Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters

(Regular Submission)

Navendu Jain
Microsoft Research, Redmond, WA
navendu@microsoft.com

Ishai Menache
Microsoft Research, Redmond, WA
ishai@microsoft.com

Joseph (Seffi) Naor
CS Department, Technion, Haifa, Israel
naor@cs.technion.ac.il

Jonathan Yaniv
CS Department, Technion, Haifa, Israel
jyaniv@cs.technion.ac.il

Abstract

We consider a market-based resource allocation model for batch jobs in cloud computing clusters. In our model, we incorporate the importance of the due date of a job by which it needs to be completed rather than the number of servers allocated to it at any given time. Each batch job is characterized by the work volume of total computing units (e.g., CPU hours) along with a bound on maximum degree of parallelism. Users specify, along with these job characteristics, their desired due date and a value for finishing the job by its deadline. Given this specification, the primary goal is to determine the *scheduling* of cloud computing instances under capacity constraints in order to maximize the social welfare (i.e., sum of values gained by allocated users). Our main result is a new $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation algorithm for this objective, where C denotes cloud capacity, k is the maximal bound on parallelized execution (in practical settings, $k \ll C$) and s is the slackness on the job completion time i.e., the minimal ratio between a specified deadline and the earliest finish time of a job. Our algorithm is based on utilizing dual fitting arguments over a strengthened linear program to the problem.

Based on the new approximation algorithm, we construct truthful allocation and pricing mechanisms, in which reporting the job true value and properties (deadline, work volume and the parallelism bound) is a dominant strategy for all users. To that end, we provide a general framework for transforming allocation algorithms into truthful mechanisms in domains of single-value and multi-properties. We then show that the basic mechanism can be extended under proper Bayesian assumptions to the objective of maximizing revenues, which is important for public clouds. We empirically evaluate the benefits of our approach through simulations on datacenter job traces, and show that the revenues obtained under our mechanism are comparable with an *ideal* fixed-price mechanism, which sets an on-demand price using oracle knowledge of users' valuations. Finally, we discuss how our model can be extended to accommodate uncertainties in job work volumes, which is a practical challenge in cloud settings.

Keywords: Resource Allocation, Economic Models, Truthful Mechanisms, Cloud Computing

1 Introduction

Background and motivation. Batch processing applications have become a significant fraction of computing workload across both public and private clouds. Examples include MapReduce/DryadLINQ jobs, web search index update, monte carlo simulations, eScience applications and data analytics. The primary challenge for many of these applications is to meet the service level agreements (SLA) on their job completion deadline. For instance, financial firms run large batch jobs to analyze daily stock trades and need the results to develop trading strategies before market opens the next day. Similarly, job completion time is critical for web search because even a small fraction of stale results can lead to a significant loss in revenue through a reduction in purchases, search queries, or advertisement click-through rates. The second challenge is concurrently *scheduling* multiple jobs with each job requiring a large number of CPU units, under the cloud capacity constraints.

Currently, cloud providers offer three pricing schemes to users: (i) *on-demand* instances, where a user pays a fixed price for a virtual machine (VM) per unit time (e.g., an hour) and can acquire or release VMs on demand, (ii) *spot-instances*, where users bid for spot instances and get allocation when the spot market price falls below their bid, and (iii) *reservation* pricing where users pay a flat fee for long term reservation (e.g., 1-3 years) of instances and a discounted price during actual use. Despite their simplicity, these approaches have several shortcomings for batch computing. First, they offer only *best-effort* execution without any guarantees on the job completion time. However, the financial firm in the above example requires SLA on the finish time rather than how VMs are allocated over time. Further, given capacity constraints, the cloud may not even be able to meet large resource requirements under unpredictable demand. Second, current resource allocation mechanisms do not differentiate jobs based on their importance/priority e.g., financial applications have (and are willing to pay for) strict job deadlines while scientific jobs are likely willing to trade a bounded delay for lower costs. As a result, cloud systems lose the opportunity to increase profits (e.g., by prioritizing jobs with strict SLA), improve utilization (e.g., by running low priority jobs at night), or both. Finally, existing schemes do not have in-built *incentives* to prevent fluctuations between high and low resource utilization. Perhaps the most desired goal of cloud operators is to keep all their resources constantly utilized.

In this paper we consider an alternative approach to resource allocation in cloud computing clusters based on an offline scheduling problem of multiple batch jobs called the *bounded flexible scheduling* problem. In this model, we explicitly incorporate the importance of the completion time of a batch job to its owner, rather than the number of instances allocated to the job at any given time. Each batch job is characterized by the work volume (or distribution thereof) of total computing units (e.g., CPU hours) along with a bound on the maximum degree of parallelism. Users report, along with these characteristics, the job deadline and a value for finishing the job by its deadline. The goal is to decide a scheduling of cloud resources under capacity constraints to optimize a *value-integrated* objective function, such as the social welfare (i.e., sum of values gained by users with resource allocations, especially relevant for private clouds). We also focus on the design of economic mechanisms, in which a public cloud charges users based on the above mentioned job parameters. While current pricing schemes do not charge based on deadlines, this work aims to advocate the merits of incorporating deadline-based pricing schemes. This scheduling model raises fundamental questions in mechanism design as users may try to game the system by misreporting their values or other job parameters to increase their utility. To address these challenges, our proposed solutions provide incentives for users to report truthfully.

