

Incentive Compatible Online Scheduling of Malleable Parallel Jobs with Individual Deadlines

Thomas E. Carroll
Pacific Northwest National Laboratory
902 Battelle Boulevard
P.O. Box 999, MSIN J4-45
Richland, WA 99352 USA
Email: Thomas.Carroll@pnl.gov

Daniel Grosu
Department of Computer Science
Wayne State University
5143 Cass Avenue
Detroit, MI 48202 USA
Email: dgrosu@cs.wayne.edu

Abstract—We consider the online scheduling of malleable jobs on parallel systems, such as clusters, symmetric multiprocessing computers, and multi-core processor computers. *Malleable jobs* is a model of parallel processing in which jobs adapt to the number of processors assigned to them. This model permits the scheduler and resource manager to make more efficient use of the available resources. Each malleable job is characterized by *arrival time*, *deadline*, and *value*. If the job completes by its deadline, the user earns the payoff indicated by the value; otherwise, she earns a payoff of zero. The scheduling objective is to maximize the sum of the values of the jobs that complete by their associated deadlines. Complicating the matter is that users in the real world are *rational* and they will attempt to manipulate the scheduler by misreporting their jobs' parameters if it benefits them to do so. To mitigate this behavior, we design an incentive compatible online scheduling mechanism. *Incentive compatibility* assures us that the users will obtain the maximum payoff only if they truthfully report their jobs' parameters to the scheduler. Finally, we simulate and study the mechanism to show the effects of misreports on the cheaters and on the system.

I. INTRODUCTION

Interest in parallel computing has risen in recent years owing to the advancements in multi-core processing, cluster computing, and symmetric multiprocessing. In a significant number of parallel computing environments, the user specifies the number of processors and the amount of execution time needed when she submits her job to the scheduler. But many parallel jobs are *malleable*: there is a choice of the number of processors to execute the job on. The job's execution time is then a function of the number of processors. Even though it is more difficult to write an application to support malleability, the scheduler and resource manager should be able to make more efficient use of the computing resources by exploiting the jobs' malleability.

We consider the online scheduling of malleable jobs in which each user submits her job to the scheduler along with parameters that characterize the job's *arrival time* (or *release time*), *deadline*, *value*, and *malleability*. The scheduler does not learn about the job arrival sequence in advance; consequently, the scheduler must make decisions without knowledge of future arrivals. The *value* is the payoff that the user earns if the job completes by its *hard deadline*; otherwise, she earns zero. The objective of the scheduler is to maximize the sum of the

values of the jobs that complete by their respective deadlines. The scheduler can use *preemption* to achieve this goal. The problem as it stands is NP-hard [1]. Further complicating the problem is that the users are *rational*. They behave in their own self interest and they are welfare maximizing. They will misreport their jobs' parameters to the scheduler if it benefits them to do so. Such manipulations have a global impact as they introduce inefficiencies and suboptimal performance.

Schedulers in these competitive environments must be designed to cope with the self interest of the users. Fortunately for us, *economics* has been studying problems involving selfish entities and it has provided tools and solutions. One such tool is *mechanism design theory*. The theory provides the foundations for developing mechanisms that optimize the *social* (or *system-wide*) objective in the presence of self-interested users. *Mechanisms* can be thought of as games in which the rules are purposely designed to reach specific system-wide *outcomes* (or *solutions*) by eliciting specific behaviors or *strategies* from the users. There are few goals that need to be considered when designing mechanisms. The most important is incentive compatibility. A mechanism is *incentive compatible* if each user's welfare-maximizing strategy is to truthfully declare her parameters, regardless of the other users' strategies. Each user has a privately known *valuation* function that quantifies the benefit or loss for a specific outcome. A general *direct mechanism* consists of a function that computes a solution to the system-wide objective and a set of *payments*, one payment for each user. The user's welfare is then the sum of the valuation and her payment. The mechanism's choice of outcome motivates the users to truthfully declare their parameters.

In this work we design MPJS (Malleable Parallel Job Scheduling), a mechanism for the online scheduling of malleable jobs with individual hard deadlines. The objective of the scheduler is to maximize the sum of the values of the jobs that complete by their deadline. As an approximation to this objective, the scheduler assigns each job the quantity of processors that minimizes the job's execution time. We show that the resulting mechanism is incentive compatible and individually rational. *Individual rationality* means that truthful users will not incur a loss, regardless of what strategies

are employed by the others. We then perform a competitive analysis on the mechanism. *Competitive analysis* is a worst case comparison of the online scheduler against an optimal offline scheduler. We show that MPJS competitive ratio is at least m , where m is the number of processors in the system. Finally, we study by simulation the mechanism's properties. We experiment with misreports and observe how they impact the cheater and the system.

Organization. The rest of the paper is structured as follows. In Section II, we formally define the malleable job model. In Section III, we describe the problem of online scheduling of malleable jobs. In Section IV, we provide a brief introduction to mechanism design theory and show how its concepts relate to our problem. In Section V, we design an online mechanism that solves the problem of scheduling parallel malleable jobs. In Section VI, we show that the mechanism satisfies our goals. In Section VII, we study by simulation the mechanism by generating random jobs and observing how the mechanism operates. In Section VIII, we discuss related work. Finally, in Section IX, we draw conclusions and present future work.

II. MALLEABLE JOBS

Feitelson and Rudolph [2] classified parallel jobs into four classes. A job is *rigid* or *non-malleable* if it requires a fixed number of processors. The allocation of additional processors does not speedup the execution. When the number of processors can be adapted at the start of execution, the job is called *moldable*. A job is classified as *evolving* if the allotment changes due to its requirements. Finally, a *malleable* job adapts itself to the availability of processors given by the scheduler or the resource manager.

