

An Incentive-Compatible Mechanism for Scheduling Non-Malleable Parallel Jobs with Individual Deadlines*

Thomas E. Carroll
Department of Computer Science
Wayne State University
5143 Cass Avenue
Detroit MI 48202, USA
tec@cs.wayne.edu

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

Abstract

We design an incentive-compatible mechanism for scheduling n non-malleable parallel jobs on a parallel system comprising m identical processors. Each job is owned by a selfish user who is rational: she performs actions that maximize her welfare even though doing so may cause system-wide suboptimal performance. Each job is characterized by four parameters: value, deadline, number of processors, and execution time. The user's welfare increases by the amount indicated by the value if her job can be completed by the deadline. The user declares the parameters to the mechanism which uses them to compute the schedule and the payments. The user can misreport the parameters, but since the mechanism is incentive-compatible, she chooses to truthfully declare them. We prove the properties of the mechanism and perform a study by simulation.

1. Introduction

Parallel jobs that are typical to the science and engineering fields are conventionally scheduled on a shared distributed computing resource (e.g., a university-wide grid or cluster computer). Schedulers are designed to achieve goals such as minimizing the completion time of the last job executed (i.e., the makespan) or minimizing the average job completion time (i.e., response times). Traditional scheduling algorithms assume that the users are *obedient* in the sense that they do not manipulate the scheduling algorithm and that they truthfully report the parameters of their jobs to the scheduler. This fact is important as the scheduler must make allocation decisions based on these parameters. The obedience assumption is not valid in the real world where no *a priori* motivation for cooperation exists and *rational*

users (self-interested and welfare-maximizing) will misreport parameters to the scheduler to maximize (or minimize) her individual objective (e.g., meeting her deadline) even though such behavior can cause system-wide performance degradation. The goal of this paper is to design a scheduling mechanism that provides incentives to the users to report their true parameters, thus, leading to improved system efficiency.

Schedulers for shared resources must be designed to account for users' rationality. Economics studies rational behavior and it provides tools and solutions. One such tool is *mechanism design theory* [16]. The theory provides the foundations for developing mechanisms that optimize the social (system-wide) objective in a competitive, self-interested setting. In this work we consider two mechanism design goals. The first goal is incentive compatibility. A mechanism satisfies *incentive compatibility* if each user maximizes her own welfare if she truthfully reports her parameters to the mechanism, irrespective of the others' declarations. 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 vector of payments, one payment for each user. The combination of these are used to motivate the users to truthfully declare their parameters. A second mechanism design goal is *individual rationality* in which a truthful user is guaranteed to never incur a loss, irrespective of the others' declaration. This property is important as users voluntarily participate. Mechanisms can be designed to satisfy both goals simultaneously.

In this work we design the NMPJS (Non-Malleable Parallel Job Scheduling) mechanism for scheduling non-malleable parallel jobs with individual deadlines. A *non-malleable* parallel job requires a fixed number of processors and a given amount of time to properly execute to comple-

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

[†]Corresponding author.

tion. It does not execute faster if additional processors are allocated and it fails if fewer than the requested amount is allocated. Each self-interested user owns a single job that must be completed on or before a deadline. Each job is characterized by a valuation function that gives a positive real number if the job is completed within the deadline, and zero otherwise. Her *utility*, a measure of her welfare for a specific outcome, is given by the difference between her value and her payment. This type of utility function is classified in the literature as being *quasi-linear*. The user reports her job parameters consisting of the value, the deadline, the number of processors, and the execution time to NMPJS. The goal of scheduling in this context is to maximize the sum of the values of the jobs that meet their deadlines. NMPJS computes a schedule using an approximation algorithm and a vector of payments such that incentive compatibility and individual rationality is satisfied. To our knowledge this is the first attempt to design an incentive-compatible mechanism for scheduling non-malleable parallel jobs. This is also an example of an incentive-compatible mechanism where the agents are characterized by multiple parameters that span two different domains, money and time.

1.1. Related work

Feitelson and Rudolph [8] classified parallel jobs into three classes. A job is *rigid* or *non-malleable* if it requires a fixed number of processors. The allocation of additional processors does not speedup execution. When the number of processors can be adapted at the start of execution, the job is called *modable*. Jobs where the number of processors can be altered during execution are called *malleable*. In this work we only consider non-malleable jobs. The problem of scheduling non-malleable parallel jobs is similar to the *strip packing problem* (SPP). This problem consists of placing rectangles in a two-dimensional bin with the goal of minimizing height. A survey of strip packing algorithms is [12]. There are several algorithms for scheduling parallel jobs in the literature. Optimal and approximation algorithms for scheduling malleable jobs with the objective of minimizing the makespan are given in [4] and [5]. A simple approximation scheduling algorithm that minimizes the makespan is provided in [14]. Schwiegelshohn *et al.* [20] developed the SMART scheduling algorithm for minimizing the weighted response time of non-malleable jobs. Kwon and Chwa [11] designed an algorithm for scheduling non-malleable jobs with individual deadlines. An approximation algorithm for scheduling malleable jobs with precedence constraints is proposed in [10]. Havill and Mao [9] examined the on-line scheduling of non-malleable and malleable jobs. More recent work considered the scheduling of parallel jobs on “clusters of clusters” and grid computers. The support of and scheduling policies for parallel jobs are examined in the context of KOALA multicluster scheduler [6].

