On the Importance of Migration for Fairness in Online Grid Markets

(Short Paper)

Lior Amar, Ahuva Mu'alem
Institute of Computer Science
The Hebrew University of Jerusalem
Jerusalem, 91904 Israel
{lior,ahumu}@cs.huji.ac.il

Jochen Stößer
Information & Market Engineering
Universität Karlsruhe (TH)
Englerstr. 14, 76131 Karlsruhe, Germany
stoesser@iism.uni-karlsruhe.de

ABSTRACT

Computational grids offer users a simple access to tremendous computer resources for solving large scale computing problems. Traditional performance analysis of scheduling algorithms considers overall system performance while fairness analysis focuses on the individual performance each user receives. Until recently, only few grids and cluster systems provided preemptive migration (e.g. [2]), which is the ability of dynamically moving computational tasks across machines during runtime. The emergent technology of virtualization (e.g. [4]) provides off-the-shelf support for migration, thus making the use of this feature more accessible (even across different OS's).

In this paper, we study the close relation between migration and fairness. We present fairness and quality of service properties for economic online scheduling algorithms. Under mild assumptions we show that it is impossible to achieve these properties without the use of migration. On the other hand, if zero cost migration is used, then these properties can be satisfied.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*; J.4 [**Social and Behavioral Sciences**]: Economics

General Terms

Algorithms, Economics, Design, Theory

Keywords

Mechanism Design, Online Scheduling, Fairness

1. INTRODUCTION

Computational grids offer users simple access to tremendous computer resources for solving large scale computing problems. Grids are often composed of shared resources owned by different owners. Hence grids are heterogeneous in nature. This paper explores the close connection between job migration and fairness.

Until recently, only few grids and cluster systems provided preemptive migration (e.g. [2]), which is the ability of dynamically moving computational tasks across machines during runtime. The emerging technology of virtualization becomes an important building block in grids (e.g. [4]). Virtualization provides off-the-shelf support for virtual machine migration, thus making the use of migration more accessible (even across different OS's). It has been shown that migration with zero cost provides better theoretical worst case bounds [8][1]. Even if the cost of migration is not negligible, it is highly beneficial in practice [6].

Traditional performance analysis of scheduling algorithms considers overall system performance. Theoretical competitive analysis and simulation-based analysis of scheduling algorithms focus on overall criteria such as makespan, sum of (weighted) completion time and average slowdown (e.g. [8, 6]). Even if such an overall criterion is optimized, the individual goal of fairness is not guaranteed. Fairness is an important goal in shared systems, such as grids, involving different owners.

Fair division has been a central problem in economic theory since the 1950's and recently gained considerable amount of focus from the computer science community ([10, 1, 5, 11] and references therein). In general, an allocation is said to be *envy-free* if every player likes his own share at least as much as the share of any other player.

In this paper, we study fairness criteria for the allocation of indivisible goods *over time*. We first present three criteria: envy-freeness, responsiveness and independence for economic online scheduling algorithms (Section 3). Under mild assumptions, we show that it is impossible to achieve these properties without the use of migration (Section 4). On the other hand, if zero cost migration is allowed, we show an algorithm satisfying all of these properties (Section 5).

2. THE SETTING

We consider an organizational grid model [2]. In this model an organization possesses a computational grid (homogeneous or heterogeneous). The organization is composed of several departments where each department i has a predefined priority v_i^0. Departments with higher v_i^0 are ranked higher in the organization. One way to think about v_i^0 is

[1]Observe that in [8] migration between machines with the same speed is called preemption.

Cite as: On the Importance of Migration for Fairness in Online Grid Markets (Short Paper), Lior Amar, Ahuva Mu'alem, Jochen Stößer, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May,12-16.,2008,Estoril,Portugal,pp. 1299-1302.

that it reflects the amount of money department i invests in the shared grid. Essentially, our fairness and quality of service criteria ensure that the more the department invests the better service it gets.

