Competing Schedulers

Itai Ashlagi
Harvard Business School
Harvard, MA, USA
iashlagi@hbs.edu

Moshe Tennenholtz
Microsoft Israel R&D Center
Herzlia, Israel and
Faculty of Industrial Eng. and Management
Technion–Israel Institute of Technology
Haifa, Israel
moshet@microsoft.com

Aviv Zohar
School of Eng. and Computer Science
The Hebrew University of Jerusalem
Jerusalem, Israel and
Microsoft Israel R&D Center
Herzlia, Israel
avivz@cs.huji.ac.il

Abstract

Previous work on machine scheduling has considered the case of agents who control the scheduled jobs and attempt to minimize their own completion time. We argue that in cloud and grid computing settings, different machines cannot be considered to be fully cooperative as they may belong to competing economic entities, and that agents can easily move their jobs between competing providers. We therefore consider a setting in which the machines are also controlled by selfish agents, and attempt to maximize their own gains by strategically selecting their scheduling policy. We analyze the equilibria that arise due to competition in this 2-sided setting. In particular, not only do we require that the jobs will be in equilibrium with one another, but also that the schedulers' policies will be in equilibrium. We also consider different mixtures of classic deterministic scheduling policies and random scheduling policies.

1 Introduction

With the advent of cloud computing, it has become more and more common for users requiring resource intensive computation to send their jobs to be executed on some remote machine. Cloud computing has often been considered a cooperative setting in which only a single entity, namely the service provider, controls all the processing power. However, it is becoming more common that such services are owned and controlled by different entities – e.g., various large commercial enterprizes.[1] We therefore believe that when modeling the behavior of rational agents in a market for computation cycles, it is crucial to also consider the behavior of the machine owners.

Previous work on incentives in scheduling domains has mostly focused on the behavior of agents from one side of the market. In these settings, the machines use a fixed scheduling policy, and the agents try to maximize their utility by selecting the machine on which their job will run.

In cooperative settings, where all machines are owned by the same entity, it is reasonable to expect that the owner will attempt to balance the load on all machines and minimize the makespan. However, in a market setting, where machine owners get paid for running each job, the goal of each machine owner is to maximize revenue, and thus to attract more jobs at the expense of the competition. In this case each owner actually attempts to maximize the running time of his machine.

Our model considers m identical machines, and n jobs. Each job has a running time that is known in advance, and is controlled by a single agent that attempts to minimize its completion time by sending it to the machine that will execute it as early as possible. On the other hand, each machine is controlled by an agent that can change the scheduling priority of the machine. For simplicity, we assume that there is a fixed cost per computation cycle, and that this price is identical for all machines. Under this assumption, optimizing the revenue obtained by the machines amounts to maximizing their total running time. We define the model more formally in the following section.

We start with a setting in which different machines use different deterministic scheduling policies, taken from a broad set of such deterministic policies. Namely, any fixed ordering on the available jobs can serve as a scheduling policy. Interestingly, we show that for any selection of such deterministic scheduling policies there exists a pure Nash equilibrium of the jobs. The only study we are aware of that refers to the existence of Nash equilibria in a setting where schedulers have a wide variety of policies to select from presents negative results (Kollias 2008).

Next, we consider schedulers that can use a standard deterministic policy or the random scheduling policy (Koutsoupias 2003). We show that if each machine uses either the random policy or the shortest-job-first policy there exists a pure Nash equilibrium. Similarly, a pure Nash equilibrium exists if each machine uses either the longest-job-first policy or the random policy. We then move on to the most elaborate setting, in which the schedulers' policies themselves need to be in equilibrium. In general, studies of competition among mechanisms are quite rare in the literature, and we are not aware of any other discussion of that topic in the context of competing schedulers. Surprisingly, we are able to show the existence of such an equilibrium whenever the schedulers can choose among any set of two policies. We also show that this is not true when there are more than two policies to select from.