In this work we consider the problem of scheduling $n \geq 1$ malleable jobs on a parallel system comprising $m \geq 1$ identical processors. We assume that jobs cannot share processors and thus, jobs must be allocated an integral number of processors. We consider the *perfectly malleable jobs with setup times* model from [3], [4]. This model accurately accounts for costs associated with communication and coordination. The model is as follows: a malleable job j , $j = 1, 2, \dots, n$, requires p_j units of time to run on a single processor. The *execution time*, t_j , for job j when $m_j \in [1, m]$ processors are assigned to it is

$$t_j(m_j) = p_j/m_j + (m_j - 1)c, \quad (1)$$

where $c > 0$ is the setup time to manage a single process. The constant c is a system parameter that depends on the specific parallel computing environment. The first term of (1) gives the *computational time*, the second gives the *setup time*. The setup time captures the time spent coordinating and communicating between processes. Figure 1 shows the execution time and the speedup for this model as functions of the number of processors. The *speedup* is the ratio $p_j/t_j(m_j)$. The minimum execution time (maximum speedup) is obtained when job j is

assigned $m_j^{\min}$ processors,

$$m_j^{\min} = \begin{cases} \lfloor \sqrt{p_j/c} \rfloor & \text{if } t_j(\lfloor \sqrt{p_j/c} \rfloor) \leq t_j(\lceil \sqrt{p_j/c} \rceil), \\ \lceil \sqrt{p_j/c} \rceil & \text{if } t_j(\lfloor \sqrt{p_j/c} \rfloor) > t_j(\lceil \sqrt{p_j/c} \rceil). \end{cases}$$

The number of processors assigned to a job is bounded by the size of the system. Therefore, $m_j = \min\{m, m_j^{\min}\}$.

A scheduler for malleable jobs must make two decisions. It must decide what jobs to execute and how many processors to assign to each. Furthermore, the scheduler must ensure that the number of processors allocated at any given instant does not exceed m .

There is little prior work on hard deadline scheduling for parallel jobs. As a consequence, few models account for preemption costs. In this work we assume that once the scheduler allocates a number of processors to a given job, the processor allocation will not change even if the job is preempted and executed again. Since the number of processors allocated to a given job does not change across the execution we assume that the preemption costs are zero.

III. PROBLEM DESCRIPTION

We consider the online scheduling of malleable jobs on a parallel system. Each user j , $j = 1, 2, \dots, n$, owns a single job j . Job j is released to the system at time a_j (the *arrival time*) and has a *hard deadline* of $d_j > a_j$. If the job completes in the interval $(a_j, d_j]$, the user earns a payoff of v_j ; otherwise, she receives nothing. The scheduling is computed *online*: the scheduler knows only about the jobs that have arrived and it cannot foresee jobs arriving in the future.

We define the *type* of job j as $\theta_j = (a_j, d_j, p_j, v_j)$, which is a point from the *type space* Θ_j . Let $\Theta = \times_j \Theta_j$. The scheduler receives the vector $\theta = (\theta_1, \theta_2, \dots, \theta_n) \in \Theta$ which it uses to compute the schedule $S(\theta, t)$. Schedule $S(\theta, \cdot)$ gives the set of jobs and the number of processors allocated to each at time t . The number of processors allocated at t cannot exceed m . The goal of the scheduler is to maximize the sum of values of the jobs that complete by their deadline. The scheduler is permitted to preempt and migrate jobs in order to satisfy its objective. This deadline scheduling problem is known to be NP-hard [1].

Further complicating the scheduling is that the users are *rational* (self-interested and welfare-maximizing): a user will misreport her type if she believes it will benefit her. Instead of declaring θ_j , user j declares $\hat{\theta}_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j) \in \Theta_j$, where the components can be different from their actual values. Now, the scheduler computes the schedule using $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_n) \in \Theta$. The consequence of this behavior is that a suboptimal schedule may result if the scheduler does not explicitly account for the misreports. As an example of the detriment of misreports suppose that a scheduler implements a policy of least slack time (LST) first. At time t , job j 's *slack time* is the difference between available time $(d_j - t)$ and the remaining computational time. Jobs with the smallest slack time are executed first. If job j would not complete in the schedule, user j can manipulate the schedule by holding onto

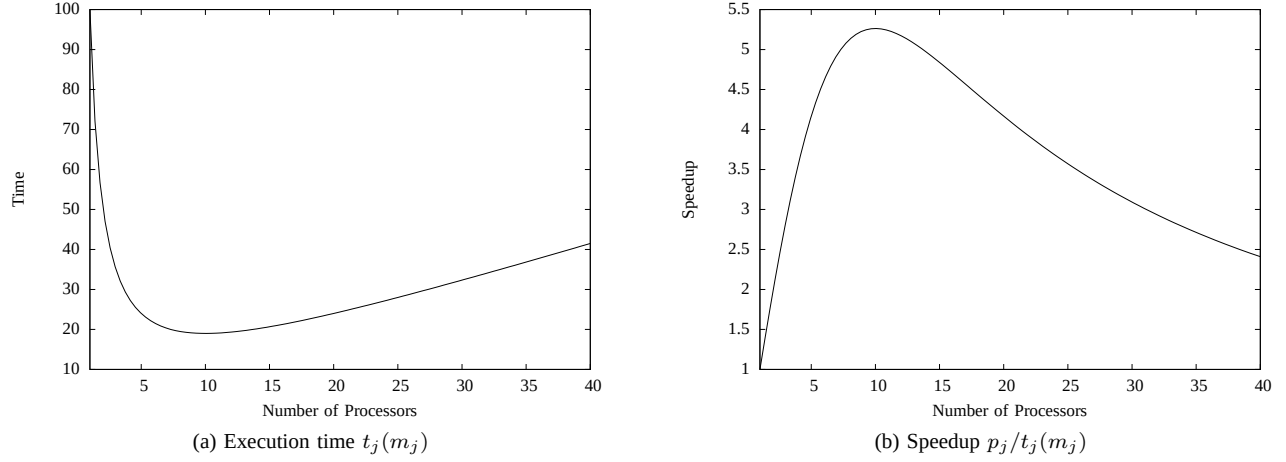


Fig. 1: Execution time and speedup of job j with $p_j = 100$ and $c = 1$. Maximum speedup is obtained when $m_j = \sqrt{p_j/c} = 10$.

the job or reporting a tighter deadline. User j 's misreports may cause other jobs to fail. In the next sections we show how to cope with the self-interest of the users, by providing them with incentives to report their true job parameters to the scheduler.

IV. MECHANISM DESIGN

In the classical (non-incentive compatible) scheduling literature, the aim is on designing algorithms and protocols to optimize some system-wide criteria such as minimizing the makespan or minimizing the response time. But most of these algorithms require users to truthfully declare their types. In the real world, users are rational. They will misreport their types if it benefits them to do so. These manipulations can result in suboptimal performance. Until recently, the parallel scheduling literature has paid little attention to rational manipulation.

Mechanism design, a sub-field of game theory, provides tools and techniques to solve this problem. Generally, a mechanism is a set of rules that implements a complex auction and it is designed to obtain specific outcomes that maximize (or minimize) the designer's objective even in the presence of self-interested users. A reference on mechanism design is [5]. In the following, we present the mechanism design framework and concepts that will be used in this paper.

