Approximation algorithms for the Multi-Organization Scheduling Problem

Pierre-François Dutot, Fanny Pascual, Krzysztof Rzadca, and Denis Trystram

Abstract

The distributed nature of new computing platforms results in the problem of scheduling parallel jobs produced by several independent organizations that have each their own rules. They have no direct control over the whole system, thus it is necessary to revisit classical scheduling with locality constraints. In this article, we consider distributed computing systems in which each organization has its own resources. Each organization aims at minimizing the execution times of its own jobs. We introduce a global centralized mechanism for designing a collaborative solution that improves the global performance of the system while respecting organizations' selfish objectives. The proposed algorithm is proved to have an approximation ratio equal to 3 over the global optimal makespan and this bound is shown to be asymptotically tight (when the number of organizations is large). Several variants of this problem are also studied. Then, we derive another algorithm that improves in practice these solutions by further balancing the schedules. Finally, we provide some experiments based on simulations that demonstrate a very good efficiency of this last algorithm on typical instances.

- *P.-F. Dutot, K. Rzadca and D. Trystram are with the LIG, Grenoble University, 51 avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France.*
- *F. Pascual is with the LIP6, Université Pierre et Marie Curie - Paris 6, 4 place Jussieu, 75005 Paris, France.*
- *K. Rzadca is also affiliated to SCE, Nanyang Technological University, Blk N4, North Spine, Nanyang Avenue, Singapore 639798.*

Approximation algorithms for the Multi-Organization Scheduling Problem

1 Introduction

When buying a production cluster, a company has to compromise between the price and the expected computing power. However, the job submissions usually follow an irregular pattern with alternative periods of peak and low utilization. The decision is then to own an oversized cluster which will sometimes be under-used or to export jobs to other computing facilities when the submissions are heavy. In the grid computing paradigm [1] several organizations share their computing resources in order to distribute peak workloads over all the participants. An organization is an administrative entity grouping users and computational resources such as private companies or academic research groups. The system has only a low central administrative control [2], and 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 computational grid, the resource management system must achieve an acceptable performance not only at the level of the community of users (as in classical scheduling), but also at the inter-organizations level. Some globally-optimal approaches may be unacceptable because they implicitly favor jobs produced by specific organizations, 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* (composed of identical processors), that together form a computational grid. The global objective is to minimize the makespan [3], defined as the time when all the jobs are completed. 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 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 classical parallel job scheduling [3] by the constraints stating that no organization's makespan can be worsened.

The main contribution of this article is the demonstration that it is always in the interest of several independent organizations to collaborate in a computational grid system composed of independent clusters. This problem (called MOSP for Multi Organization Scheduling Problem) has been introduced in [4]. We propose a new algorithm producing solutions that guarantee that no organization's makespan is increased. This algorithm is proved to have a constant approximation ratio (worst-case performance) regarding the globally-optimal solution. More precisely, assuming that each job can fit into every cluster, the proposed algorithm achieves a 3-approximation if the local schedules are obtained by the highest first list scheduling policy (HF in short). For any other local policy, the approximation ratio is equal to 4. Then, we improve these solutions by a centralized iterative load-balancing that is hard to analyze theoretically. We run many experiments that assess that the proposed algorithms work very well on average.

This paper is organized as follows. Section 2 formally defines the model and the problem and presents some motivating examples. It also introduces the notations used in the rest of the paper. Section 3 considers the basic problem of scheduling local jobs on a single organization cluster with list algorithms. In particular, we analyze the highest first scheduling policy (HF) where the jobs are sorted according to the decreasing number of required processors. This algorithm is a 2-approximation and we exhibit interesting structural properties that are useful for the general analysis. Section 4 presents the algorithm for N organizations and proves that it is a 3-approximation. This bound is shown to be asymptotically tight (for a large number of organizations). In section 5, we discuss some extensions including the case where the local scheduling policies are not HF leading to a 4-approximation ratio and the case where the jobs are sequential. Section 6 describes a load-balancing algorithm that is used to improve the results of the base algorithm. Results of experiments are analyzed in Section 7. Related studies are discussed in Section 8. Finally, Section 9 summarizes the obtained results and concludes the paper.

2 Preliminaries

2.1 Model of the multi-organization Grid and notations

The basic model is a computational grid composed of independent clusters belonging to various organizations. $\mathcal{O} = \{O_1, \dots, O_N\}$ denotes the set of independent organizations. Each organization O_k owns a cluster M_k ($1 \leq k \leq N$). Each M_k is composed of m_k identical processors, and without loss of generality, let us assume that $m_1 \geq m_2 \geq \cdots \geq m_N$. Finally, let $m = \sum m_k$.

The set of all the jobs *produced* by O_k is denoted by $\mathcal{I}_k$, whose elements are $\{J_{k,i}\}$. $\mathcal{J}_k$ denotes the set of

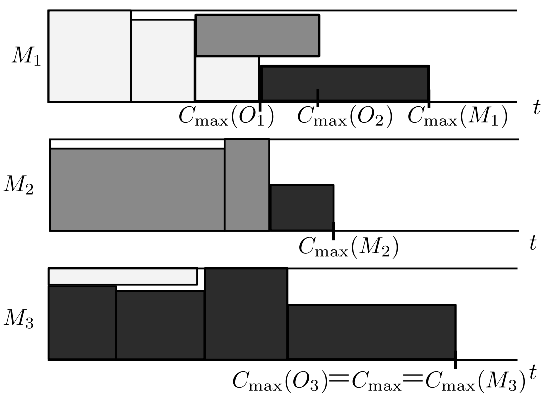


Fig. 1. Example of a schedule with three organizations: O_1 produces light gray jobs, O_2 – gray, and O_3 – dark gray. O_1 has the earliest makespan $C_{\max}(O_1)$. Cluster M_1 is then used for executing jobs of the other organizations. The makespan of M_1 corresponds to the last completion time of a job belonging to O_2, thus, it is equal to $C_{\max}(O_2)$. The global makespan $C_{\max}$ is determined by O_3 on M_3.

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* to another cluster. The underlying computational model is parallel rigid jobs where $J_{k,i}$ must be executed in parallel on $q_{k,i}$ processors of exactly one cluster during $p_{k,i}$ time units [5]. It is not possible to distribute a job between two, or more, clusters. $p_{\max} = \max_{i,k}(p_{k,i})$ denotes the maximum length of the jobs. $J_{k,i}$ is said a *low* job with respect to cluster M_k if its height $q_{k,i}$ is at most equal to $\frac{m_k}{2}$, otherwise it is *high*. Moreover, we assume that all the jobs can fit in all clusters, which means that each job has a height lower than or equal to each cluster size ($\max q_{k,i} \leq m_N$).

$C_{k,i}$ denotes the completion time of job $J_{k,i}$. For an organization O_k, the maximum completion time (makespan) is computed as $C_{\max}(O_k) = \max_i(C_{k,i} : J_{k,i} \in \mathcal{I}_k)$. The global makespan $C_{\max}$ is the maximum makespan over all the organizations, $C_{\max} = \max_k C_{\max}(O_k)$. All these notations are natural extensions of standard notations in scheduling, they are illustrated in Figure 1.

For cluster M_k, a *schedule* is an allocation of jobs $\mathcal{J}_k$ to processors together with a starting time, such that at each time, no processor is assigned to more than one job. The *makespan* $C_{\max}(M_k)$ *of cluster* M_k is defined as the maximum completion time of jobs $\mathcal{J}_k$ assigned to that cluster, $C_{\max}(M_k) = \max_{i,j}(C_{j,i} : J_{j,i} \in \mathcal{J}_k)$. 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_k. A *scheduler* is an application which 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 [6] and which require the user to define the run-time of the submitted jobs. The objective of each organization O_k is 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 the local cluster M_k, if the last job to be executed does not belong to 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.

Informally, the *Multi-Organization Scheduling Problem* (MOSP) corresponds to 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 local jobs. More formally, let $C^{loc}_{\max}(O_k)$ be the makespan of O_k when the set of jobs executed by M_k (that is $\mathcal{J}_k$) corresponds only to local jobs. MOSP is defined as:

$$\text{Minimize } C_{\max} \text{ such that}$$

$$\forall k \in \{1, \ldots, n\} \; C_{\max}(O_k) \leq C^{loc}_{\max}(O_k). \qquad (1)$$

In the sequel, we denote by $C^*_{\max}$ the makespan of an optimal solution of the problem where the aim is to minimize the makespan without local constraints. Thus, $C^*_{\max}$ is a lower bound of the makespan of an optimal solution of MOSP. We denote by LB a lower bound of the makespan of an optimal solution of MOSP: $LB = \max\left(\frac{\sum_{k,i} q_{k,i} p_{k,i}}{\sum_k m_k}, \max_{k,i}(p_{k,i})\right)$.

The complexity analysis of MOSP is straightforward by restricting the number of organizations to $N = 1$, the size of the cluster to $m = 2$ and the jobs to sequential ones ($q_{k,i} = 1$). In this case, MOSP is exactly the classical scheduling problem $P2||C_{\max}$ which is NP-hard [7].