[1] Indeed, the competition in cloud computing between Microsoft Azure, Google and Amazon web services fits squarely into this setting.

2 Related Work

There are two main lines of research our work fits into. One is the work dealing with selfish job scheduling. Classical examples of game-theoretic studies in this regard are in the work by (Koutsoupias and Papadimitriou 2009) and work that followed and extended these results to unrelated machines (see e.g. (Azar, Jain, and Mirrokni 2008) and references therein). Much of this work concentrated on the study of the price of anarchy in restricted scheduling settings, rather than on a general game-theoretic analysis. A non-trivial existence of pure Nash equilibrium under random ordering where a participant can choose more than one resource, has been shown in (Penn, Polukarov, and Tennenholtz 2009) in the context of congestion games.

The other relevant line of research has to do with competing mechanisms. The literature on competing mechanisms is not rich. A few exceptions in the context of auctions can be found in (Burguet and Sakovics 1999; Monderer and Tennenholtz 2004). In that context full two-stage equilibria do not exist even in restricted models. More generally, the AI and multi-agent systems literature is full with attempts of providing rigorous game-theoretic analysis of settings originating from classical multi-agent systems (see (Wooldridge 2000; Shoham and Leyton-Brown 2009)). Interestingly, while competition in resource selection appears frequently in that literature (e.g. (Galstyan, Kolar, and Lerman 2003)) and related computational aspects of multi-agent scheduling (e.g. (van Hoevel et al. 2007) are also frequently discussed, the analysis of actual competing schedulers has been mainly neglected.

3 The Model

Let $M = \{1, \ldots, m\}$ be the set of machines and $N = \{1, 2, \ldots, n\}$ be the set of jobs. Every job $i \in N$ can be processed on any of the machines in $t_i > 0$ time units and each job or machine is controlled by a different agent (we therefore make no distinction between the agent and the job or machine that it controls).

A scheduling policy receives as input a subset of the jobs, and outputs the order in which they will be processed. Hence, the output of a scheduling policy is a vector $\mathbf{o} = (o_1, \ldots, o_k)$, such that $o_i \in N$ for every i, and job o_i is processed in the i^{th} place.

For any set T, denote by $\Pi(T)$ the set of all ordered vectors (permutations) of the elements in T.

Definition 1 *A (deterministic)* scheduling policy *is a function f such that for every nonempty subset of jobs T, $f(T) \in \Pi(T)$ is some permutation of the jobs in T.*

To illustrate, suppose machine i uses policy f, and suppose $f(\{2,4,3\}) = (3,4,2)$, then if exactly the subset of jobs $\{2,4,3\}$ choose machine i, job 3 will be processed first, followed by job 4 and job 2.

Definition 2 *We say that a scheduling policy is* consistent *if for every two jobs i, j such that $i \neq j$, job i is always scheduled before job j, or j is always scheduled before i (in all cases where they are both on the same machine).*

A randomized policy is then defined as follows:

Definition 3 *A randomized scheduling policy assigns each possible permutation of the jobs some probability. I.e., for every set of jobs T, the policy assigns a probability function over $\Pi(T)$.*

For convenience we will denote three natural scheduling policies as follows:

- S: processes jobs from the shortest to the longest.
- L: processes jobs from the longest to the shortest.
- R: processes jobs in a random order, with equal probability for each permutation.

In the two deterministic policies above, we assume that ties are broken in favor of the job with the lowest index. I.e., if two jobs have the same length then the one with the lower index has a higher priority.

4 An Equilibrium of Jobs

In this section we analyze the one-stage game in which each machine has a fixed scheduling policy (and is therefore not acting strategically) and every job selfishly chooses the machine on which it will be executed.

Every profile of scheduling policies $\mathbf{f} = (f_1, \ldots, f_m)$ defines a game for the jobs as follows. The action set of the agent associated with each job i is $X_i = M$ (i.e., the machine the job will run on). Denote by X the set of the action profiles. That is $X = \times_{i=1}^{n} X_i$.