The above algorithms were not designed to tolerate manipulation by rational users. Manipulations cause system degradation and inefficiencies. In this work we employ the mechanism design theory to develop a scheduling algorithm that operates in a competitive, self-interested setting and that prevents such manipulations. Nisan and Ronen [15] were credited with the development of algorithmic mechanism design which examines mechanism design problems in computational settings. A recent reference on algorithmic mechanism design is [16]. Incentive compatibility is heavily reviewed in the literature. A general framework for designing incentive-compatible mechanisms for one parameter agents was proposed by Archer and Tardos [2]. They developed a general method to design incentive-compatible mechanisms for optimization problems that have general objective functions and restricted forms of valuations. The restrictions on valuations were eased in [1]. Mechanisms with verification were studied by Auletta *et al.* [3]. A mechanism with verification bases the payments on the agents’ outputs. Consequently, these mechanisms are pertinent to a large problem domain. All the mechanisms discussed so far are offline mechanisms, *i.e.*, the mechanism has full knowledge of all declarations before computing the output. Online mechanisms compute outputs without complete knowledge of the future. A reference on online mechanisms is [17]. Porter [18] designed an online mechanism for scheduling real-time jobs with hard deadlines.

1.2. Our contributions

In this work we design NMPJS, an incentive-compatible mechanism for scheduling non-malleable parallel jobs on a parallel system. Since NMPJS is an incentive-compatible mechanism it provides incentives to the users to report their true parameters. NMPJS computes a schedule using an algorithm based on the Bottom-Left packing heuristic. The payment scheme is designed in such a way to guarantee incentive compatibility. We prove that the mechanism satisfies incentive compatibility and individual rationality. We also study by simulation the properties of NMPJS. To the best of our knowledge, this is the first attempt at designing incentive-compatible mechanisms for scheduling non-malleable parallel jobs. NMPJS is also an example of incentive-compatible mechanism for multi-parameter, multi-domain agents.

1.3. Organization

The paper is structured as follows. In Section 2 we discuss the model for scheduling parallel jobs. In Section 3 we present the framework for designing mechanisms. In Section 4 we present NMPJS, a mechanism for scheduling parallel jobs. In Section 5 the properties of NMPJS are proved. In Section 6 we investigate the properties of the mechanism by simulation. In Section 7 we draw conclusions and present future work.

2. Model

Consider a parallel system comprising m identical processors. Denote each processor by p_i , $i = 1, 2, \dots, m$. There are n users attempting to execute their jobs on this system. Each user i , $i = 1, 2, \dots, n$, owns a single job i that requires w_i consecutive processors and executes in h_i units of time. The value of job i is v_i . If the job completes on or before its deadline, d_i , the user obtains v_i units of welfare. We denote the type of job i as the tuple $\theta_i = (v_i, d_i, w_i, h_i)$ from the space Θ_i of all possible types. Denote the type space of all jobs by $\Theta = \Theta_1 \times \Theta_2 \times \dots \times \Theta_n$ and $\Theta_{-i} = \Theta_1 \times \dots \times \Theta_{i-1} \times \Theta_{i+1} \times \dots \times \Theta_n$. Let $\theta = (\theta_1, \theta_2, \dots, \theta_n) \in \Theta$ denote the vector of types for all jobs and let $\theta_{-i} = (\theta_1, \dots, \theta_{i-1}, \theta_{i+1}, \dots, \theta_n) \in \Theta_{-i}$. The jobs must execute without preemption or migration and must be scheduled on contiguous processors to minimize communication latency (*i.e.*, scheduling on a *line* of processors [13]).

The scheduler allocates a line of processors of size w_i starting with processor p_i and a start time t_i to job i such that job i is the only job executing on the w_i processors and that $t_i + h_i \leq d_i$. Not all deadlines can be satisfied due to the limited nature of the resource. Therefore, the objective of the scheduler is

$$\max \sum_i v_i \cdot \mu(t_i + h_i \leq d_i)$$

where $\mu(\cdot)$ is an indicator function that evaluates to one if the enclosed expression is true and zero otherwise. In words, the objective of the scheduler is to maximize the sum of the values of the jobs that meet their respective deadlines.