The bounded flexible scheduling problem was first considered in our prior work [13], which proposed the first algorithm for the social welfare objective with a 2-approximation factor. That algorithm, designed for general user valuation functions, served as the base to design a truthful-in-expectation mechanism where reporting valuations truthfully maximized the expected utility for each user. However, it had three key *practical* shortcomings. First, from the mechanism design perspective, job work volume and parallelism bounds are assumed to be truthfully reported and hence it was only necessary to guarantee truthfulness with respect to values and deadlines. Second, to guarantee truthfulness, the proposed mechanism risks low utilization with at least half of the resources unutilized. Further, the solution cannot be extended to deal with uncertainties in work volume (we use the terms work volume and job demand interchangeably). Finally, the solution requires solving a linear problem which might be computationally expensive to run frequently for a large number of jobs and resources, as common in the cloud. See Appendix D for a complete survey of related work.

Our contributions. This paper makes the following contributions.

• *Modeling.* We propose a flexible resource allocation framework for scheduling batch jobs with deadlines. In particular, the model comprises an urgency parameter s which given a job work volume and parallelism bound determines the earliest deadline that the job owner can request. This parameter enables cloud the flexibility in scheduling jobs thereby yielding higher profits and improved performance over existing approaches.

• *Near-optimal and computationally-efficient scheduling.* We design a new $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation algorithm for the optimal social welfare, where k is the maximal parallelism bounds over all jobs, and C is the cloud capacity. Note that the approximation factor approaches one under the (plausible) assumption that $k \ll C$, and s is sufficiently large. The approximation algorithm is semi-greedy - it considers the jobs one by one, and to accommodate a new job it is allowed to make certain changes to the allocation of previously allocated jobs. To analyze the algorithm, we formulate the problem as a linear program with strengthened constraints which are somewhat reminiscent of knapsack constraints [9]. We then apply the technique of *dual fitting* to prove the approximation factor. The algorithm is specifically designed to maintain unique properties, which are realized under clever dual fitting arguments. Thus, each allocation step (of a job) goes hand in hand with the construction of a feasible dual solution, such that its cost is later bounded by sophisticated charging techniques.

• *Full incentive compatibility.* The proposed scheduling algorithm is incentive compatible with respect to *all* job parameters (value, deadline, work volume and parallelism bound). This property allows achieving the same approximation bound for both the social welfare mechanism design and the profit maximization problem (the latter, under standard Bayesian assumptions).

• *Empirical study.* We perform experiments with job traces taken from a large cloud provider. Our simulation study provides a surprising result: The mechanism we propose generates higher revenue than an ideal fixed-price mechanism, which has full knowledge of the private values and deadlines of users.

• *Dealing with demand uncertainties.* We propose a new model to handle cases where job volumes are stochastic, adapt the scheduling algorithm and provide performance bounds.

2 The Model

We consider a single cloud provider (or simply, the cloud), which allocates resources (CPUs) to jobs over time. Specifically, the time horizon is divided into T time slots $\mathcal{T} = \{1, 2, \ldots, T\}$. For example, in a cloud computing setting, each slot t represents an actual time interval of one hour. The cloud has a fixed capacity of C, given in CPU time units. This paper focuses on an offline setting, where all jobs that wish to be served until time T arrive to the system at time 0 and can be executed immediately.

There are n users (clients), each of them owning a single job that needs to be executed. We use the terms jobs, users and clients interchangeably. Every job is associated with a type profile $\tau_j = (v_j, d_j, D_j, k_j)$ representing the job parameters, fully described hereupon. Every user has a value $v_j \in \mathbb{R}^+$ representing the worth of a successful execution of its job, i.e., its job being fully executed before the *deadline* d_j. The size of each job j, also referred to as the *demand* of the job, is given by $D_j \in \mathbb{R}^+$ (in CPU time units). The cloud provider can *flexibly* allocate resources to jobs, that is, the amount of resources allocated to a job can change over time. For instance, a job may even be preempted and continued later on. However, each job has an upper bound k_j on the number of resources it may receive in a time slot. This bound stays fixed throughout the whole time horizon $\mathcal{T}$. Formally, an *allocation* of a job j is a function $y_j : [1, d_j] \to [0, k_j]$ representing the number of CPU units job j receives per time slot, which completes the job without violating the job parallelism bound k_j. We assume that the maximal parallelism bound $k \triangleq \max_j \{k_j\}$ is much smaller than the capacity C. A solution to the problem is a set $y = (y_1, \ldots, y_n)$ of allocations satisfying capacity constraints. The objective of an allocation algorithm is to maximize the *social welfare*, which is the sum of values of jobs that are completed before their deadline. We emphasize that partial execution of a job does not yield partial value.

Let $len_j = \lceil D_j/k_j \rceil$ denote the minimal length (or duration) of a complete allocation of resources to job j. We assume that the cloud will consider scheduling a job only if its deadline satisfies $d_j \geq s \cdot len_j$, where s is a *slackness* (or urgency) parameter advertised by the cloud. We refer to this condition as the *slackness condition*. Intuitively, the slackness condition gives the cloud the time margins to schedule jobs. In the extreme case where $s = 1$, each user requests that its job is allocated the maximal amount of resources until termination, leaving no scheduling flexibility to the cloud.