For every action profile of the jobs $\mathbf{x}$, and for every profile of scheduling policies $\mathbf{f}$, we denote the completion time of job i by $c_i^{\mathbf{f}}(\mathbf{x})$ (i.e., the amount of time needed to complete job i and all jobs that chose the same machine and are scheduled before it). We assume that every agent wishes to minimize the completion time of its own job.

Definition 4 *An action profile $\mathbf{x} = (x_1, \ldots, x_n)$ is a* pure Nash equilibrium *in the game with policies $\mathbf{f} = (f_1, \ldots, f_m)$ if for every job i $c_i^{\mathbf{f}}(x_i, x_{-i}) \leq c_i^{\mathbf{f}}(y_i, x_{-i})$ for every $y_i \in X_i$.*

For any two scheduling policies ϕ, ψ, we call the game $f = (f_1, \ldots, f_m)$ a (ϕ, ψ)-*game* if for every i, f_i is either the scheduling policy ϕ or ψ (we thus define (S, L)-games,(S, R)-games and (L, R)-games).

In the next section we study the one-shot game in which each machine uses a consistent deterministic policy.

Equilibria when Machines use Deterministic Policies

Theorem 1 *There exists a pure equilibrium for the jobs in every game in which the machines use deterministic and consistent scheduling priorities.*

We will prove Theorem 1 by construction. The following algorithm computes an equilibrium in the game. During the algorithm we will add jobs to the machines one at a time. For every machine i we have a variable q_i which denotes the load on machine i so far (total amount of time the machine will process jobs). As the machines are empty at the beginning we initialize $q_i = 0$ for every i.

Algorithm "Compute Deterministic Job Eq."

1. Input: A scheduling game.
2. Until all jobs are assigned do:
   (a) let μ be a machine with an earliest completion time q_μ (if some machines are tied, choose arbitrarily).
   (b) let job i be the job that has the highest priority on μ out of all unassigned jobs.
   (c) Assign job i to machine μ (thereby increasing q_μ by t_i).

Claim 1 *The algorithm above constructs a pure equilibrium in any job scheduling game with fixed deterministic consistent policies.*

Proof: Observe some arbitrary job i that has been placed on machine μ. Notice that when job i is placed, all jobs that are already assigned to machines have a higher priority. This is because every time a job is placed on some machine it has the highest priority on that machine among all unplaced jobs and so, if i is placed after j has been placed, then job i cannot have higher priority on j's machine. Therefore, when i is placed on μ its running time is minimized (it will start later on all other machines, as μ is chosen to be the machine with the earliest job completion time). Since Job i is the job with the highest priority of all unplaced jobs when it is placed on μ, all jobs that are placed on μ later will run after Job i. Therefore, we have shown that i's starting time on μ is the earliest time it will have among all machines. □

In particular, from the theorem above, it follows that there exists an equilibrium for the jobs in any scheduling scenario in which the machines use only S, L strategies. With random strategies things get more complicated and the existence of a pure equilibrium is harder to show. In the next two sections we will prove the existence of pure Nash equilibrium in (L, R) and (S, R) games.

An Equilibrium for the Jobs in L-R Games

Note that when using a machine of type R, each job has an expected completion time that is exactly its own running time plus half the running time of all other jobs that are using this machine. This is because, for a fixed job j, any other job j' will precede j with probability $\frac{1}{2}$.

Theorem 2 *There exists a pure equilibrium for the jobs in every L-R game.*

We shall prove this theorem in stages. First, by showing an algorithm that reaches an equilibrium in R-type machines, and then an algorithm that reaches an equilibrium in a setting where both L- and R-type machines are used.

Algorithm "Balance R-machines":
Given a placement of jobs on R-type machines do:

1. Let i be the job with the highest expected starting time.
2. Move job i to the machine that will give it the lowest expected starting time (if the current machine gives the best starting time, leave the job on the same machine).
3. If job i stayed on the same machine, then stop. Otherwise, return to stage 1.