The closest existing model to the one presented above is by Kwon and Chwa [11]. In their paper Kwon and Chwa proposed an algorithm for scheduling non-malleable jobs which we denote as the KC algorithm. In their model the value of a job is not general and is equal to the product of the processor quantity and the execution time. They refer to this product as “work”. Larger jobs perform greater amounts of work. The algorithm works as follows. The jobs are partitioned into two disjoint sets, $J_W = \{i | w_i > m/2\}$ and $J_N = \{i | w_i \leq m/2\}$. Each set of jobs is scheduled independently from the other. The algorithm selects the schedule that maximizes the sum of the work performed. Note that only one job in J_W can be scheduled at any instant. Thus, single processor scheduling algorithms such as *Job Sequencing with Deadlines* (JSD) [19] are pertinent. JSD is a dynamic programming algorithm that schedules jobs to maximize the sum of the values. With certain modifications it becomes a FPTAS. The jobs in J_N are further subdivided into two sets, $J_{NS} = \{i | i \in J_N, h_i > 0.5d_i\}$ and $J_{NL} = \{i | i \in J_N, h_i \leq 0.5d_i\}$, by how tight the job’s deadline is. The algorithm re-indexes the jobs in J_{NS} beginning from one to satisfy $d_i \geq d_{i+1}$. Similarly, it

indexes J_{NL} , ordering the jobs so that $d_i \leq d_{i+1}$. A job $i \in J_{NS}$ is scheduled such that $t_i = 0$. If job i cannot start at $t_i = 0$, the job is discarded. The jobs in J_{NL} are scheduled using *Earliest Deadline First* (EDF) heuristic.

Users can manipulate the KC algorithm by misreporting their types. Assume a system of two processors and two self-interested users, user 1 and 2, each with a single job. The types of the jobs are $\theta_1 = (8, 5, 2, 4)$ and $\theta_2 = (8, 6, 2, 4)$, where $v_i = w_i h_i$, $i = 1, 2$. The KC algorithm schedules job 1 and discards job 2 as 1’s deadline is earlier than 2’s. If job 2 declares $\hat{\theta}_2 = (8, 4, 2, 4)$, job 2 is scheduled and 1 is discarded. By manipulating the deadline, job 2 can make the scheduler discard job 1 even though job 2 does not require the earlier deadline.

Another problem with KC is that the values are coupled to the dimensions of the jobs. A general valuation scheme is preferred as the self-interested users can set the values of their jobs independent of the characteristics of their jobs. Our goal in this paper is to design a mechanism for scheduling non-malleable parallel jobs that prevents manipulations by the users and that also allows the users to assign values to their jobs independent of the amount of work required by the jobs.

The problem of scheduling non-malleable jobs with individual deadlines is a variation of the strip packing problem (SPP). There are various approximation algorithms for SPP, but most of them group items by their height and then contiguously pack items group by group. The grouping is suboptimal when the item’s value is not general and not correlated to the item’s height. Algorithms with the bottom-left stability property are more appropriate for the general value model. An algorithm satisfies the *bottom-left stability* if an item is positioned at the most bottom and left position available at the time of placement. This corresponds to the earliest possible start time for a job. We use this heuristic in designing our scheduling mechanism.

3. Mechanism Design

A rational user will misreport her type to the scheduler if it benefits her to do so. As seen with the KC algorithm, user i will report her job type as $\hat{\theta}_i$ instead of θ_i , where $\hat{\theta}_i \neq \theta_i$, as this increases the likelihood of her job being scheduled. 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 implement 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. In the following we present the mechanism design framework and concepts that will be used in this paper.

3.1. Formulation

The mechanism design framework we use assumes that there is a center that implements the mechanism. Agents (or

users) interact with the center through a direct mechanism $\Gamma = (\Theta_1, \Theta_2, \dots, \Theta_n, g(\cdot))$ in which agents declare the type of their job, denoted $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i) \in \Theta_i$. The function $g : \Theta \rightarrow O$ maps the declared types to an outcome $o \in O$. An outcome $o = (S(\cdot), r_1(\cdot), r_2(\cdot), \dots, r_n(\cdot))$ consists of a schedule $S(\hat{\theta})$ and a payment $r_i(\hat{\theta})$ from each agent to the mechanism, where $\hat{\theta} = (\hat{\theta}_1, \hat{\theta}_2, \dots, \hat{\theta}_n) \in \Theta$. Let $t_i(S(\hat{\theta}))$ denote the start time of job i under schedule $S(\hat{\theta})$. If i is not scheduled, then $t_i(S(\hat{\theta})) = \perp$.

User i 's utility is the quasi-linear function

$$u_i(g(\hat{\theta}), \theta_i) = -r_i(\hat{\theta}) + v_i \cdot \mu(t_i(S(\hat{\theta})) \neq \perp) \cdot \mu(t_i(S(\hat{\theta})) + h_i \leq d_i) \cdot \mu(\hat{w}_i \geq w_i) \cdot \mu(\hat{h}_i \geq h_i).$$

The second term of the utility function captures the fact that user i obtains utility $u_i = r_i + v_i$ if job i is scheduled, it completes within the deadline d_i , it is allocated w_i or more processors, and it reports an execution time \hat{h}_i greater than or equal to its true execution time h_i . 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. It is common practice for a scheduler to terminate jobs exceeding their execution times. Therefore, users will choose to declare execution times no less than their true execution times. Furthermore, the rigid job model supposes that a job cannot complete unless the required number of processors are allocated to it. Thus, the declared number of processors cannot be less than the minimum number of processors required by the job. The declaration $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ is such that $\hat{h}_i \geq h_i$ and $\hat{w}_i \geq w_i$.