The type θ_j of job j is $\theta_j = (r_j, t_j, v_j^0) \in \mathbb{R}_+ \times \mathbb{R}_+ \times \mathbb{R}_+$ where r_j is the release time of the job, t_j represents the processing time of job j on a standard machine (with speed 1) and v_j^0 is the priority of the department j belongs to. Unless stated explicitly, we assume that the user does not have to report t_j to the scheduling algorithm. In this setting, t_j is not necessarily known to the user before the completion of the job.

Let $\theta_{-}j$ be the types of all other jobs except j. Let d_j^{Alg} be the time job j finishes to run given $\theta_j, \theta_{-}j$ and the scheduling algorithm Alg. For simplicity, when it is clear from the context, we use the notation d_j. We assume that users are more happier if their jobs finish earlier. Note that this is the only assumption we make on users' preferences.

The action of suspending a job on a machine is generally called *preemption*, while the moving of jobs across machines is called *migration*. In our organizational grid model the operating system may or may not support migration of jobs. We assume that the machines may only differ by their speed, and thus we consider a uniform related machines model [8]. Recall that the *homogeneous* machine setting is a special case of the heterogeneous machine setting. The stronger heterogeneous machine setting is a natural theoretical model to study realistic grid environments. We say that a setting is *strictly heterogeneous* if there are no two machines with the same speed.

We define a partial order between jobs in the following way:

DEFINITION 1. *Let j, k be any two jobs. We say that $j \succeq k$ if $r_j \leq r_k$, $v_j^0 \geq v_k^0$ and $t_j \leq t_k$. We say that $j \succ k$ if $j \succeq k$ and at least one of the following inequalities is strict: $r_j \leq r_k$ or $v_j \geq v_k$.*

Note that the $\succ$ order does not require strictness with respect to the running times. As will be discussed later, this is due to the fact that in our possibility result, we do not assume that players declare the running times of their jobs to the algorithm. In what follows, we will not restrict the inputs of our scheduling algorithms. Rather, we will define the required behavior of the scheduling algorithms on a restricted family of instances.

3. DIVISION CRITERIA

3.1 Fairness Criterion

Our first criterion is a fairness criterion for the allocation of indivisible goods over time. We define the Envy-Free (EF) property of a scheduling algorithm to be:

DEFINITION 2. *An online scheduling algorithm is said to provide the* EF *property if $j \succ k$ implies that $d_j(r_j, v_j, t_j) \leq d_k(r_k, v_k, t_k)$, for every two jobs j, k.*

The EF property means that if job j has an earlier release time, higher priority and shorter processing time than job k, it will finish earlier than job k. Note that our property is weaker than the general notion of *envy-freeness* in the sense that two portion of the allocation are compared only if the associated jobs are comparable.

3.2 Responsiveness Criterion

We next define a responsiveness criterion indicating that the scheduler is sensitive to the requirements of the consumers. The Weak-Performance-Monotonicity (WPM) property of a scheduling algorithm is as follows:

DEFINITION 3. *An online scheduling algorithm is said to provide the* WPM *property if $d_j(\theta_j, \theta_{-}j) \leq d_{j'}(\theta_{j'}, \theta_{-}j)$ whenever $j \succeq j'$.*

The WPM property means that, fixing the types of all other jobs $\theta_{-}j$, job j will not complete later if it is released earlier or reports a higher priority (or consumes less CPU time). Thus, this property essentially requires a reasonable behavior from the scheduler. Moreover, this property incentivizes the user to not manipulate the system by reporting a lower priority, a later arrival and / or a longer running time.