Claim 2 *The algorithm "Balance R-Machines" terminates.*

Proof: The algorithm continually moves jobs to the machine that will give job i the earliest starting time. If this is a different machine than the original machine of that job, then the expected starting time it gives job i is also half the total running time of that machine. That is, the machine with the lowest running time has its running time increased (and if there are several machines with the same lowest running time, then the number of such machines is decreased). Notice that the machine job i leaves has a worse running time after it does so, otherwise that job would not have left. □

Claim 3 *When the algorithm "Balance R-Machines" terminates, the jobs are in equilibrium.*

Proof: Let us assume to the contrary (for the purpose of reaching a contradiction) – that there is a job i that gains by moving to another machine. In particular, job i would also gain from a move to the machine that terminates first, because its expected termination time on that machine would be the lowest. Therefore, any job that starts running after job i would also wish to move to the earliest terminating machine (and is not already on that machine, otherwise, it will not be starting after job i). This especially applies to the job that has the highest expected starting time. We reach a contradiction: The job with the highest expected running time wishes to move to the earliest terminating machine, and so the algorithm could not have terminated. □

The following claim gives a useful property of the algorithm that we will use later on:

Claim 4 *Let T be the termination time of the earliest terminating machine at some configuration of jobs on R-machines. Then after running the "Balance R-Machines" algorithm, the earliest terminating machine terminates at a time $\geq T$.*

The proof is immediate from the arguments made in Claim 2.

We now proceed to show that there exists an equilibrium for any setting that involves L-R machines. The following algorithm reaches this equilibrium:

Algorithm "Construct L-R Equilibrium"
Initialization: Place all jobs only on R-machines, and use the "Balance R-machines" algorithm to get them in an equilibrium (when only R-Machines are allowed).
While there exists a job that wishes to move from an R-machine to an L-machine iterate over the following actions:

1. Let i be the largest job that that gains from moving to an L-machine, and let μ be the current machine of job i.
2. Move job i to the L-machine that would give it the earliest starting time under the current configuration.
3. Move any job that is running on an L-machine, and has a shorter running time than i to machine μ.
4. Run the "Balance R-machines" algorithm to get the jobs on the R-machine into equilibrium (when ignoring the L-machines and jobs currently on them).

Claim 5 *The algorithm "Construct L-R equilibrium" terminates.*

Proof: Without loss of generality, let us assume that the jobs $1, \ldots, n$ are sorted in decreasing order of size (i.e., that $t_1 \geq t_2 \geq \ldots \geq t_n$). For a given allocation A of jobs to machines let the vector $\vec{v}(A) = (v_1, v_2, \ldots, v_n)$ be defined as follows: $v_i = 1$ if job i is on an L-machine. Otherwise, $v_i = 0$. Now, notice that during each iteration of the algorithm, a job is moved from an R-machine to an L-machine, and (only) smaller jobs are moved from R-machines to an L-machine. This implies, that in every iteration of the algorithm, if it moved from state A to state A' then $\vec{v}(A)$ is lexicographically smaller than $\vec{v}(A')$. The value of $\vec{v}()$ is therefore strictly increasing (lexicographically) and since the the vectors are all bounded from above (by the vector of all 1's) the algorithm must terminate. □

Claim 6 *When the "Construct L-R equilibrium" algorithm terminates, the jobs on all machines are in equilibrium.*

Proof: First, notice that if the algorithm has terminated, then any job on an R-machine does not gain by switching machines. It does not gain by switching to another R-machine because the last thing that was executed was the "Balance R-machines" algorithm, and it does not gain by switching to an L-machine, because in this case the algorithm would not terminate.

Let us therefore consider some job i on an L-machine. Notice, that job i cannot gain from switching to another L machine. This is because job i was placed on its most preferred L-machine (when it was last moved from an R-machine), and no longer job has been placed on or removed from an L-machine after that point in time.

