A Truthful Mechanism for Interval Scheduling

Jugal Garg and Peter McGlaughlin(✉)

University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
{jugal,mcglghl2}@illinois.edu

Abstract. Motivated by cloud computing, we study a market-based approach for job scheduling on multiple machines where users have hard deadlines and prefer earlier completion times. In our model, completing a job provides a benefit equal to its present value, i.e., the value discounted to the time when the job finishes. Users submit job requirements to the cloud provider who non-preemptively schedules jobs to maximize the social welfare, i.e., the sum of present values of completed jobs. Using a simple and fast greedy algorithm, we obtain a $1+s/(s-1)$ approximation to the optimal schedule, where $s > 1$ is the minimum ratio of a job's deadline to processing time. Building on our approximation algorithm, we construct a pricing rule to incentivize users to truthfully report all job requirements.

1 Introduction

Cloud computing's explosive growth over the past decade is attributable to its flexible computing resources and the economy of scale provided by large data centers. This framework allows users to rent computing resources on demand, avoiding the need for costly infrastructure investment. Typically, pricing is pay-as-you-go where users pay per unit time. While simple, this pricing scheme does not reflect current market conditions, i.e., user demand versus the cloud provider's capacity, nor does it account for important job requirements such as deadlines.

We investigate an alternative market based approach for the fair allocation of reusable resources by introducing a new scheduling problem, Present Value Scheduling (PVS). Abstractly, the problem is to non-preemptively schedule jobs with hard deadlines on m identical machines. Each job $\mathcal{J}_i = (v_i, t_i, d_i)$ is defined by a processing time t_i, a deadline d_i, and a value v_i if completed immediately. Users prefer earlier completion times, leading job values to decay over time as determined by the discount factor $0 < \beta < 1$ shared by all jobs. Then, completing job $\mathcal{J}_i$ at time $\tau \in [t_i, d_i]$ provides a benefit of $v_i\beta^\tau$. Note that this is the job's present value, the standard economic model for the time value of money.

Users submit job requests to the cloud provider who determines an allocation of resources based on the jobs' requirements with the objective of maximizing

J. Garg—Supported by NSF CRII Award 1755619.

X. Deng (Ed.): SAGT 2018, LNCS 11059, pp. 100–112, 2018.
https://doi.org/10.1007/978-3-319-99660-8_10

social welfare, i.e. the sum of present values of completed jobs. The inherently difficult scheduling problem is further complicated as users may misreport any job parameter (v_i, t_i, d_i) in an effort to increase their utility, defined as present value minus payment. We aim to construct scheduling and pricing rules to incentivize truthful reporting of all job information.

Our Contribution. This paper addresses a fundamental issue in mechanism design, non-preemptive job scheduling for social welfare maximization. Our model, PVS, includes the natural preference for early completion times by discounting the value of job to the time when it finishes. In other words, we consider maximizing the present value of completed jobs.

PVS is a special case of interval scheduling with arbitrary values, i.e., a job's value is an arbitrary function of time. Theoretical work in this area centers on constant factor approximations as the allocation (scheduling) problem is NP-hard. However, most existing algorithms do not ensure truthfulness. While there is a black-box method to construct truthful mechanisms from these approximations, the conversion comes at high computational cost. Fortunately, PVS provides sufficient structure to achieve a deterministic truthful mechanism through simple and efficient allocation and pricing rules.

First we provide a $1 + s/(s-1)$ approximation to PVS for any discount factor $0 < \beta < 1$, where $s = \min_i d_i/t_i > 1$. Our algorithm greedily schedules jobs in decreasing order of weights $w_i = v_i\beta^{t_i}/(1-\beta^{t_i})$ on the machine which gives the earliest completion time, as long as jobs complete by their deadlines. Our method achieves significantly faster run time compared to other applicable approximations which do not require truthfulness, an important consideration for large scale problems encountered in practice. The $1 + s/(s-1)$ bound is essentially tight for $\beta \approx 1$ and $s \gg 1$. However, it is conservative for $\beta \ll 1$ or $s \approx 1$. Second, we show a few key properties of our greedy approximation algorithm that allow an extension of Myerson's lemma [12] to PVS. As a result, we obtain a deterministic mechanism which is truthful with respect to all parameters (v_i, t_i, d_i).

1.1 Related Work

We provide a survey of related work in interval scheduling and mechanism design, taking care to highlight competing approaches to design truthful mechanisms for PVS.

Interval Scheduling. The allocation problem's theoretical foundations lie in interval scheduling. Essentially the discrete version of machine scheduling, each job is defined by explicitly listing all available scheduling times (intervals), with each interval potentially providing a different value. In other words, jobs' values are arbitrary functions of time. Nearly all versions of the problem are NP-hard, confining theoretical work to constant factor approximations. We focus on the best known approximations applicable to PVS and note that none of the following works require truthfulness.

