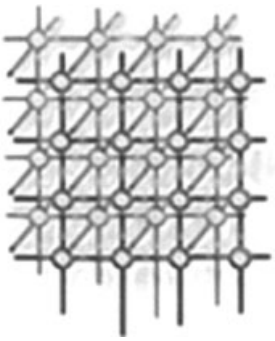
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **21**:905–921
Published online 9 September 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1378



Cooperation in multi-organization scheduling

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

[1]*LIP6, Université Pierre et Marie Curie-Paris 6, 104 Avenue du Président Kennedy, 75016 Paris, France*
[2]*LIG, Grenoble University, 51, Avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France*
[3]*Polish-Japanese Institute of Information Technology, Koszykowa 86, 02-008 Warsaw, Poland*

SUMMARY

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 in which each organization owns a cluster of processors. Each organization wants its tasks to be completed as soon as possible. In this paper, we model an off-line system consisting of N identical clusters of m processors. We show that it is always possible to produce a collaborative solution that respects participants' selfish goals, at the same time improving the global performance of the system. We propose an algorithm (called MOLBA) with a guaranteed worst-case performance ratio on the global makespan, equal to 4. Next, we show that a better bound (equal to 3) can be obtained in a specific case when the last completed job requires at most $m/2$ processors. Then, we derive another algorithm (called ILBA) that in practice improves the proposed, guaranteed solution by further balancing the schedules. Finally, by an extensive evaluation by simulation, we show that the algorithms are efficient on typical instances. Copyright © 2008 John Wiley & Sons, Ltd.

Received 25 January 2008; Accepted 26 May 2008

KEY WORDS: grid; scheduling; approximation algorithms; multi-objective optimization

*Correspondence to: Denis Trystram, LIG, Grenoble University, 51, Avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France.
†E-mail: trystram@imag.fr

Contract/grant sponsor: INRIA
Contract/grant sponsor: European Commission; contract/grant number: IST-2002-004265

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 at the between-organizations level. Some globally optimal approaches may be unacceptable because they implicitly favor jobs produced by one organization, thus 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 forms a computational grid. The global goal is to minimize the makespan [3], the date 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 that 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 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 classic, off-line parallel job scheduling [3] by a series of constraints stating that no organization's makespan can be increased.

The main contribution of this paper is the demonstration that several independent organizations always have an incentive to collaborate in the proposed load-balancing grid system. The proposed algorithm produces 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 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. As simulations show, the proposed algorithms work much more efficiently both on average and in the worst-case instances.

This paper is organized as follows. Section 2 introduces notations, formally defines the model and the problem, and presents some motivating examples. Section 3 considers the problem of scheduling local and foreign jobs on a single multiprocessor with guaranteed performance for local jobs. Section 4 presents the algorithm for N identical multiprocessor clusters and proves the approximation ratios. Section 5 shows an algorithm that is used to improve the results of the base algorithm. Results of simulations are analyzed in Section 6. Related work is discussed in Section 7. Section 8 summarizes the obtained results and concludes this 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. By $\mathcal{M}$ we denote the set of all clusters. Each cluster M_k has m_k

DOI: 10.1002/cpe

identical processors. For the remainder of this paper, we consider that all the clusters have the same number, m, of processors.

The set of all the jobs *produced* by O_k is denoted by $\mathscr{I}_k$, with elements $\{J_{k,i}\}$. By $\mathscr{J}_k$ we denote the set of jobs *executed* on O_k's cluster M_k. If $J_{k,i} \in \mathscr{J}_k$, 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 $p_{\max} = \max_{k,i}\ p_{k,i}$ the maximum job length. $J_{k,i}$ is *low* if it needs no more than half a cluster's processors ($q_{k,i} \leq m/2$), 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 compute the maximum completion time (makespan) as $C_{\max}(O_k) = \max_i\{C_{k,i}$ such that $J_{k,i} \in \mathscr{I}_k\}$. 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 $\mathscr{J}_k$ to processors and start times in such a manner that, at each time, no processor is assigned to more than one job. We define the *makespan* $C_{\max}(M_k)$ *of cluster* M_k as the maximum completion time of jobs $\mathscr{J}_k$ assigned to that cluster, $C_{\max}(M_k) = \max_{i,j}\{C_{j,i}$ such that $J_{j,i} \in \mathscr{J}_k\}$. 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 that produces schedules, given the sets of jobs produced by each organization.

2.2. Problem statement

We consider off-line, clairvoyant scheduling with no preemption. 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 jobs. The objective of each organization O_k is to minimize the date $C_{\max}(O_k)$ at which all the locally produced jobs $\mathscr{I}_k$ are finished. Organization O_k does not care about the performance of other organizations or 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 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 the additional constraint that no makespan is increased compared with a preliminary schedule in which all the clusters compute only locally produced jobs. More formally, let us denote $C^{loc}_{\max}(O_k)$ as the makespan of O_k when $\mathscr{J}_k$, the set of jobs executed by M_k, is equal to the set of locally produced jobs, i.e. $\mathscr{J}_k = \mathscr{I}_k$. 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 $P2\|C_{\max}$ [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

DOI: 10.1002/cpe

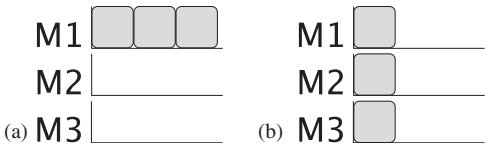


Figure 1. Executing all the jobs locally (a) may lead to N approximation ratio regarding the globally optimal solution (b). All the jobs were produced by organization O_1, the owner of M_1.