3.3 Independence Criterion

We now consider an independence criterion which ensures that the quality of service a job receives depends only on jobs with higher priority. We define the Independence of Low Value jobs (ILV) property of an economic scheduling algorithm to be:

DEFINITION 4. *Let J be an instance of jobs and let $\theta_j \in J$. Define $h_{\theta_j}(J) = \{\theta_k \in J | v_k^0 > v_j^0\} \cup \{\theta_k \in J | v_k^0 = v_j^0$ and $r_k \leq r_j\}$.*

Let $\theta = (r, v^0, t)$. Let J and J' be any arbitrary instances of jobs, where:

1. *$\exists j$ s.t. $\theta = \theta_j \in J$ and $\exists k$ s.t. $\theta = \theta_k \in J'$.*
2. *$r_{j'} \neq r_{k'}$ for every j' and $k' \in J$.*
3. *$r_{j'} \neq r_{k'}$ for every j' and $k' \in J'$.*

An online scheduling algorithm is said to provide the ILV *property if $h_\theta(J) = h_\theta(J')$ implies that $d_\theta(J) = d_\theta(J')$.*

The ILV property means that the completion time of job θ is independent of jobs with lower priority than θ (when considering instances with no simultaneous arrivals of jobs).

4. IMPOSSIBILITIES

In what follows we consider the family of busy scheduling algorithms [8]. We will show that without migration this family of scheduling algorithms can not provide any of our properties.

DEFINITION 5. *A busy-scheduling algorithm is an online algorithm satisfying the following conditions:*

1. *If there is an idle machine then there is no waiting job (that has not started yet).*
2. *A job is returned to the user as soon as it finishes.*
3. *Jobs can not be killed by the system before completion and there are no restarts.*

The first condition means that the scheduler immediately gives "work" to any machine that becomes idle. By work, we do not mean preempted jobs that has already started on another machine. The second condition means that the

job is not kept before returned to the user. The second and the third conditions were implicitly assumed in [8]. Many classical scheduling algorithms (e.g. FCFS, Round Robin) belong to the natural family of busy schedulers. Moreover real grid and cluster systems such as MOSIX [2] and those described in [3] are using busy scheduling policies. Busy schedulers with "reservation" prices were considered in [7, 9]. A job can only use an idle machine if its willingness to pay can cover the machine's price.

PROPOSITION 1. *In a strictly heterogeneous system without migration, any busy scheduling algorithm cannot provide the EF property.*

PROOF. We will show this for every algorithm A, any large enough $X > 0$ and any number of machines $n \geq 2$. WLOG we are given 2 machines, where machine n_1 is having speed of 1, and a faster machine n_2 having speed of 2 (clearly, when using a busy scheduler we can make all other $n-2$ machines occupied indefinitely). In what follows we assume that $v_i^0 = 1, i = 1 \ldots 5$. Assume that at time $t = 0$, job j_1 is submitted. We can divide the family of busy schedulers into two groups: the first group of algorithms assigns j_1 to n_1, while the second group assigns it to n_2. According to the group algorithm A belongs to, we will build an instance of jobs that violates the EF property.

If algorithm A belongs to the first group, then we will use the following instance of jobs: j_1 with $r_1 = 0, t_1 = X$, j_2 with $r_2 = 1, t_2 = X$, j_3 with $r_3 = X/2 + 1, t_3 = X$, j_4 with $r_4 = X, t_4 = X$, j_5 with $r_5 = X + 1, t_5 = X + 2$. Note that $j_4 \succ j_5$. Since job j_1 is assigned to n_1, then j_2 will be assigned to n_2 and job j_3 will also be assigned to n_2. At time $t = X$ when j_4 is released, it will be assigned to n_1 since n_1 is idle while n_2 is busy. When j_5 is released it will be assigned to n_2, since n_2 will be idle and n_1 busy (running j_4). The result will be that j_5 will finish at time $\frac{3}{2}X + 2$, which is sooner than the time j_4 will finish ($2X$), thus the EF property is violated. Proving the case where A belongs to the second group is similar using the following instance: j_1 with $r_1 = 0, t_1 = X$, j_2 with $r_2 = 1, t_2 = X - 1$, j_3 with $r_3 = X/2, t_3 = X + 2$, j_4 with $r_4 = X, t_4 = X$, j_5 with $r_5 = X + 1, t_5 = X + 1$. Note again that $j_4 \succ j_5$. □