3 The GreedyRTL Algorithm

In this section we construct a new approximation algorithm for the bounded flexible scheduling problem called GreedyRTL, which obtains a $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation to the optimal social welfare. We start off by considering in Section 3.2 a simple greedy algorithm, as follows. Sort the jobs in non-increasing order of their marginal values (i.e., their value-demand ratio). Then, allocate them one-by-one (if possible), according to the sorted order, where a job is scheduled depending on the remaining resources. Henceforth, we assume that the jobs are ordered such that $\frac{v_1}{D_1} \geq \frac{v_2}{D_2} \geq \cdots \geq \frac{v_n}{D_n}$. To analyze the performance of this simple algorithm, we formulate the bounded flexible scheduling problem as a linear program (Section 3.1). The proofs of performance rely on a *dual fitting* argument. We construct a feasible solution to the dual linear program at cost proportional to the total value gained by the greedy algorithm. Since the cost of a feasible solution to the dual program is an upper bound to the optimal solution, we obtain the approximation factor.

To obtain the near-optimal approximation factor, we develop a single-job allocation rule (Section 3.3) called AllocateRTL(j). When applied, AllocateRTL(j) will allocate resources to job j, possibly reallocating previously scheduled jobs $i < j$. Our allocation rule will maintain a property we define called β-consistency. This property allows us to improve the feasible solution to the dual program, significantly reducing its cost.

3.1 Preliminaries

We begin by introducing the primal and dual linear programs. Let $y_j(t)$ denote the amount of CPU hours dedicated to the execution of job j in a time slot t, $t \leq d_j$. We consider a relaxed linear program of the bounded flexible scheduling problem suggested by [13]:

$$\text{(P)} \quad \max \quad \sum_{j=1}^{n} \frac{v_j}{D_j} \sum_{t \leq d_j} y_j(t)$$

$$\text{s.t.} \quad \sum_{t \leq d_j} y_j(t) \ \leq \ D_j \qquad \forall j \tag{1}$$

$$\sum_{j: t \leq d_j} y_j(t) \ \leq \ C \qquad \forall t \tag{2}$$

$$y_j(t) - \frac{k_j}{D_j} \cdot \sum_{t' \leq d_j} y_j(t') \ \leq \ 0 \qquad \forall j, t \leq d_j \tag{3}$$

$$y_j(t) \geq 0 \qquad \forall j, t \leq d_j \tag{4}$$

Note that in the relaxed linear program, a job can be *fractionally* allocated, under constraints (1)–(3). Constraints (1)–(2) are job demand and capacity constraints, whereas the set of constraints in (3) is a strengthened version of the natural parallelism bound constraints of the form $y_j(t) \leq k_j$. This set of strengthened constraints, formulated by [13], is related to *knapsack cover* constraints [9] and is used to decrease the integrality gap of the relaxed linear program.

In the corresponding dual program we have a cover constraint for each job j and time slot $t \leq d_j$:

$$\begin{aligned} \text{(D)} \quad \min \quad & \sum_{j=1}^{n} D_j \alpha_j + \sum_{t=1}^{T} C\,\beta(t) \\ \text{s.t.} \quad & \alpha_j + \beta(t) + \pi_j(t) - \frac{k_j}{D_j} \cdot \sum_{t' \leq d_j} \pi_j(t') \ \geq \ \frac{v_j}{D_j} \qquad \forall j, t \leq d_j \qquad (5) \\ & \alpha_j, \beta(t), \pi_j(t) \geq 0 \qquad \forall j, t \qquad (6) \end{aligned}$$

Before we describe our approximation algorithms, we give some definitions and notations that we will use later on. Given an allocation y_j, denote by $s\left(y_j\right) = \min\left\{t : y_j\left(t\right) > 0\right\}$ and $e\left(y_j\right) = \max\left\{t : y_j\left(t\right) > 0\right\}$ the *start time* and *completion time* of allocation y_j, which are the first an last time slots in which resources are allocated to user j, respectively. Given a solution consisting of allocations y_j, we define $W(t) = \sum_{j=1}^{n} y_j(t)$ to be the total *workload* at time t and $\bar{W}(t) = C - W(t)$ to be the amount of available resources at time t. A time slot is *saturated* if $\bar{W}(t) < k$ and *unsaturated* otherwise. Finally, given a time slot t, we define:

$$R(t) = \max\left\{t' \geq t : \forall t'' \in \left(t, t'\right], \bar{W}(t) < k\right\} \qquad (7)$$

Intuitively, if there are saturated time slots adjacent to t to the right, $R(t)$ is the rightmost timeslot out of the saturated block to the right of t. Otherwise, $R(t) = t$.

3.2 A Simple Greedy Algorithm

The first algorithm we construct is called the SimpleGreedy algorithm, which serves as the basis for the GreedyRTL algorithm. Recall that the jobs are sorted in non-increasing order of their marginal values v_j/D_j. The algorithm, fully described in Appendix B, works as follows. We initialize an empty solution $y \leftarrow 0$ and go over the jobs in their sorted order. For every job j in this order we check whether we can fully allocate D_j free resource units, meeting the deadline d_j, and without violating the parallelism bound k_j. Formally, a job j can be scheduled successfully if $\sum_{t \leq d_j} \min\left\{\bar{W}(t), k_j\right\} \geq D_j$, where $\bar{W}(t)$ is the amount of available resources in time slot t. If so, we schedule job j by allocating resources arbitrarily (without violating capacity constraints and the parallelism bound). Otherwise, job j is not scheduled. We note that the job allocation phase of SimpleGreedy may seem somewhat too permitting, however, we will be able to give a good bound on the total value gained by the algorithm. Later on, we present a more sophisticated allocation rule, replacing the arbitrary assignment of resources, that will further improve the bound.