We must therefore consider the possibility that some arbitrary job i wishes to move from an L to an R-machine. Let k be the shortest job on any L-machine. If job i gains by moving to an R-machine, then so must job k. However, by construction, job k must be the last job that was moved to an L-machine. Let T be the expected starting time of job k before it was moved to an L-machine (in the last iteration of the algorithm). When job k was moved, the expected starting on its current machine was the lowest it could get (under the current configuration) from any R-machine. As it was moved, any shorter job that was assigned to any L-machine was moved to the R-machine job k was taken off of. This can only increase the minimal termination time of the R-machines. Next, the "Balance R-machines" algorithm was executed for the last time. According to Claim 4, this can only increase the earliest termination time of the R-machines. Therefore, if job k moves back it will have an expected starting time $\geq T$, which is less than it gets on an L-machine. A contradiction. □

An Equilibrium for the Jobs in S-R Games

Theorem 3 *There exists a pure equilibrium for the jobs in every S-R game.*

We present the algorithm that finds such an equilibrium, but omit the remainder of the proof due to lack of space.

Algorithm "Construct S-R Equilibrium"
Input: A scheduling game in which all machines use S or R scheduling policies only.

1. Sort all jobs according to size (where jobs of equal length are ordered consistently with their execution priority on the S-machines).
2. While there are any un-assigned jobs, do:
   (a) Let s and l be the shortest and longest unassigned jobs. (If there is only 1 remaining unassigned job then it may very well be that $l = s$)
   (b) Let μ be the machine that job s prefers the most (with ties broken in favor of S-machines).
   (c) If μ is an S-machine then assign job s to it. Otherwise assign job l to μ.

Since a job is assigned to a machine (and never taken off) at every iteration of the algorithm, it is clear that the algorithm terminates.

Claim 7 *Algorithm "Construct S-R Equilibrium" terminates with an assignment of the jobs to the machines which is a pure Nash Equilibrium.*

5 The Two-Stage Game

In this section we study the following two stage game in which the machine owners can choose their own policies, and do so strategically to attract jobs. In the first stage every machine owner i simultaneously chooses a policy from some given set of policies F_i, and in the second stage, *after the choices of the machine owners become common knowledge* the job owners simultaneously choose which machine they will send their job to. Consequently, the strategy of each machine owner i is the set of all scheduling policies F_i. A strategy of every job j is now a function $x_j : \times_{i=1}^{m} F_i \rightarrow M$, i.e. for every profile of policies $(f_1, \ldots, f_m)$ job j chooses a machine in M. Hence, $x(\mathbf{f}) = (x_1(\mathbf{f}), \ldots, x_n(\mathbf{f}))$ denotes the action profile of the jobs when the machines profile is $\mathbf{f}$. We denote by $q_i(x(\mathbf{f}))$ the total load on machine i when the action profile is $x(\mathbf{f})$.

We assume that every machine owner wishes to maximize the amount of time it operates, i.e. the total length of jobs it processes, and so the utility of machine i is defined as:

$$u_i((f_1, \ldots, f_m), (x_1, \ldots, x_n)) = q_i(x(\mathbf{f}))$$

We denote the above two stage game by $G(F_1, \ldots, F_m)$.

Definition 5 *A tuple $(f_1, \ldots, f_m, x_1, \ldots, x_n)$ is a* subgame-perfect equilibrium *in the game $G(F_1, \ldots, F_m)$ if*

1. *for every machine i and for every $g_i \in F_i$, $u_i((f_i, f_{-i}), (x_1, \ldots, x_n)) \leq u_i((g_i, f_{-i}), (x_1, \ldots, x_n))$*
2. *the profile of strategies $(x_1(\mathbf{f}), \ldots, x_n(\mathbf{f}))$ is an equilibrium in the game $\mathbf{f} = (f_1, \ldots, f_m)$.*

We are now ready to show the following result:

Theorem 4 *There exists a subgame-perfect equilibrium in every two-stage-game in which the machines are restricted to* any *two strategies for which the job-game has an equilibrium.*

The proof of Theorem 4 is by construction. In particular, we will show an algorithm that finds a subgame-perfect equilibrium in this game.