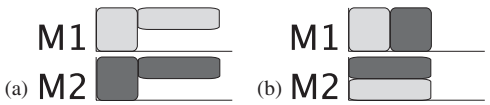


Figure 2. By matching certain types of jobs, cooperative solution (b) delivers better makespans for both organizations than a solution scheduling all the jobs locally (a). The light gray jobs were produced by organization O_1, the dark gray ones by O_2.

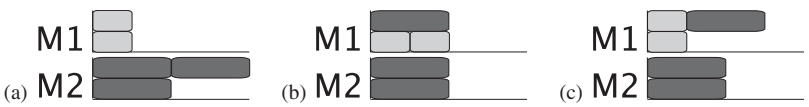


Figure 3. Globally optimal solution (b) is inadmissible, as it extends the makespan of organization O_1 (the producer of light gray jobs) in comparison with the local solution (a). The best solution not extending O_1's makespan (c) is $\frac{3}{2}$ from the global optimum.

local clusters. However, such a solution can be as far as N times worse than the optimal one (see Figure 1). Moreover, 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 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

DOI: 10.1002/cpe

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 $2-1/m$. 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 are used to solve the general problem of multi-organization scheduling in Section 4.

3.1. List scheduling

List scheduling [6] is a class of scheduling algorithms that work in two phases. In the first phase, jobs are ordered into a list (using any criterion). 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.

Although straightforward, list scheduling of rigid parallel jobs is an approximation algorithm with guaranteed worst-case performance of $2-1/m$ [7], regardless of 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 $2-1/m$ approximation ratio of list scheduling does not depend on the particular order of jobs in the list. Therefore, we may choose a criterion that gives some interesting properties of the resulting schedule without loosing the approximation ratio.

Let us consider jobs ordered according to the highest first (HF) order, i.e. by non-increasing q_i. The following proposition, illustrated in Figure 4, holds:

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

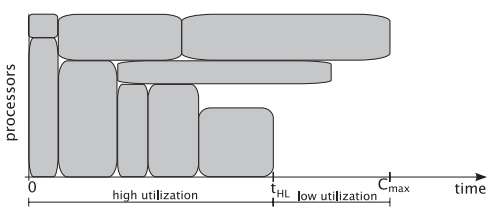


Figure 4. When jobs are presorted according to number of required processors, the schedule can be divided into two regions with utilization $U(t)>\frac{1}{2}$ (up to t_{HL}) and $U(t) \le \frac{1}{2}$ (after that moment).

DOI: 10.1002/cpe

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 order). Thus, there is no low utilization period before 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 ($q_i \leq m/2$) 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 LOAD BALANCING ALGORITHM

In this section we present an algorithm that addresses the MOSP. This algorithm has a guaranteed approximation ratio regarding the global makespan, at the same time not worsening the local solutions that are produced by the organizations while computing independently. In this section, we first present the algorithm. Then, we show some general properties about the resulting schedules. Finally, we prove approximation ratios.

4.1. Description of the algorithm

The Multi-Organization Load Balancing Algorithm, denoted by MOLBA(α) (with $\alpha \geq 1$), computes lower bounds of the global makespan, schedules all the jobs on their local cluster, selects all the organizations whose makespan is larger than $(\alpha+1)$ times these bounds, and then reschedules some of their jobs onto the less loaded clusters.

Two lower bounds on the global makespan $C_{\max}$ can be defined. Let $C^*_{\max}$ be the global makespan of an optimal solution. Let us denote by $W = \sum p_{k,i} q_{k,i}$ the total *surface* of the jobs and by $p_{\max} = \max p_{k,i}$ the length of the longest job. Firstly, all the jobs must fit into available processors; hence, $C^*_{\max} \geq \bar{W} = W/Nm$. Secondly, the longest job must be executed; hence, $C^*_{\max} \geq p_{\max}$.

MOLBA(α) starts with scheduling jobs $\mathcal{J}_k$ on local cluster M_k with list scheduling algorithm in HF order (i.e. non-increasing number of required processors). Local makespans $C^{loc}_{\max}(O_k)$ obtained at this stage of the algorithm are constraints of the MOSP.

Then, the algorithm computes the subset $\mathcal{O}'$ of all organizations that have local makespan $C_{\max}(O_k) \geq \alpha\bar{W} + p_{\max}$. Parameter α expresses the trade-off between the global efficiency and the constraints on local makespans (see the proofs in the following sections). Hereafter, we refer to the algorithm with a particular value of α as MOLBA(α), e.g. MOLBA(2).