3.2. Goals

In this section we outline the goals of mechanism design. There are two primary goals for most mechanisms, incentive compatibility and individual rationality.

In the following we denote by $\hat{\theta}_{-i} = (\hat{\theta}_1, \dots, \hat{\theta}_{i-1}, \hat{\theta}_{i+1}, \dots, \hat{\theta}_n) \in \Theta_{-i}$ the vector of declarations without user i 's declaration.

Definition 1. A direct mechanism Γ satisfies incentive compatibility (IC) if, for all i , $\hat{\theta}_i \in \Theta_i$, and $\hat{\theta}_{-i} \in \Theta_{-i}$,

$$u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\hat{\theta}_i, \hat{\theta}_{-i}), \theta_i).$$

A mechanism satisfies IC if the agent always maximizes her utility if she truthfully declares her type irrespective of the other agents' declarations. This leads to a single strategy profile in which all rational agents choose to declare their true values.

Definition 2. A direct mechanism Γ satisfies individual rationality (IR) if, for all i and $\hat{\theta}_{-i} \in \Theta_{-i}$,

$$u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq 0.$$

If an agent truthfully declares her type, she will never incur a negative utility. This is important as agents will freely choose to interact with the mechanism as losses are not possible.

The foundations defined in this section are used in the following to design a mechanism for scheduling non-malleable parallel jobs.

4. The NMPJS Mechanism

In this section we present the design of NMPJS (Non-Malleable Parallel Job Scheduling), an incentive-compatible mechanism that schedules non-malleable parallel jobs with individual deadlines with the objective of maximizing the sum of the values of the scheduled jobs, *i.e.*, $\max \sum_i v_i \cdot \mu(t_i + h_i \leq d_i)$. NMPJS is a direct mechanism comprising two parts, a scheduling algorithm and a payment scheme. Each user i declares her job type $\hat{\theta}_i$ to the scheduler (referred as the center in mechanism design theory). Using this information, the scheduler computes the schedule and the payments.

User i 's job type θ_i comprises four parameters: value v_i , deadline d_i , number of processors w_i , and execution time h_i . We do not use all the parameters in designing the objective due to the different domains of money (v_i) and time (d_i , w_i , and h_i). Designing functions to convert between domains is simple but often require assumptions such as exchanging money for time. We further decouple v_i from the amount of work required by job i , which is $w_i h_i$, to allow a generalized value system. It can be the case that small jobs can be of greater value than larger tasks. We now interpret v_i as how much user i is willing to pay to have her job completed on or before the deadline.

As we mentioned before, the problem of scheduling non-malleable parallel jobs to satisfy deadlines is a variant of the SPP. A non-malleable parallel job i can be viewed as a rectangular item of width w_i equal to the number of processors required to execute job i , and height h_i equal to the required execution time of job i . Since SPP is NP-hard, the problem of scheduling non-malleable parallel jobs is also NP-hard. Thus, we need to rely on an approximation mechanism in order to solve the problem in a feasible amount of time. An approximation to the true objective ($\max \sum v_i \cdot \mu(t_i + h_i \leq d_i)$) is simply scheduling the jobs in non-increasing order of values. Any job whose deadline cannot be met is not scheduled.

4.1. Scheduling algorithm

An approximate scheduling algorithm is needed to schedule the jobs. There are several algorithms for strip packing, but many of them are *shelf* or *level* packing [12] in which rectangular items (or jobs) of approximately the same height are grouped together and placed on sequential levels. All the bottoms of the rectangles at a level are positioned at the same distance from the strip bottom. Since the jobs are

ordered by height, the various heuristics can determine the height of the next level in a straight forward fashion. But in this problem, the value determines the objective and the value is not correlated with the job's dimensions. Therefore, we need to employ an algorithm that does not require ordering by height. One such algorithm is the polynomial-time *Bottom Left* (BL) [7] packing heuristic. BL satisfies the *bottom-left stability* condition: there are no positions to the bottom or to the left to which a packed item can slide. We employ the BL heuristics in computing the schedule for our proposed mechanism. Algorithm 1 is used to compute the schedule.

Algorithm 1

```

1: procedure NMPJS-SCHEDULE( $\hat{\theta}$ )
2:   Sort  $\hat{\theta}$  in non-increasing order of  $\hat{v}_i$ 
3:   for  $i \leftarrow 1, n$  do
4:     Find bottom-left position  $(t_i, p_i)$  in  $S$ 
       for  $(\hat{w}_i, \hat{h}_i)$ 
5:     if  $t_i + \hat{h}_i \leq \hat{d}_i$  then
6:       Schedule  $i$  at position  $(t_i, p_i)$  in  $S$ 
7:     else
8:       Mark  $i$  as unscheduled in  $S$ 
9:     end if
10:  end for
11:  return  $S$ 
12: end procedure

```

The polynomial-time Algorithm 1 works as follows. It sorts the jobs in order of non-increasing values. If there are jobs having the same value, those jobs are sorted in order of non-decreasing deadlines. If there are jobs with the same value and the same deadline then they are arbitrarily ordered. BL schedules each job at the earliest start time subject to the placement of the more valuable jobs. For each job i , BL finds the bottom leftmost available position that is at least \hat{w}_i wide and at least \hat{h}_i tall. If BL computes a position at which i cannot meet its deadline, the job is discarded. Otherwise, i is scheduled in that position. From this point forward, we assume that the jobs are indexed in order of non-increasing \hat{v}_i , non-decreasing \hat{d}_i (as the secondary ordering), and that the ties are broken arbitrarily. In the above algorithm, BL determines the start time, t_i , and the first processor in the line, p_i , in S for job i . If job i can be scheduled by the deadline, then job i is scheduled in S . Otherwise, i is marked as not scheduled.