2.3 Motivation for cooperating

Let us first recall that there exist several papers addressing the scheduling of centralized jobs on hierarchical platforms with several clusters without local constraints. A theoretical analysis of the problem has been studied in [8] where the problem has been proved to be NP-hard and any list scheduling policy is arbitrarily far from the optimal (for a large number of clusters). Then, a first algorithm based on several separate lists has been proposed in [9], it leads to a 3-approximation in the case of non-clairvoyant jobs (it also holds for clairvoyant jobs with a larger ratio). A complete theoretical analysis has been provided recently in [10] leading to costly polynomial approximation schemes and shelves-based algorithms. A recent alternative approach based on evolutionary fuzzy systems has been proposed in [11].

Many cases motivate independent organizations to cooperate and accept non-local jobs, even if the resulting configuration is not necessary globally optimal. A non-cooperative solution is that all the organizations compute their jobs on their local clusters. However, such a

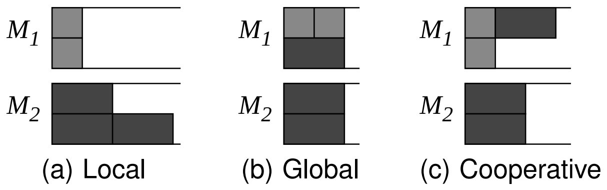


Fig. 2. Globally-optimal solution (b) is inadmissible, as it extends the makespan of organization O_1 in comparison with the local solution (a). The best solution which does not increase any local makespan (c) has a larger value than the global optimum.

solution can be arbitrarily far from the global optimum. Let us consider the following instance: Organization O_1 owns N jobs of unit length and all the others (from O_2 to O_N) have no local jobs. The global makespan obtained without cooperation is equal to N while the optimal makespan is 1 when each cluster executes one job. Thus, the ratio is arbitrarily large when the number of organizations increase. Let us remark also that careful scheduling may offer more than simple load balancing of the previous example. By matching certain types of jobs, bilaterally profitable solutions are also possible. Nevertheless, a certain price must be paid in order to produce solutions in which all the organizations have incentive to cooperate. Figure 2 presents a lower bound of 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 a makespan equal to $\frac{3}{2}$ of the value of the globally-optimal solution ($C^*_{\max}$). Let us remark that this bound still holds for the sub-case where the machines are reduced to only one processor.

3 Scheduling on One Cluster

Let us first focus on the preliminary case of scheduling rigid parallel jobs on one cluster of m identical processors. In this section, we define a job as *low* if its height is at most $\frac{m}{2}$; otherwise it is *high*. Notice 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 the classical list scheduling algorithm, which has an approximation ratio equal to $2-\frac{1}{m}$ [12]. We show that if the jobs are ordered according to decreasing number of required processors, the resulting schedule achieves fairly homogeneous utilization.

3.1 List Scheduling

List scheduling [12] is a class of algorithms which work as follows. First the jobs are ordered into a list. Then, the schedule is constructed by assigning jobs to processors in a greedy manner. Let us assume that at time t, m' processors are idle in the schedule under construction. The scheduler chooses the first job J_i of the list requiring no more than m' processors, it schedules the job at time t, and finally it removes the job 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 in its principle, list scheduling of rigid parallel jobs is an approximation algorithm with guaranteed worst case performance of $2 - \frac{1}{m}$ [13], no matter the order of jobs in the first phase.

3.2 Highest First (HF) Job Order

It is well-known that 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 losing the approximation ratio.

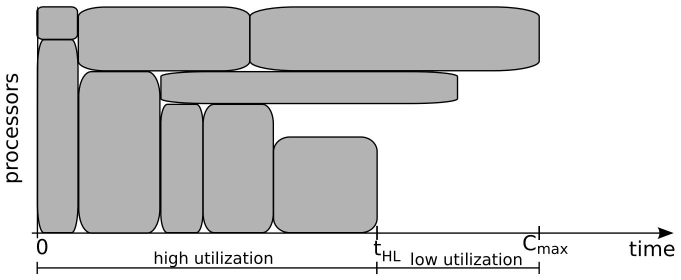


Fig. 3. When jobs are sorted according to the number of required processors, the schedule can be divided into two successive regions with utilization $U(t) > \frac{1}{2}$ (up to t_{HL}) and $U(t) \leq \frac{1}{2}$ (after this moment).

Assuming that the jobs are ordered according to HF, i.e. by non-increasing values of 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 $0 \leq t_{HL} \leq C_{\max}$. Moreover, for $t_{HL} \leq t \leq C_{\max}$, $U(t)$ is non-increasing and the period length $C_{max} - t_{HL}$ is lower or equal to LB. (Figure 3 illustrates the structure).

Proof: First, notice 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 noticed 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.

A similar argument can be used to show that no job starts after t_{HL}. Therefore, for $t_{HL} \leq t \leq C_{\max}$, $U(t)$ is non-increasing. Since the longest job has a processing time smaller or equal to LB, this also proves the property on the length of the low utilization period. □

The cost complexity for Local HF is of order $O(n_k \log n_k)$ on each cluster, where $n_k = |\mathcal{J}_k|$.

4 MOCCA

In this section we present an algorithm for solving MOSP. We start by presenting the principle of the algorithm and detail how it runs on an example. Then, a formal description is given before proving the approximation ratio.

4.1 Principle of the algorithm

MOCCA is an algorithm with a guaranteed worst-case performance on the global makespan. The algorithm consists of three phases. It starts by scheduling all the jobs on their local clusters using the Highest First ordering (HF) (described and analyzed in Section 3.2) in phase 1. Then, the algorithm unschedules the jobs that would break a given threshold (i.e., in the local schedules, those that finish after the performance bound equal to three times a lower bound of the optimal global makespan. This lower bound is determined by the maximum between the total workload divided by m and the maximum processing time).

All these unscheduled jobs are sorted according to the HF order. They are scheduled in phase 2 on all possible clusters as late as possible, from the threshold value backwards. These jobs are scheduled without interfering with the local schedules, i.e., already scheduled jobs ending before the threshold remain unchanged.

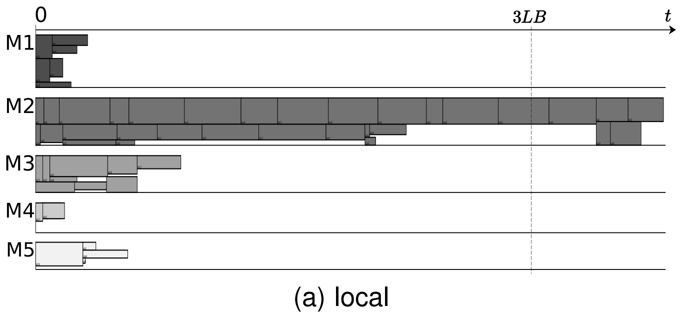
Then, the remaining (small) jobs are inserted in phase 3 using an HF list algorithm over all the clusters between the initial local jobs and the imported jobs. We will prove that following this procedure, all the jobs are scheduled. A formal description of the algorithm is given in Section 4.3.

4.2 Example Run

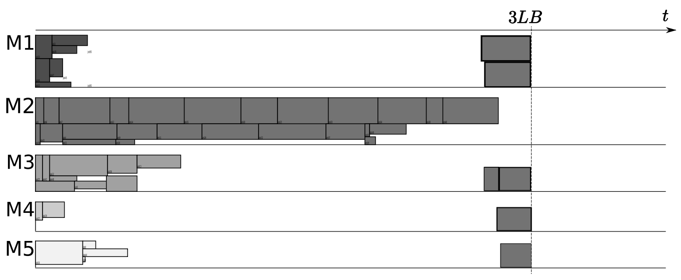
Figure 4 presents the result of a typical execution of MOCCA on an instance with $N = 5$ organizations. Scheduling all the jobs locally (Figure 4(a)) results in large global makespan $C_{\max}$, that is determined here by the makespan of M_2. O_2 is the only organization whose local makespan is larger than $3LB$. MOCCA (Figure 4(b)) improves $C_{\max}$ by rescheduling the last jobs of O_2 on the other clusters. Notice that no local makespan is increased in the resulting schedule. The global makespan meets the theoretical bound of 3 times the lower bound of $C^*_{\max}$. In Section 6 we show that a further improvement is possible.

4.3 Formal Description

In order to provide a complete and formal definition of MOCCA, some additional notations are introduced inside the algorithm description. At the end of phase 1, we denote by $C_k^{\geq j}$ the earliest time moment when at least j processors are idle on cluster M_k until the end of the schedule (or at least until a certain deadline D_k when D_k is defined). These values are computed immediately after phase 1. Let D_k be the deadline that it is fixed on machine M_k for the jobs scheduled in phase 2. Let us recall that the clusters are sorted in non-increasing order of number of processors ($m_1 \geq m_2 \geq \cdots \geq m_N$). In the sequel, (p, q) will denote a job using q processors with a processing time p. The cost complexity of MOCCA is $O(n \log n)$, where n is the total number of jobs.