Bar-Noy et al. [3] presents a 2 approximation based on LP rounding. Their algorithm starts by finding the optimal fractional allocation to the natural LP relaxation of the problem. The fractional solution is rounded into a set of polynomially many integer valued (feasible) solutions using a graph coloring argument, the largest of which yields at least 1/2 the optimal schedule's value. Bar-Noy et al. [2] uses the local ratio technique to derive a combinatorial 2 approximation in a generalization of interval scheduling where each job has a width. Independently, Berman and DasGupta [4] obtained a similar algorithm which achieves better runtime by specializing to the standard interval scheduling problem.

Mechanism Design. There is extensive literature on scheduling in the context of mechanism design, starting with the seminal work of Nisan and Ronen [13]. However, the majority of existing literature focuses on makespan minimization, e.g., see [9,11]. One particularly interesting and relevant approach for PVS is the black-box method of Lavi and Swamy [10] which converts approximation algorithms for set packing problems to truthful randomized mechanisms with the same approximation ratio. The procedure starts by applying a fractional VCG mechanism to the natural LP relaxation of the problem. Using the approximation algorithm as a separation oracle, the rescaled fractional allocation is decomposed into a convex combination of integer valued allocations. These integer solutions provide a truthful in expectation randomized mechanism with the same performance guarantees as the initial approximation algorithm. Although applicable to the previously mentioned interval scheduling approximations, and therefore PVS, the technique raises practical concerns for computational efficiency. First, one must solve an LP with a large number of variables and constraints multiple times for the fractional VCG mechanism. Then, decomposing the fractional allocation requires multiple calls to the approximation algorithm.

Recent work in the related field of batch computing in cloud systems offers an alternative to the black-box method. Batch computing generalizes interval scheduling by allowing jobs to run on multiple machines in parallel, up to a given threshold. Drawing on the LP rounding approximation of [3] and the black-box method of [10], Jain et al. [6] construct a truthful in expectation mechanism which approaches a 2 approximation as the number of machines goes to infinity. Requiring only one solution to the natural LP relaxation, their mechanism addresses some of the computational efficiency issues of the black-box approach, and it also allows job values to be arbitrary non-increasing functions of time.

In the preemptive version of batch computing, Jain et al. [7] develop a deterministic truthful mechanism with near optimal performance as the number of machines goes to infinity under a slackness condition on jobs, i.e., a lower bound on the ratio of a job's deadline to its processing time d_i/t_i. From the perspective of PVS, this paper is of interest as the allocation rule is akin to our own. Their approximation greedily schedules jobs in decreasing order of value density v_i/t_i, the natural analog of our weights w_i when the discount factor $\beta = 1$. Azar et al. [1] extend this mechanism to the online setting. We note both works assume job values are constant over time.

2 Definitions and Notation

PVS Model. In PVS a set of n jobs, $\mathcal{J} = \{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$, compete for processing time on m identical machines. Each job $\mathcal{J}_i$ is defined by the tuple (v_i, t_i, d_i) where: v_i is the job's valuation, t_i is the job's processing time, and d_i is the job's deadline. Note that under the identical machines assumption, a job's processing time is the same on all machines. We assume integer valued processing times and deadlines, though techniques generalize naturally to positive real values. The value of completing a job decays over time, determined by the discount factor $0 < \beta < 1$ shared by all jobs. Specifically, the value of completing job $\mathcal{J}_i$ at time $\tau \in [t_i, d_i]$ is $v_i\beta^\tau$.

A schedule for machine j is an ordered subset of jobs: $\mathcal{S}_j = \{\mathcal{J}_{k_1}, \ldots, \mathcal{J}_{k_a}\} \subseteq \mathcal{J}$, to be completed in the given order, i.e., $\mathcal{J}_{k_1}$ is completed first, then $\mathcal{J}_{k_2}$ and so on. The value of a schedule on machine j is the sum of values of completed jobs. That is, if $\mathcal{S}_j = \{\mathcal{J}_{k_1}, \ldots, \mathcal{J}_{k_a}\}$, then job $\mathcal{J}_{k_i}$ completes at time $\tau_{k_i} = \sum_{b=1}^{i} t_{k_b}$ and the value of the schedule $\mathcal{S}_j$ is: $V(\mathcal{S}_j) = \sum_{i=1}^{a} v_{k_i}\beta^{\tau_{k_i}}$. A full schedule consists of a schedule for each machine: $\mathcal{S} = \{S_1, \ldots, S_m\}$. We will simply say schedule when the distinction between full and machine specific schedules is clear. A schedule is feasible if each job is contained in no more than one machine schedule: $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$, $\forall i, j$. In words, each job is processed at most once across all machines. The social welfare maximizing schedule $\mathcal{S}^*$ is the feasible schedule with maximum value: $V(\mathcal{S}^*) \geq V(\mathcal{S})$ for all feasible $\mathcal{S}$. We will often refer to the social welfare maximizing schedule simply as the optimal schedule. We call the problem of finding the optimal schedule the allocation problem. We say that a schedule $\mathcal{S}$ is an α approximation (to optimal schedule) if $V(\mathcal{S}) \geq V(\mathcal{S}^*)/\alpha$.