4.2. Payment scheme

The second component of our mechanism is the payment scheme. Algorithm 2 implements our payment scheme.

If i 's job is scheduled, she pays

$$r_i(\hat{\theta}) = \arg \min_v (t_i(S((v, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i})) \neq \perp)$$

Algorithm 2

```

1: procedure NMPJS-PAYMENT( $\hat{\theta}$ )
2:   Sort  $\hat{\theta}$  in non-increasing order of  $\hat{v}_i$ 
3:   for  $i \leftarrow 1, n$  do
4:     if  $t_i(S) \neq \perp$  then
5:       for  $j \leftarrow i + 1, n$  do
6:          $\bar{\theta} \leftarrow ((\hat{v}_j, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i})$ 
7:         Reorder  $\bar{\theta}$  to satisfy  $\hat{v}_i > \hat{v}_{i+1}$ 
8:         if  $t_i(\text{NMPJS} - \text{Schedule}(\bar{\theta})) \neq \perp$  then
9:            $r_i \leftarrow \hat{v}_j$ 
10:        end if
11:       end for
12:        $\bar{\theta} \leftarrow ((0, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i})$ 
13:       Reorder  $\bar{\theta}$  to satisfy  $\hat{v}_i > \hat{v}_{i+1}$ 
14:       if  $t_i(\text{NMPJS} - \text{Schedule}(\bar{\theta})) \neq \perp$  then
15:          $r_i \leftarrow 0$ 
16:       end if
17:     else
18:        $r_i \leftarrow 0$ 
19:     end if
20:      $r \leftarrow r \cup \{r_i\}$ 
21:   end for
22:   return  $r$ 
23: end procedure

```

In words, user i pays the smallest value for which Algorithm 1 schedules her job i . The payment is conceptually similar to the online mechanism's *critical value* [16]. If user i 's job is discarded, she pays $r_i = 0$. The polynomial-time Algorithm 2 implements the payment scheme. A dummy job with value 0 is appended to the list of jobs. For a job i , the algorithm iteratively sets \hat{v}_i to the decreasing values of the jobs that follow job i . It then attempts to schedule the jobs. If job i can be scheduled, the current value is saved. Otherwise, the algorithm is terminated. Payment r_i is then equal to the value of the last successful scheduling attempt.

5. NMPJS Mechanism's Properties

In this section we prove that NMPJS satisfies the IR and IC mechanism design goal objectives. Then we provide the approximation ratio of NMPJS. We first show that NMPJS satisfies IR.

Theorem 1. *NMPJS mechanism satisfies the IR condition.*

Proof. Let job i be of type $\theta_i = (v_i, d_i, w_i, h_i)$, which is its true type. If job i is scheduled, then user i pays r_i , the smallest value for which the job is scheduled. Since job i is scheduled, then $r_i \leq v_i$, which implies $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq 0$. If the job is not scheduled, $r_i = 0$ and $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) = 0$. Thus, NMPJS satisfies the IR condition. \square

The following lemmas are formulated to prove that NMPJS satisfies the IC condition. As a summary of the follow-

ing, we first show that deadline restrictions arise naturally. Then, we show that if a job is scheduled when one or all of deadline, number of processors, and execution time parameters are misreported, then the job would also be scheduled if these parameters are truthfully declared. We show that payment increases if these parameters are misreported. We show that if a user's job is scheduled when either the value is declared truthfully or misreported, then the user pays the same amount for both declarations. Combining these results, we prove that NMPJS satisfies the IC condition.

Lemma 1. *User i restricts her deadline misreports to $\hat{d}_i \leq d_i$.*

Proof. Let job i with declared type $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ be scheduled, where $\hat{d}_i > d_i$. The deadline is not considered when the BL position is computed and thus, it does not affect t_i . Let t_i be such that $t_i + \hat{h}_i \leq \hat{d}_i$, and $t_i + h_i > d_i$, which means that job i 's completion time is within the reported deadline but the job is missing the true deadline. The user pays $r_i \geq v_i = 0$. This implies $u_i(g(\hat{\theta}_i, \theta_i)) \leq 0$, that means that user i obtains negative utility. Thus, user i does not have incentives to report an earlier deadline. If $\hat{d}_i \leq d_i$ and i is scheduled, then $v_i \geq r_i \geq 0$ and $u(g(\hat{\theta}_i, \theta_i)) \geq 0$. Thus, the user i restricts her misreports to $\hat{d}_i \leq d_i$. \square