A. Formulation

The mechanism design framework we use assumes that there is a center (i.e., a scheduler) that implements the mechanism. Users¹ interact with the scheduler through a direct mechanism $\Gamma = (\Theta_1, \Theta_2, \dots, \Theta_n, g(\cdot))$ in which the users declare the type of their job, denoted $\theta_i \in \Theta_i$. The function $g : \Theta \rightarrow \mathcal{O}$ maps the declared types to an outcome $o \in \mathcal{O}$. An outcome $o = (S(\cdot), r_1(\cdot), r_2(\cdot), \dots, r_n(\cdot))$ consists of a schedule $S(\hat{\theta}, t)$ and a payment $r_i(\hat{\theta})$ from each agent to the mechanism, where $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_n) \in \Theta$.

¹The mechanism design theory literature refers to users as *agents*.

User j 's utility is the quasi-linear function

$$u_j(g(\hat{\theta}), \theta_j) = -r_j(\hat{\theta}) + V_j(\hat{\theta}, \theta_j),$$

where $V_j(\hat{\theta}, \theta_j)$ evaluates to v_j if the job completes by its deadline in $S(\hat{\theta}, \cdot)$ and zero otherwise. We assume that each user is rational and risk indifferent, i.e., she chooses her actions to maximize her expected utility. In doing so, she may declare a type different from her true type.

B. Goals

There are two primary goals for most mechanisms, incentive compatibility and individual rationality.

In the following let $\hat{\theta}_{-j} = (\hat{\theta}_1, \dots, \hat{\theta}_{j-1}, \hat{\theta}_{j+1}, \dots, \hat{\theta}_n) \in \Theta_{-j}$, where $\Theta_{-j} = \times_{i \neq j} \Theta_i$, be the vector of declarations without user j 's declaration.

Definition 1. A direct mechanism Γ satisfies incentive compatibility (IC) if, $\forall j = 1, 2, \dots, n$, $\forall \hat{\theta}_j \in \Theta_j$, and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$,

$$u_j(g(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq u_j(g(\hat{\theta}_j, \hat{\theta}_{-j}), \theta_j).$$

A mechanism satisfies IC if the user's utility maximizing strategy is to truthfully declare her type, regardless of the strategies employed by the others. This is beneficial as the user does not need to expend resources to determine her optimal strategies. Furthermore, IC leads to a single strategy profile in which all rational agents choose to declare their true values.

The second goal in mechanism design is individual rationality².

Definition 2. A direct mechanism Γ satisfies individual rationality (IR) if, $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$,

$$u_j(g(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq 0.$$

In a mechanism that satisfies IR, a user who truthfully declares her type never incurs a loss. This goal is important

²Mechanism design theory literature sometimes refers to this goal as *voluntary participation*.

as rational users will freely choose to participate in the mechanism.

The foundations defined here are used in the following sections to design an online mechanism for scheduling malleable parallel jobs.

V. MECHANISM FOR SCHEDULING MALLEABLE JOBS

In this section we outline the design of a mechanism for the incentive-compatible online scheduling of malleable parallel jobs with individual deadlines. The scheduling objective is $\max \sum_j \hat{V}_j(\hat{\theta}_j, \theta_j)$, where $\hat{V}_j$ gives $\hat{v}_j$ if job j completes by its deadline, and zero otherwise.

To assist us in designing an incentive compatible mechanism, we examine how the misreports can be restricted. First, assume that user j learns her type at time a_j . Then, by considering *no early arrivals*, we have $\hat{a}_j \geq a_j$. Second, the system can ensure *no early departures* by holding each completed job j until $\hat{d}_j$. Thus, rational users declare $\hat{d}_j \leq d_j$. Finally, let the scheduler forcibly terminate all jobs executing longer than declared. User j then reports $\hat{p}_j \geq p_j$, allowing enough time for her job to complete execution.

The system invokes the scheduler upon an event. An *event* is either a job arriving or a job completing. The scheduler at these times must determine the set of jobs to execute and the number of processors to assign to each. Jobs executing on the system may need to be preempted to run the chosen jobs. These suspended jobs may be restarted in the future.

Notation. In the following let $m_j^*(p_j) = \arg \min_{1 \leq m_j \leq m} p_j/m_j + (m_j - 1)c$ denote the number of processors that minimizes the execution time of job j . Let $t_j^*(p_j) = \min_{1 \leq m_j \leq m} p_j/m_j + (m_j - 1)c$ denote the minimum execution time of job j . Finally, the users are indexed such that if $j < j'$ then $\hat{a}_j \leq \hat{a}_{j'}$. Ties are broken arbitrarily.

A scheduler needs to make two decisions when scheduling jobs. These decisions may be interdependent. The first is to determine the number of processors to assign; the second is to determine what jobs to execute. Alg. 1 implements these decisions as follows. First, any job j that executes is assigned $m_j^*(\hat{p}_j)$ processors. For the second decision, let $e_j(\hat{\theta}, t)$ give the amount of time that job j has executed by time t in schedule $S(\hat{\theta}, \cdot)$. Job j is completed if $e_j(\hat{\theta}, \hat{d}_j) \geq t_j^*(\hat{p}_j)$. At any time t there is a set of jobs available to run. Denote this set by *Avail*. Job j is in *Avail* if and only if it is released on or before t ($t \geq \hat{a}_j$), its deadline has not been exceeded ($e_j(\hat{\theta}, t) + \hat{d}_j - t \geq t_j^*(\hat{p}_j)$), and it has not completed ($e_j(\hat{\theta}, t) < t_j^*(\hat{p}_j)$). From *Avail*, jobs are scheduled to execute at t in non-increasing order of value (Alg. 1 line 5–13). The highest value jobs are extracted from *Avail*, one by one (line 6 of Alg. 1). Ties are broken by lower job index. This continues until a job is found that cannot be executed due to lack of processors. The algorithm then scans the available jobs for the highest value jobs which fit on the remaining free processors (Alg. 1 line 14–22). Line 15 of Alg. 1 returns $\perp$ if a job cannot be found to execute on the remaining free processors. Jobs can