Mechanisms. A mechanism $\mathcal{M}$ is an algorithm to produce an allocation (schedule) and a set of payments p_i. Each job in a schedule $\mathcal{J}_i \in \mathcal{S}$ is charged a payment p_i, earning utility $u_i(\mathcal{J}_i, \mathcal{J}_{-i}) = v_i\beta^{\tau_i} - p_i$, where $\tau_i > 0$ is time when $\mathcal{J}_i$ completes. Note $p_i = 0$ for all $\mathcal{J}_i \in \mathcal{J} \setminus \mathcal{S}$. Further, we assume agents receive no benefit for partially completing their job, or completing their job after their deadline. Jobs seek to maximize utility. The true parameters of a job $\mathcal{J}_i = (v_i, t_i, d_i)$ are private information and a job may misreport any or all of the values $\mathcal{J}_i' = (v_i', t_i', d_i')$ to gain higher utility. A mechanism is truthful if accurately reporting all job parameters is a dominant strategy: $u_i(\mathcal{J}_i, \mathcal{J}_{-i}) \geq u_i(\mathcal{J}_i', \mathcal{J}_{-i})$, $\forall \mathcal{J}_i'$. In words, truthfully reporting job parameters maximizes utility. We say a mechanism is social welfare maximizing if the scheduling algorithm returns the social welfare maximizing schedule $\mathcal{S}^*$, and an α approximation if it returns an α approximation.

3 Approximation for PVS

Due to space restrictions, we do not provide all proofs. Complete, detailed proofs can be found in the full version of this paper. Our first goal is solving the PVS allocation problem, i.e., maximizing the present value of completed jobs. In the

Algorithm 1. Greedy Scheduling Algorithm (GS)

Input : Job parameters (v_i, t_i, d_i) for each job $\mathcal{J}_i$
Output: Schedule $\mathcal{S}$ of jobs

Define: $w_i = \frac{v_i \beta^{t_i}}{1-\beta^{t_i}}$
Sort and relabel jobs in descending order of w_i
$\mathcal{S}_j \leftarrow \emptyset$, $j = 1, \ldots, m$ (schedule on machine j)
$\tau_j \leftarrow 0$, $j = 1, \ldots, m$ (total processing time of machine j)
$k = 1$ (machine offering fastest completion time)
for $i = 1$ ***to*** n **do**
  if $\tau_k + t_i \leq d_i$ **then**
    $\mathcal{S}_k \leftarrow (\mathcal{S}_k, \mathcal{J}_i)$; $\tau_k \leftarrow \tau_k + t_i$; $k \in \arg\min_j \tau_j$

full version of this paper, we show that this problem is strongly NP-hard. For this reason, we design a greedy algorithm which achieves a $1 + s/(s-1)$ approximation to the optimal schedule, where $s = \min_i d_i/t_i > 1$. Before presenting our algorithm, it is instructive to consider a simpler scheduling problem on single machine without deadlines. This special case admits an exact solution.

Proposition 1. *Define the weight of job $\mathcal{J}_i$ as:*

$$w_i = \frac{v_i \beta^{t_i}}{1 - \beta^{t_i}}. \tag{1}$$

If there is a single machine, and there are no deadlines, then placing jobs in decreasing order of w_i maximizes the social welfare.

This result follows from a simple interchange argument. Proposition 1 provides the basis for a natural greedy approximation: schedule jobs in decreasing order of weights $w_i = v_i \beta^{t_i}/(1 - \beta^{t_i})$ on the machine providing the earliest completion time, as long as jobs complete by their deadlines. A formal algorithm is shown in Greedy Scheduling Algorithm (GS). Despite its simplicity, GS provides performance guarantees for any discount factor $0 < \beta < 1$, under an assumption on the minimum slackness of any job $s = \min_i d_i/t_i > 1$.

Theorem 1. *Assume the minimum slackness of any job $s = \min_i d_i/t_i > 1$, then GS provides an approximation of $1+s/(s-1)$ to the PVS allocation problem.*

Remark 1. Intuitively, the assumption $s > 1$ means all agents are willing to wait at least a small amount of time proportional to the length of their job. For some applications, it is plausible that $s \approx 1$ making the performance guarantee vacuous. However, this bound is conservative for $s \approx 1$. In the full version of this paper, we show the actual approximation factor approaches $(2 - \beta)/(1 - \beta)$ as s goes to 1. Meaning GS gives a constant factor approximation for all $\beta < 1$. For practical application, we assume that cloud providers can use historical data to estimate s for their platform and assess our mechanism's guarantee from there.