Lemma 2. *If job i with declared type $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ is scheduled, then the same job i with declared type $\hat{\theta}'_i = (\hat{v}_i, d_i, w_i, h_i)$ is scheduled, where $\hat{d}_i \leq d_i$, $\hat{w}_i \geq w_i$, and $\hat{h}_i \geq h_i$.*

Proof. Algorithm 1 computes the start position independent of deadline. The tighter deadline harms the user as her job is more difficult to schedule. BL positions a job into the first bottom-left hole that fits the size of the job. Larger jobs (*i.e.*, jobs with greater w_i and h_i) are more difficult to schedule as larger holes become scarce. Thus, job i with type $\hat{\theta}'_i$ is scheduled if it is scheduled with type $\hat{\theta}_i$. \square

Lemma 3. *User i pays more if she declares her job type $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ than if she declares $\hat{\theta}'_i = (\hat{v}_i, d_i, w_i, h_i)$, where $\hat{d}_i \leq d_i$, $\hat{w}_i \geq w_i$, and $\hat{h}_i \geq h_i$.*

Proof. The job of type $\hat{\theta}_i$ is more difficult to schedule than when it has type $\hat{\theta}'_i$. BL places jobs into the first position that they fit in. There are fewer positions where the algorithm can place large jobs compared to the case of placing small jobs. Thus, the smallest value for which job i of type $\hat{\theta}_i$ can be scheduled is larger than in the case it has type $\hat{\theta}'_i$. This implies that the payment is higher and thus $u_i(g(\hat{\theta}_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\hat{\theta}'_i, \hat{\theta}_{-i}), \theta_i)$. \square

Lemma 4. *If job i is scheduled when its declared type is $\hat{\theta}_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ and it is also scheduled when its declared type is $\hat{\theta}'_i = (v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$, where $\hat{v}_i \neq v_i$, then user i pays the same amount.*

Proof. Suppose $v_i < \hat{v}_i$. Let r_i be the payment when job i is declared with value v_i . Since r_i is the smallest value for which job i is scheduled and $r_i \leq v_i < \hat{v}_i$, then r_i is also the smallest value for which job i with declared type $\hat{\theta}'_i$ is scheduled. Thus, declaring either v_i or \hat{v}_i results in the same payment r_i . The proof for $v_i > \hat{v}_i$ is similar. \square

Theorem 2. *NMPJS mechanism satisfies the IC condition.*

Proof. Let job i be of type $\theta_i = (v_i, d_i, w_i, h_i)$, which is its true type. Misreporting d_i , w_i , and h_i does not improve the schedulability of job i (Lemma 2), and in fact may increase payments (Lemma 3). Thus, $u_i(g((\hat{v}_i, d_i, w_i, h_i), \hat{\theta}_{-i}), \theta_i) \geq u_i(g((\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i}), \theta_i)$ for $\hat{d}_i \leq d_i$, $\hat{w}_i \geq w_i$, and $\hat{h}_i \geq h_i$. From Lemma 4, user i pays r_i if her job is scheduled when either $\hat{\theta}'_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ or $\hat{\theta}_i = (v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$. Thus, $u_i(g((v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i}), \theta_i) = u_i(g((\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i}), \theta_i)$. Furthermore, $u_i(g((v_i, d_i, w_i, h_i), \hat{\theta}_{-i}), \theta_i) \geq u_i(g((v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i), \hat{\theta}_{-i}), \theta_i)$.

If job i is scheduled when $\hat{\theta}'_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ and not scheduled when $\hat{\theta}_i = (v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$, then $\hat{v}_i \geq r_i > v_i$ and $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\hat{\theta}_i, \hat{\theta}_{-i}), \theta_i) = 0 \geq u_i(g(\hat{\theta}'_i, \hat{\theta}_{-i}), \theta_i)$.

If job i is scheduled when $\hat{\theta}_i = (v_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$ and not scheduled when $\hat{\theta}'_i = (\hat{v}_i, \hat{d}_i, \hat{w}_i, \hat{h}_i)$, then $v_i \geq r_i > \hat{v}_i$ and thus, $u_i(g(\theta_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\hat{\theta}_i, \hat{\theta}_{-i}), \theta_i) \geq u_i(g(\hat{\theta}'_i, \hat{\theta}_{-i}), \theta_i) = 0$. Thus, NMPJS satisfies the IC condition. \square

In the following we show that NMPJS is a n -approximation mechanism. A mechanism is a ρ -approximation mechanism if the solution it computes is guaranteed to be within a multiplicative constant ρ from the optimal solution.