(a) local



(b) MOCCA

Fig. 4. An example instance with $N = 5$ organizations. Jobs of different organizations are denoted by different shades of gray. MOCCA (b) improves the system-wide makespan comparing to the local scheduling (a).

```
Algorithm 1 MOCCA phase 1
Require: Local queues for all clusters
Ensure: A list of jobs named $Sorted$ and a partial schedule
  $LB \leftarrow \max\left(\frac{\sum_{k,i} q_{k,i} p_{k,i}}{\sum_k m_k}, \max_{k,i}(p_{k,i})\right)$
  for $k \leftarrow 1$ to $N$ do
    Schedule local jobs on $M_k$ using HF up to $3LB$.
  end for
  $Sorted \leftarrow$ remaining jobs sorted according to the number of processors required in non-increasing order.
  compute the values $C_k^{\geq j}$.
  for $k \leftarrow 1$ to $N$ do
    On each cluster $M_k$, the processors are numbered in non-increasing order of their completion times. Processor 1 ends its jobs the latest, while the $m_k$-th processor ends the soonest.
  end for
```

Algorithm 2 MOCCA phase 2

```
Require: A list of jobs $Sorted$ and a partial schedule
Ensure: A list of jobs $Unscheduled$ and a partial schedule
  $Unscheduled \leftarrow \emptyset$
  for $k \leftarrow 1$ to $N$ do
    $D_k \leftarrow 3LB$
  end for
  while $Sorted \neq \emptyset$ do
    $(p, q) \leftarrow$ first job of $Sorted$
    Remove job $(p, q)$ from $Sorted$
    $scheduled \leftarrow$ false
    $k \leftarrow N$
    while ((not $scheduled$) and ($\frac{m_k}{2} < q$) and ($k \geq 1$)) do
      if ($C_k^{\geq q} + p \leq D_k$) then
        Schedule job $(p, q)$ starting at time $D_k - p$ on $M_k$. The $q$ processors used are those which are idle at time $D_k - p$ and have the highest index.
        $D_k \leftarrow D_k - p$
        $scheduled \leftarrow$ true
      else
        // The job does not fit on $M_k$
        $k \leftarrow k - 1$
      end if
    end while
    if not $scheduled$ then
      Add job (p,q) to $Unscheduled$
    end if
  end while
```

Algorithm 3 MOCCA phase 3

```
Require: A list of jobs $Unscheduled$ and a partial schedule
Ensure: A complete schedule not longer than $3LB$
  $potential\_start \leftarrow \min_{j,k} C_k^{\geq j}$
  while $Unscheduled \neq \emptyset$ do
    $possible\_configurations \leftarrow \{(j,k) | C_k^{\geq j} = potential\_start\}$
    $max\_idle \leftarrow \max\{j | \exists k, (j,k) \in possible\_configurations\}$
    $scheduled\_one \leftarrow$ false
    $candidate\_jobs \leftarrow \{(p,q) | (p,q) \in Unscheduled$ and $q \leq max\_idle\}$
    Sort $candidate\_jobs$ in non increasing order of processor used
    while ($candidate\_jobs \neq \emptyset$ and not $scheduled\_one$) do
      $(p, q) \leftarrow$ First job of $candidate\_jobs$
      for $k \leftarrow 1$ to $N$ do
        if not $scheduled\_one$ and $C_k^{\geq q} \leq potential\_start$ and $potential\_start + p \leq D_k$ then
          Schedule $(p, q)$ starting at $potential\_start$ on $M_k$
          $scheduled\_one \leftarrow$ true
          Recompute the new $C_k^{\geq j}$ for all $j$ in $M_k$
          Remove job $(p, q)$ from $Unscheduled$
        end if
      end for
      if not $scheduled\_one$ then
        Remove job $(p, q)$ from $candidate\_jobs$
      end if
    end while
    if not $scheduled\_one$ then
      $next\_potential\_starts \leftarrow \{C_k^{\geq j} | C_k^{\geq j} > potential\_start\}$
      // The set $next\_potential\_starts$ is not empty (as proved in proposition 6).
      $potential\_start \leftarrow \min(next\_potential\_starts)$
    end if
  end while
```

4.4 Feasibility

The algorithm is guaranteed by construction to be a 3-approximation. We prove in this section that it is feasible (all the unscheduled jobs fit before $3LB$). First, we will start by some structural propositions which give important information on the partial schedules built at each step of MOCCA. Proposition 2 states that some clusters are computing their fair share of workload if certain jobs cannot be added during phase 2. Proposition 4 states that during phase 3 each time a job is scheduled on M_k, then the global processor usage is strictly greater than $\frac{m_k}{2}$ during an initial period, followed by a limited period of low utilization (when at most $\frac{m_k}{2}$ processors are used).

Proposition 2: During phase 2, when a job requiring q processors is inserted in the $Unscheduled$ set, all the clusters M_k such that $\frac{m_k}{2} < q$ have a workload W_k which is at least $m_k LB$.

Proof: When a job is inserted in the $Unscheduled$ set, the boolean variable $scheduled$ is false since the instruction is guarded by a test on this variable. Hence, the preceding *while* loop stopped either because $\frac{m_k}{2} \geq q$ (the job is no longer big enough for the remaining clusters) or $k < 1$ (all the clusters have been considered). Anyway, all the clusters M_k such that $\frac{m_k}{2} < q$ have been considered, and for each of them we have $C_k^{\geq q} + p > D_k$.

Since $p \leq LB$, the previous inequality can be simplified to $C_k^{\geq q} + (3LB - D_k) > 2LB$.

If $C_k^{\geq q} \leq LB$, the proof is obtained by a simple addition of the workload done before $C_k^{\geq q}$ and the workload done after D_k. According to Proposition 1, the utilization before $C_k^{\geq q}$ is either larger than $\frac{1}{2}$, or non increasing (and by definition of $C_k^{\geq q}$, larger than $\frac{m_k - q + 1}{m_k}$). After D_k, the utilization is at least $\frac{q}{m_k}$ since the jobs are added in non-increasing height. The workload W_k being larger than the sum of these two parts, we can conclude:

$$\begin{aligned} W_k &\geq C_k^{\geq q}(m_k - q + 1) + (3LB - D_k)q \\ &> C_k^{\geq q}(m_k - q + 1) + (2LB - C_k^{\geq q})q \\ &= C_k^{\geq q}(m_k + 1) + (2LB - 2C_k^{\geq q})q \\ &\geq C_k^{\geq q}(m_k + 1) + (LB - C_k^{\geq q})(m_k + 1) \end{aligned}$$

$$= \quad LB(m_k + 1)$$

If $C_k^{\geq q} > LB$, the proof is obtained by a slightly more complicated decomposition. First, let us remark that since phase 1 consists of a local HF list scheduling, where jobs are independent and shorter than LB, all jobs running at time $t + LB$ were not running at time t. Hence, for all t such that $U(t + LB) > 0$, we have $U(t) + U(t + LB) \geq \frac{m_k+1}{m_k}$. On one hand, if $C_k^{\geq q} \geq 2LB$, W_k is larger than $m_k \int_0^{2LB} U(t)dt$ which is larger than $(m_k + 1)LB$, and on the other hand if $C_k^{\geq q} < 2LB$, W_k can be expressed as a sum of four terms:

$$\begin{aligned}
W_k &\geq m_k \int_0^{C_k^{\geq q}-LB} U(t)dt + m_k \int_{C_k^{\geq q}-LB}^{LB} U(t)dt \\
&\quad + m_k \int_{LB}^{C_k^{\geq q}} U(t)dt + (3LB - D_k)q \\
&= m_k \int_0^{C_k^{\geq q}-LB} (U(t) + U(t + LB))\, dt \\
&\quad + m_k \int_{C_k^{\geq q}-LB}^{LB} U(t)dt + (3LB - D_k)q \\
&\geq (m_k + 1)(C_k^{\geq q} - LB) \\
&\quad + (m_k - q + 1)(2LB - C_k^{\geq q}) + (3LB - D_k)q \\
&= (m_k + 1)LB + q(LB - D_k + C_k^{\geq q}) \\
&> (m_k + 1)LB
\end{aligned}$$

□

Proposition 3: During phase 3, if job (p, q) is scheduled on M_k then $q \leq \frac{m_k}{2}$.

Proof: Since the job was not scheduled during phase 2, either $q \leq \frac{m_k}{2}$ or $C_k^{\geq q} + p > D_k$. The proposition is straightforward in the first case, and in the later case there are no possible $potential\_start$ such as $C_k^{\geq q} \leq potential\_start$ and $potential\_start + p \leq D_k$ which is a necessary condition to schedule (p, q). □

Proposition 4: During phase 3, each time a job is scheduled on M_k, the cluster utilization is strictly greater than $\frac{1}{2}$ during an initial period of time (up to the new $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$), followed by a limited period of low utilization when at most $\lfloor \frac{m_k}{2} \rfloor$ processors are used (during at most LB units of time).