Let ϕ and ψ be two policies such that for any job-game in which each machine chooses either ϕ or ψ there exist a pure Nash equilibrium. For every profile $\mathbf{f}$ in which $f_i \in \{\phi, \psi\}$ denote by $\phi(\mathbf{f})$ and $\psi(\mathbf{f})$ the sets of machines that choose ϕ and ψ respectively in a given pure equilibrium.

The algorithm below constructs a subgame perfect equilibrium in the two stage game in which each machine can choose among the scheduling policies ψ and ϕ. It consists of two stages: In the first stage we construct the equilibrium from the point of view of the agents who control the jobs. For every profile $\mathbf{f}$ of the machines, we define an assignment of the jobs to the machines that is in equilibrium from the perspective of the jobs. Notice that since machines that choose the same policy are identical, several equilibria exist, and we have the freedom to permute the entire bundle of jobs between two machines that are using the same policy. We can use this fact to construct an equilibrium with the following properties: if for two profiles $\mathbf{f}, \mathbf{f}'$ of the machines we have that $|\psi(\mathbf{f})| = |\psi(\mathbf{f}')|$ and $|\phi(\mathbf{f})| = |\phi(\mathbf{f}')|$, then the jobs will in fact use the same equilibrium, up to re-naming of the machines. In addition, we will permute the bundles among machines that use the same scheduling policy so that we give more utility (a later completion time) to machines that chose ϕ if they have a lower index, and more utility to machines that chose ψ if they have a high index.

At the second stage of the algorithm we begin with a profile in which all machines choose ψ and we change the profile of the machines one by one, starting with the machines with the lowest index. Each machine switches to policy ϕ as long as it derives benefit from the change. Our construction guarantees that no machine that changed its profile to ϕ will ever want to change back to ψ.

Algorithm "Find Machine and Job Equilibrium"

1. For every profile of policies $\mathbf{f}$ compute an equilibrium for the jobs $x(\mathbf{f})$ with the following properties:

   (a) for every pair of machines $i, i' \in \phi(\mathbf{f})$, $q_i \leq q'_i$ if and only if $i \geq i'$, and

   (b) for every couple of machines $i, i' \in \psi(\mathbf{f})$, $q_i \leq q'_i$ if and only if $i \leq i'$.

   (c) For every two profiles $\mathbf{f}, \mathbf{f}'$ if $|\phi(\mathbf{f})| = |\phi(\mathbf{f}')|$ and $|\psi(\mathbf{f})| = |\psi(\mathbf{f}')|$, then there exists a permutation π such that $\pi\phi(\mathbf{f}) = \phi(\mathbf{f}')$ and $\pi\psi(\mathbf{f}) = \psi(\mathbf{f}')$ and $\pi x(\mathbf{f}) = x(\mathbf{f}')$.

2. Initialize $\mathbf{f}$ to be $\mathbf{f} = (\psi, \ldots, \psi)$. Repeat the following:
   - If $\psi(\mathbf{f}) = \emptyset$, output $(\mathbf{f}, \mathbf{x})$.
   - Let i be the machine with the lowest index in $\psi(\mathbf{f})$. If $q_i(x(\mathbf{f})) < q_i(x(f_{-i}, \phi))$ then let $f_i = \phi$. Otherwise output $(\mathbf{f}, \mathbf{x})$.