Algorithm 1 MPJS-SCHEDULE($\hat{\theta}$)

```

1: for all  $t$  do
2:    $Avail \leftarrow \{j | (t \geq \hat{a}_j) \wedge (e_j(\hat{\theta}, t) < t_j^*(\hat{p}_j)) \wedge (e_j(\hat{\theta}, t) + \hat{d}_j - t \geq t_j^*(\hat{p}_j))\}$ 
3:    $procs \leftarrow m$ 
4:    $jobs \leftarrow \emptyset$ 
5:   while  $(Avail \neq \emptyset) \wedge (procs > 0)$  do
6:      $j \leftarrow \arg \max_{i \in Avail} \hat{v}_i$   $\triangleright$  Break ties by index
7:      $Avail \leftarrow Avail / \{j\}$ 
8:     if  $m_j^* > procs$  then
9:       break
10:    end if
11:     $jobs \leftarrow jobs \cup \{j\}$ 
12:     $procs \leftarrow procs - m_j^*$ 
13:  end while
14:  while  $(Avail \neq \emptyset) \wedge (procs > 0)$  do
15:     $j \leftarrow \arg \max_{i \in Avail | m_i^* \leq procs} \hat{v}_i$   $\triangleright$  Break ties by index
16:    if  $j = \perp$  then  $\triangleright$  Return  $\perp$  if no job found
17:      break
18:    end if
19:     $Avail \leftarrow Avail / \{j\}$ 
20:     $jobs \leftarrow jobs \cup \{j\}$ 
21:     $procs \leftarrow procs - m_j^*$ 
22:  end while
23:   $S(\hat{\theta}, t) \leftarrow jobs$ 
24: end for
```

Algorithm 2 MPJS-PAYMENT($\hat{\theta}$)

```

1: for all  $j$  do
2:   if  $e_j(\hat{\theta}, \hat{d}_j) \geq t_j^*(\hat{p}_j)$  then
3:      $r_j(\hat{\theta}) \leftarrow \arg \min_{\hat{v}_j' \geq 0} [e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j')), \hat{d}_j) \geq t_j^*(\hat{p}_j)]$ 
4:   else
5:      $r_j(\hat{\theta}) \leftarrow 0$ 
6:   end if
7: end for
```

be preempted and suspended. Jobs will always be assigned m_j^* processors, no matter the percentage that has been completed.

One question is why the simple priority rule of $\hat{v}_j$ is chosen over another priority rule that takes into account the elapsed time? One of the biggest problems in designing a priority rule for our setting is that there is no connection between the job's value and the job's dimensions. Furthermore, if we account for elapsed time, we cannot guarantee $\sum V_j \geq \max \hat{v}_j$, a useful property for our scheduling objective.

The payments $r_1(\hat{\theta}), r_2(\hat{\theta}), \dots, r_n(\hat{\theta})$ are computed using Alg. 2. The payments take the form of the so called *critical payments* [6]: user j pays zero if her job does not complete ($e_j(\hat{\theta}, \hat{d}_j) < t_j^*(\hat{p}_j)$); otherwise, she pays the minimum value $\hat{v}_j'$ that allows her job to complete by its deadline. Since the job completes when declared $\hat{v}_j$, we have $\hat{v}_j' \leq \hat{v}_j$. The payment

TABLE I: Declared job types.

j	$\hat{a}_j$	$\hat{d}_j$	$\hat{p}_j$	$\hat{v}_j$
1	0	40	100	5
2	4	25	36	4
3	5	25	64	10
4	10	25	64	10
5	22	35	36	5

$r_j(\hat{\theta})$ can be computed at any time $t \geq \hat{d}_j$ when all other jobs that overlap with j 's time window $(\hat{a}_j, \hat{d}_j]$ are known. In certain circumstances it would be advantageous to compute the payments in batch, offline.

The following example illustrates the execution of the mechanism. Consider scheduling the five jobs with declared types given in Tab. I on a system defined by $m = 16$ and $c = 1$. Job 1 arrives first and is allocated ten processors. Job 2 arrives at time 4 and requires six processors. The job begins executing immediately since there are six unassigned processors. Job 3 then arrives. This job needs eight processors, but all processors are accounted for. Since job 3 is declared a value greater than the executing jobs, preemption occurs. The scheduler then allocates jobs to processors in non-increasing order of declared value. This results in job 2 and job 3 being selected for execution. Even though $\hat{v}_1 > \hat{v}_2$, job 2 is selected over job 1 because the number of available processors (8) is less than the number needed for job 1 (10). Job 2 is preempted at time 10 when job 4 arrives. Job 3 completes at time 20 and job 2 resumes execution until job 5 arrives at time 22. Job 2 is preempted and suspended in favor of executing job 5, since job 5 has a greater value than job 2. Job 4 completes at time 25. At this point job 1 resumes execution. Job 5 completes at time 33 and job 1 completes at time 39. Job 2 never completes.

The payments are assigned as follows. Job 2 does not complete, thus, $r_2 = 0$. If either job 3 or job 4 declares a value less than 5 job 1 would never be preempted and 3 and 4 would never be selected. This results in a payment of $r_3 = r_4 = 5$. Job 5 pays $r_5 = 4$ because if less job 2 would resume in its place and it would never complete. Finally, $r_1 = 0$ since job 1 would complete if declared a value of zero. The users obtain the following utilities: $u_1 = 5 - 0 = 5$, $u_2 = 0$, $u_3 = 10 - 5 = 5$, $u_4 = 10 - 5 = 5$, and $u_5 = 5 - 4 = 1$.

Let's examine the case of user 2 misreporting the value of its job. That is job 2 is declared a value $\hat{v}_2 = 12 > v_2 = 4$. As discussed above job 1 is scheduled first and then at time 4 job 2 is scheduled on 6 processors. Job 3 arrives at time 5. This job needs 8 processors, but not enough processors are available. Since job 3 is declared a value greater than the executing jobs, preemption occurs. Job 3 is scheduled on 8 processors and job 2 is continuing its execution on 6 processors. At time 10 job 4 arrives. Job 4 does not preempt any of the jobs since it does not have a value greater than the jobs that are currently scheduled (job 2 and 3). Job 2 completes execution at time 15. We are now able to compute the payment for job 2. If job 2 is declared a value less than 10, it would not have been able to complete its execution, since job 4 would have preempted it. Thus, the

payment of job 2 is $r_2 = 10$ (i.e., the minimum value that allows the completion of job 2 by the deadline). Job 2 obtains a utility of $u_2 = -r_2 + \hat{v}_2 = -10 + 4 = -6$. Similarly, we can show that misreporting the other parameters would lead to reduced utility. Job 2 would not have incentives to misreport its parameters to the scheduler, since the mechanism that is used by the scheduler is incentive compatible (the proof of incentive-compatibility is provided in the next section).