Proof: At the very beginning of phase 3, this property is verified since the schedule up to D_k is exactly the schedule produced by HF in phase 1 and it verifies Proposition 1.

We have just proved in Proposition 3 that (p, q) is a low job for M_k. We will now prove that job (p, q) is scheduled to start no later than $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$. The phase 3 of our algorithm relates to list algorithms in the sense that it relies on a moving starting date ($potential\_start$). The starting date is shifted to a later date (namely the next $C_j^{\geq i}$) only when no jobs can be scheduled starting at this time instant. If at some point in phase 3 there is a k such that $C_k^{\geq \lceil \frac{m_k}{2} \rceil} < potential\_start$, it means that at a previous step of the algorithm we had $C_k^{\geq \lceil \frac{m_k}{2} \rceil} = potential\_start$ and no jobs could be scheduled on k. Hence, for any remaining job (p, q) such that $q \leq \frac{m_k}{2}$, we have $C_k^{\geq \lceil \frac{m_k}{2} \rceil} + p > D_k$. Which proves that job (p, q) cannot be scheduled during phase 3 on k at a later date than $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$.

Therefore, when a job (p, q) is scheduled, the utilization of M_k may be augmented before and after $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$, but there are no gap of low utilization created by scheduling (p, q). This proves that the property of having a high utilization period followed by a low utilization period is conserved throughout phase 3. Since all jobs have a duration lower or equal to LB, the low utilization phase cannot be extended further than $C_k^{\geq \lceil \frac{m_k}{2} \rceil} + LB$, which completes the proof. □

Proposition 5: For each job (p, q) in $Unscheduled$, for each cluster M_k such that $C_k^{\geq \lceil \frac{m_k}{2} \rceil} + p > D_k$, cluster M_k has a workload which is at least $m_k LB$.

Proof: Cluster M_k has a high utilization before $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$, and after D_k up to the fixed deadline of $3LB$. The sum of both periods is equal to $C_k^{\geq \lceil \frac{m_k}{2} \rceil} + (3LB - D_k)$, and since $p \leq LB$ this sum is larger than $2LB$. □

Proposition 6: The set $next\_potential\_starts$ is never empty and therefore $potential\_start$ is always well defined.

Proof: The proof is by contradiction, assuming that at some point the set $next\_potential\_starts$ is empty. Then, the set $Unscheduled$ contains at least one job which could not be scheduled at any step of the algorithm, while all the $C_k^{\geq \lceil \frac{m_k}{2} \rceil}$ have been considered as potential starting dates. Considering the characteristics of this particular job (p_0, q_0), from the previous propositions we derive the following facts:

- Any cluster M_k such that $q_0 > \frac{m_k}{2}$ was processing a workload of at least $m_k LB$ when (p_0, q_0) was considered in phase 2 (by Proposition 2).
- Any cluster M_k such that $q_0 \leq \frac{m_k}{2}$ was processing a workload of at least $m_k LB$ when (p_0, q_0) was considered in the last step of phase 3 (by Proposition 4).

□

Theorem 1: The schedule produced by MOCCA is feasible, it includes all the jobs and completes before $3LB$. Hence MOCCA is a 3-approximation algorithm.

Proof: Proposition 6 states that the schedule is feasible and that all the jobs have been properly scheduled. Moreover, by construction no jobs complete after $3LB$. □

4.5 Tightness

By construction, the algorithm produces schedules which have a makespan close to the $3LB$ deadline on some organization, since jobs ending after this deadline in local schedules are rescheduled to end exactly at $3LB$. Therefore, the bound is obviously tight for this algorithm. In this section, we will analyze several interesting properties related to tightness.

4.5.1 Adequacy of the lower bound

In our analysis of MOCCA, we proved that the local schedules ended before $3LB$. Actually, for some instances this deadline is asymptotically close to the optimal makespan.

Proposition 7: For any ϵ, there exist instances where $3LB - \epsilon$ is lower than C^*_{max}.

Proof: Consider for example an instance composed of N organizations of m processors each, and $2N+1$ jobs of size $m/2+1$ and length $\frac{Nm}{(2N+1)(m/2+1)}$. By construction the lower bound LB is then equal to 1. However, no organization is large enough to run two jobs at the same time. Since at least one organization has to execute three jobs, the optimal makespan is $\frac{3Nm}{(2N+1)(m/2+1)}$ which asymptotically tends towards 3 when N or m grows. □

This proves that no algorithm can produce a schedule with a makespan always strictly lower than $3LB$.

4.5.2 Unscheduling more jobs

To go further in the analysis we can also consider the potential performance guarantee for all the algorithms which respect the MOCCA fundamental property. This property is to leave untouched all the jobs scheduled before a given threshold. Since an optimal (non-polynomial) algorithm can schedule all the unscheduled jobs in C^*_{max} units of time, whatever the threshold $d \geq 0$, the value $d + C^*_{max}$ can be trivially achieved by an optimal algorithm. Considering this, and the fact that MOCCA has a performance ratio of 3, we will prove in this section that no algorithm can improve this performance ratio unless the threshold is smaller than $2C^*_{max}$.

Proposition 8: Any algorithm keeping local schedules unchanged up to $2C^*_{max}$ cannot have a performance ratio better than 3.

Proof: We consider an instance of N organizations of m processors, where $m = 2N-1$. All the organizations use list scheduling with jobs sorted in non-increasing sizes to produce their original schedule. The jobs of the first organization are N jobs of size m and length $\frac{2}{N}$ and N jobs of size N and length $1 - \frac{2}{N}$. The other organizations each have N jobs of size 1 and length $1 - \frac{2}{N}$. Original schedules are shown in figure 5.

The LB value for this instance is equal to 1. □

Notice that this remains true even if LB is lower than C^*_{max} and the threshold is set to $2LB$.

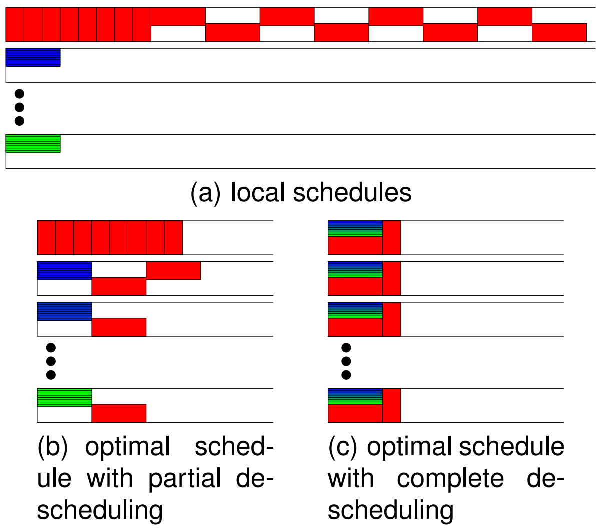

(a) local schedules
(b) optimal schedule with partial descheduling
(c) optimal schedule with complete descheduling

Fig. 5. Worst-case instance, where partially descheduling jobs lead to inefficient schedules.

Proposition 9: In the case of two organizations, if the only constraint on the job's heights is that no job is higher than the size of the cluster of its organization, then MOCCA is a 3-approximation.

Sketch of the proof: As in the previous proof, since MOCCA returns by construction 3-approximate schedules, we only show that the returned scheduled is feasible. The principle of the proof is by contradiction, assuming that one unscheduled job does not fit, a case study on the job height shows that the total surface is larger than expected in LB.

5 Extensions

5.1 Two clusters and jobs of any heights

We present in this Section the following result: if there are two organizations O_1 and O_2 with clusters of different sizes m_1 and m_2 (with $m_1 \geq m_2$), and if the height of each job is at most equal to the number of machines of its own organization, then the approximation ratio of MOCCA is also 3. Notice that, contrary to what it is considered elsewhere, we do not assume that all the jobs fit in all the clusters.

5.2 Any local policy

We have considered up to now that the local schedule of each organization has been obtained by HF. However, it is not always the best schedule. We consider in this section that each organization produces its local schedule as it wishes: thus, it may produce a schedule with a local makespan smaller that the makespan it would have obtained using HF. In this case, knowing the local schedules which have been computed, our goal is to return a schedule minimizing the global makespan under the constraints that no local makespan has been increased (regardless of the local policies used by the organizations). Let us consider the following modified version of MOCCA:

The differences with MOCCA are the following: we keep local schedules if their makespan is better than the one obtained with HF; we unschedule the jobs completed after $4LB$ (and not $3LB$); unscheduled jobs are rescheduled only backward, from $4LB$ (as in Phase 2 of MOCCA– there is no phase 3 here).

Since the schedule is cut at time $4LB$ we will denote this algorithm by MOCCA (4). More generally, the same algorithm which cut the schedule at time αLB is denoted by MOCCA (α).