Next, jobs belonging to $\mathcal{O}'$, which start after $2\bar{W}$, are removed from local clusters, mixed, and rescheduled on all the clusters. These 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_l ($t < C^{loc}_{\max}(O_l)$) only if at least $q_{k,i}$ processors are free on M_l from time t to time $t + p_{k,i}$. Such a strategy is similar to the well-known conservative backfilling in first come first serve (FCFS) [8]. Therefore, a job incoming to one of the clusters that do not belong to $\mathcal{O}'$ is scheduled either in a gap or at the end of the schedule.

DOI: 10.1002/cpe

4.2. An example run

Figure 5 presents a typical result of executing MOLBA(2) on an instance with $N = 5$ organizations. Scheduling all the jobs locally (Figure 5(a)) results in large system makespan $C_{\max}$, which is determined by the makespan of O_1. O_1 is the only organization whose local makespan is larger than $2\bar{W} + p_{\max}$. MOLBA(2) (Figure 5(b)) improves $C_{\max}$ by rescheduling several of O_1 jobs on M_3 and M_5. Note that, in the resulting schedule, no makespan is increased. The makespan meets the theoretical bound of 4 times the lower bound of $C^*_{\max}$. In Section 5 we show that further improvement is possible.

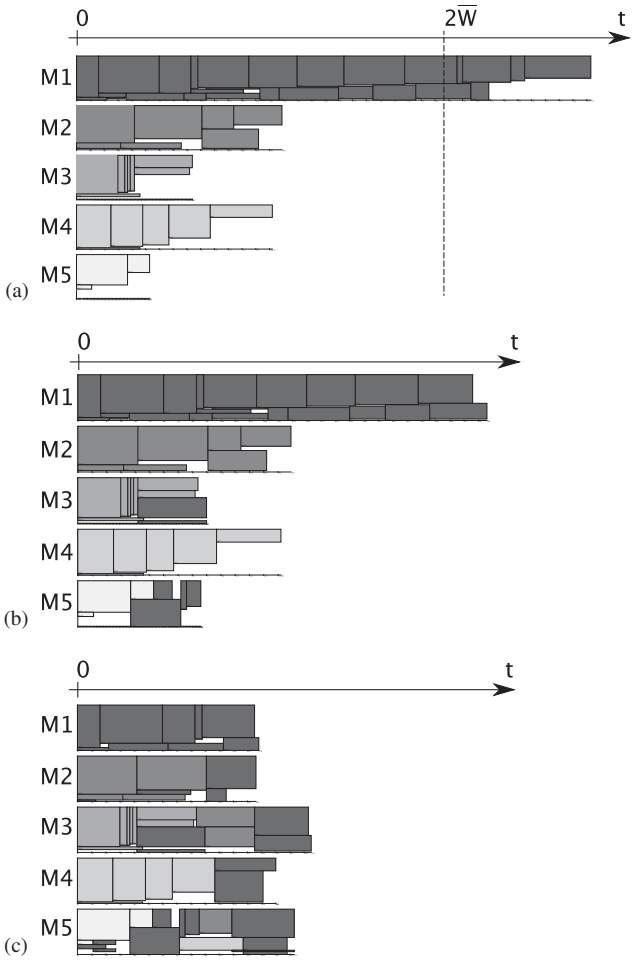


Figure 5. An example instance with $N = 5$ organizations. Jobs of different organizations are denoted by different shades of gray. MOLBA(2) (b) improves the system-wide makespan compared with the local scheduling (a). ILBA (c) (Section 5) improves both the system makespan and the local makespans of O_1, O_2 and O_4.

DOI: 10.1002/cpe

4.3. General properties

We prove in Sections 4.4 and 4.5 that MOLBA(3) is a 4-approximation of the global makespan $C_{\max}$ in the general case and that MOLBA(2) is a 3-approximate algorithm when the last completed task is a low task. We also show that these algorithms do not increase the local makespans of the organizations (therefore holding the constraint in Equation (1)). We start with a lemma that characterizes the structure of all the clusters' schedules.

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

Lemma 1. *In the schedule returned by* MOLBA(α), *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; hence, 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 $p_{\max}$. □

Lemma 2. *After* MOLBA(α) *finishes*, *there is at least one cluster with* $t_L^{start} \leq 2\bar{W}$.

Proof. The proof is by contradiction. Suppose that there exists $\varepsilon>0$ such that all the clusters have high utilization from 0 till $2\bar{W} + \varepsilon$. Then, the total surface of jobs computed by all the clusters is greater than $2\bar{W} \cdot mN \cdot 0.5 = W$, i.e. greater than the total work available, which leads to a contradiction. □

4.4. Approximation ratio for arbitrary jobs

Let us now consider the case where the last completed job can have any height. MOLBA(3) has an approximation ratio of 4 in the general case.