Theorem 3. *NMPJS is a n -approximation mechanism for the objective $\max \sum_i v_i \cdot (t_i + h_i \leq d_i)$.*

Proof. As always, assume that the declarations are ordered by non-increasing value. In the pathological case, NMPJS schedules only job 1 while the optimal algorithm schedules the other $n - 1$ jobs. Let OPT be the sum of values of the jobs scheduled by the optimal algorithm. We have

$$OPT = \sum_{i=2}^n v_i \leq \sum_{i=1}^n v_i \leq \sum_{i=1}^n v_1 \leq n v_1.$$

It follows that the schedule computed by NMPJS is $1/n \cdot OPT$. Therefore, NMPJS is an n -approximation mechanism. \square

Note that NMPJS does not solve the traditional scheduling problem that has the objectives of minimizing makespan or response time. The objective considered in this paper is the sum of the job values. The values considered in this paper do not have any relationship to the dimensions of the jobs. Therefore, we will not focus here on a direct comparison between NMPJS and the existing scheduling algorithms in the literature.

6. Experimental Results

In this section we study by simulation the properties of the NMPJS mechanism. A simulator written in the Ruby programming language was developed specifically for this purpose. We study a system comprising $m = 64$ processors and $n = 16$ jobs. The true types of the users jobs are as follows.

$$\theta_i = (v_i, d_i, w_i, h_i), \quad i = 1, 2, \dots, 16$$

$$v_i = 5 * i$$

$$d_i = \begin{cases} 100 & \text{if } 1 \leq i \leq 3 \\ 150 & \text{if } i = 4 \\ 200 & \text{if } 5 \leq i \leq 8 \\ 300 & \text{if } 9 \leq i \leq 12 \\ 400 & \text{if } 13 \leq i \leq 16 \end{cases}$$

$$w_i = \begin{cases} 32 & \text{if } i \text{ is odd} \\ 24 & \text{if } i \text{ is even} \end{cases}$$

$$h_i = 50$$

We examine the system from user 4's perspective. We consider seven cases of potential misreports by user 4:

1. User 4 declares her type truthfully.
2. User 4 declares her type such that $v_4 = r_4^1$, where r_4^1 is the payment computed in case 1.
3. User 4 declares her type such that $\hat{v}_4 > v_4$.
4. User 4 declares her type such that $\hat{v}_4 < r_4^1$, where r_4^1 is the payment computed in case 1.
5. User 4 declares her type such that $\hat{w}_4 > w_4$.
6. User 4 declares her type such that $\hat{w}_4 < w_4$.
7. User 4 declares her type such that $\hat{d}_4 < d_4$.

The parameters for the above cases are given in Table 1.

The types were selected so that when the users truthfully report their types, all jobs can be scheduled. Due to space limitation not all possible cases of misreports are presented in this paper.

We first consider the consequences of user 4's behavior on her utility and payment. Figure 1 gives u_4 and r_4 for the

Table 1. Job 4's parameters.

Case	1	2	3	4	5	6	7
\hat{v}_4	65	50	83	47	65	65	65
\hat{d}_4	150	150	150	150	150	150	100
\hat{w}_4	24	24	24	24	48	12	24
\hat{h}_4	50	50	50	50	50	50	50

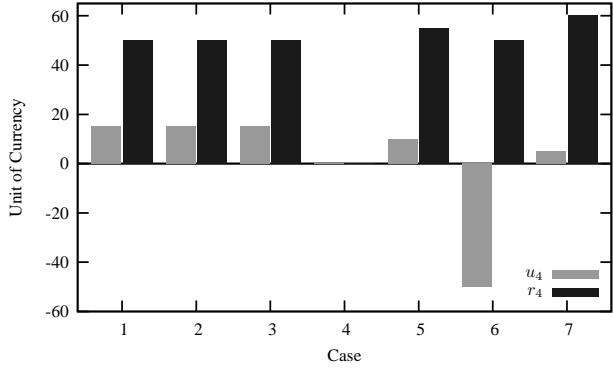


Figure 1. User 4's utility and payment.

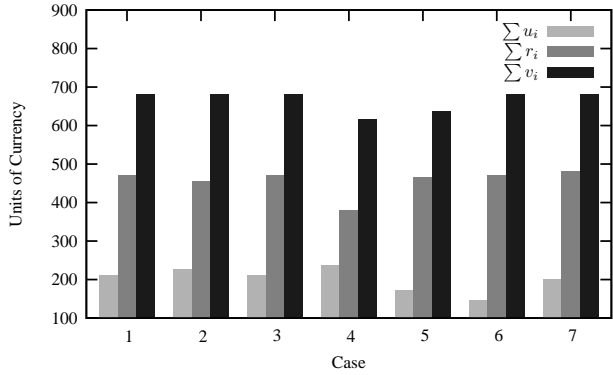


Figure 2. Sum of utility, payments, and values for the entire system.

seven cases. Let r_4^1 be the payment when 4 is truthful. In case 2, since $\hat{v}_4 = r_4^1$, the job is scheduled and she pays $r_4 = r_4^1$. Her utility is unchanged as she values the job the same in all cases. In case 3, she declares the job more valuable, but as expected, she pays the same and her utility is unchanged. In case 4, the job is not scheduled as $\hat{v}_4 \leq r_4^1$ which results in zero utility for job 4. Since the user requests more processors in case 5, her job is more difficult to schedule and it can be positioned in fewer locations. This results in a payment $r_4 > r_4^1$ and in a corresponding reduction in her utility. Even though the job is scheduled in case 6, the dearth of allotted processors result in job failure. Thus, $v_4 = 0$ and $v_4 - r_4 \leq 0$. In the final case, the decreased deadline reduces the available positions where it can be scheduled and thus, the payment increases leading to decreased utility.