Algorithm 4 MOCCA (4)

```
1) Let $LB = \max\left(\frac{\sum_{k,i} q_{k,i} p_{k,i}}{\sum_k m_k}, \max_{k,i}(p_{k,i})\right)$.
   On each cluster, if the local makespan is smaller than $4LB$ and smaller than the makespan obtained by scheduling the local jobs with HF then keep the actual local schedule, otherwise replace it by the HF schedule.
2) Unschedule all the jobs that are completed after $4LB$.
3) Sort the unscheduled jobs according to HF.
4) List-schedule the unscheduled jobs backwards, starting from $4LB$ (without moving the already scheduled jobs).
```

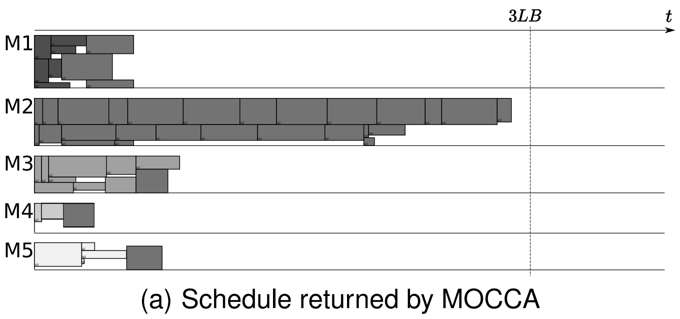
Proposition 10: The schedule produced by MOCCA (4) is feasible: it includes all the jobs and all the organizations have incentive to cooperate, regardless of the algorithms they use to compute their local schedules. Thus, MOCCA (4) is a 4-approximate algorithm.

Proof: At step 1, each local schedule smaller than $4LB$ is modified only if its corresponding HF schedule has a smaller makespan: no local makespan (smaller than $4LB$) is increased. At step 2, the only jobs which are unscheduled are those which are completed after $4LB$. These jobs will be, at the end of the algorithm, completed before $4LB$. Hence, if the schedule returned by MOCCA (4) is always feasible, no organization obtains a makespan larger than the makespan of its original local schedule. We show now that MOCCA (4) indeed returns feasible schedules, i.e., that all the jobs can be scheduled by the algorithm.

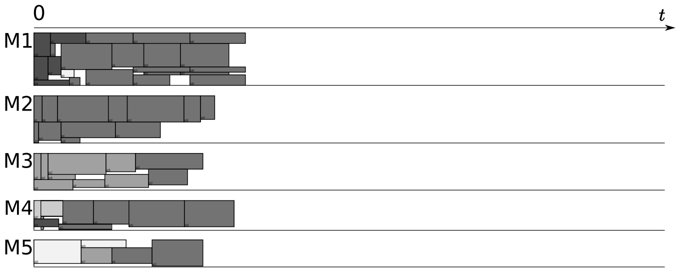
By contradiction, let us assume that at step 4 of MOCCA (4) some jobs do not fit on any cluster. Let $J = (p, q)$ be the highest of these jobs. We show in this case that the work already scheduled on each cluster M_k is larger than $m_k\, LB$. This implies that the work already scheduled is larger than the total work available, which is a contradiction.

Let S be the schedule on cluster M_k at the end of the algorithm. We denote by S_{loc} the schedule of the local jobs of S: S_{loc} is the schedule S in which we remove all the jobs which have been added at step 4 of the algorithm (i.e., those which have been list scheduled backward from $4LB$). We denote by $C_{max}(S_{loc})$ the makespan of S_{loc}. Let S_{HF} be the schedule that we would have obtained by scheduling on M_k the local jobs using HF (and by scheduling only these local jobs). We denote by $C_{max}(S_{HF})$ the makespan of S_{HF}. Thus, all the jobs in S_{HF} are those of S_{loc}: the work in S_{HF} is the same as in S_{loc}.

Let $x = C_{max}(S_{loc})$. We know that the work done in S_{loc} is the same as the work done in S_{HF}, and the makespan of S_{HF} is larger than or equal to x (recall that at step 1 we keep the original local schedule if and only if its makespan is smaller than $C_{max}(S_{HF})$). According to Proposition 1, the work done in S_{HF} (and thus also in S_{loc}) is larger than or equal to $W_1 = \frac{m_k}{2}(x - LB)$ since the low utilization zone in S_{HF} is at most equal to LB. In S, the jobs which are not in S_{loc} are scheduled backward from $4LB$, using the HF order. Job J cannot be scheduled in S, so the gap between the completion time of the last job of S_{loc} and the high job which starts first among the jobs scheduled "backwards" is thus smaller than p and thus smaller than LB. Otherwise, J would fit in the schedule (if J is a high job, recall that with the HF list algorithm, as soon that a high job has been scheduled the highest remaining job is scheduled – unless there is no room for it). Thus there is a high utilization zone from $x + LB$ to $4LB$. The work done in this zone is at least $W_2 = \frac{m_k}{2}(4LB - x - LB)$. Thus the work done in S is at least $W_1 + W_2 = \frac{m_k}{2}(x - LB + 4LB - x - LB) = m_k LB$. This completes the proof. □



(a) Schedule returned by MOCCA



(b) Schedule returned by ILBA

Fig. 6. ILBA (b) refines the schedule returned by MOCCA (a) and improves both the global and local makespans of O_1, O_2 and O_4. Notice that (a) is a compacted version of the schedule from Figure 4 (balanced jobs are scheduled as soon as possible).

We show finally that this bound is tight. Consider for example two clusters M_1 and M_2 with respectively three and one processor. If O_1 does not have any local job, and if O_2 owns 4 jobs of length 1, then the makespan of its local schedule is 4, which is equal to $4LB$, and thus MOCCA does not change this local schedule. The returned schedule is 4-approximate. Notice that we do not know if the schedule returned by MOCCA (α) for $3 \leq \alpha < 4$ is feasible (Proposition 8 states that any algorithm keeping local schedules unchanged up to $2C^*_{max}$ cannot have a performance ratio better than 3).

6 Iterative Load Balancing Algorithm

The MOCCA algorithm guarantees that the global makespan $C_{\max}$ is not greater than $3\,C^*_{\max}$. Nevertheless, the load distribution resulting from running the algorithm can be unbalanced. For instance, MOCCA does not usually modify the schedule of an organization with $C_{\max}(O_k) = 3\,C^*_{\max} - \epsilon$. Yet, it is possible that jobs on such clusters are not yet scheduled while some other clusters are idle. The Iterative Load Balancing Algorithm (ILBA) presented in this section improves the schedule returned by MOCCA by balancing the load between clusters. We also show that in a restricted case of scheduling sequential jobs, ILBA has the same approximation ratio as any list-scheduling algorithm.

The basic principle is to start with the least loaded clusters and iteratively add the more loaded ones, ILBA permits also to reduce the makespans on the less loaded clusters. An example of applying ILBA to a schedule returned by MOCCA is presented in Figure 6. Notice that, as ILBA does not take into account the owner of a job, in this section we will omit the index of organization.

6.1 Algorithm Description

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

ILBA does not delay any job, thus, given a schedule produced by MOCCA, ILBA fulfills the locality constraint in and does not worsen the approximation ratio (see the next Section 6.2 for a formal proof).

Notice 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. The cost complexity of ILBA is $O(Nn)$, where N is small.

6.2 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 11: ILBA does not delay any job comparing with the base schedule.

Proof: As ILBA considers clusters one after the other, 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 assume that the jobs are numbered according to non-decreasing starting 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 after (nor by jobs incoming from different clusters in the subsequent parts of the algorithm). Notice 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 to 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 $t = 0$.

Let now assume that jobs J_1 to J_{i-1} are scheduled no later than their starting times in the original schedule. Consequently, they finish no later than in the original schedule. Thus, at J_i's original starting 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 starting time. It is migrated only if it can be started earlier than this initial starting time. Consequently, J_i finishes at the latest when it finished in the base schedule. $\square$

6.3 Approximation Ratio for Sequential Jobs

If all the jobs are sequential ones, ILBA can be run directly after running local scheduling (i.e., executing MOCCA is not needed). ILBA is a 2-approximation algorithm.

Proposition 12: If all the jobs are sequential (i.e., $q_{k,i} = 1 \forall i, k$), then the approximation ratio of ILBA is equal to $(2 - \frac{1}{\sum_{k=1}^{N} m_i})$ where all the organizations have incentive to cooperate.

Proof: In the obtained schedule, no organization has a makespan larger than the makespan it had by scheduling its jobs locally, thus all the organizations have incentive to cooperate. Moreover, no job starts after any processor (of any cluster) becomes idle. Thus, the schedule could have been returned by a list scheduling algorithm. The approximation ratio of any list algorithm which schedules jobs on m processors is $2 - \frac{1}{m}$ [12]. Thus, ILBA also has the approximation ratio of $(2 - \frac{1}{\sum_{k=1}^{N} m_i})$. $\square$

Notice that this bound is tight: consider for instance one organization with m clusters, which uses the SPT list algorithm (Shortest processing Times) to schedule its jobs, and whose jobs are the ones used in Graham's worst case example of SPT (that is $m(m-1)$ jobs of length 1 and one job of length m). ILBA does not move any job, and the approximation ratio of the resulting schedule is $2 - \frac{1}{m}$.