Proposition 2. *MOLBA*(3) *is a* 4*-approximate algorithm and all the organizations have incentive to cooperate.*

Proof. Let us prove this proposition by contradiction (see Figure 6 for an illustration). Let $C^*_{\max}$ be the makespan of an optimal schedule, and let us suppose that a job starts after time $3C^*_{\max}$ in the schedule returned by MOLBA(3). This means that this job could not have been started before: for all $i \in \{1, \ldots, n\}$, $C_{\max}(M_i) \geq 3C^*_{\max}$. Lemma 1 shows that, on each cluster, the zone where at most half of the processors are busy is smaller than or equal to $p_{\max} \leq C^*_{\max}$. Thus, on each cluster, the zone where at least half of the processors are busy is larger than or equal to $2C^*_{\max}$. As

DOI: 10.1002/cpe

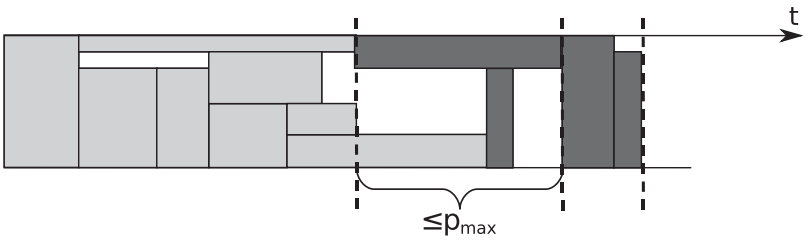


Figure 6. *Low* foreign jobs (in dark gray) fill the zone of low utilization at the end of the schedule of local jobs (in light gray). *High* foreign jobs are placed at the end of the schedule.

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

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

There are some special cases in which the presented approximation ratio can be improved. When all the jobs are *high*, no two jobs can be scheduled in parallel on one cluster. Thus, the problem corresponds to scheduling sequential jobs on N processors. Any list scheduling algorithm is, in this case, $2 - 1/N$ approximate. It is straightforward to guarantee that all the organizations have incentive to cooperate. Each job J_i is scheduled on its local machine M_j, unless there is a free machine M_k that has already scheduled all its local jobs at the moment when J_i would start on M_j: in this case J_i is scheduled on M_k.

4.5. Approximation ratio for low jobs

We show in this section that, in the schedule returned by MOLBA(2), if the last completed job is a *low* job, then MOLBA(2) is a 3-approximate algorithm.

Proposition 3. *The makespan of the schedule returned by* MOLBA(2) *is a 3-approximation of the optimal makespan, if the last completed job is* low. *Moreover, all the organizations have incentive to cooperate.*

Proof. If the last job is *low* it is started no later than $2\bar{W}$. Indeed, by Lemma 2, there is at least one cluster whose $t_L^{start} \leq 2\bar{W}$, and in the final schedule the utilization cannot be high on all the clusters until a time larger than $2\bar{W}$, as this would mean that the total work is larger than W. As we use a list algorithm to schedule the migrated jobs, no low job starts after $2\bar{W}$. Therefore, $J_{k,i}$ finishes before $2\bar{W} + p_{k,i} \leq 2\bar{W} + p_{\max} \leq 3C^*_{\max}$.

The proof that all the organizations have incentive to cooperate is analogous to the proof of Proposition 2. □

DOI: 10.1002/cpe

5. ITERATIVE LOAD BALANCING ALGORITHM

MOLBA(3) guarantees that the global makespan $C_{\max}$ is not higher than $4C^*_{\max}$. Nevertheless, the load distribution resulting from running the algorithm can be uneven. For instance, MOLBA(3) does not modify the schedule of an organization with $C_{\max}(O_k) = 4C^*_{\max} - \varepsilon$. Yet, it is possible that jobs on M_k are not yet scheduled while the other clusters are idle. The Iterative load balancing algorithm (ILBA) presented in this section improves the schedule returned by MOLBA(α) by balancing the load between clusters.

Note that, as ILBA does not take into account the owner of a job, in this section we will omit the index of organization, as in Section 3.

5.1. Algorithm description

ILBA starts by sorting clusters by non-decreasing makespans. Suppose that $C_{\max}(M_1) \leq C_{\max}(M_2) \leq \cdots \leq C_{\max}(M_N)$. The algorithm, for all $k = 2, \ldots, N$, reschedules all the jobs $\mathscr{J}_k$ executed on M_k on $\{M_1, \ldots, M_k\}$. Jobs $\mathscr{J}_k$ are scheduled sequentially in the order of execution of the original schedule. Each job J_i is allocated to the cluster that has an earliest strip of free processors of width of at least p_i and height of at least q_i.

An example of applying ILBA on a schedule returned by MOLBA is presented in Figure 5(c).

ILBA does not delay any job. Therefore, given a schedule produced by MOLBA(α), ILBA fulfills the constraint in Equation (1) and does not worsen the approximation ratio (see Section 5.2 for a formal proof).

Note that many other load balancing policies are possible (such as redistributing load in pairs or starting with the most loaded machine). However, by starting with the least loaded clusters and iteratively adding the more loaded ones, ILBA permits also to reduce the makespans on the less loaded clusters. Consequently, the resulting load distribution is more equitable.

5.2. Principle of algorithm

ILBA does not delay any job and thus does not increase the makespan of any organization. The following proposition formally states this result.