Analysis. To bound the total value gained by SimpleGreedy, we construct a feasible solution $\left(\alpha, \beta, \pi\right)$ to the dual program. Recall the dual constraints (5). For every job j, we must cover every time slot $t \leq d_j$ by at least v_j/D_j. Initially, we set all dual variables to be 0. For allocated jobs j, we set $\alpha_j = v_j/D_j$. This covers all the dual constraints associated with j, since the variable α_j is common to all of them. Note that the cost added to the dual objective function is exactly $D_j\alpha_j = v_j$.

Dual constraints of unallocated jobs will be covered by the $\beta(t)$ variables. Note that the variable $\beta(t)$ appears in all of the dual constraints associated with t. By setting $\beta(t)$ we are able to cover (or at least partially cover) other dual constraints. We will maintain three useful properties regarding the $\beta(t)$ variables: (1) A variable $\beta(t)$ is always set to be a marginal value v_j/D_j of an unallocated job; (2) Once a variable

$\beta(t)$ has been set, it remains unchanged throughout the rest of the algorithm; and (3) The $\beta(t)$ variables are monotonically non-increasing in t. We prove property 3 in Lemma 3.1, yet for now assume that it is given. Now, consider the case where a job j is unallocated by SimpleGreedy. Notice that for every time slot t with $\beta(t) > 0$, by property 1 we have $\beta(t) \geq v_j/D_j$, since SimpleGreedy considers jobs in non-increasing order of marginal values. Now, if $\beta(d_j) > 0$, then by property 3 all the $\beta(t)$ variables for $t \leq d_j$ have been set to values larger than v_j/D_j. By the property 2, they remain so until the end of the algorithm. Otherwise, if $\beta(d_j) = 0$, we apply a method which we call **β-cover(j)** (lines 17–21) to cover the uncovered dual constraints of job j. Let $t_{cov} = \min\{t \mid \beta(t) = 0\}$ be the first time slot currently set to 0. By similar arguments, all of the dual constraints associated with time slots $t < t_{cov}$ are covered by variables $\beta(t) \geq v_j/D_j$. The β-cover(j) method sets all of the unset $\beta(t)$ variables up to time d_j to be v_j/D_j, in order to cover the remaining unsatisfied constraints of j. In fact, β-cover keeps setting $\beta(t) = v_j/D_j$ for every $d_j \leq t \leq R(d_j)$, that is, we keep setting $\beta(t)$ variables of saturated time slots $t \geq d_j$ until we reach an unsaturated time slot. We note that in the context of the SimpleGreedy algorithm, it is enough to cover the constraints up to time d_j. However, covering the $\beta(t)$ up to $R(d_j)$ instead of d_j will be useful in the next subsection. Thus to simplify arguments later on we introduce this step here. One can simplify the analysis of the SimpleGreedy algorithm by replacing $R(d_j)$ with d_j. It remains to prove Lemma 3.1, based on the β-cover step.

Lemma 3.1 *At the end of every call to β-cover(j), $\beta(t)$ is monotonically non-increasing in t.*

Corollary 3.2 *At the end of the SimpleGreedy algorithm, (α, β, π) is a feasible solution to the dual program.*

It now remains to bound the total cost of the dual solution constructed by SimpleGreedy. Let S denote the set of jobs fully allocated by SimpleGreedy. The cost of covering the dual constraints associated with allocated jobs is exactly $\sum_{j=1}^{n} D_j\alpha_j = \sum_{j\in S} v_j$. To bound the remaining cost of $\sum_t C\beta(t)$, we use a charging argument: We charge allocated jobs for the $\beta(t)$ variables, such that the total amount charged exceeds $\sum_t C\beta(t)$, and then bound the total amount charged. Specifically, let $charge_i(t)$ be the amount charged from job i at time t. We will charge i at time t an amount proportional to $y_i(t)$, the number of resources it received at time t. Every such pair (i,t) will be charged only once, according to the following rule: whenever β-cover(j) is called, we charge from every uncharged pair (i,t):

$$charge_i(t) \leftarrow \left[\frac{C}{C-k}\cdot\frac{s}{s-1}\right]\cdot\frac{v_j}{D_j}\cdot y_i(t) \tag{8}$$

By the order SimpleGreedy consider jobs, a charged job i must have been allocated beforehand. Thus, $i < j$ and therefore $charge_i(t) \leq \left(\frac{C}{C-k}\cdot\frac{s}{s-1}\right)\cdot v_i/D_i\cdot y_i(t)$. The total amount charged from all jobs satisfies:

$$\sum_{i\in S}\sum_{t\leq d_i} charge_i(t) \leq \left[\frac{C}{C-k}\cdot\frac{s}{s-1}\right]\sum_{i=1}^{n}\sum_{t\leq d_i}\frac{v_i}{D_i}\cdot y_i(t) = \left[\frac{C}{C-k}\cdot\frac{s}{s-1}\right]\cdot\sum_{i\in S} v_i \tag{9}$$

We note that it is possible to use similar charging arguments to prove this bound, however the form used here will be useful in the context of the GreedyRTL algorithm. It remains to show that total amount charged from jobs is an upper bound to $\sum_t C\beta(t)$. Define E_j to be the set of unsaturated time slots (i.e., $\bar{W}(t) < k$) up to time $R(d_j)$ during the call to β-cover(j).