6.4 Approximation in the General Case

We show in this Section that the approximation ratio of ILBA can be as high as wanted, contrarily to MOCCA which is a 3-approximate algorithm for MOSP. This is true even if the local schedules use the HF order.

Proposition 13: The approximation ratio of ILBA is unbounded.

Proof: Let $x \in \mathbb{N}$. Let us show that ILBA has an approximation ratio larger than x for MOSP. In the sequel (p, q) will denote a job using q processors with processing time p. The considered instance is as follows: Let $n = x^2 - x + 2$. There are $N = n + 2x$ organizations. The n first organizations have each one a job $(L, 1 + x)$ and a job $(1, m - x - 1)$. For each $i \in \{1, \ldots, x\}$, the local jobs of organization O_{n+2i-1} are: i jobs (L, m), $n + 3i - 2$ jobs $(1, m)$, and $n + 2i - 1$ jobs $(1, m - x + i - 1)$; and the local jobs of organization O_{n+2i} are: i jobs (L, m), i jobs $(1, m)$, 1 job $(1, m - x + i - 1)$, and $n + 2i$ jobs $(L, x - i + 1)$. Figure 7(a) depicts the local schedules (in the HF order) in the case where $x = 3$ (for the sake of simplicity, we do not distinguish the heights of the long thin tasks, likewise we do not distinguish the heights of the jobs of length 1 and heights between $m - 1$ and $m - x$). The idea is that organizations $O_1, \ldots, O_n$ will only receive jobs. For each $i \in \{1, \ldots, x\}$, organization O_{n+2i-1} will export its last jobs of height m -one on each of the less loaded clusters -, each of these jobs being followed by another heigh job, slightly too heigh to fit in any gap but thin enough to be scheduled with a long thin job that organization O_{2i} will export at the next step of the algorithm. Thus, the $n + 2i$ long thin jobs exported by organization O_{2i} will be each one on a different cluster. Note that these jobs could not have been scheduled earlier since the length of each task is L and the gaps in the schedule are of length $L - 1$. The schedule returned by ILBA on this instance is depicted in Figure 7(b).

ILBA returns a schedule whose makespan is $(x+1)L + x$. The optimal schedule of these tasks would be:

- $N - 2 = n + 2x - 2$ clusters schedule each one one job (L, m). Since $n = x^2 - x + 2$, we have $n + 2x - 2 = 2\sum_{i=1}^{x} i$: it is possible to schedule each job (L, m) alone on one of these $n + 2x - 2$ clusters.
- one cluster schedules all the other jobs of processing time L. The sum of the heights of all these jobs is $H = \sum_{i=1}^{x}(n + 2i)(x - i + 1)$. By setting $m = H$, the makespan of this cluster is L.
- the last cluster schedules all the other jobs. Each of these jobs has a processing time of 1. By setting L equal to the number of these jobs, the makespan of this cluster is also L.

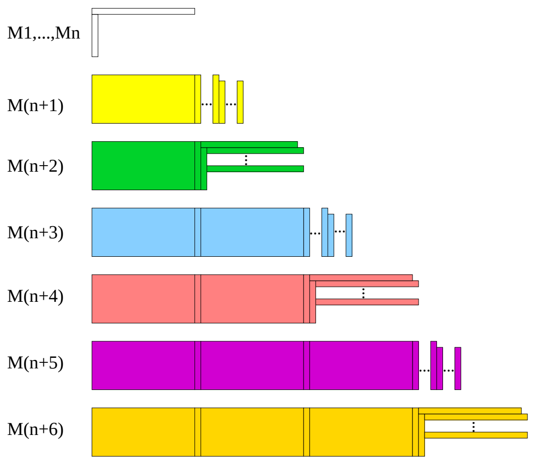
The makespan of an optimal schedule is thus L. The approximation ratio of ILBA on this instance is thus $\frac{(x+1)L+x}{L} > (x + 1)$. Since we can fix x as large as wanted, the approximation ratio of ILBA for MOSP is unbounded. □

7 Experiments

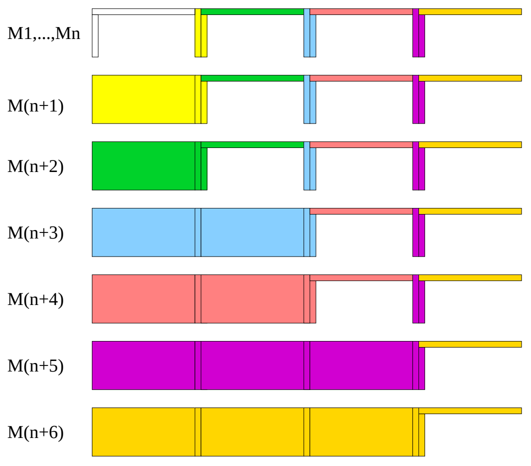
In this section, we carry out the experimental evaluation of MOCCA. We start with performance measures and the methods used to generate the workloads.

7.1 Description of the methods

We considered three algorithms for the experiments, namely: (1) the algorithm that schedules for each organization its jobs on its local cluster according to HF



(a) local schedules



(b) schedule returned by ILBA

Fig. 7. Instance which shows that the approximation ratio of ILBA is unbounded. Here, the makespan of the schedule returned by ILBA is more than 4 times larger than the optimal makespan.

(called *local*); (2) MOCCA presented in Section 4 (we implemented a slightly different version that combines the second and the third phase, i.e., both high and low jobs are scheduled together starting from 3LB; moreover, in order to improve the average performance, then, the schedules are compacted by advancing the moved jobs if there is a gap in the schedule before them (see Figure 6 a); (3) ILBA presented in Section 6 that iteratively improves the schedule returned by MOCCA.

The algorithms were compared regarding the produced global makespan. For each instance and each algorithm, we computed the ratio of the produced makespan to the lower bounds defined in Section 4.1:

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

where $s(alg, I)$ is the score of algorithm alg on instance I (and thus, an upper bound of the approximation ratio), $C_{max}(alg, I)$ is the makespan produced by algorithm alg on instance I, $\bar{W}(I)$ and $p_{\max}(I)$ are respectively the average work and the length of the longest job in instance I.

The algorithms were tested on two sets of randomly-generated instances.

- *uni* instances in which jobs' sizes $p_{k,i}$ and numbers of required processors $q_{k,i}$ were produced by two independent uniform distributions over respectively, $\{1, \ldots, 50\}$ and $\{1, \ldots, m\}$;
- *swf* instances produced by a realistic workload generator [14]. We adjusted the generator as follows: The maximum required number of processors is smaller than m and all the release dates are set to 0.

Each instance produced consisted of n (a parameter of the experiments) jobs. To determine the owner O_k of each job $J_{x,i}$, we used a Zipf's distribution. We set the number of elements equal to the total number of organizations (N) and exponent $s = 1.4267$. These parameters correspond to the distribution of Virtual Organizations owning the jobs in data analyzed in [15].

7.2 Performance in Homogeneous Grids

In this series of experiments, we analyze the performance of the presented algorithms in grids composed of clusters that have the same number of processors. Both *uni* and *swf* instances were generated with $N \in \{2, 5, 10, 20\}$, $n \in \{10, 50, 100, 500\}$ and $m \in \{32, 128, 512\}$. N was chosen to represent academic grids, where the number of participating laboratories is about 10. The number of processors varied between small and large clusters currently used in grid systems. For instance, clusters in Grid'5000 [16] 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 were generated. In total, 4,800 instances were generated. The average scores of algorithms in function of n and N are presented on Figure 8 (*uni* dataset) and Figure 9 (*swf* dataset).

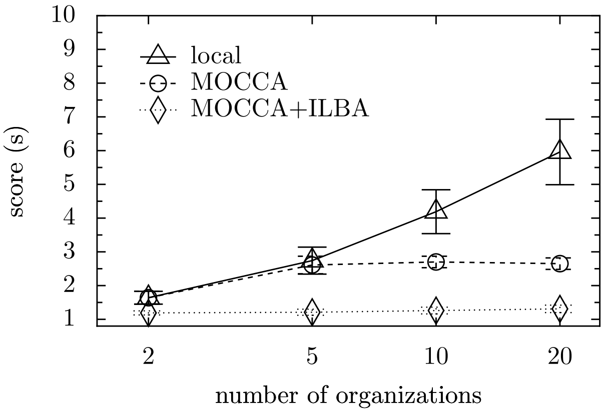
The following phenomena can be observed.