3.1 Analysis of GS

The analysis of GS relies on dual fitting, an approach for proving approximation guarantees on greedy algorithms [16]. At a high level, we consider an LP relaxation of PVS and its dual. For any dual feasible variables λ, define $cost(\lambda)$ as the value of the dual problem evaluated at λ. Abusing notation, let GS be the value of the greedy schedule. Suppose we can show $\alpha GS \geq cost(\lambda)$ for some $\alpha \geq 1$, then weak duality implies GS is an α approximation to the optimal schedule. Under standard terminology, we say the algorithm GS is charged α to pay for the dual variables λ.

LP Relaxation of PVS and its Dual. We begin with the natural LP relaxation of PVS and its dual. Let $\mathcal{I}_i(t) = \{s : s \leq d_i, t \leq s \leq t + t_i - 1\}$ be the set of feasible finishing times for job $\mathcal{J}_i$ that overlap the time interval $[t-1, t)$, then an Integer Programming formulation of PVS is:

$$\max_x \sum_{i=1}^{n} \sum_{t \in [t_i, d_i]} v_i \beta^t x_{i,t} \tag{P}$$

$$\text{subject to: } \frac{1}{m} \sum_{i: d_i \geq t} \sum_{s \in \mathcal{I}_i(t)} x_{i,s} \leq 1 \quad t = 1, 2, \ldots, T \tag{C1}$$

$$\sum_{t \in [t_i, d_i]} x_{i,t} \leq 1 \quad i = 1, 2, \ldots, n \tag{C2}$$

$$x_{i,t} \in \{0, 1\} \quad \forall i, t,$$

where $T = \max_i d_i$, is the last deadline. Here, the variables $x_{i,t}$ indicate that job $\mathcal{J}_i$ finishes at time $t \in [t_i, d_i]$. The constraints $(C1)$ require that at most m jobs are scheduled at any point in time, and the constraints $(C2)$ require that each job is scheduled at most once. We obtain the natural LP relaxation with $x_{i,t} \geq 0$. Note that the constraints $x_{i,t} \leq 1$ are redundant due to $(C2)$.

The dual problem has a simple form. There is one dual variable γ_i for each job, and there is one dual variable λ_t for each time slot $t = 1, 2, \ldots, T$. Note that we define time slots in terms of their right endpoints, so that λ_t corresponds to the time interval $(t-1, t]$. The dual problem is:

$$\min_{\lambda, \gamma} \sum_{i=1}^{n} \gamma_i + \sum_{t=1}^{T} \lambda_t \tag{D}$$

$$\text{subject to: } \gamma_i + \frac{1}{m} \sum_{s=t-t_i+1}^{t} \lambda_s \geq v_i \beta^t \quad \forall i, \forall t \in [t_i, d_i] \tag{C3}$$

$$\gamma_i \geq 0, \forall i, \quad \lambda_t \geq 0, \forall t$$

Note that, for each job $\mathcal{J}_i$ there is exactly one constraint for each possible finishing time $t \in [t_i, d_i]$.

Approximation Guarantee. First we show how to construct dual feasible γ and λ to satisfy $(C3)$ for jobs used by GS. For each time slot $t = 1, \ldots, T$, there are at most m jobs processing in the greedy schedule, say $\mathcal{J}_{k_1}, \ldots, \mathcal{J}_{k_m}$. These jobs have weights $w_{k_1}, \ldots, w_{k_m}$. We set:

$$\lambda_t = \sum_{j=1}^{m} w_{k_j}(1-\beta)\beta^{t-1}. \tag{2}$$

Suppose GS finishes job $\mathcal{J}_i$ at time τ_i, then we set:

$$\gamma_i = v_i \beta^{\tau_i} \tag{3}$$

and $\gamma_i = 0$ otherwise.

Lemma 1. *The dual variables γ and λ ensure dual constraints $(C3)$ are satisfied for each job $\mathcal{J}_i$ used in GS.*

Proof (Sketch). We consider the three cases:

Case 1: $(C3)$ for $t \geq \tau_i$. From (3): $\gamma_i + \frac{1}{m} \sum_{l=t-t_i+1}^{t} \lambda_l \geq \gamma_i \geq v_i \beta^t$.