Lemma 3.3 *After every call to **β-cover(j)**:*

$$\sum_{i=1}^{n}\sum_{t\leq d_i:\bar{W}(t)<k} charge_i(t) - \sum_{t=1}^{T} C\beta(t) \;\geq\; C\cdot\frac{v_j}{D_j}\cdot\frac{s}{s-1}\cdot\left[\frac{R(d_j)}{s} - |E_j|\right] \tag{10}$$

Theorem 3.4 *SimpleGreedy gives a $\left(1+\frac{C}{C-k}\cdot\frac{s}{s-1}\right)$-approximation to the optimal social welfare.*

Proof. Denote by S the set of jobs allocated by SimpleGreedy. Let j be the last job for which β-cover has been called. Since j was not allocated, we must have $|E_j| < len_j$, otherwise j could have been allocated. By the slackness assumption and since $d_j \leq R(d_j)$, we have $s \cdot |E_j| < s \cdot len_j \leq d_j \leq R(d_j)$. By Lemma 3.3 and by (9), the dual cost is at most $\left(1 + \frac{C}{C-k} \cdot \frac{s}{s-1}\right) \cdot \sum_{j \in S} v_j$. $\square$

3.3 Improving the Greedy Algorithm - The GreedyRTL Algorithm

The GreedyRTL algorithm presented in this subsection is similar in nature to the SimpleGreedy algorithm. Here, we will also sort the jobs according to their marginal values and decide whether to allocate a job using the same decision rule (if it is possible to fully allocate the job using unused resources). The main difference between the two algorithms is the allocation rule of a single job. SimpleGreedy allowed any arbitrary allocation of resources to jobs that were taken to the solution, whereas for GreedyRTL we construct a specific allocation rule called AllocateRTL(j). Unlike the former case, AllocateRTL may also choose to reallocate previously scheduled jobs, to be described later.

Before beginning, we give some intuition behind the suggested algorithm. Our goal will be to reduce the dual cost associated with allocated jobs j, which consists of $D_j\alpha_j$ and $charge_j(t)$ for every t. Consider some monotonically non-increasing vector β and ignore for now the π variables. To satisfy the dual constraints of an allocated job j, we must set α_j to be $v_j/D_j - \beta(d_j)$, since β is monotonically non-increasing. We would like the charged values $charge_j(t)$ to be as low as possible, preferably proportional to $D_j\beta(d_j)$. Ideally, we would want all of the jobs to be *aligned to the right*. Formally speaking, in an allocation aligned to the right, we allocate k_j resource units to j in every time slot, starting from the job deadline d_j moving towards earlier time slots. The last time slot in which we allocate resources to j will not necessarily receive k_j resource units, only the remaining amount of resources needed to complete the job[1]. Yet, even if allocations took this ideal form, this would still not necessarily mean that we could construct a dual solution of low cost. To do so, we need to incorporate the dual variables $\pi_j(t)$ into our dual solution.

The AllocateRTL rule will maintain a property called β-consistency over all allocated jobs, in spirit of the discussion above. To define the β-consistency property, we first need the following definition:

Definition 1 *The* breakpoint $bp(y_j)$ *of an allocation* y_j *of job* j *is defined as:*

$$bp(y_j) = \max\Big(\ \{s(y_j)\} \cup \{\ t \mid y_j(t) < k_j\ \}\ \Big) \tag{11}$$

The breakpoint $bp(y_j)$ is essentially the first time slot t, starting from the deadline moving towards earlier time slots, such that $y_j(t)$ does not coincide with the ideal aligned-to-right form of allocation. If such a time slot does not exist, we define $bp(y_j) = s(y_j)$.

Definition 2 *An allocation* y_j *is called* β-consistent *if for every time slot* t, $s(y_j) < t \leq bp(y_j)$, *either* t *is saturated or* $\beta(t) > 0$.

The AllocateRTL(j) rule allocates job j as follows. First, we initialize an empty allocation $y_j = 0$. We begin at the deadline $t = d_j$ and move towards earlier time slots (hence the name Right-To-Left). In every time slot t, the algorithm will attempt to allocate $\Delta = \min\{k_j, D_j - \sum_{t \leq d_j} y_j(t)\}$ resource units to job j. That is, give job j either the maximal amount of resources k_j it can get at time t, or the remaining unallocated portion, so that eventually y_j will be β-consistent. If $\bar{W}(t) \geq \Delta$, we allocate Δ resource units to j and continue to an earlier time slot. Otherwise, if $\bar{W}(t) < \Delta$ (specifically, t is saturated since $\Delta \leq k$), we attempt to free resources at time t by moving existing jobs to earlier time slots. AllocateRTL searches for the first unsaturated time slot to the left of t, denoted by t'. Notice that if $\beta(t') > 0$ then y_j will definitely be β-consistent, and therefore we can allocate the remaining portion of j arbitrarily in the interval $[1, t]$ (for

[1]When D_j and k_j are integers, this amount equals $D_j \ mod \ k_j$.

the sake of consistency, we will keep allocating j from right to left, giving j in each time slot the maximal amount of resources it can get). Otherwise, the key idea is that there must be a job j' with $y_{j'}(t) > y_{j'}(t')$, since t is saturated and t' is unsaturated. As long as this condition holds, we increase $y_{j'}(t)$ in expense of $y_j(t)$, until either (i) $\bar{W}(t) = \Delta$, in which case we set $y_j(t) = \Delta$ and continue, or (ii) $y_{j'}(t) = y_{j'}(t')$, and then we keep repeating this process. It is easy to see that this operation does not violate the parallelism bound of j', since we do not continue this process if equality is reached, nor change the completion time of j'. The fully detailed implementation of AllocateRTL(j) can be found in the appendix.

We begin our analysis by making two important observations. Using these observations, we prove that all of the allocations are β-consistent (Claim 3.7).

