no matter the order of jobs in the first phase. A polynomial time algorithm with better approximation ratio is not known.

3.2 Highest First (HF) Job Order

The $2-\frac{1}{m}$ approximation ratio of list scheduling does not depend on the particular order of jobs in the list. Therefore, we may choose a criterion which gives some interesting properties of the resulting schedule without loosing the approximation ratio.

Let us consider jobs ordered according to the Highest First (HF) rule, i.e. by non-increasing q_i. The following proposition holds:

Proposition 1. *All HF schedules have the same structure consisting of two consecutive regions of high ($t \in [0, t_{HL}) : U(t) > \frac{1}{2}$) and low ($t \in [t_{HL}, C_{\max} : U(t) \leq \frac{1}{2}$) utilization, where $0 \leq t_{HL} \leq C_{\max}$ (Figure 4).*

Proof: First, note that no high job is scheduled after a period of low utilization. Indeed, as soon as a high job is completed, the following highest job is scheduled (according to the HF rule). Thus there is no low utilization period before that all the high jobs have been completed. The proof is now by contradiction. Let us assume that at time t the utilization is low ($U(t) \leq \frac{1}{2}$), and that at time $t' > t$ the utilization is high ($U(t') > \frac{1}{2}$). Let us consider a job J_i scheduled at time t'. It is not possible that J_i is a high job because no high job can be scheduled after a period of low utilization, as noted before. If J_i is low ($q_i \leq \frac{m}{2}$) then it could have been scheduled at time t, and scheduling it after t contradicts the greedy principle of the list scheduling algorithm. □

4 Multi-organization Scheduling

In this section we present an algorithm that address the Multi-Organization Scheduling Problem. This algorithm has a guaranteed approximation ratio regarding to the global makespan, at the same time not worsening the local solutions that are produced by the organizations while computing independently.

The algorithm that we propose computes a lower bound of the global makespan and then moves all the jobs which start after twice this date to the end of the schedules of less-loaded clusters. Details follow.

Let us denote by $W = \sum p_{k,i}q_{k,i}$ the total work in the system, or the total *surface* of the jobs. As all the jobs must fit into available processors, the global makespan $C_{\max}$ is not less than the lower bound $LB = \frac{W}{mn}$.

Let us assume that all the organizations list-scheduled their jobs on their local machines according to HF order. The *Multi-Organization Load Balancing Algorithm (MOLBA)* is the following one. It starts to compute LB. All the clusters with local makespans between $2LB$ and $2LB + p_{\max}$ are ignored (we do not move their local jobs). For the rest of the clusters, all the jobs that start after time $2LB$ are moved from their local clusters to a migration queue. Finally, the jobs from the migration queue are list-scheduled onto all available clusters. The jobs are scheduled sequentially in a greedy manner. No migrated job can delay a local job: a job $J_{k,i}$ is scheduled before the original makespan of the host cluster M_j ($t < C^{loc}_{\max}(O_j)$) only if at least $q_{k,i}$ processors are free on M_j from time t to time $t + p_{k,i}$. Such a strategy is similar to the well-known conservative backfilling in FCFS (First Come First Serve).

We prove in Sections 4.1 and 4.2 that this algorithm is a $3-$approximation of the global makespan $C_{\max}$ when the last completed task is a low task, and that is it a 4-approximate algorithm in the general case. We also show that this algorithm does not increase the local makespans of the organizations (therefore holding the constraint in Eq. 1). We start with a lemma that characterizes the structure of all the clusters' schedules.