Before we prove the correctness of the algorithm, we illustrate its execution using an example:

Example 1 *Consider a two stage game with 3 machines where each machine can choose either S or L, and 5 jobs that have running times of* $1.5, 2, 3, 4$ *and* 5. *Figure 1 depicts the four possible ways to select strategies for the 3 machines (all other options have the same number of machines that choose S,L and are thus built equivalently). In each of the four cases, the jobs are assigned to the machines in an assignment that is stable from the perspective of the owners of the jobs. For each profile of the machines, the equilibrium for the jobs is found (using the algorithm we have shown before) and the bundles of jobs on each machine are swapped so that on L-machines the load will increase as the index of the machine increases, and on S-machines, the load will decrease. The algorithm then begins with all machines using the longest-first scheduling policy (Figure 1-I), and all jobs in the equilibrium which is ordered as described above. Then, one of the machines (the one with the lowest index which is by construction worse off in the equilibrium) switches to the shortest-first policy (Figure 1-II). The jobs are again assigned in a stable manner and ordered on the machines in the same manner (and so the machine that switched will become the machine that is the worst off among all S machines in this new configuration). Notice that the machine that switched from L to S now has a later completion time, and the change is thus adopted. The algorithm then continues to switch yet another machine (the second one) to the S scheduling policy (Figure 1-III). Once again this machine gains from the change and the change is adopted. Finally, when the last machine switches to the S policy, its situation is worse (Figure 1-IV), and so the algorithm rejects this change and terminates. The profile* (S, S, L) *is the equilibrium for the machines.*

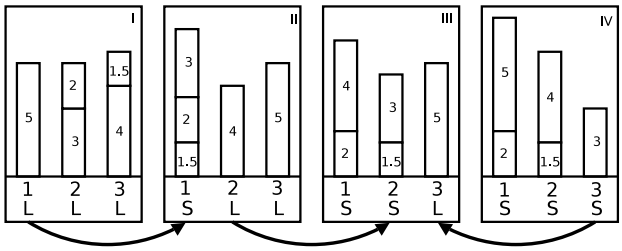

Figure 1: An example of the computation to find an equilibrium of the two-stage game. The machines use strategies S, L only, and jobs that appear lower in the diagram are scheduled earlier. The profile (S, S, L) is the one that yields the equilibrium in the 2-stage game.

Claim 8 *Algorithm "Find Machine and Job Equilibrium" constructs an equilibrium in any* (ϕ, ψ)*-two-stage-game.*

Proof: First note that the first step is feasible, i.e., it is possible to construct $\mathbf{x}(\mathbf{f})$ for every profile $\mathbf{f}$ in a manner that will satisfy conditions (a) (b) and (c), since we can just permute the machines that jobs are assigned to to get an equilibrium for the jobs with these properties. Since for every profile the assignment of jobs is in equilibrium, then when the algorithm halts, with a profile $\mathbf{f}$ that is the policy profile of the machines, then $x(\mathbf{f})$ is an equilibrium in the single stage

game $\mathbf{f}$. We now show that at any step no machine that uses a ϕ policy, is better off changing to the ψ policy. Suppose machine i changed its policy from ψ to ϕ and let $\mathbf{f}$ be the policy profile right after this change. By the construction of $x(\mathbf{f})$, $q_i(x(\mathbf{f})) \leq q_{i'}(x(\mathbf{f}))$ for every $i' \in \phi(\mathbf{f})$. Moreover by the construction $q_i(x(\psi, \mathbf{f}_{-\mathbf{i}})) = q_{i'}(x(\psi, \mathbf{f}_{-\mathbf{i}'}))$ for every $i' \in \phi(\mathbf{f})$ since every $i'' > i'$ for every $i''' \in \psi(\mathbf{f})$. Therefore, since i prefers ϕ given f_{-i} every $i' \in \phi(\mathbf{f})$ prefers ϕ given $f_{-i'}$. Similarly, after the algorithm halts, then in the final policy $\mathbf{f}$ every machine $i \in \psi(\mathbf{f})$ prefers policy ψ over ϕ given f_{-i}. □

If the machines are not constrained to use only two deterministic scheduling strategies, there are settings for which there is no pure equilibrium in the two stage game.