Claim 3.5 *For every t, the total workload $W(t)$ does not decrease after a call to AllocateRTL(j). Specifically, GreedyRTL does not turn a satisfied time slot into an unsatisfied one.*

Claim 3.6 *Let j be an uncharged job such that y_j is β-consistent. If j is charged by the algorithm, then from that point on the allocation y_j does not change.*

Claim 3.7 *In each step of the GreedyRTL algorithm, all of the allocations are β-consistent.*

It remains to show how to set the dual variables α_j, $\pi_j(t)$ for a job j allocated according to a β-consistent allocation y_j. First, notice the following: by setting a variable $\pi_j(t)$ to be some value ε, we incur a loss of $(k_j/D_j) \cdot \varepsilon$ in all of the dual constraints associated with job j. We will cover this loss by increasing α_j accordingly. For every allocated job, we apply a method called **$\boldsymbol{\alpha}$-correct(j)** (see algorithm in the appendix for complete details) to set the dual variables of job j. Initially, all of these variables are set to 0. We start by setting $\pi_j(t) = \beta(bp(y_j)) - \beta(t)$ for time slots $bp(y_j) < t \leq d_j$, and increase α_j accordingly to cover the loss incurred from setting the $\pi_j(t)$ variables. Now, all dual constraints are covered by at least $\beta(bp(y_j))$. To finish, we increase α_j by $v_j/D_j - \beta(bp(y_j))$. The following theorem proves the solution we constructed is feasible and bounds its total cost compared to the total value gained by GreedyRTL, completing the analysis.

Theorem 3.8 *GreedyRTL gives a $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation to the optimal social welfare.*

4 Incentive Compatible Mechanisms

In the previous section we constructed algorithms that allocate cloud resources to scheduled jobs, while ignoring the *incentives* issue, namely how to make sure that users report their true value v_j, as well as job properties (e.g., deadline, demand and parallelism bound). In this section we present a general framework for designing incentive compatible (*truthful*) allocation and pricing mechanisms, in single-value multi-property domains. The framework requires extending well-known results for single-parameter settings, where the private information held by each user consists of a single scalar. While extensions to multi-parameters auctions do exist (see, e.g., [8] and references therein), we provide here a general framework for truthfulness, which covers our model as special case.

4.1 Preliminaries

A *mechanism* $\mathcal{M} = (f, p)$ consists of an allocation rule f and a pricing rule p_j for every user j. Every user is associated with a private true type $\tau_j = \langle v_j, \mathcal{P}_j \rangle$, where $\mathcal{P}_j = \langle \rho_j^1, \rho_j^2, \ldots, \rho_j^m \rangle$ is a set of m properties of the job (specific to our context, $\mathcal{P}_j = \langle d_j, D_j, k_j \rangle$), and $v_j \in \mathbb{R}^+$ represents (as before) the value gained by user j if its job is successfully completed, i.e., fully allocated according to the requested properties. Users report a bid type b_j to the cloud, which may differ from their true type τ_j. The mechanism, given a reported bid vector $b = (b_1, b_2, \ldots b_n)$, allocates the jobs according to $f(b)$ and charges a non-negative payment

$p_j(b)$ from user j. We define $f_j(b \mid \mathcal{P}_j)$ to be a binary function that returns 1 if and only if the job of user j has fully completed with respect to the job properties $\mathcal{P}_j$[2]. We assume allocation functions f are *rational*, that is, if user j submitted a bid type of $b_j = \langle v'_j, \mathcal{P}'_j \rangle$ and $f_j(b) = 1$, then the allocation user j receives complies with $\mathcal{P}'_j$. Every user strives to maximize its *utility* u_j, defined to be the value it gains from the allocation f minus the payment it is charged:

$$u_j(b) = v_j f_j(b \mid \mathcal{P}_j) - p_j(b) . \tag{12}$$

One desired property of mechanisms is that reporting the true valuation function of users is a dominant strategy. Given some vector x, let x_{-j} denote the vector x without its j-th entry. Specifically, τ_{-j} denotes a vector of valuation functions of all players except for j. Let (τ_j, τ_{-j}) denote the concatenated vector of τ_j and τ_{-j}. A mechanism is said to be *incentive compatible* (IC) or *truthful* if for every user j and for every choice of τ_{-j}, truth-telling is a dominant strategy, i.e., maximizes their utility:

$$\forall j \quad \forall b_j, \tau_{-j} \quad u_j(\tau_j, \tau_{-j}) \geq u_j(b_j, \tau_{-j}) . \tag{13}$$

Apart from incentive compatibility, we would like to construct rational mechanisms, in which users do not lose by participating in the mechanism. A mechanism is *individually rational (IR)* if users do not receive negative utility when reporting their true valuation functions.

4.2 A General Sufficient Condition for Truthfulness

In order to construct truthful mechanisms based on the algorithms presented in Section 3, we extend known results for single-value domains [2, 12] to the case in which users also hold private job properties. For user j, we shorten notation by omitting the term b_{-j}. For example, we write $f_j\big(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j\big)$ instead of $f_j\big((v'_j, \mathcal{P}'_j), b_{-j} \mid \mathcal{P}_j\big)$. We first define two conditions on allocation algorithms f called value-monotonicity and property-monotonicity, and prove that they are sufficient for truthfulness for any binary rational f.