In the schedule returned by MOLBA, on each cluster, we denote by t_L^{start} the first moment when the utilization is lower than or equal to $\frac{1}{2}$. Similarly, t_L^{end} is the last moment when the utilization is larger than 0 and lower than or equal to $\frac{1}{2}$. We first prove the following lemma:

Proposition 2. *In the schedule returned by MOLBA, on each cluster, the length of the time interval between t_L^{start} and t_L^{end} (denoted by P_L) is shorter than or equal to $p_{\max}$.*

Proof: Each cluster schedules its local jobs with HF. Then, it may add jobs from other organizations, also in HF order. Proposition 1 shows that, in a schedule returned by HF, the only zone of low utilization is at the end of the schedule. Thus, on each cluster, there are at most two zones of low utilization: possibly one at the end of the schedule of the local jobs, and also possibly one at the end of the schedule.

Let $J_{k,i}$ be the *low* job that finishes last on cluster M_j. After $J_{k,i}$ finishes, utilization is either high, or zero. Thus, by P_L definition, $J_{k,i}$ cannot finish before t_L^{end}. $J_{k,i}$ does not start after t_L^{start}, as utilization at t_L^{start} is low, so there are enough free processors to execute a *low* job. Thus, the length of P_L is smaller than or equal to the length of $J_{k,i}$, which is not longer than p_{max}. □

Proposition 3. *After MOLBA finishes, there is at least one cluster whose $t_L^{start} \leq 2\,LB$.*

Proof: The proof is by contradiction. Suppose that there exists $\epsilon > 0$ such that all the clusters have high utilization until time $2\,LB + \epsilon$. Then, the total surface of jobs

computed by all the clusters is greater than $2\,LB \cdot mn \cdot 0.5 = W$, i.e. greater than the total work available, which leads to a contradiction. □

4.1 Low Jobs

We show in this section that, in the schedule returned by MOLBA, if the last completed job is a *low* job, then MOLBA is a $3-$approximate algorithm.

Proposition 4. *The makespan of the schedule returned by MOLBA is a 3-approximation of the optimal makespan, if the last completed job is* low. *Moreover, all the organizations have incentive to cooperate.*

Proof: The proof uses two well known lower bounds of the optimal makespan $C^*_{\max}$. Firstly, the longest job (of length $p_{\max}$) must be completed, so $C^*_{\max} \geq p_{\max}$. Secondly, all the jobs must fit onto available processors, so $C^*_{\max} \geq LB = \frac{W}{mn}$. The last job $J_{k,i}$ finishes at $C_{\max}$. Recall that this job is a low job. Proposition 3 guarantees that there is at least one cluster with low utilization before or at time $2\,LB$. Thus, job $J_{k,i}$ does not start after $2\,LB$, since we use a list scheduling algorithm. Hence, $C_{\max} \leq 2LB + p_{k,i} \leq 2LB + p_{\max} \leq 3C^*_{\max}$.

As no migrated job can delay a local job, makespans of organizations that were receiving tasks are not modified. The organizations that were sending tasks have their makespan reduced because of the global approximation ratio. The schedule of the rest of organizations is not modified. Thus, the constraint in Equation (1) is satisfied and all the organizations have incentive to cooperate. □

4.2 General Case

Let us now consider the case where the last completed job can have any height. We now show that MOLBA achieves an approximation ratio of 4 on the global makespan. We suppose here that we "cut" the schedule where each organization schedule its local jobs at time $3\,LB$ (and not $2\,LB$ as in the previous case). We do not move the local tasks of organizations with local makespans smaller than $4\,LB$. For the rest of the clusters, all the jobs which start after time $3\,LB$ before the load balancing procedure are moved into a queue and then scheduled using the HF list algorithm.