Proposition 4. *ILBA does not delay any job when compared with the base schedule.*

Proof. As ILBA considers clusters sequentially, it is sufficient to show that the proposition holds for any cluster M_k (as the jobs from the following clusters do not modify the jobs already scheduled).

Let us assume that jobs are numbered according to non-decreasing start times in the base schedule.

The proof is by induction on the scheduled jobs. In ILBA, a scheduled job J_i cannot be influenced by jobs $J_{i+1}, \ldots, J_n$ scheduled afterward (or by jobs incoming from different clusters in the subsequent parts of the algorithm). Note also that at this phase of the algorithm, no new job is allocated to M_k. Thus, it is sufficient to show that if jobs $J_1, \ldots, J_{i-1}$ are completed no later than in the original schedule, J_i is also not delayed.

The proposition trivially holds for J_1, as the first job will be started at $t = 0$.

DOI: 10.1002/cpe

Let us now assume that jobs $J_1, \ldots, J_{i-1}$ are scheduled no later than their start times in the original schedule. Consequently, $J_1, \ldots, J_{i-1}$ finish no later than in the original schedule. Thus, at J_i's original start time, the number of free processors is at least the same as in the original schedule. Hence, J_i can be scheduled at its original start time. J_i is migrated only if it can be started earlier than this original start time. Consequently, J_i finishes at the latest when it finished in the base schedule. □

6. SIMULATIONS

In this section, we carry out the experimental evaluation of the proposed algorithms. We start with performance measures and the methods used to generate workload. Then, we show the performance of the considered algorithms.

6.1. Methods

We compare three algorithms:

Local: This algorithm, for each organization, schedules its jobs on its local cluster with a HF list scheduling algorithm; this solution is used as a reference, non-cooperative solution.

MOLBA: The algorithm presented in Section 4, that $\alpha = 2$;

ILBA: The algorithm presented in Section 5, which iteratively improves the schedule returned by MOLBA(2), yet it has the same worst-case performance.

We choose to use MOLBA with $\alpha = 2$ as a trade-off between performance (better when α is smaller) and worst-case guarantees (as there are no guarantees for $\alpha < 2$).

Note that MOLBA(2) is an approximation algorithm only when the last job is *low*. The theoretical analysis in Section 4.4 showed that, in the general case, MOLBA(3) should be used. In our simulations, we use the following meta-algorithm. Firstly, we run MOLBA(2). Then, if in the resulting schedule any organization had its makespan increased, or the global makespan is greater than $3\max(p_{\max}, \bar{W})$, the schedule is discarded and MOLBA(3) is run instead. However, during our simulations, those conditions were never fulfilled. Thus, all the presented results are obtained with MOLBA(2).

The algorithms are compared with respect to the system-wide makespan that was produced. As the instances differ, e.g. in the number of jobs or their sizes, for each instance and each algorithm we compute the ratio of the produced makespan to the lower bounds defined in Section 4.1, i.e.

$$s(alg, I) = \frac{C_{\max}(alg, I)}{\max(\bar{W}(I), p_{\max}(I))}$$

where $s(alg, I)$ is the algorithm alg score on instance I, $C_{\max}(alg, I)$ is the makespan of the schedule produced by algorithm alg on instance I, $\bar{W}(I)$ is the average work in this instance, and $p_{\max}(I)$ is the length of the longest job in the instance. Thus, $s(alg, I)$ is an upper bound of the approximation ratio of alg on instance I. Low values of $s(alg, I)$ denote better performance.

DOI: 10.1002/cpe

We test the algorithms on two sets of randomly generated instances.

- *uni* instances in which the sizes $p_{k,i}$ and the numbers of required processors $q_{k,i}$ of jobs are produced by two independent, uniform distributions over, respectively, $\{1, \ldots, 50\}$ and $\{1, \ldots, m\}$.
- *swf* instances produced by a realistic workload generator [9]. We adjust the generator as follows. The maximum required number of processors is smaller than m. The number of generated jobs is set to n. All the release dates are set to 0.

Each instance consists of n (a parameter of the simulations) jobs. To determine the owner O_k of each job $J_{x,i}$, we use a Zipf distribution. We set the number of elements equal to the total number of organizations and the exponent equal to 1.4267. This parameter corresponds to that found in [10], which analyzes real-world data from virtual organizations and their submitted jobs.

Both *uni* and *swf* instances are generated with the total number of organizations $N \in \{2, 5, 10, 20\}$, the total number of jobs $n \in \{10, 50, 100, 500\}$, and the number of processors in each cluster $m \in \{32, 128, 512\}$. N is chosen to represent academic grids, where the number of participating laboratories is often around 10. The number of processors varies between small and large clusters currently used in grid systems. For instance, clusters in Grid'5000 [11] currently have between 32 and 342 nodes.

For each possible distribution type (*uni* and *swf*) and each possible combination of values of n, N and m, 50 instances are generated. In total, 4800 instances are generated.

6.2. Analysis of results

The average scores of algorithms in function of the total number of jobs n and the number of organizations N are presented in Figure 7. A number of phenomena can be observed.