PROPOSITION 2. *In a strictly heterogeneous system without migration, any busy scheduler cannot provide the WPM property.*

PROOF. We will show this for every algorithm A, any large enough $X > 0$ and any number of machines $n \geq 2$. Again, we can divide the family of busy schedulers into two groups, whether the algorithm assigns j_1 to n_1 or not. According to the group algorithm A belongs to, we will build an instance of jobs that violates the WPM property.

In case algorithm A belongs to the first group, then we will use the following sequence of jobs: j_1 with $r_1 = 0, t_1 = X$, j_2 with $r_2 = 1, t_2 = X$, j_3 with $r_3 = X/2 + 2, t_3 = X$, j_4 with $r_4 = X, t_4 = X$, j_5 with $r_5 = X + 1, t_5 = 2$. Since j_1 is assigned to n_1, then j_2 will be assigned to run on n_2 and since n_2 is twice as fast as n_1, job j_2 finishes at $X/2 + 1$. When j_3 is submitted, machine n_2 is idle while n_1 is busy, this means j_3 is assigned to n_2. Now at time $t = X$, j_4 is submitted and will run on n_1 since n_2 is still busy. However, if j_4 is submitted later at $t = X+2$, then j_5 will be scheduled to n_1, and j_4 must be scheduled to run on n_2. This means that the completion time of j_4 can be strictly improved by releasing it later.

If algorithm A belongs to the second group, we will use the following sequence of jobs: j_1 with $r_1 = 0, t_1 = X$, j_2 with $r_2 = 1, t_2 = X - 1$, j_3 with $r_3 = X/2 + 2, t_3 = X$, j_4 with $r_4 = X, t_4 = X$, j_5 with $r_5 = X + 1, t_5 = 2$. A similar argument shows that if j_4 is submitted later at $t = X + 2$, it would be scheduled to run on the faster machine n_2. □

Observe that in the two proofs above, the schedules will not change independently of whether the algorithm does or does not know the runtimes of jobs. Additionally, even if migration is not allowed but preemption is allowed, this will not change the schedule of the instances.

We complete this section by showing that in a homogeneous system, a busy scheduler without migration fails to provide the ILV property.

PROPOSITION 3. *In a homogeneous system, any* busy *scheduler which does not use migration (including preemption) does not provide the ILV property.*

PROOF. Let n be the number of machines in the system. Let J' be instance containing only j_{n+1}, where $r_{n+1} = n + 1$, $t_{n+1} = 3n$, $v_{n+1}^0 = 2$. Clearly, as we use a busy scheduler, j_{n+1} will be scheduled immediately on arrival. Now, consider the following instance: $J = j_1, j_2, \ldots, j_n, j_{n+1}$ where $j_1 = (v_1^0 = 1, r_1 = 1, t_1 = 2n - 1), j_2 = (v_2^0 = 1, r_2 = 2, t_1 = 2n - 2), \ldots, j_n = (v_n^0 = 1, r_n = n, t_1 = n)$ and j_{n+1} as above. For every busy scheduler A, the machines are occupied from time n onward. When j_{n+1} is introduced (at time $n + 1$) it can not be scheduled to run immediately (as preemption is not allowed). This means that the completion time of j_{n+1} depends on jobs in J which have a lower v^0, contradicting the ILV property. □

Observation. Note that if preemption is allowed (without migration), the example in the proof above can satisfy the ILV property, provided that, upon arrival of j_{n+1}, the scheduler will preempt one of the running jobs and start to immediately run j_{n+1}. However, at time $2n + 1$, there will be $n - 1$ idle machines, while the machine running j_{n+1} has also a preempted job, waiting to be completed.