These mechanism can be implemented naively using a simple ordered list. When jobs arrive, they are inserted into the list in non-increasing order of declared value. Each insertion costs $O(n)$ in the worst case. The two inner loops of Alg. 1 are implemented with a single scan of the list from front to back. Again, this requires $O(n)$. The total cost for n jobs is then $O(n^2)$. Alg. 2 can be implemented using Alg. 1 by sequentially setting the value of each job j to the value of the jobs with lesser value. Therefore, the naïve mechanism implementation runs in worst case time $O(n^4)$.

In the next section, we show that MPJS is an incentive compatible mechanism and that it satisfies individual rationality.

VI. MPJS MECHANISM'S PROPERTIES

In this section we study the properties of MPJS. We first prove that MPJS is incentive compatible. To prove this, we first show that the schedule $S(\hat{\theta}, \cdot)$ obtained by MPJS is monotone.

Definition 3 (Monotonicity). A schedule $S(\hat{\theta}, \cdot)$ is monotone if, $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$,

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j)$$

then

$$e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}'_j, \hat{v}'_j)), \hat{d}'_j) \geq t_j^*(\hat{p}'_j),$$

where $\hat{a}'_j \leq \hat{a}_j$, $\hat{d}'_j \geq \hat{d}_j$, $\hat{p}'_j \leq \hat{p}_j$, and $\hat{v}'_j \geq \hat{v}_j$.

If a monotone $S(\hat{\theta}, \cdot)$ schedules job j with a "constrained" type, it must also schedule job j with a "relaxed" type. We show that $S(\hat{\theta}, \cdot)$ obtained by MPJS is monotone. The combination of $S(\hat{\theta}, \cdot)$ being monotone and the critical payments guarantee that MPJS is incentive compatible [7].

To prove monotonicity, we first show that by relaxing the job's arrival time or deadline the job is still scheduled. Then, we prove that by decreasing the amount of processing time it requires the job is still scheduled. Finally, we prove that increasing the value of the job it does not harm the job's schedulability. Combining these three results we prove monotonicity.

We begin the proof of monotonicity by showing that relaxing job j 's arrival time or deadline does not impact its schedulability.

Lemma 1. The following holds for schedule $S(\hat{\theta}, \cdot)$: $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$, if

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j)$$

then

$$e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}_j, \hat{v}_j)), \hat{d}'_j) \geq t_j^*(\hat{p}_j),$$

where $\hat{a}'_j \leq \hat{a}_j$ and $\hat{d}'_j \geq \hat{d}_j$.

Proof: Assume for contradiction that job j completes when declared $\hat{\theta}_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)$ but not when declared $\hat{\theta}'_j = (\hat{a}'_j, \hat{d}'_j, \hat{p}_j, \hat{v}_j)$, where $\hat{a}'_j \leq \hat{a}_j$ and $\hat{d}'_j \geq \hat{d}_j$. The change affects when job j becomes available for execution and placed in the set *Avail* in Alg. 1. The availability conditions $t \geq \hat{a}_j$ and $e_j(\hat{\theta}, t) + d_j - t \geq t_j^*$ are affected such that the former condition is true earlier and the latter condition is true later when job j is declared with a type $\hat{\theta}'_j$ instead of $\hat{\theta}_j$. The change of type does not affect when the job is selected to execute as this is based on $\hat{v}_j$. Since job j with type $\hat{\theta}_j$ is completed in the shorter time window, job j with type $\hat{\theta}'_j$ will also complete, thus, contradicting our assumption. ■

Next, we show that decreasing the amount of processing time does not damage schedulability.

Lemma 2. *The following holds for schedule $S(\hat{\theta}, \cdot)$: $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$, if*

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j)$$

then

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}'_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}'_j),$$

where $\hat{p}'_j \leq \hat{p}_j$.

Proof: Assume for contradiction that job j completes when declared $\hat{\theta}_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)$ but not when declared $\hat{\theta}'_j = (\hat{a}_j, \hat{d}_j, \hat{p}'_j, \hat{v}_j)$, where $\hat{p}'_j \leq \hat{p}_j$. The change affects the second availability condition $e_j(\hat{\theta}', t) + d_j - t \geq t_j^*(\hat{p}'_j)$. This condition will be true later for the latter type as $t_j^*(\hat{p}'_j) \leq t_j^*(\hat{p}_j)$. Second, the latter type requires fewer processors and thus, can be executed in situations where the former type cannot. By assumption job j with the former type completes. Thus, job j with the latter type must complete, a contradiction. ■

Lastly, we show that increasing the values does not harm schedulability.

Lemma 3. *The following holds for schedule $S(\hat{\theta}, \cdot)$: $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$, if*

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j),$$

then

$$e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}'_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j),$$

where $\hat{v}'_j \geq \hat{v}_j$.

Proof: Assume for contradiction that job j completes when declared $\hat{\theta}_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)$ but not when declared $\hat{\theta}'_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}'_j)$, where $\hat{v}'_j \geq \hat{v}_j$. The schedule $S(\hat{\theta}, \cdot)$ executes jobs in non-increasing order of declared value. Job j with the latter type will be earlier in this order than when declared the former type. Because of this, job j with the latter type has a better chance of being executed. Since job j completes when its type is $\hat{\theta}_j$, it will also complete when its type is $\hat{\theta}'_j$, thus, contradicting our assumption. ■

We prove that MPJS is monotone by combining the above three lemmas.