Proposition 5. *MOLBA is a* 4*-approximate algorithm and all the organizations have incentive to cooperate.*

Proof: Let us prove this Proposition by contradiction. Let C^*_{max} be the makespan of an optimal schedule, and let us suppose that a job starts after time $3\,C^*_{max}$ in the schedule returned by MOLBA. This means that this job could not have been started before : for all $i \in \{1, \ldots, n\}$, $C_{max}(M_i) \geq 3\,C^*_{max}$. Proposition 2 shows that, for each cluster, the zone where at most half of the processors are busy is smaller than or equal to C^*_{max}. Thus, on each cluster, the zone where at least half of the processors are busy is larger than or equal to $2\,C^*_{max}$. As we have seen it previously, $C^*_{max} \geq \frac{W}{n\,m}$,

where $W = \sum p_{k,i}q_{k,i}$. Thus, the total work which is done before $3\,C^*_{max}$ in the zones of high utilization is larger than or equal to $\frac{n\,m}{2}(2\,\frac{W}{n\,m}) = W$. This is not possible since the total work which has to be done is equal to W. Thus, no job starts after time $3\,C^*_{max}$, and no job is completed after time $4\,C^*_{max}$.

The proof that all the organizations have incentive to cooperate is analogous to the proof of Proposition 4. □

There are some other special cases in which the presented approximation ratio can be improved. When there are $n = 2$ organizations, the original version of the algorithm ("cutting" the schedules at $2\,LB$) is $3-$approximate. We omit the proof because of the lack of space. For n clusters, and when all the jobs are *low*, the algorithm is also $3-$approximate, since this special case is included in the proof presented in Section 4.1. Finally, when all the tasks are high, no two tasks can be scheduled in parallel on one cluster. Thus, the problem corresponds to scheduling sequential tasks on n processors. Any list scheduling algorithm is, in this case, $2-\frac{1}{n}$ approximate. It is straightforward to guarantee that all the organizations have incentive to cooperate. Each task is scheduled on its local processor, unless there is a free processor that already scheduled its local tasks.

5 Related Work

In this paper we have studied the interest of collaboration between independent parties. We have claimed that if a proposed, collaborative solution does not deteriorates any participant's selfish goal, it will be adopted by all the participants.

Using a reasonable set of assumptions, we have demonstrated that it is always possible to produce such collaborative solutions. Moreover, we have developed an algorithm which has a worst-case guarantee on the social goal (the makespan of the system), at the same time respecting selfish goals of participants.

In this section we will briefly summarize how the concept of collaboration and the distributed nature of systems has been understood by and used in other works.

Non-cooperative game theory studies situations in which a set of selfish agents optimize their own objective functions, which also depend on strategies undertaken by other agents. The central notion is the Nash equilibrium [8], a situation in which no agent can improve its own objective function by unilaterally changing his/her strategy. It can be useful to define a *social* (global) objective function, which expresses the performance of the system as a whole. The ratio between the values of this function in the worst Nash equilibrium and in an optimal solution is called the Price of Anarchy (PoA)[9]. This can be interpreted as the cost of no cooperation and can be high. In the context of scheduling, [10] measures PoA when selfish sequential jobs choose one of the available processors. A related measure, Price of Stability (PoS) [11,12] compares the socially-best Nash equilibrium with the socially-optimal result. Usually, in order to find such an equilibrium, a centralized protocol gathers information from, and then suggests a strategy to, each participant. Since the proposed solution is a Nash equilibrium, the

participants do not have incentive to unilaterally refuse to follow it. [13] computes PoS in the same model as [10], but relaxes the selfishness of jobs by a factor of α and studies the trade-off between α and the approximation ratio of the global makespan. The collaborative solution proposed by our algorithm approximates the socially-best Nash equilibrium, because it optimizes the global goal with a guarantee that no participant has the incentive to deviate from the proposed solution.

Cooperative game theory studies similar situations, but assumes that players can communicate and form coalitions. The members of a coalition split the sum of their payoffs after the end of the game. Note that this requires that the payoffs are transferable, which is not the case in our problem.

Papers proposing *distributed* resource management or distributed load balancing usually solve the problem of optimizing a common goal with a decentralized algorithm. [14] shows a fully decentralized algorithm that always converges to a steady state. [15] presents a similar algorithm with the divisible load job model. Those approaches contrast with our algorithm. Although the algorithm is centralized, it respects the decentralized goals of participants. We are, however, aware that a load balancing algorithm in large scale systems must be decentralized. In [16] a fully distributed algorithm balances selfish *identical* jobs on a network of identical processors. The aim of each job is to be on the least loaded machine. The work focus on the time needed to converge towards a Nash equilibrium. Alternative approaches propose to balance the load by an implicit barter trade of CPU power [17], or explicit computational economy [18].