Case 2: $(C3)$ for $t \leq \tau_i - t_i$. GS schedules jobs in decreasing order of weight on the machine providing the earliest completion time. Since $\mathcal{J}_i$ starts processing at time slot $\tau_i - t_i \geq t$, GS must be processing m jobs with higher weight than w_i for all times $l \leq t$. By (2):

$$\gamma_i + \frac{1}{m} \sum_{l=t-t_i+1}^{t} \lambda_l \geq \frac{1}{m} \sum_{l=t-t_i+1}^{t} \lambda_l \geq \sum_{l=t-t_i+1}^{t} w_i(1-\beta)\beta^{l-1} = \beta^{t-t_i} w_i(1-\beta^{t_i}) = v_i \beta^t.$$

The last equality follows from: $w_i(1-\beta^{t_i}) = v_i \beta^{t_i}$, which is easily seen from (1).

Case 3: $(C3)$ for $\tau_i - t_i < t < \tau_i$. This case is handled with a technique similar to Case 2. □

We still need to satisfy $(C3)$ for jobs not used in GS. This is easy if GS always uses jobs with higher weight up to the unused job's deadline. That is, suppose job $\mathcal{J}_j$ is not used in GS, and for all $t \leq d_j$ GS uses a job $\mathcal{J}_i$ with $w_i \geq w_j$, then all of $\mathcal{J}_j$'s dual constraints are satisfied. The argument is essentially the same as Case 2 of Lemma 1. If this is not true, then there is some smallest time $u < d_j$ so that for all time slots $t \in [u+1, d_j]$ GS uses a job with lower weight on some machine. We call $\mathcal{J}_j$ a missed job. Covering dual constraints for missed jobs is the most challenging part of proof. For clarity, we present the remaining argument for a single machine. We show how to generalize the result to multiple machines in the full version of this paper.

Let $\mathcal{J}_j$ be a missed job. Note that $d_j - t_j$ is the last time we can start processing $\mathcal{J}_j$ and have it finish before its deadline. Let $\mathcal{J}_k$ be the job used by GS during this time slot, and τ_k be the time it completes. Since GS schedules jobs in decreasing order of weight, $w_k \geq w_j$ and $u > \tau_k \geq d_j - t_j + 1$. We say that $\mathcal{J}_j$

is missed at time τ_k. In our approach, we will go through each job $\mathcal{J}_k$ in GS and cover dual constraints for any missed jobs at τ_k by increasing λ_t's. Let $w(t)$ be the weight of job GS uses at time t. To cover all of $\mathcal{J}_j$'s dual constraints, we need to increase λ_t's by: $\hat{\lambda}_t = (w_j - w(t))(1-\beta)\beta^{t-1}$, for all times slots $t \in [u+1, d_j]$. This follows from (2), since $\lambda_t + \hat{\lambda}_t = w(t)(1-\beta)\beta^{t-1} + (w_j - w(t))(1-\beta)\beta^{t-1}$ so that:

$$\sum_{t=d_j-t_j+1}^{u} \lambda_t + \sum_{t=u+1}^{d_j} (\lambda_t + \hat{\lambda}_t) = \sum_{t=d_j-t_j+1}^{u} w_k(1-\beta)\beta^{t-1} + \sum_{t=u+1}^{d_j} w_j(1-\beta)\beta^{t-1} \geq v_j\beta^{d_j}.$$

We pay for the $\hat{\lambda}_t$'s using a portion of the value of GS up to time τ_k. In fact, we will show that an extra $1/(s-1)$ copies of the greedy schedule are enough to pay for the increased cost in dual variables for all missed jobs.

Let $Q(\tau_k)$ be the pool of resources available to cover the additional costs $\hat{\lambda}_t$'s needed to satisfy dual constraints for any missed jobs at τ_k. Formally, we set $Q(\tau_1) = v_1\beta^{\tau_1}/(s-1)$, the discounted value of the first job used by the greedy schedule scaled by $1/(s-1)$. We use $Q(\tau_1)$ to pay $C(\tau_1) = \sum_{t=u+1}^{d_j} \hat{\lambda}_t$, the cost of covering $(C3)$ for all missed jobs at τ_1. Suppose $\mathcal{J}_2$ is the second job used by GS, then the available resources to pay for missed jobs at time τ_2 is: $Q(\tau_2) = Q(\tau_1) - C(\tau_1) + v_2\beta^{\tau_2}/(s-1)$. In words, the available resources at time τ_2 are the resources remaining after covering all missed jobs up to time τ_1 plus the discounted value of the next job used in GS. Define $Q(\tau_k)$ similarly for all times τ_k when GS completes job $\mathcal{J}_k$. The following lemma provides a key result.

Lemma 2. *Assume $s = \min_i d_i/t_i > 1$, and let $\mathcal{J}_j$ be a missed job at time τ_k. If $Q(\tau_k)$ satisfies:*

$$\frac{Q(\tau_k)}{1-\beta^{\tau_k}} \geq \frac{w_j}{s-1}, \tag{4}$$

then $Q(\tau_k)$ is enough value to pay for the $\hat{\lambda}_t$'s required to cover $\mathcal{J}_j$'s dual constraints. In addition, if the next job used by the greedy schedule $\mathcal{J}_{k+1}$ completes after d_j, then:

$$\frac{Q(\tau_{k+1})}{1-\beta^{\tau_{k+1}}} \geq \frac{w_{k+1}}{s-1}. \tag{5}$$

Lemma 2 is essentially a technical result, a proof is provided in the full paper.