7.2.1 The Inefficiency of Local Scheduling

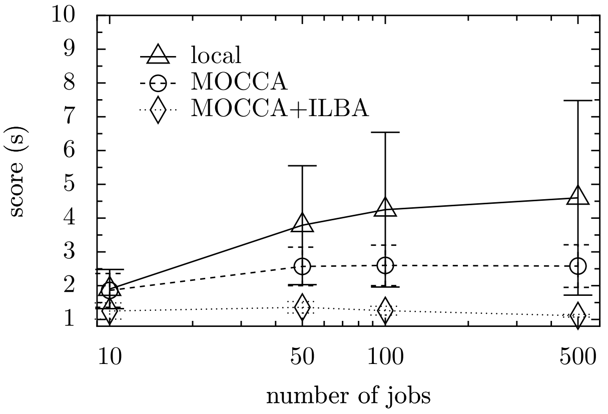
The inefficiency of *local* scheduling can be observed in *uni* instances (Figure 8(a)). The average score of *local* is deteriorating quickly with the increased number of organizations. The trend is even more visible when the smallest instances ($n = 10$) are omitted. The average score grows from 1.57 ($N = 2$) 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 when both data sets are combined). This supports our claim that in larger systems the drop in performance resulting from the lack of cooperation is proportional to N.

7.2.2 Improvement by Iterative Load Balancing Algorithm

The final refinement of schedules performed by ILBA considerably improves the results in all settings. The overall average score for *uni* instances drops from 2.40 (for MOCCA) to 1.24 (for ILBA), whereas for *swf* from 1.16 to 1.03. Moreover, ILBA returned optimal schedules (with score equal to 1) for at least 40% of instances, whereas MOCCA returned such schedules in 26% of the considered instances. Notice that the score of an optimal schedule can be higher than 1 (as we are comparing the makespan with its lower bound). In this case, even if an algorithm returns the optimal schedule, such a simple analysis will not consider it as an optimal.



(a) *uni*, average over n

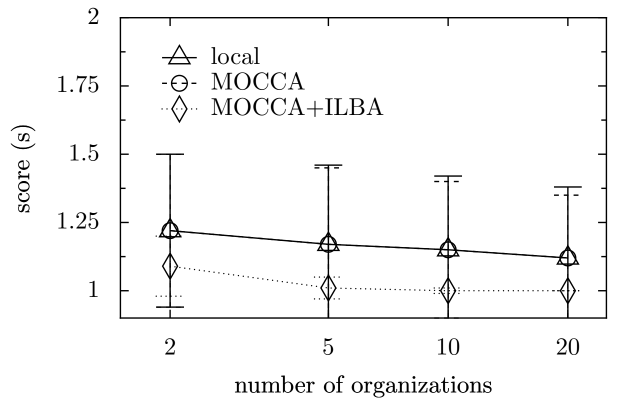


(b) *uni*, average over N

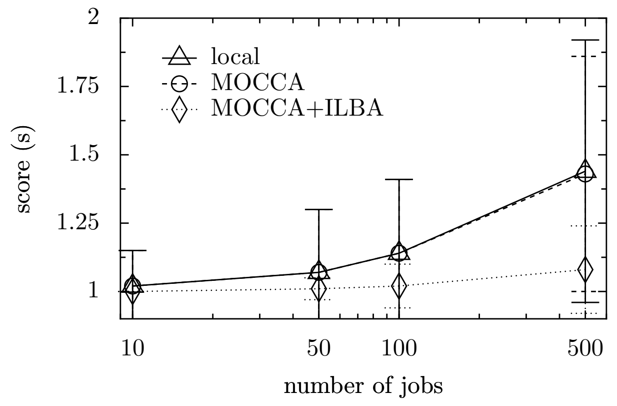
Fig. 8. Average scores, or the ratios of the returned makespan to the lower bound, of algorithms in function of number of jobs n or the number of organizations N. Error bars denote the standard deviation.

7.2.3 Performance on Realistic Instances

The performance of all the algorithms is much better on instances created with the realistic workload generator (*swf* dataset, Figures 9(a) and 9(b)). These instances are easier to schedule because of the increased diversity of both the sizes of jobs and the number of required processors. For example, in the smallest ($n = 10$) *swf* instances, ILBA was optimal in all but one instances, in which the makespan was determined by the length of the longest job.



(a) *swf*, average over n



(b) *swf*, average over N

Fig. 9. Average scores, or the ratios of the returned makespan to the lower bound, of algorithms in function of the total number of jobs n or the number of organizations N.

7.2.4 Influence of the Increased Load

The lack of cooperation makes the system inefficient under heavy load, i.e. n is large (Figure 8(b) and 9(b)). For all but *uni* instances with $N = 2$ and $N = 5$, the average score of *local* scheduling rises with n. The differences of load between organizations result from Zipf's distribution. The increased number of jobs makes the load differences more visible. Scores of MOCCA and ILBA deteriorate in *swf* instances, but improve (or stay similar) in *uni* instances (after omitting the smallest instances). In *swf* instances, the deterioration is directly caused by the increased number of jobs. In contrast to *swf*, 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 (notice that the resulting score is still higher than for *swf* instances).

7.2.5 Influence of the Number of Processors

The number of processors on each cluster 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 $m = 128$ are 0.35, 0.11 and 0.28, for the *local*, MOCCA and ILBA scores, respectively. Similarly p-values for the difference between $m = 128$ and $m = 512$ are, respectively, 0.56, 0.24 and 0.31. Consequently, we do not report results averaged over different values of m.

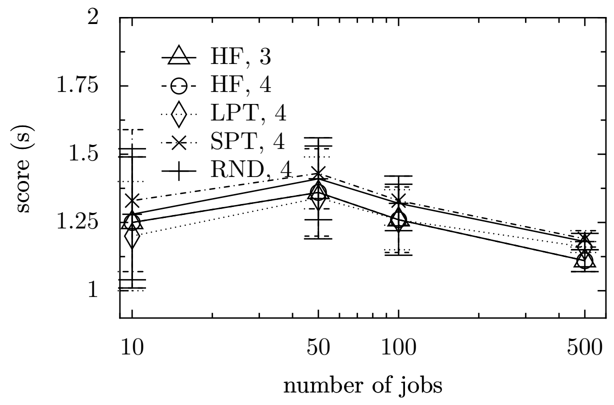


Fig. 10. Average scores, or the ratios of the returned makespan to the lower bound, of algorithms under different local policies in function of the number of organizations N.

7.2.6 Worst Cases

The worst result of the *local* algorithm was 12.12 on an *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. MOCCA achieved the worst results in 9% of instances. This is caused by the construction of the algorithm. Notice that the score of the algorithm is computed relatively to the lower bound of the makespan, and not the optimal solution that, in this instance, is definitely higher than the lower bound.

7.3 Performance with Different Local Policies

MOCCA is a 3-approximation only when HF is used in the local schedules. If the order of jobs is different, the approximation ratio becomes 4 (Section 5.2). The following series of experiments investigates the influence of the local ordering on the performance of the algorithm. We compare five algorithms using the same workloads as in the previous section. Firstly, in local scheduling phase, we order jobs by different factors: decreasing height (HF), largest processing times (LPT), shortest processing times (SPT), and random order (RND). Notice that in order to emphasize the role of different local scheduling policies, we do not reschedule jobs with HF as done in Section 5.2. Then, we run MOCCA algorithm, that either deschedules jobs after $3LB$ (HF3) or after $4LB$. Finally, we run ILBA on the resulting schedules. The results are summarized in Figure 10 for *uni* dataset (we do not show the results for *swf* dataset as they are similar).

Different local policies do not influence the performance of the algorithm. Firstly, the algorithm was feasible in all the instances (notice that Section 5.2 proves that MOCCA (4) returns feasible solutions if the local makespans are not larger than their corresponding HF local makespans, otherwise local jobs have to be rescheduled with HF). Secondly, HF3 is very similar to HF4 with the largest difference in average score around 0.01 and p-value of two-tailed, type 1 t-test equal to 0.02. Moreover, on the average, all the algorithms achieved very similar scores (1.01 – 1.02 in *swf* and 1.24 – 1.31 in *uni*). Out of all the tested local policies, LPT has best average-case behavior with the lowest average score (1.0113, $\sigma = 0.0379$ in *swf* and 1.2422, $\sigma = 0.0808$ in *uni*).

7.4 Performance in Heterogeneous Grids

In this series of experiments, we analyze how the heterogeneity of the grid influences the performance of the presented algorithms.

We simulated the heterogeneity of the grid by varying the number of processors between clusters. We analyzed grids with $N = 10$ organizations. The number of processors in each cluster was generated by a Pareto distribution with minimal number of processors $m_{\min} = 128$, and a cutoff value $m_{max} = 1024$. To generate grids with various measures of heterogeneity, we varied the parameter of Pareto distribution by setting it between 1 and 10 Values higher than 5 correspond to highly homogeneous grids, similar to those tested in the previous round of experiments. In total, $6{,}000$ instances are considered.

The heterogeneity of the grid does not worsen the behavior of ILBA. In *uni* instances, the average score varied between 1.15 (for the most heterogeneous grid) and 1.12, with standard deviation less than 0.04. In *swf* instances, MOCCA +ILBA is almost optimal, with average score less than $1 + 10^{-4}$ and variance less than 10^{-3}. In both cases, local policy and pure MOCCA are significantly worse.