Definition 3 *An allocation function f is* value-monotonic *if for every user j, $\mathcal{P}'_j$ and b_{-j}, the function $f_j\big(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j\big)$ is monotonically non-decreasing in v. That is, for every v'_j, v''_j, $v'_j \leq v''_j$:*

$$f_j\big(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j\big) \;\leq\; f_j\big(v''_j, \mathcal{P}'_j \mid \mathcal{P}_j\big) . \tag{14}$$

Definition 4 *An allocation function f is* property-monotonic *if for every user j, b_{-j}, $\mathcal{P}_j$, $\mathcal{P}'_j$ the following condition holds: If there is some value v for which $f_j\big(v, \mathcal{P}'_j \mid \mathcal{P}_j\big) = 1$, then:*

$$\forall s \leq v \quad f_j\big(s, \mathcal{P}'_j \mid \mathcal{P}'_j\big) \;\leq\; f_j\left(s, \mathcal{P}_j \mid \mathcal{P}_j\right) . \tag{15}$$

Theorem 4.1 *If a binary allocation algorithm f for a single-value multi-property problem satisfies value-monotonicity and property-monotonicity and is rational, then the mechanism $\mathcal{M} = (f, p)$ that sets prices for every user j with true type $\tau_j = \langle v_j, \mathcal{P}_j \rangle$ according to:*

$$p_j(v'_j, \mathcal{P}'_j) = v'_j\, f_j\big(v'_j, \mathcal{P}'_j \mid \mathcal{P}'_j\big) - \int_0^{v'_j} f_j\big(s, \mathcal{P}'_j \mid \mathcal{P}'_j\big) ds, \tag{16}$$

is truthful and individually rational.

Notice that for binary monotone allocation functions f, the payment charged from each allocated user in (16) is actually the minimal value bid that would have guaranteed the job being scheduled.

[2] A job is fully completed with respect to $\mathcal{P}_j$ if according to the solution $f(b)$ job j receives D_j resource units before the deadline d_j, without violating the parallel execution bound k_j.

4.3 Profit Maximization in Bayesian Settings

The objective of profit maximizing is of course significant for public commercial clouds. When assuming no a-priori knowledge on clients' private valuation functions, it is well known that a truthful mechanism might charge very low payments from clients to ensure truthfulness, yielding low revenues. Thus, following a standard approach in game-theory, we consider a *Bayesian* setting, in which each user's value v_j is assumed to be drawn from a distribution with a probability density function g_j, which is common knowledge. We denote by G_j the respective cumulative distribution function (cdf). The properties of the job are assumed, as before, to be private information with no additional distribution information.

The goal of the mechanism in the current context is to maximize the optimal expected profit, with the expectation taken over the random draws of clients' values. For single-value domains, it is well known that the problem of maximizing profits can be reduced to the problem of maximizing social welfare over virtual values; this basic property is due to celebrated work by Myerson [16], and has been extended in different contexts (see [18]). To formally state the result, we first need the following definitions.

Definition 5 *The* revenue curve *of j is defined as $R_j(q) = q \cdot G_j^{-1}(1-q)$. The* ironed virtual valuation function $\bar{\phi}_j$ *of client j is defined as:* $\bar{\phi}_j(v) = \frac{d}{dq}\Big[ConcaveHull\left(R_j(\cdot)\right)\Big]$ *for* $q = 1 - G_j(v)$

That is, $\bar{\phi}_j(v)$ the derivative of the concave hull of $R(\cdot)$. Note that $\bar{\phi}_j(\cdot)$ is monotonically non-decreasing. Thus, if a single-value allocation rule f^{sv} is value-monotone, then so is $f^{sv}\left(\bar{\phi}(\cdot)\right)$.

Theorem 4.2 ([16, 18]) *For any single-value truthful mechanism $\mathcal{M}^{sv}$ that gives an α-approximation to the optimal social surplus, the mechanism $\mathcal{M}^{sv}(\bar{\phi}(\cdot))$ gives an α-approximation to the optimal expected profit.*

For our purposes, we prove that this reduction due to Myerson extends to domains of single-value and multi-properties. Formally,

Theorem 4.3 *Let f be a binary rational allocation algorithm for a single-value multi-property problem satisfying value-monotonicity and property-monotonicity, giving an α-approximation to the optimal social welfare. Let $f_{\bar{\phi}}$ be an allocation rule that replaces every type $\langle v_j, \mathcal{P}_j\rangle$ with $\langle \bar{\phi}_j(v_j), \mathcal{P}_j\rangle$ and calls f. Then, the mechanism $M_{\bar{\phi}}$ with allocation rule $f_{\bar{\phi}}$ that charges payments according to* (16), *with respect to $f_{\bar{\phi}}$, is truthful, and is an α-approximation to the optimal expected profit under Bayesian assumptions.*

4.4 Truthfulness of GreedyRTL

We return to the GreedyRTL algorithm and prove that it satisfies the sufficient conditions for truthfulness presented in Section 4.2: value-monotonicity, property-monotonicity and rationality. Rationality follows since if GreedyRTL schedules a job j, the allocation it receives always complies with the reported property set $\mathcal{P}_j$. We now prove the remaining two monotonicity conditions.

Claim 4.4 *GreedyRTL is value-monotone.*

Claim 4.5 *GreedyRTL is property-monotone.*

This is leads to the main result of this section.

Corollary 4.6 *GreedyRTL implements a truthful mechanism obtaining a $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation to the optimal social welfare. Moreover, if the value v_j of every user is drawn from a known distribution G_j, then GreedyRTL applied on virtual values $\bar{\phi}_j(v_j)$ implements a truthful mechanism obtaining a $\left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right)$-approximation to the optimal expected profit.*

5 Empirical Study

In this section we describe the highlights of the experiments we conducted to further evaluate the benefits of our scheduling framework. Due to space limitations, details of our empirical study can be found in App. A.