6 Conclusion and Perspectives

In this work we have considered the problem of cooperation between selfish participants of a computational grid. More specifically, we studied a model of the grid in which selfish organizations minimize the maximum completion time of locally-produced jobs. Under some basic assumptions (off-line, clairvoyant system, idle time of machines is free) we have demonstrated that it is always possible to respect the selfish goals at the same time improving the performance of the whole system. The cooperative solutions have a constant worst case performance, a significant gain compared to selfish solutions that can be arbitrary far from the optimum. We deliberately focused on the analysis of the worst-case performance in order to avoid the plethora of problems of the experimental methodology in grid systems.

Our aim was not to find an algorithm solving the general problem of grid resource management, which complexity is overwhelming for any kind of mathematical modeling. However, we claim that the positive results given by this paper proves that cooperation achieved at the algorithmic (as opposed to e.g. economic) level is possible. Note that it should be fairly straightforward to relax some of our assumptions, e.g. to use on-line scheduling in batches instead of off-line. An interesting direction would also be to consider this mutiorganization scheduling problem with heterogeneous clusters.

In our future work, we would like to study the effect of the increased effort of individuals on the global goal. More specifically, we would like to relax the hard constraint of "not being worse than the local solution" to an approximation of "not being worse than α times local solution".

References

1. Foster, I., Kesselman, C.: The Grid 2. Blueprint for a New Computing Infrastructure. Elsevier, Amsterdam (2004)
2. Foster, I.: What is the grid, http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf
3. Blazewicz, J.: Scheduling in Computer and Manufacturing Systems. Springer, Heidelberg (1996)
4. Shmoys, D., Wein, J., Williamson, D.: Scheduling parallel machines on-line. SIAM Journal on Computing 24 (1995)
5. Garey, M.R., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. WH Freeman & Co, New York, USA (1979)
6. Graham, R.: Bounds on multiprocessor timing anomalies. SIAM J. Appl. Math 17(2) (1969)
7. Eyraud-Dubois, L., Mounie, G., Trystram, D.: Analysis of scheduling algorithms with reservations. In: Proceedings of IPDPS, IEEE Computer Society Press, Los Alamitos (2007)
8. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
9. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 99. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
10. Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms for selfish scheduling. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 345–357. Springer, Heidelberg (2003)
11. Schultz, A.S., Stier Moses, N.: On the performance of user equilibria in traffic networks. In: Proceedings of SODA, 86–87 (2003)
12. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proceedings of FOCS, pp. 295–304 (2004)
13. Angel, E., Bampis, E., Pascual, F.: The price of approximate stability for a scheduling game problem. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, Springer, Heidelberg (2006)
14. Liu, J., Jin, X., Wang, Y.: Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization. IEEE TPDS 16(7), 586–598 (2005)
15. Rotaru, T., Nageli, H.H.: Dynamic load balancing by diffusion in heterogeneous systems. J. Parallel Distrib. Comput. 64(4), 481–497 (2004)
16. Berenbrink, P., Friedetzky, T., Goldberg, L., Goldberg, P., Hu, Z., Martin, R.: Distributed Selfish Load Balancing. In: Proceedings of SODA, pp. 354–363. ACM Press, New York (2006)
17. Andrade, N., Cirne, W., Brasileiro, F., Roisenberg, P.: Ourgrid: An approach to easily assemble grids with equitable resource sharing. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 61–86. Springer, Heidelberg (2003)
18. Buyya, R., Abramson, D., Venugopal, S.: The grid economy. Special Issue on Grid Computing 93, 698–714 (2005)