Proof (Theorem 1). Lemma 1 shows that setting λ and γ according to (2) and (3) respectively satisfies all dual constraints for jobs used in the greedy schedule. Clearly, this costs two copies of GS.

Lemma 2 implies that $1/(s-1)$ extra copies of GS are enough to cover dual constraints of all missed jobs. We start with $\mathcal{J}_1$ the first job of GS. Let $\mathcal{J}_j$ be the missed job at τ_1 with longest processing time. We may assume that $\mathcal{J}_j$ also has the highest weight of all missed jobs at time τ_1 since this means $\mathcal{J}_j$ is the missed job with the highest value. Therefore, satisfying $(C3)$ for $\mathcal{J}_j$ will satisfy $(C3)$ for all other missed jobs at τ_1. Since GS schedules jobs in decreasing order of weight $w_j \leq w_1$. By (1) and $Q(\tau_1) = v_1\beta^{t_1}/(s-1)$, condition (4) of Lemma 2

is satisfied. This means $Q(\tau_1)$ is enough to pay for the increased cost of dual variables needed to satisfy $(C3)$ for all jobs missed at time τ_1.

We only pay for a portion of required increase in dual variables now and defer the remaining payment until time τ_2, when the second job of GS completes. Specifically, at τ_1 we only pay for the portion of w_j which exceeds w_2, i.e. $\hat{\lambda}_t = (w_j - w_2)(1-\beta)\beta^{t-1}$ for $t = \tau_1 + 1, \ldots, d_j$. Effectively, this artificially extends the deadline of $\mathcal{J}_2$ to d_j, allowing application of the second condition (5) of Lemma 2 to yield $Q(\tau_2)/(1-\beta^{\tau_2}) \geq w_2/(s-1)$. However, artificially extending the deadline of $\mathcal{J}_2$ also increases the value of GS. To account for this, we add a fictitious missed job $\hat{\mathcal{J}}_j$ at time τ_2 with weight w_2, processing time t_j, and deadline d_j. It is easily seen that this matches the value added to GS. Further, all missed jobs at τ_2, including the newly added fictitious job, have weight less than w_2 and (4) is satisfied again at τ_2. As a result, $Q(\tau_2)$ is enough value to cover the cost of all missed jobs at τ_2. Repeating the above argument for each job used in the greedy schedule we see that our $1/(s-1)$ extra copies of GS are enough to pay for the dual constraints of all missed jobs. In total we require $2 + 1/(s-1) = 1 + s/(s-1)$ copies of GS to construct dual feasible λ and γ. □

The above proof extends easily from a single machine to m identically machines. First we use $w_m(t) = m^{-1}\sum_{i=1}^{m} w(t)$ in place of $w(t)$ in Lemma 2. Then, we proceed through the jobs of GS in increasing order of completion time, covering missed jobs as we go. The full version of the paper provides all details.

Remark 2. The $1 + s/(s-1)$ performance guarantee is essentially tight for $\beta \approx 1$ and $s \geq 2$. However, the bound is conservative for $\beta \ll 1$ or if $s \approx 1$. This is due to the somewhat loose analysis in Lemma 2. More careful treatment reveals $C(\beta, s) = \beta^{s-1}(1-\beta)/(1-\beta^{s-1})$ copies of GS are sufficient to cover all missed jobs, giving the performance guarantee of $1 + (1-\beta^s)/(1-\beta^{s-1})$, showing the algorithms dependence on β. We state the conservative bound since we assume most applications require β close to 1. Indeed, $\beta > 0.9$ is common in economics and finance literature.

4 Truthful Mechanism

Our $1 + s/(s-1)$ approximation to the allocation problem is only half of the mechanism design problem. As rational agents, job owners may lie about any or all of their job's parameters (v_i, t_i, d_i) to increase utility. We seek a pricing rule to ensure truthful reporting is a dominate strategy. The task is well understood in single parameter domains where the celebrated Myerson's lemma [12] provides the unique payment rule for any monotone allocation rule. Multi-parameter domains, as our own problem, present a challenge. VCG payments [14] create a truthful mechanisms when the allocation problem can be solved exactly, but many problems of interest require an approximation algorithm. It is known that generalizations of monotonicity are necessary and sufficient in these situations, see [5,8,15], but the conditions are difficult to check. Instead, we show a few

simple properties of the GS allocation rule allow us to construct a pricing rule which yields a truthful mechanism. We note that this is an extension of a result first obtained by Jain et al. in [7].