6.2.1. The inefficiency of local scheduling

The inefficiency of *local* scheduling can be observed in *uni* instances (Figure 7(a)). The average score of the *local* algorithm deteriorates quickly as the number of organizations increases. The trend is even more visible when the smallest instances ($n = 10$) are omitted. The average score grows from 1.57 ($N = 2$), through 3.00 ($N = 5$) and 4.93 ($N = 10$), till 7.35 ($N = 20$). The linear correlation coefficient between N and the score of *local* is equal to 0.69 for *uni* (and 0.38 for the both data sets combined). This supports our claim that in larger systems the drop in performance resulting from the lack of cooperation is proportional to the size of the system N.

6.2.2. Improvement by ILBA

The final refinement of schedules, performed by ILBA, considerably improves the results in all settings. The overall average score for *uni* instances drops from 1.96 (for MOLBA(2)) to 1.25 (for ILBA), whereas for *swf* it drops from 1.09 (for MOLBA(2)) to 1.03 (for ILBA). Moreover, ILBA returned optimal schedules (with score equal to 1) for at least 40% of instances, whereas MOLBA(2) returned such schedules in 29% of the considered instances. Note that the score of an optimal schedule can be higher than 1 (as we are comparing the makespan with its lower bound). In

DOI: 10.1002/cpe

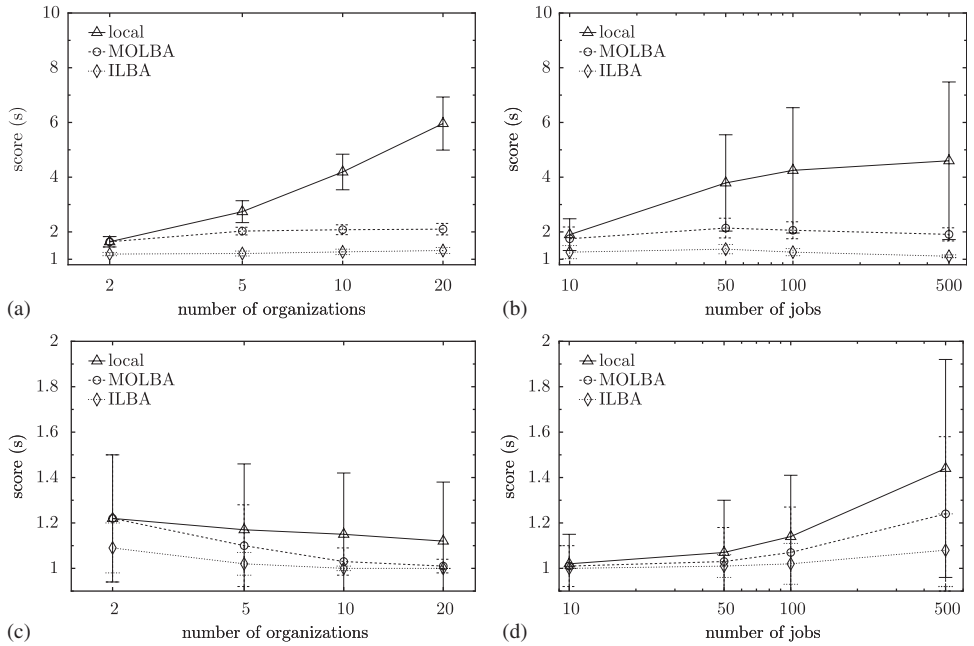


Figure 7. Average scores s (upper bounds for the approximation ratio) in function of number of jobs n or the number of organizations N. Data averaged over instances and the number of processors in each cluster. Error bars denote standard deviation: (a) *uni*, average over n; (b) *uni*, average over N; (c) *swf*, average over n; and (d) *swf*, average over N.

this case, even if an algorithm returns the optimal schedule, such a simple analysis will not consider it as an optimal.

6.2.3. *Performance on realistic instances*

The performance of all the algorithms is much better on instances created with the realistic workload generator (*swf* data set, Figures 7(c) and (d)). Because of the increased diversity of both the sizes of jobs and the number of required processors, these instances simply schedule better. For example, in *swf* data set, ILBA was optimal in all the smallest instances considered (with $n = 10$), in which the makespan was determined by the length of the longest job.

6.2.4. *Influence of load increase*

The lack of cooperation makes the system inefficient under high load, i.e. when the number of jobs n is large (Figures 7(b) and (d)). For all but *uni* instances with $N = 2$ and 5, the average score of *local* scheduling rises with n. The differences of load between organizations result from the Zipf distribution. The increased number of jobs makes the load differences more visible. Scores of

DOI: 10.1002/cpe

MOLBA(2) and ILBA deteriorate in *swf* instances but improve in *uni* instances (after omitting the smallest instances). In *swf* instances, the deterioration is directly caused by the increased number of jobs. In contrast, jobs produced by the uniform distribution are, in general, harder to match and, thus, to pack efficiently. If the number of such jobs is increased, it is easier to match jobs of different sizes, and thus the algorithm achieves better packing (note that the resulting approximation ratio is still higher than for *swf* instances).