The total utility, $\sum_i u_i$, the total payments, $\sum r_i$, and the total value $\sum v_i$ for the jobs that are scheduled in the simulation are given in Figure 2. For the first three cases, the sums of the values are equal. The sum of the utilities is greater in case 2 as the smaller \hat{v}_4 results in smaller payments and a resulting increase in utility for users 1, 2, and 3. The value reduction does not alter any other user's payment.

In case 3, the positions of jobs 2 and 4 are interchanged, but this change does not affect their payment. In case 4, the sum of the values decreases as 4 is not scheduled. The resulting additional available processors reduce contention among jobs, which lowers payments. Since job 4 requests more processors in case 5, job 8 is no longer scheduled resulting in a decrease in the sum of values. Without job 8, there is greater resource availability, which lowers payments with respect to case 1. In case 6, the sum of utilities is small due to user 4's decrease in utility. The sum of utilities is lower and the sum of payments is higher due to the limited schedulability of 4. The configuration results in user 4 paying more, which reduces her utility. The other users are not affected by this misreport. The experiments show how the misreports of jobs' parameters can lead to decreased utility for the users and decrease in the total value for the system.

7. Conclusion

In this paper we designed the incentive-compatible NMPJS mechanism for scheduling non-malleable parallel jobs on a parallel system. Since the mechanism is incentive-compatible, each user will truthfully report their type to the mechanism as this action results in utility maximization, irrespective of the others' actions. The mechanism also satisfies individual rationality: truthful agents will never incur a loss. To the best of our knowledge, NMPJS is the first incentive-compatible mechanism for scheduling non-malleable parallel jobs. It is also an example of incentive-compatible mechanism for multi-parameter, multi-domain agent types.

This work studied only non-malleable parallel jobs. For future work we are planning to investigate the design of mechanisms for moldable and malleable job scheduling problems.

References

- [1] N. Andelman and Y. Mansour. A sufficient condition for truthfulness with single parameter agents. In *Proc. of the 7th ACM Conf. on Electronic Commerce (EC '06)*, pages 8–17, 2006.
- [2] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science (FOCS '01)*, pages 482–491, 2001.
- [3] V. Auletta, P. Penna, R. De Prisco, and P. Persiano. The power of verification for one-parameter agents. In *Proc. of the 31st Int. Coll. on Automata, Languages, and Programming (ICALP '04)*, volume 3142 of *Lecture Notes In Computer Science*, pages 171–182, 2004.
- [4] J. Błażewicz, M. Machowiak, G. Mounié, and J. Węglarz. Approximation algorithms for scheduling independent malleable tasks. In *Proc. of 7th Int. Euro-Par Conference on Parallel Processing (Euro-Par '01)*, volume 2150 of *Lecture Notes in Computer Science*, pages 191–197, 2001.
- [5] J. Błażewicz, M. Machowiak, and J. Węglarz. Scheduling malleable tasks on parallel processors to minimize the makespan. *Ann. Oper. Res.*, 129:65–80, 2004.
- [6] 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)*, pages 372–381, 2007.
- [7] B. Chazelle. The bottom-left bin-packing heuristic: an efficient implementation. *IEEE Trans. Comput.*, 32(8):697–707, 1983.
- [8] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer-Verlag, 1996.
- [9] J. T. Havill and W. Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European J. Oper. Res.*, 187:1126–1142, 2008.
- [10] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, 2006.
- [11] O.-H. Kwon and K.-Y. Chwa. Scheduling parallel tasks with individual deadlines. *Theoret. Comput. Sci.*, 215(1):209–223, 1999.
- [12] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European J. Oper. Res.*, 141(2):241–252, 2002.
- [13] W. T. Ludwig. *Algorithms for scheduling malleable and nonmalleable parallel tasks*. PhD thesis, University of Wisconsin at Madison, Madison, WI, USA, 1995.
- [14] G. Mounié, C. Rapine, and D. Trystram. Efficient approximation algorithms for scheduling malleable tasks. In *Proc. of the 11th ACM Symp. on Parallel Algorithms and Architectures (SPAA '99)*, pages 23–32, 1999.
- [15] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games Econom. Behav.*, 35:166–196, 2001.
- [16] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University, 2007.
- [17] D. C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University, 2007.
- [18] R. Porter. Mechanism design for online real-time scheduling. In *Proc. 5th ACM Conf. on Electronic Commerce (EC '04)*, pages 61–70, 2004.
- [19] S. K. Sahni. Algorithms for scheduling independent tasks. *J. ACM*, 23(1):116–127, 1976.
- [20] U. Schwiegelshohn, W. Ludwig, J. L. Wolf, J. Turek, and P. S. Yu. Smart SMART bounds for weighted response time scheduling. *SIAM J. Comput.*, 28(1):237–253, 1999.