Properties of GS Allocation. We begin by introducing some notation used throughout this section. In PVS, each agent i reports a bid $b_i = (v_i, r_i)$ of their job's value v_i and requirements $r_i = (t_i, d_i)$. Given the set of bids $b = (b_1, \ldots, b_n)$, the cloud provider determines a completion time τ_i for i's job. We say i receives the allocation $\mathcal{A}_i(b) = \beta^{\tau_i}$ and note that i receives a value of $v_i\mathcal{A}_i(b) = v_i\beta^{\tau_i}$.

Assume $b_i = (v_i, r_i)$ are the agent's true valuation and job requirements, but they may misreport any of these values. If the agent submits a false bid $b'_i = (v'_i, r'_i)$, the actual allocation they receive may be different from $\mathcal{A}_i(b'_i, b_{-i})$ depending on their true requirements. For example, if an agent reports $t'_i > t_i$ and is scheduled for the time slot $[0, t'_i)$, then their actual allocation is the interval $[0, t_i)$ as their job only requires t_i units of processing time. Define $\mathcal{A}_i(b|r)$ as the actual allocation received by agent i assuming the requirements r. Continuing the earlier example with $t'_i > t_i$, then $\mathcal{A}_i(b'_i, b_{-i}|r_i) = [0, t_i)$. We assume $\mathcal{A}_i(b'_i, b_{-i}|r') = \mathcal{A}_i(b'_i, b_{-i})$. Finally, we note that the true benefit received from bidding b'_i is $v_i\mathcal{A}_i(b'_i, b_{-i}|r_i)$ and the utility received is:

$$u_i(b') = v_i\mathcal{A}_i(b'|r_i) - p_i(b'). \tag{6}$$

We show how a few simple properties of the allocation $\mathcal{A}_i$ allow us to construct a pricing rule $p_i(b_i, b_{-i})$ which yields a truthful mechanism. For notational convenience, we drop the b_{-i} argument and write $\mathcal{A}_i(v, r|r')$ instead of $\mathcal{A}_i(b_i, b_{-i}|r')$ or $p_i(v, r)$ instead of $p_i(b_i, b_{-i})$.

Definition 1: An allocation rule $\mathcal{A}$ is rational if for all agents i, all bids b_{-i}, all requirements r, r': $\mathcal{A}_i(v, r'|r) > 0 \implies \mathcal{A}_i(v, r'|r') > 0$.

Definition 2: An allocation rule $\mathcal{A}$ is value monotonic if for all agents i, all bids b_{-i}, all requirements r, r', and all valuations $v < v'$:

$$\mathcal{A}_i(v, r'|r) \leq \mathcal{A}_i(v', r'|r). \tag{7}$$

Definition 3: An allocation rule $\mathcal{A}$ is requirement monotonic if for all agents i, all bids b_{-i}, all requirements r, r' the following property holds: if there exists a v such that $\mathcal{A}_i(v, r'|r) > 0$ then:

$$\mathcal{A}_i(v, r'|r') \leq \mathcal{A}_i(v, r'|r) \quad \text{and} \quad \mathcal{A}_i(v', r'|r') \leq \mathcal{A}_i(v', r|r), \forall v'. \tag{8}$$

Intuitively these definitions have the following meanings: Rationality says if an allocation satisfies an alternative set of job requirements r, then it must also satisfy the requested requirements r'. Value monotonicity asks that the allocation is non-decreasing in the valuation v after fixing a set of job requirements. Finally, requirement monotonicity says if an allocation meets an alternate set of requirements r, then these requirements must be easier to satisfy and will

always receive an allocation at least as good as the original request r'. Before establishing that the GS algorithm satisfies these properties, we show how they contribute to a truthful mechanism.

Proposition 2. *Let $\mathcal{A}$ be a non-negative, rational, value monotonic and requirement monotonic allocation rule, then mechanism $\mathcal{M}(\mathcal{A}, p)$ using the pricing rule:*

$$p_i(v, r) = v\mathcal{A}_i(v, r|r) - \int_0^v \mathcal{A}_i(x, r|r)dx \tag{9}$$

is truthful and individually rational.

Note that the form of the payment rule is the same as that of Myerson's lemma, the important distinction being that jobs have additional requirements which must be satisfied, e.g., complete before their deadline. This result is an extension of [7] in which allocations are binary functions, i.e., jobs have constant value and are either completed or not. In PVS, the allocation is piece-wise constant. This means pricing rule (9) reduces to: the sum over (change in allocation) * (value where the allocation changes). The derivation is similar to the familiar single parameter case of Myerson's lemma. For more details, see the full version of the paper.