Theorem 5 *There exists a two-stage scheduling game in which the machines are allowed to use any deterministic consistent strategy, but there is no pure strategy equilibrium.*

Proof: In order to prove this result one needs to find a set of jobs, together with a number of machines, such that for every given profile of scheduling policies, there exists a machine which is better off deviating from its policy in the profile. We proved this theorem using a computer program in the following setting: 2 machines and 5 jobs, where the job lengths were $t_1 = 1$, $t_2 = 2.5$, $t_3 = 3$, $t_4 = 4.01$, and $t_5 = 4.6$.[2] Our computer program iterated over all possible profiles and found a machine that gains by deviating in each case. □

Even limiting the set of scheduling policies to exactly three policies can result in a setting with no equilibrium:

Theorem 6 *There exists a two-stage scheduling game in which the machines are allowed to use only 3 deterministic, consistent strategies, that has no pure equilibrium.*

Proof: [sketch] Consider 2 machines and 7 jobs, where the job lengths are $1, 2.5, 3, 4, 4.2, 5$ and 7. Suppose the machines can choose one of the following three policies: S, L and $\bar{f}$ where $\bar{f}(N) = (6, 5, 2, 7, 4, 1, 3)$ and for every $T \subseteq N$ $\bar{f}(T)$ is the order in $\bar{f}(N)$ restricted to the jobs in T. The reader can verify that in this case, one of the machines will deviate in any profile of scheduling policies. □

6 Conclusions and Future Work

We introduced a study of *competitive schedulers*. In particular, we looked at a 2-sided setting where both machines and users are strategic. We have shown that equilibria exist for the agents who control the jobs in a variety of situations, and that there are pure equilibria when the machines are restricted to sets of two policies.

The focus of our work is aimed at creating an initial model for the competition that is beginning to emerge within the growing world of cloud computing. This environment, where service providers naturally compete and attempt to attract customers is quite different from the setting where a central entity controls all the machines. Our analysis has introduced some basic (and mostly positive) results on the existence of pure strategy equilibrium in such settings.

There are many directions for future work, which include some more detailed and elaborated settings with competing schedules. For example: dynamic arrival of jobs, noisy execution times, machine and job failures and incomplete information about job sizes can all be considered with respect to a model with competing schedulers. We believe that due to the recent emerging trends, the scenario of competitive schedulers will become increasingly relevant, and that further exploration in this direction will prove to be highly important.

[2] One useful property of the profile we chose for running the program is that there are no ties in any assignment of the jobs to the machines.

References

Azar, Y.; Jain, K.; and Mirrokni, V. 2008. (almost) optimal coordination mechanisms for unrelated machine scheduling. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, 323–332. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Burguet, R., and Sakovics, J. 1999. Imperfect competition in auction designs. *International Economic Review* 40(1):231–47.

Galstyan, A.; Kolar, S.; and Lerman, K. 2003. Resource allocation games with changing resource capacities. In *In Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 145–152.

Kollias, K. 2008. Non-preemptive coordination mechanisms for identical machine scheduling games. In *SIROCCO '08: Proceedings of the 15th international colloquium on Structural Information and Communication Complexity*, 197–208.

Koutsoupias, E., and Papadimitriou, C. H. 2009. Worst-case equilibria. *Computer Science Review* 3(2):65–69.

Koutsoupias, E. 2003. Selfish task allocation. *Bulletin of EATCS* 81:79–88.

Monderer, D., and Tennenholtz, M. 2004. K-price auctions: Revenue inequalities, utility equivalence, and competition in auction design. *Economic Theory* 24(2):255–270.

Penn, M.; Polukarov, M.; and Tennenholtz, M. 2009. Random order congestion games. *Math. Oper. Res.* 34(3):706–725.

Shoham, Y., and Leyton-Brown, K. 2009. *Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press.

van Hoevel, W.-J.; Gomes, C. P.; Selman, B.; Lombardi, M.; and Risorgimento, V. 2007. Optimal multi-agent scheduling with constraint programming. In *In Proceedings of the Nineteenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*.

Wooldridge, M. J. 2000. *Reasoning about Rational Agents*. The MIT Press, Cambridge, Massachusetts.