Theorem 1. *Schedule $S(\hat{\theta}, \cdot)$ obtained by MPJS is monotone.*

Proof: Assume that job j completes when declared $\hat{\theta}_j = (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)$ but not when $\hat{\theta}'_j = (\hat{a}'_j, \hat{d}'_j, \hat{p}'_j, \hat{v}'_j)$, where $\hat{a}'_j \leq \hat{a}_j$, $\hat{d}'_j \geq \hat{d}_j$, $\hat{p}'_j \leq \hat{p}_j$, and $\hat{v}'_j \geq \hat{v}_j$. By Lem. 1, $e_j((\hat{\theta}_{-j}, (\hat{a}_j, \hat{d}_j, \hat{p}_j, \hat{v}_j)), \hat{d}_j) \geq t_j^*(\hat{p}_j)$ implies $e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}_j, \hat{v}_j)), \hat{d}'_j) \geq t_j^*(\hat{p}_j)$ $\forall j = 1, 2, \dots, n$ and $\forall \hat{\theta}_{-j} \in \Theta_{-j}$. By Lem. 2, $e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}_j, \hat{v}_j)), \hat{d}'_j) \geq t_j^*(\hat{p}_j)$ implies $e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}'_j, \hat{v}_j)), \hat{d}'_j) \geq t_j^*(\hat{p}'_j)$. Finally by Lem. 3, $e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}'_j, \hat{v}_j)), \hat{d}'_j) \geq t_j^*(\hat{p}'_j)$ implies $e_j((\hat{\theta}_{-j}, (\hat{a}'_j, \hat{d}'_j, \hat{p}'_j, \hat{v}'_j)), \hat{d}'_j) \geq t_j^*(\hat{p}'_j)$. Therefore, job j with type $\hat{\theta}'_j$ will complete if it completes when its declared type is $\hat{\theta}_j$. ■

We have proved that $S(\hat{\theta}, \cdot)$ is monotone. Using this and the following theorem from [7], we prove that MPJS satisfies incentive compatibility.

Theorem 2. *A mechanism is incentive compatible if and only if: (1) The scheduling algorithm is monotone, and (2) the payments are critical payments.*

Theorem 3. *MPJS is an incentive-compatible mechanism.*

Proof: This follows directly from Thm. 1 and Thm. 2. ■

We have shown that MPJS is an incentive compatible mechanism. Next, we show that it satisfies individual rationality.

Theorem 4. *Mechanism MPJS is individually rational.*

Proof: Let job j have true type $\theta_j = (a_j, d_j, p_j, v_j)$. If job j does not complete by its deadline, then $V_j((\hat{\theta}_{-j}, \theta_j)) = 0$ and $r_j((\hat{\theta}_{-j}, \theta_j)) = 0$. For this case, j 's utility $u_j((\hat{\theta}_{-j}, \theta_j), \theta_j) = -r_j((\hat{\theta}_{-j}, \theta_j)) + V_j((\hat{\theta}_{-j}, \theta_j)) = 0$. If job j completes, then $V_j((\hat{\theta}_{-j}, \theta_j)) = v_j$ and $r((\hat{\theta}_{-j}, \theta_j)) \leq v_j$. This implies $u_j((\hat{\theta}_{-j}, \theta_j), \theta_j) \geq 0$. Therefore, $u_j((\hat{\theta}_{-j}, \theta_j), \theta_j) \geq 0$, proving that MPJS is individually rational. ■

We next perform a competitive analysis on MPJS. *Competitive analysis* considers the worst case performance of an online algorithm to an optimal offline algorithm that sees the sequence of inputs in advance. A typical analysis has an “adversary” choosing the inputs to make the performance as poor as possible. We say that an algorithm is c competitive if the ratio between the performance of the online algorithm to the offline algorithm is $1/c$, where $c > 1$ is a constant.

Theorem 5. *The competitive ratio of MPJS is at least m , where $m \geq 2$, when all jobs arrive at time zero.*

Proof: Consider $m + 1$ jobs. Job 1's processing time, deadline, and value are $p_1 = cm^2$, $d_1 = (2m - 1)c$, and $v_1 = v$, respectively. For each job j , $p_j = c$, $d_j = c$ and $v_j = v - \epsilon$, where $j = 2, 3, \dots, m + 1$ and $\epsilon > 0$ is small. MPJS schedules job 1 on m processors for $(2m - 1)$ time units. Jobs 2, 3, $\dots$, $m + 1$ miss their deadline. The offline scheduler allocates jobs 2, 3, $\dots$, $m + 1$ a single processor for c time units. The schedule computed by MPJS has a value of v , while the offline variant obtains a value of $m(v - \epsilon)$. The

competitive ratio is then at least m .

VII. EXPERIMENTAL RESULTS

We study by simulation the MPJS mechanism. The simulation was conducted using software written specifically for the purpose of modeling the mechanism using Ruby, a general purpose object-oriented programming language. In this simulation we investigate the effects of misreports on the scheduling of 50 parallel jobs ($n = 50$) on a parallel computer comprising 64 identical processors ($m = 64$) and $c = 1.0$.

The true values for the jobs were generated as follows. For each job j , $j = 1, 2, \dots, 50$, we uniformly draw $m_j^{\min}$, the number of processors that minimizes the execution time, from the set $\{16, 17, \dots, 48\}$. Then, we set $p_j = m_j^{\min}(m_j^{\min} + 1)$, which gives an execution time of $t_j = 2 * m_j^{\min}$ on $m_j^{\min}$ processors. The arrival time a_j is randomly drawn from the uniform distribution on the set $\{0, 1, \dots, 2000\}$. Then, the deadline is $d_j = a_j + \rho_j t_j$, where ρ_j is drawn from the uniform distribution on the interval $[1, 5)$. Finally, we randomly draw v_j from the set $\{100, 101, \dots, 1000\}$. After generating the values, we ran the mechanism with all jobs reporting truthfully and then selected a job that completed, had a positive payment, and earned a positive utility. Without loss of generality, let us label this job as j . The true type for this job is $\theta_j = (932, 1320, 1560, 600)$. The minimum completion time is achieved when $m_j^{\min} = 39$ processors are assigned to it. In the following study, user j (the owner of job j) attempts various misreports.

We consider nine cases from the perspective of user j . The cases are:

- (1) User j reports truthfully.
- (2) User j reports $(a_j, d_j, p_j, \hat{v}_j)$, where $\hat{v}_j > v_j$.
- (3) User j reports $(a_j, d_j, p_j, r_j(\theta))$, where $r_j(\theta)$ is her payment when all users declare truthfully.
- (4) User j reports $(a_j, d_j, p_j, \hat{v}_j)$, where $\hat{v}_j < r_j(\theta)$.
- (5) User j reports $(a_j, d_j, \hat{p}_j, v_j)$, where $\hat{p}_j > p_j$.
- (6) User j reports $(a_j, d_j, \hat{p}_j, v_j)$, where $\hat{p}_j < p_j$.
- (7) User j reports $(a_j, \hat{d}_j, p_j, v_j)$, where $\hat{d}_j < d_j$.
- (8) User j reports $(a_j, \hat{d}_j, p_j, v_j)$, where $\hat{d}_j > d_j$.
- (9) User j reports $(\hat{a}_j, d_j, p_j, v_j)$, where $\hat{a}_j > a_j$.