More formally, define the Non-Wastefulness property:

If there is an idle machine then there is no waiting job which has previously been *preempted*. We essentially show that: In a homogeneous system, any *busy* scheduler which uses preemption but not migration, can not satisfy the Non-Wastefulness property.

5. ADDING MIGRATIONS: A POSSIBILITY RESULT

In this section we will show that in a heterogeneous system with migration, all our fairness properties can hold, using the Online Greedy Migration algorithm (GM) (see Algorithm 1). The system executes the GM algorithm upon arrival of a new job or termination of a running job. Upon arrival, job j declares its priority v_j^0 (but not t_j). The obtained reallocation is performed using the migration capabilities of the system.

Assumptions: In what follows, we assume first that the time it takes to migrate a job between two machines is 0 ("zero migration cost"). Second, every termination and submission event occurs in an integral time.

LEMMA 1. *The GM algorithm provides the EF property.*

Algorithm 1 Online Greedy Migration (GM)

Upon job arrival or job termination do:

1. Sort the set of current submitted and uncompleted jobs in descending order according to their v^0 and break ties according to earlier release time (and job id).
2. Sort the machines in descending order according to their speed.
3. Repeatedly assign the highest ranked job to the highest ranked machine.

PROOF. Let j and k be any two jobs with $j \succ k$. If $v_j^0 > v_k^0$, *GM* will always rank job j higher than job k (and thus job j gets at least the same service level as k). Otherwise, if $v_j^0 = v_k^0$ then it must be the case that $r_j < r_k$. Similarly, *GM* will rank j higher, according to its tie breaking rule. Since $t_j \leq t_k$, job j will finish no later than job k in both cases, resulting in $d_j(r_j, v_j, t_j) \leq d_k(r_k, v_k, t_k)$. □

LEMMA 2. *GM provides the WPM property.*

PROOF. First, fix r_j and v_j^0. By the greediness of our algorithm, it is clear that the smaller t_j the earlier d_j. More formally $d_j(\tilde{t_j}) \geq d_j(t_j)$, for $\tilde{t_j} \geq t_j$.

Second, fix v_j^0 and t_j. We will show that $d_j(\tilde{r_j}) \geq d_j(r_j)$, for $\tilde{r_j} \geq r_j$. By a step by step argument, consider $\tilde{r_j} = r_j + 1$. If job j was not assigned to a machine at time r_j, then obviously $d_j(r_j) = d_j(\tilde{r}_j)$. Similarly, if job j was assigned to a machine at time r_j, and no other job was assigned to this machine if the job was submitted at time $\tilde{r_j}$, then $d_j(r_j) < d_j(\tilde{r}_j)$. Now, assume that job j was assigned to a machine at time r_j. Let K be the set of jobs that were assigned to faster machines if job j would be released at $\tilde{r_j}$. Then it must be the case that $v_k^0 \leq v_j^0, \quad \forall k \in K$.

If $v_k^0 < v_j^0$ for every $k \in K$ then each job k would be ranked lower than j by the GM algorithm at each step. Therefore, in the time interval starting from time $\tilde{r_j}$ until $d_j(r_j)$, in both scenarios, job j would be assigned to the same machines. However, since in the first scenario job j was scheduled to run at r_j, then it should be given some extra processing time in the second scenario (after $d_j(r_j)$). Thus $d_j(r_j) < d_j(\tilde{r}_j)$.

Otherwise, there is at least one job $k' \in K$ with $v_{k'}^0 = v_j^0$. However, $r_{k'} < \tilde{r_j}$ and thus k' will be ranked higher than j in the GM algorithm. As a result, in the time interval starting at $\tilde{r_j}$ until $d_j(r_j)$, the schedule of j can only get worse. Meaning $d_j(r_j) < d_j(\tilde{r}_j)$.