6.2.5. Influence of the number of processors

The number of processors on each node m does not influence the performance of the considered algorithms. After averaging over all the other variables, p-values of two-tailed, type 3 t-test, for $m = 32$ and 128 are 0.35, 0.17 and 0.53, for the *local*, MOLBA(2) and ILBA scores, respectively. Similarly, for the difference between $m = 128$ and 512, we have 0.56, 0.62 and 0.35. Consequently, we do not present results averaged over different values of m.

6.2.6. Worst cases

The worst result for the *local* algorithm was 12.12, for a *uni* instance with $N = 20$ organizations and $n = 100$ jobs. The makespan is determined by a number of relatively short jobs executed on one cluster. The worst result of MOLBA(2) was 2.98 for a *uni* instance with $N = 20$ and $n = 50$. In this instance, the makespan is determined by an organization that does not take part in the MOLBA(2) algorithm, as its local makespan is between $2\bar{W}$ and $2\bar{W} + p_{\max}$. The worst result of ILBA is 1.92 (also for $N = 20$ and $n = 50$), which is significantly lower that the theoretical upper bound of 3. In this instance, the last finished job is a long, *high* job, started after a period of low utilization.

6.3. Summary of the results

The results of our simulations show that MOLBA(2) with ILBA is an efficient algorithm that considerably improves the performance in comparison with local scheduling. Even the worst observed results remain far from the theoretical approximation ratios. At the same time, no organization saw its makespan increased. We also observed that the workloads generated by the realistic workload generator were easier to schedule, and thus the improvement of MOLBA(2) with ILBA was reduced compared to uniform instances.

7. RELATED WORK

In this paper we have studied the incentive for independent parties to collaborate. We have claimed that, if a proposed, collaborative solution does not deteriorate 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 that has a worst-case guarantee on the social goal (the makespan of the system), at the same time respecting selfish goals of participants.

DOI: 10.1002/cpe

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 [12] studies situations in which a set of selfish users (called players) optimize their own objective functions, which also depend on strategies undertaken by other players. Moreover, in order to measure the performance of the system as a whole, a *social* (global) objective function is often defined. The central notion is the *Nash equilibrium* [13], a situation in which no player can improve its own objective function by unilaterally changing his/her strategy.

The ratio between the values of the social objective function in the worst Nash equilibrium and in the optimal solution is called the *price of anarchy* [14]. The price of anarchy is interpreted as the cost of the lack of cooperation. For example, in the context of scheduling, Christodoulou *et al*. [15] measure the price of anarchy when selfish sequential jobs choose one of the available processors: the goal of each job is to minimize its completion time, whereas the social goal is to minimize the makespan of the schedule.

A related measure, the *price of stability* [16,17], compares the socially best Nash equilibrium with the socially optimal result. Usually, in order to reach such an equilibrium, a centralized protocol gathers information from and then suggests a strategy to each player. As the proposed solution is a Nash equilibrium, the players do not have incentive to unilaterally refuse to follow it. For example, Angel *et al*. [18] compute the price of stability in the same model as [15], but relaxes the selfishness of jobs by a factor α and studies the trade-off between α and the approximation ratio of the global makespan. The price of stability can be interpreted as the cost due to the fact that players optimize their own objective functions rather than the global objective function. It is possible to view our problem as a non-cooperative game where the players are the organizations (which have the possibility to accept the proposed solution or to choose to execute their own jobs locally) and the social goal is to minimize the global makespan. The collaborative solution proposed by our algorithm approximates the socially best Nash equilibrium, because it optimizes the social goal with a guarantee that no player has incentive to deviate from the proposed solution.

In cooperative game theory [12], players still have their selfish goals, but they can communicate and enforce agreements on their strategies. In the resulting game, coalitions rather than individual players compete with each other. The payoff of such a game describes how much a set of players gains by forming a coalition. The players choose which coalitions to form, according to the way the allocation of gains obtained by the coalition will be divided among the coalition members. In cooperative games, the goal is often to find division rules with which every player gains most by joining the grand coalition that gathers all the players. However, cooperative game theory usually requires transferable payoffs, i.e. that the coalition could divide its gain between the members in an arbitrary manner. This is certainly not the case in our problem, as we assume that the only payoff of a player is the completion time of his/her jobs.

Papers proposing *distributed* resource management or distributed load balancing usually solve the problem of optimizing a common goal with a decentralized algorithm. Liu *et al.* [19] show a fully decentralized algorithm that always converges to a steady state. Rotaru and Nageli [20] present a similar algorithm with the divisible load job model. Those approaches contrast with our algorithm. In our approach, a centralized algorithm respects decentralized goals of participants. We are, however, aware that a load balancing algorithm in large-scale systems must be decentralized. In [21], a distributed algorithm balances selfish, identical jobs on a network of identical processors. There are several differences with our work as there is no notion of organization, and agents are

DOI: 10.1002/cpe

jobs whose aim is to be on the least loaded machine. Berenbrink *et al*. [21] focus on the time needed to converge to a Nash equilibrium and does not consider, like we do, a social goal to optimize (e.g. the price of anarchy is not computed).