User j 's parameters for the above cases are provided in Tab. II.

We first consider the effects of user j 's cheating on her utility and payment. Figure 2 gives the payment, r_j , and utility,

■ u_j , for the nine cases. As we expected, case (1) in which user j is truthful results in utility at least as great as the other cases. She pays 392 units and earns 208 units of utility. In case (2), the user reports $\hat{v}_j > v_j$ but she pays the same as in case (1) due to the properties of critical payments. The user in case (3) declares $\hat{v}_j = r_j(\theta)$ and, again due to critical payments, she pays the same as she did in case (1). Her job does not complete in case (4) as she declared $\hat{v}_j < r_j(\theta)$. Therefore, $r_j = 0$ and $u_j = 0$. In case (5) user j pays 11.7% more because she reports a larger job, $\hat{p}_j > p_j$, that is more difficult to schedule. Consequently, the larger payment reduces her utility by 22.1%. In case (6), she declares that her job is smaller than what it actually is. The system believes that the job completes, but the job does not actually finish due to the lack of processing time. The user pays 114 units but her payoff is zero, giving a negative utility. The user declares an earlier deadline in case (7). Again, this makes the job more difficult to schedule; therefore, the user pays 11.7% more, which reduces her utility by 22.1%. In case (8), the job is held beyond its actual deadline. Since the user does not earn a payoff, she experiences a loss. In the final case, case (9), the user reports an arrival time $\hat{a}_j > a_j$. This makes the job more difficult to schedule, which increases her payment by 17.3%.

We now consider the effects of misreports on the overall system utility. Incentive compatibility does not guarantee that the sum of the utilities is maximized when all users are truthful. Misreports may result in increasing or decreasing the sum and the actual effect cannot be precisely predicted. The sum of payments and the sum of utilities are given in Fig. 3. In the truthful case, 38 out of the 50 jobs complete. In case (2), the same 38 jobs complete, but the higher value report results in two jobs paying more. The opposite happens in case (3), with three jobs paying less than they do in the truthful case. In case (4), 37 of the 38 jobs from case (1) complete. As we mentioned above, job j does not complete. Some jobs pay more, while other jobs pay less. In case (5), 38 jobs complete. One job that fails to complete in case (1), completes, which forces another job to be abandoned. Some jobs pay more while others pay less. In case (6), 37 jobs complete (job j fails to complete). The jobs that conflict with job j pay less. In case (7) and case (8), the change in the sum was strictly due to changes in j . In the last case, 37 jobs complete. Similar to the other cases, some jobs pay more, while others pay less.

VIII. RELATED WORK

A significant body of work on malleable job scheduling focuses on the objective of minimizing the makespan. Such work in an *offline* (batch) context includes [8]–[12]. A framework for scheduling malleable jobs on cluster of clusters was presented in [13]. Their job model is perfectly malleable between a minimum and a maximum number of processors. If fewer processors than the minimum are assigned to the job, the job would fail. Any number above the maximum do not achieve any further speedup. Kettimuthu *et al.* [14] are one of the few researchers to consider preemption in the scheduling of parallel jobs. They examined the benefits of preemption to

TABLE II: Job j 's parameters

Case	$\hat{a}_j$	$\hat{d}_j$	$\hat{p}_j$	$\hat{v}_j$
1	932	1320	1560	600
2	932	1320	1560	800
3	932	1320	1560	392
4	932	1320	1560	391
5	932	1320	1980	600
6	932	1320	1260	600
7	932	1120	1560	600
8	932	1450	1560	600
9	1200	1320	1560	600

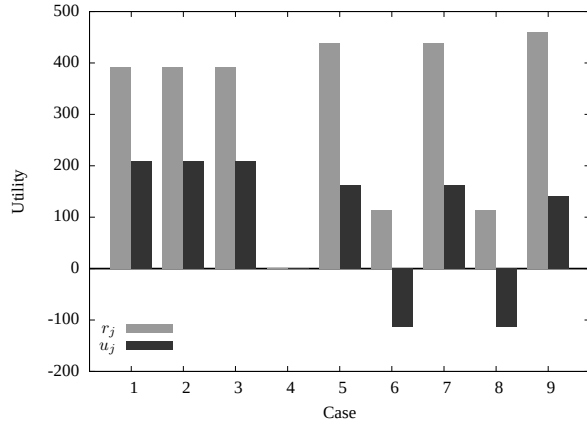


Fig. 2: User j 's utility, u_j , and payment, r_j .

improve *response time* by preempting long running jobs to execute short running jobs. Jansen and Zhang [15] considered the scheduling of malleable jobs with precedence constraints. They solved the problem in two phases. First, they select an allotment of processors and then they schedule the jobs. The only work that we know that considers the scheduling of malleable jobs with deadlines is [16]. The proposed algorithm divides the jobs into two groups by the tightness of deadlines. In our prior work [17], we showed that this algorithm is not monotone. Furthermore, the algorithm cannot be readily adapted to the online scenarios. Imreh [18] proposed an algorithm to minimize the height for the online strip packing problem. This problem is identical to the problem of scheduling perfectly malleable jobs to minimize makespan. Dutton and Mao [3] proposed the ECT algorithm for online scheduling to minimize makespan. For each job ECT chooses the number of processors to minimize the job's completion time. In followup work, Havill and Mao [4] gave an online algorithm that allocates solely at the minimum execution time.

The above algorithms were not designed to tolerate manipulation by rational users. As we said above, manipulations cause system degradation. We employ mechanism design theory to develop a scheduling algorithm that operates in a competitive, self-interested environment. Nisan and Ronen [19] were credited with the development of algorithmic mechanism design which examines mechanism design problems in competitive settings. A reference on mechanism design is [5]. An incentive compatible mechanism for batch scheduling with priorities was designed by Mutz *et al.* [20]. Carroll and Grosu [17] designed an offline mechanism for scheduling rigid parallel jobs. How the online setting changes standard mechanism design is discussed in [21]. Porter [22] proposed an incentive compatible online mechanism for the scheduling of a single processor real time system. To the best of our knowledge our research is the first to consider mechanism design and incentives within the context of scheduling parallel malleable jobs.