Finally, fix r_j and t_j. Let $d = d_j(v_j^0)$. That is, d is the completion time of j. We give extra power to j and assume that at the beginning of any time slot the job can modify its v_j^0 and declare different value of v_j^0 to the *GM* algorithm. By a step by step argument, where in each step we assume that at times $r_j, \ldots, d-i-1$ the declared value of j is v_j^0 and at times $d-i, d-i+1, \ldots, d-1$ the declared value of j is $\tilde{v}_j^0 > v_j^0$. We will show that at each step the completion time can only improve.

In step 1, we assume that at beginning of the last time slot $(d-1)$, job j declares $\tilde{v}_j^0$. Clearly this can not increase d_j. Now, at step 2 assume that at $(d-2)$, the player declares $\tilde{v}_j^0 > v_j^0$. Clearly, in this time interval ($[d-2, d-1]$) the job will get a (weakly) faster machine. As a result, there might be a set of jobs K that will be allocated to slower machines. Clearly, for every $k \in K$, $v_k^0 \leq \tilde{v}_j^0$. Thus as j declares $\tilde{v}_j^0 > v_j^0$ also at $d-1$, this will not increase d_j. □

LEMMA 3. *The GM algorithm provides the ILV property.*

PROOF. Let J and J' be instances as in definition 4. Clearly, the GM algorithm ranks the jobs in $h_\theta(J)$ and in $h_\theta(J')$ in the same order at the top of the queue (since there are no jobs with the same release time in J and J', and as a result, no ties occur). And so, the jobs in $h_\theta(J') = h_\theta(J')$ receive the same service (regardless of the low ranked jobs in $J \setminus h_\theta(J)$ and in $J' \setminus h_\theta(J')$) and thus complete at the same time. In particular $d_\theta(J) = d_\theta(J')$. □

THEOREM 1. *The* GM *algorithm provides the EF, the WPM, and the ILV property.*

PROOF. See lemmas 1, 2, 3. □

6. CONCLUSION & FUTURE WORK

In this paper, we investigated the benefit of performing preemptions and migrations in economic settings where users have time-dependent valuations. We identified several fairness criteria which represent a first step towards inducing selfish users to sincerely reveal their job characteristics. At the core of this paper, we showed that, without migrations, so called economic busy schedulers cannot satisfy these criteria. In contrast, if migration is allowed, a simple allocation mechanism can indeed achieve these notions of fairness.

One basic assumption in our possibility result is zero migration cost. A natural extension of this work will thus be to consider such a cost in realistic scenarios. We are currently developing a simulator to investigate economic scheduling mechanisms which achieve an "adequate" balance between incentives and fairness performance on the one hand and migration cost on the other hand.

7. REFERENCES

[1] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. In *STOC-07*, 2007.

[2] A. Barak, A. Shiloh, and L. Amar. An organizational grid of federated mosix clusters. In *CCGrid-05*, May 2005.

[3] Y. Etsion and D. Tsafrir. A short survey of commercial cluster batch schedulers. Technical Report 2005-13, Hebrew University, May 2005.

[4] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machines. In *ICDCS-03*, 2003.

[5] V. Guruswami, J. D. Hartline, A. R. Karlin, D. Kempe, C. Kenyon, and F. McSherry. On profit-maximizing envy-free pricing. In *SODA-05*, 2005.

[6] M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. *ACM Trans. Comput. Syst.*, 15(3):253–285, 1997.

[7] L. Kang and D. C. Parkes. A decentralized auction framework to promote efficient resource allocation in open computational grids. In *Proc. NetEcon-IBC*, San Diego, CA, 2007.

[8] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. In M. J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*. CRC Press, 1997.

[9] R. Lavi and N. Nisan. Online ascending auctions for gradually expiring items. In *SODA-05*, 2005.

[10] R. J. Lipton, E. Markakis, E. Mossel, and A. Saberi. On approximately fair allocations of indivisible goods. In *EC-04*, 2004.

[11] A. Mu'alem and M. Schapira. Setting lower bounds on truthfulness. In *SODA-07*, 2007.