PVS Mechanism. Before showing GS satisfies the assumptions of Proposition 2, we impose a few natural constraints on what agents may misreport. It is important to note that these are not additional assumptions, rather certain types of misreporting are dominated by truthfulness. Therefore, a rational job owner would not misreport values in these ways. We assume $b_i = (v_i, t_i, d_i)$ are $\mathcal{J}_i$'s true valuation and requirements, and $b'_i = (v'_i, t'_i, d'_i)$ are alternative values. First, agents may only misreport longer processing times $t'_i \geq t_i$. This holds since agents receive no benefit from partially completed jobs. As such, a job owner reporting $t'_i < t_i$ gains no benefit from any allocation but is charged a non-negative price, implying $u_i(b'_i, b_{-i}) \leq 0$. Therefore, no rational job owner would report $t'_i < t$. Second, we assume agents may only under report their deadlines $d'_i \leq d_i$. This case is similar to the first, completing a job after the deadline provides no benefit but requires a non-negative payment creating the possibility for negative utility. We now show that GS satisfies the conditions of Proposition 2.

Proposition 3. *The GS allocation is rational, price monotonic, and requirement monotonic.*

These properties follow easily from the fact that GS greedily schedules jobs in decreasing order of weight: $w_i = v_i\beta^{t_i}/(1-\beta^{t_i})$, which is increasing in v_i and decreasing in t_i. Full details are provided in the full paper. Proposition 3 shows

GS satisfies the assumptions of Proposition 2, providing a truthful mechanism when using pricing rule (9). Combining this with Theorem 1 we obtain:

Corollary 1. *Assuming* $s = \min_i d_i/t_i > 1$*, the mechanism consisting of the GS allocation rule and the pricing rule (9) gives a truthful* $1+s/(1-s)$ *approximation to the social welfare maximizing schedule.*

5 Conclusion

In this paper, we propose a new scheduling problem, PVS, where jobs have hard deadlines and their values decay over time. Our simple and fast greedy scheduling algorithm, GS, provides reasonable performance guarantees under the relatively mild assumption that users are willing to wait for at least a short period of time, i.e. $s > 1$. Further, we exploit the greedy nature of GS to extend the celebrated Myerson's Lemma to a special multi-parameter domains where users report processing time and deadline in addition to their job value. From this, we obtain a mechanism for PVS which truthful with respect to all job parameters.

Our model does suffer from some over simplifications. Most notably, all users must have the same discount factor β. Allowing user specific discount factor β_i is more realistic. Further, β_i should be private information so that users report bids $b_i = (v_i, t_i, d_i, \beta_i)$. This presents interesting and challenging problems in both the design of an approximation algorithm and a truthful mechanism. We leave this to future work. Another avenue for future work is the design of a truthful revenue maximizing mechanism.

References

1. Azar, Y., Kalp-Shaltiel, I., Lucier, B., Menache, I., Naor, J.S., Yaniv, J.: Truthful online scheduling with commitments. In: Proceedings of the Sixteenth ACM Conference on Economics and Computation, pp. 715–732. ACM (2015)
2. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM (JACM) **48**(5), 1069–1090 (2001)
3. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comput. **31**(2), 331–352 (2001)
4. Berman, P., DasGupta, B.: Multi-phase algorithms for throughput maximization for real-time scheduling. J. Comb. Optim. **4**(3), 307–323 (2000)
5. Bikhchandani, S., Chatterji, S., Lavi, R., Mu'alem, A., Nisan, N., Sen, A.: Weak monotonicity characterizes deterministic dominant-strategy implementation. Econometrica **74**(4), 1109–1132 (2006)
6. Jain, N., Menache, I., Naor, J.S., Yaniv, J.: A truthful mechanism for value-based scheduling in cloud computing. Theory Comput. Syst. **54**(3), 388–406 (2014)
7. Jain, N., Menache, I., Naor, J.S., Yaniv, J.: Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. ACM Trans. Parallel Comput. **2**(1), 3 (2015)
8. Kovács, A., Vidali, A.: A characterization of n-player strongly monotone scheduling mechanisms. In: IJCAI, pp. 568–574 (2015)

9. Lavi, R., Swamy, C.: Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. In: Proceedings of the 8th ACM Conference on Electronic Commerce, pp. 252–261. ACM (2007)
10. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. J. ACM (JACM) **58**(6), 25 (2011)
11. Mu'alem, A., Schapira, M.: Setting lower bounds on truthfulness. In: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1143–1152. Society for Industrial and Applied Mathematics (2007)
12. Myerson, R.B.: Optimal auction design. Math. Oper. Res. **6**(1), 58–73 (1981)
13. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pp. 129–140. ACM (1999)
14. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory, vol. 1. Cambridge University Press, Cambridge (2007)
15. Rochet, J.C.: A necessary and sufficient condition for rationalizability in a quasi-linear context. J. Math. Econ. **16**(2), 191–200 (1987)
16. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2013)