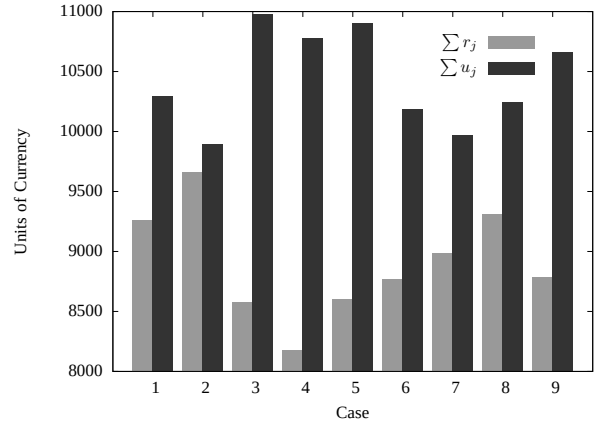


Fig. 3: Sum of payments, $\sum r_j$, and utility, $\sum u_j$, for the system.

IX. CONCLUSION

In this work we designed MPJS, an incentive compatible mechanism for the online scheduling of malleable parallel jobs. Scheduling malleable jobs requires two decisions: (1) how many processors to assign to jobs, and (2) which jobs to schedule. These decisions may be interdependent. The mechanism assigns the number of processors to achieve minimum execution time. It then schedules jobs in non-increasing order of their reported values. When the first job is found that does not fit on the remaining free processors, the mechanism then searches for the highest value jobs that fit. We show that MPJS is incentive compatible and that it satisfies individual rationality. We then show that MPJS has a competitive ratio of m . Finally, we study by simulation the properties of the mechanism.

For future work, we plan to examine how this mechanism operates in a real world setting. We also plan to re-examine batch scheduling of rigid jobs with the goal of designing a mechanism with a better approximation ratio.

ACKNOWLEDGMENTS

This research was supported, in part, by the NSF grant DGE-0654014.

REFERENCES

- [1] J. Du and J. Y.-T. Leung, "Complexity of scheduling parallel tasks systems," *SIAM J. Discrete Math.*, vol. 2, no. 4, pp. 473–487, 1989.
- [2] D. G. Feitelson and L. Rudolph, "Toward convergence in job schedulers for parallel supercomputers," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer-Verlag, 1996, pp. 1–26.
- [3] R. A. Dutton and W. Mao, "Online scheduling of malleable parallel jobs," in *Proc. of the IASTED Int. Conf. on Parallel and Distributed Computing and Systems (PDCS '07)*, 2007.
- [4] J. T. Havill and W. Mao, "Competitive online scheduling of perfectly malleable jobs with setup times," *European J. Oper. Res.*, vol. 187, pp. 1126–1142, 2008.
- [5] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, Eds., *Algorithmic Game Theory*. Cambridge University, 2007.
- [6] N. Nisan, "Introduction to mechanism design (for computer scientists)," in *Algorithmic Game Theory*, N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, Eds. Cambridge University, 2007, pp. 209–241.

- [7] A. Mu'alem and N. Nisan, "Truthful approximation mechanisms for restricted combinatorial auctions," *Games Econom. Behav.*, vol. 64, pp. 612–631, 2008.
- [8] J. Błażewicz, M. Machowiak, G. Mounié, and D. Trystram, "Approximation algorithms for scheduling independent malleable tasks," in *Proc. of 7th Int. Euro-Par Conference on Parallel Processing (Euro-Par '01)*, ser. Lecture Notes in Computer Science, vol. 2150, 2001, pp. 191–197.
- [9] J. Błażewicz, M. Machowiak, J. Węglarz, M. Y. Kovalyov, and D. Trystram, "Scheduling malleable tasks on parallel processors to minimize the makespan," *Ann. Oper. Res.*, vol. 129, pp. 65–80, 2004.
- [10] G. Mounié, C. Rapine, and D. Trystram, "A $3/2$ -approximation algorithm for scheduling independent monotonic malleable tasks," *SIAM J. on Computing*, vol. 37, pp. 401–412, 2007.
- [11] K. Jansen and L. Porkolab, "Linear-time approximation schemes for scheduling malleable parallel tasks," *Algorithmica*, vol. 32, no. 3, pp. 507–520, 2002.
- [12] K. Jansen, "Scheduling malleable parallel tasks: an asymptotic fully polynomial time approximation scheme," *Algorithmica*, vol. 39, pp. 59–81, 2004.
- [13] J. Buisson, O. Sonmez, H. Mohamed, W. Lammers, and D. Epema, "Scheduling malleable applications in multicluster systems," in *Proc. of the IEEE Int. Conf. on Cluster Computing (Cluster '07)*, 2007, pp. 372–381.
- [14] R. Kettimuthu, V. Subramani, S. Srinivasan, T. Gopalasamy, D. K. Panda, and P. Sadayappan, "Selective preemption strategies for parallel job scheduling," *Int. J. High Performance Comput.*, pp. 122–152, 2005.
- [15] K. Jansen and H. Zhang, "An approximation algorithm for scheduling malleable tasks under general precedence constraints," *ACM Trans. Algorithms*, vol. 2, no. 3, pp. 416–434, 2006.
- [16] O.-H. Kwon and K.-Y. Chwa, "Scheduling parallel tasks with individual deadlines," *Theoret. Comput. Sci.*, vol. 215, no. 1, pp. 209–223, 1999.
- [17] T. E. Carroll and D. Grosu, "An incentive-compatible mechanism for scheduling non-malleable parallel jobs with individual deadlines," in *Proc. of the 37th Int. Conf. on Parallel Processing (ICPP '08)*, 2008, pp. 107–114.
- [18] C. Imreh, "Online strip packing with modifiable boxes," *Oper. Res. Lett.*, vol. 29, no. 2, pp. 79–85, 2001.
- [19] N. Nisan and A. Ronen, "Algorithmic mechanism design," *Games Econom. Behav.*, vol. 35, pp. 166–196, 2001.
- [20] A. Mutz, R. Wolski, and J. Brevik, "Eliciting honest value information in a batch-queue environment," in *Proc. of the 7th IEEE/ACM Int. Conf. on Grid Computing (GRID 2007)*, 2007, pp. 291–297.
- [21] E. J. Friedman and D. C. Parkes, "Pricing WiFi at Starbucks — issues in online mechanism design," in *Proc. of the 4th ACM Conf. On Electronic Commerce*, 2003.
- [22] R. Porter, "Mechanism design for online real-time scheduling," in *Proc. 5th ACM Conf. on Electronic Commerce (EC '04)*, 2004, pp. 61–70.