Alternative approaches propose to balance the load by implicit barter trade of CPU power [22] or by an explicit computational economy [23].

8. 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 and, at the same time, to improve the performance of the whole system. The cooperative solutions have a constant worst-case performance, a significant gain compared with selfish solutions that can be arbitrarily far from the optimum. Moreover, proposed algorithms turned out to be efficient during experimental evaluation by simulation.

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

In our future work, we would like to, firstly, extend the presented algorithms to heterogeneous clusters, and, secondly, study the effect of the increased effort of individuals on the global goal, especially in the case of an on-line system. 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 β times local solution'.

ACKNOWLEDGEMENTS

The authors acknowledge the helpful suggestions of the anonymous reviewers. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr). This work has been done when Fanny Pascual was postdoc in the INRIA project MOAIS. This research was partly supported by the FP6 Network of Excellence CoreGRID funded by the European Commission IST-2002-004265.

REFERENCES

1. Foster I, Kesselman C (eds.). *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 [4 July 2008].
3. Blazewicz J, Ecker KH, Schmidt G, Weglarz J. *Scheduling in Computer and Manufacturing Systems*. Springer: Berlin, 1996.
4. Shmoys D, Wein J, Williamson D. Scheduling parallel machines on-line. *SIAM Journal on Computing* 1995; **24**: 1313–1331.

DOI: 10.1002/cpe

5. Garey M, Johnson D. *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*. Freeman: New York, NY, 1979.
6. Graham R. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics* 1969; **17**(2):416–429.
7. Eyraud-Dubois L, Mounie G, Trystram D. Analysis of scheduling algorithms with reservations. *Proceedings of IPDPS*. IEEE Computer Society Press: Silver Spring, MD, 2007.
8. Lifka D. The ANL/IBM SP scheduling system. *Job Scheduling Strategies for Parallel Processing* (*Lecture Notes in Computer Science*, vol. 949). Springer: Berlin, 1995.
9. Lublin U, Feitelson D. The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing* 2003; **63**(11):1105–1122.
10. Iosup A, Dumitrescu C, Epema D, Li H, Wolters L. How are real grids used? The analysis of four grid traces and its implications. *Seventh IEEE/ACM International Conference on Grid Computing*, Barcelona, Spain, 2006; 262–269.
11. Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, Jégou Y, Lantéri S, Leduc J, Melab N, Mornet G, Namyst R, Primet P. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications* 2006; **20**(4):481–494.
12. Osborne M, Rubinstein A. *A Course in Game Theory*. MIT Press: Cambridge, MA, 1994.
13. Nash J. Non-cooperative games. *Annals of Mathematics* 1951; **54**:286–295.
14. Koutsoupias E, Papadimitriou C. Worst-case equilibria. *Proceedings of STACS* (*Lecture Notes in Computer Science*, vol. 1563). Springer: Berlin, 1999; 404–413.
15. Christodoulou G, Koutsoupias E, Nanavati A. Coordination mechanisms for selfish scheduling. *Proceedings of ICALP* (*Lecture Notes in Computer Science*, vol. 2719). Springer: Berlin, 2003; 345–357.
16. Schultz AS, Stier Moses N. On the performance of user equilibria in traffic networks. *Proceedings of SODA*. ACM: New York, 2003; 86–87.
17. Anshelevich E, Dasgupta A, Kleinberg JM, Tardos É, Wexler T, Roughgarden T. The price of stability for network design with fair cost allocation. *Proceedings of FOCS*, Rome, Italy, 2004; 295–304.
18. Angel E, Bampis E, Pascual F. The price of approximate stability for a scheduling game problem. *Proceedings of Euro-Par* (*Lecture Notes in Computer Science*, vol. 4128). Springer: Berlin, 2006.
19. Liu J, Jin X, Wang Y. Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization. *IEEE Transactions on Parallel and Distributed Systems* 2005; **16**(7):586–598. DOI: http://doi.ieeecomputersociety.org/10.1109/TPDS.2005.76.
20. Rotaru T, Nageli HH. Dynamic load balancing by diffusion in heterogeneous systems. *Journal of Parallel and Distributed Computing* 2004; **64**(4):481–497. DOI: http://dx.doi.org/10.1016/j.jpdc.2004.02.001.
21. Berenbrink P, Friedetzky T, Goldberg L, Goldberg P, Hu Z, Martin R. Distributed selfish load balancing. *Proceedings of SODA*. ACM: New York, 2006; 354–363.
22. Andrade N, Cirne W, Brasileiro F, Roisenberg P. Ourgrid: An approach to easily assemble grids with equitable resource sharing. *Proceedings of JSSPP* (*Lecture Notes in Computer Science*, vol. 2862). Springer: Berlin, 2003; 61–86.
23. Buyya R, Abramson D, Venugopal S. The grid economy. *Special Issue on Grid Computing*, vol. 93. IEEE Press: New York, 2005; 698–714.