7.5 Summary of the Results

The results of our experiments show that MOCCA 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. We also observed that the workloads generated by the realistic workload generator were easier to schedule.

8 Discussion on Related Work

In this paper we have studied the interest of collaboration between independent parties. We based our work on the idea that if a proposed collaborative solution does not deteriorate any participant's selfish goal, it should be adopted by all the participants. Using reasonable 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 a global objective (the makespan of the system) which represents a social goal, at the same time respecting selfish interests 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 [17] studies situations in which selfish users (called players) optimize their own objectives, 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*, which describes a situation in which no player can improve their own objective by unilaterally changing their 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* [18]. It is interpreted as the cost of the lack of cooperation. For instance, in the context of scheduling, [19] measures 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* [20] 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. Since the proposed solution is a Nash equilibrium, the players do not have incentive to unilaterally refuse to follow it. The price of stability can be interpreted as the cost when 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 [17], players still have their selfish goals, but they can communicate and enforce agreements on their strategies. As a result, coalitions appear and compete instead of individual players. 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. As the only payoff of a player in our problem is the completion time of his or her jobs, payoffs are not arbitrarily transferable. Therefore, our solution is based on an extra centralized mechanism enforcing the cooperation.

Papers proposing *distributed* resource management usually solve the problem of optimizing a common objective with a decentralized algorithm. [21] shows a fully decentralized algorithm that always converges to a steady state. [22] presents a similar algorithm with the divisible load job model. Those approaches contrast with our algorithm. Although the algorithm is centralized, it respects decentralized goals of participants. We are, however, aware that load balancing algorithms in large scale systems tend to be decentralized. In [23], a distributed algorithm balances selfish, identical jobs on a network of identical processors. There are several differences with our work since there is no notion of organization, and agents are jobs whose aim is to be on the least loaded machine. The authors focus on the time needed to converge to a Nash equilibrium, and do not consider, like we do, a social goal to optimize.

Finally, more there exist some conventional combinatorial optimization methods for scheduling independent jobs on clusters or grids composed of multiple clusters. For one cluster, the best algorithms are based on 2D packing (also called strip packing). A survey of such methods has been published in [24] and the best known approximation is 2 for low cost algorithms (the inapproximation bound is $\frac{3}{2}$). Tchernykh et al. proposed a hierarchical 3-approximation algorithm for multiple clusters under the same hypotheses as our's, without the locality constraints [9].

9 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 in which selfish organizations minimize the maximum completion time of their local jobs. We have proved that it is always possible to respect the selfish objectives and, at the same time, improve the performance of the whole system. The cooperative solutions have a constant approximation ratio, a significant gain compared to selfish solutions that can be arbitrarily far from the optimum. Moreover, the proposed algorithms turned out to be efficient during experimental evaluation by simulation. Our aim was not to design 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 in this paper proves that cooperation achieved at the algorithmic (as opposed to e.g. economic) level is possible.

As a perspective, we would like to study the effect of the increased effort of individuals on the global goal, especially in the case of on-line systems. More specifically, we would like to relax the hard constraint of "not being worse than the initial local solutions" to an approximation of "not worsening too much the local solutions".

References

[1] Foster, I., Kesselman, C., eds.: The Grid 2. Blueprint for a New Computing Infrastructure. Elsevier (2004)

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

[3] Blazewicz, J., Ecker, K.H., Schmidt, G., Weglarz, J.: Scheduling in Computer and Manufacturing Systems. Springer (1996)

[4] Pascual, F., Rzadca, K., Trystram, D.: Cooperation in multi-organization scheduling. Concurrency and Computation: Practice and Experience **21**(7) (2009) 905–921

[5] Feitelson, D.G., Rudolph, L., Schwiegelshohn, U.: Parallel job scheduling a status report. In: Proceedings of JSSPP 2004. Volume 3277 of LNCS., Springer (2005) 1–16

[6] Shmoys, D., Wein, J., Williamson, D.: Scheduling parallel machines on-line. SIAM Journal on Computing **24**(6) (1995) 1313–1331

[7] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. WH Freeman & Co. New York, NY, USA (1979)

[8] Bampis, E., Giroudeau, R., König, J.C.: An approximation algorithm for the precedence constrained scheduling problem with hierarchical communications. Theoretical Computer Science **290**(3) (2003) 1883–1895

[9] Schwiegelshohn, U., Tchernykh, A., Yahyapour, R.: Online scheduling in grids. In: Proceedings of IPDPS. (2008)

[10] Bougeret, M., Dutot, P.F., Jansen, K., Otte, C., Trystram, D.: Approximation algorithms for multiple strip packing. In in Computer Science, L.N., ed.: Proceedings of the 7th Workshop on Approximation and Online Algorithms (WAOA), Springer (2009) to appear.

[11] Folling, A., Grimme, C., Lepping, J., Papaspyrou, A.: Robust load delegation in service grid environments. IEEE Transactions on Parallel and Distributed Systems **21** (2010) 1304–1316

[12] Graham, R.: Bounds on multiprocessor timing anomalies. SIAM J. Appl. Math **17**(2) (1969) 416–429

[13] Eyraud-Dubois, L., Mounié, G., Trystram, D.: Analysis of scheduling algorithms with reservations. In: Proceedings of IPDPS, IEEE Computer Society (2007)

[14] Lublin, U., Feitelson, D.: The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. Journal of Parallel & Distributed Computing **11**(63) (2003) 1105–1122

[15] Iosup, A., Dumitrescu, C., Epema, D., Li, H., Wolters, L.: How are real grids used? the analysis of four grid traces and its implications. In: Grid Computing, 7th IEEE/ACM International Conference on. (2006) 262–269

[16] 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., Quetier, B., Richard, O., Talbi, E.A., Touche, I.: Grid'5000: a large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications **20**(4) (2006) 481–494

[17] Osborne, M., Rubinstein, A.: A Course in Game Theory. MIT Press (1994)

[18] Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Proceedings of STACS. Volume 1563 of LNCS., Springer (1999) 404–413

[19] Christodoulou, G., Koutsoupias, E., Nanavati, A.: Coordination mechanisms for selfish scheduling. In: Proceedings of ICALP. Volume 2719 of LNCS., Springer (2003) 345–357

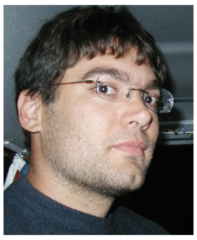
[20] Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proceedings of FOCS. (2004) 295–304

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

[22] Rotaru, T., Nageli, H.H.: Dynamic load balancing by diffusion in heterogeneous systems. J. Parallel Distrib. Comput. **64**(4) (2004) 481–497

[23] Berenbrink, P., Friedetzky, T., Goldberg, L., Goldberg, P., Hu, Z., Martin, R.: Distributed Selfish Load Balancing. In: Proceedings of SODA, ACM Press (2006) 354–363

[24] Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: A survey. European Journal of Operational Research **141** (2002) 241–252



Pierre-Francois Dutot is an Assistant Professor at the University Pierre Mendes France, Grenoble, France. His research interests include theoretical aspects of scheduling, approximation algorithms and multi-objective optimization. He obtained his M.S. from the Ecole Normale Superieure de Lyon, France in 2000 and his Ph.D. from the Institut National Polytechnique de Grenoble in 2004.

Fanny Pascual received the PhD degree in computer science from Evry University, France, in 2007, and she did a postdoc for one year at INRIA (French National Institute for Research in Computer Science and Control), in Grenoble, France. She is currently assistant professor at Pierre et Marie Curie University, in Paris, France. Her research interests include design and analysis of algorithms for combinatorial optimization problems, scheduling problems, and algorithmic game theory.

Krzysztof Rzadca received his MSc in Software Engineering from Warsaw University of Technology, Poland in 2004, and a PhD in Computer Science in 2008 jointly from Institut National Polytechnique de Grenoble (INPG), France and from Polish-Japanese Institute of Information Technology (PJIIT), Poland. Between 2008 and 2010, he worked as a research fellow in Nanyang Technological University (NTU), Singapore. Since 2010, he has been an assistant professor in the Institute of Informatics, University of Warsaw, Poland.

Denis Trystram obtained a PhD in Applied Mathematics at INP (Institut National Polytechnique) Grenoble in 1984 and a second one in Computer Science in 1988 from the same institute. Since 1991 he is Professor at INP Grenoble and is now distingued professor in this Institute. He has been nominated recently to the Institut Universitaire de France. He is leading a research group on Scheduling and Combinatorial Optimization within the MOAIS team of the LIG laboratory. He is currently Regional Editor for Europe for the Parallel Computing Journal and served few years ago in the board of IEEE TPDS. Denis Trystram participates regularly to the Program Committee of major conferences of the field (EUROPAR, IPDPS, HiPC, PARCO, SPAA, ...). The main research interests of professor Trystram concern the design of efficient approximation algorithms for multi-objective scheduling problems and algorithmic Game Theory. He has published several books and more than 90 articles in international journals and as many international conferences.