Resource utilization. We demonstrate that GreedyRTL reaches a utilization level which is very close to an upper bound on the optimal utilization, while the mechanism of [13] achieves around 35% of the upper bound. The results are consistent regardless the number of jobs that we consider. The utilization results not only provide an explanation to the social welfare improvements we obtain, but also stand on their own – given the significance of utilization in large cloud clusters.

Revenues. We show that the revenues obtained by our mechanism are comparable with the revenues of an *idealistic* fixed-price mechanism, which optimizes a fixed per-unit price based on oracle knowledge of the private values of users and other job parameters (such mechanism is *not* guaranteed to be truthful).

6 Extension: Coping with Demand Uncertainties

Up until now, we have assumed that the job work volume (or demand) D_j is a deterministic quantity. However, it turns out to be a restrictive assumption in many applications as the exact volume is either unknown, predicted using prior executions, or often overestimated. Further, the demand might be sensitive to stochastic fluctuations, especially in jobs where some tasks have dependencies on the completion of other tasks (see, e.g., [1] and references therein). From a theoretical perspective, these demand uncertainties introduce new challenges for mechanism design and impossibility results can indeed be shown ([10]). In this section we discuss how to address demand uncertainties while maintaining the benefits of our scheduling framework. We present below one plausible model, however a comprehensive study of alternatives to address this challenge is beyond the scope of this paper.

The model. To incorporate demand uncertainties, we extend the basic model we used for both jobs and the cloud. In particular, we consider a more general job model, where the demand of each job j is drawn from a distribution $\mathcal{D}_j$. The distribution has a finite support over $\big(0, D_j^{\mathbb{E}}(1+\delta)\big]$, where $D_j^{\mathbb{E}} \equiv \mathbb{E}\big[\mathcal{D}_j\big]$ is the *expected* volume, and δ is a positive parameter. For simplicity, this distribution is assumed to be common knowledge. The cases where $\mathcal{D}_j$ doesn't meet this assumption may be handled using a repeated auction framework, however, it is beyond the scope of this paper. From the cloud provider's side, we assume that the cloud is able to generate additional resources on-demand. We note that providing these additional resources might increase the operation cost of the cloud, hence in practice the cloud may charge the users an additional fee; however, we do not consider this aspect here for simplicity.

The solution. We propose the following modified GreedyRTL mechanism. Jobs are scheduled via the original offline allocation rule using their expected work volume as input. If the job does not complete after utilizing $D_j^{\mathbb{E}}$ resource units, it is allocated additional resources. To accommodate demand uncertainties, the offline allocation rule should guarantee that a job exceeding its initial estimate may still be completed before the deadline, taking into account the parallelism bounds. To that end, jobs are scheduled according to deadlines which are set earlier than their true deadlines, leaving an empty gap per job in which additional resources can be generated to fully complete the job. Specifically, the offline allocation rule will schedule jobs according to deadlines that are set earlier $d'_j = \lfloor d_j(1-\delta/s) \rfloor$. Note that in the remaining $\lceil d_j(\delta/s) \rceil$ time slots, the cloud can allocate at least $D_j\delta$ resource units, since $d_j \geq s \cdot len_j = s \cdot \lceil D_j/k_j \rceil$, as required. However, a job that exceeding the multiplicative bound of $(1+\delta)$ cannot be guaranteed completion by its deadline. We prove that the decrease in total value of the modified GreedyRTL is relatively small. To obtain this result, we use the dual solution $\big(\alpha, \beta, \pi\big)$ constructed in Section 3.3 to bound the gap between the value gained by the modified mechanism and the optimal social welfare (with respect to the original deadlines).

Theorem 6.1 *Let RTL' denote the social welfare obtained by the modified GreedyRTL algorithm, and let OPT denote the optimal social welfare (with original deadlines). Then,* $RTL' \geq \big(1 - \frac{\delta}{s} - \frac{1}{T}\big)\big(\frac{C-k}{C}\frac{s-1}{s}\big)OPT$.

References

[1] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the outliers in map-reduce clusters using mantri. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–16. USENIX Association, 2010.

[2] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *FOCS*, pages 482–491, 2001.

[3] Aaron Archer and Robert Kleinberg. Characterizing truthful mechanisms with convex type spaces. *SIGecom Exchanges*, 7(3), 2008.

[4] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48:1069–1090, 2001.

[5] Amotz Bar-Noy, Sudipto Guha, Joseph Naor, and Baruch Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal of Computing*, 31(2):331–352, 2001.

[6] Sushil Bikhchandani, Shurojit Chatterji, Ron Lavi, Ahuva Muálem, Noam Nisam, and Arunava Sen. Weak monotonicity characterizes deterministic dominant strategy implementations. *Econometrica*, 74:1109–1132, 2006.

[7] Peter Brucker. *Scheduling Algorithms*. Springer, 4th edition, 2004.

[8] Yang Cai, Constantinos Daskalakis, and S. Matthew Weinberg. On optimal multidimensional mechanism design. *ACM SIGecom Exchanges*, 10(2):29–33, 2011.

[9] Robert D. Carr, Lisa K. Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA*, pages 106–115, 2000.

[10] Uriel Feige and Moshe Tennenholtz. Mechanism design with uncertain inputs: (to err is human, to forgive divine). pages 549–558, 2011.

[11] A. Greenberg, J. Hamilton, D.A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[12] Mohammad Taghi Hajiaghayi, Robert Kleinberg, Mohammad Mahdian, and David C. Parkes. Online auctions with re-usable goods. pages 165–174, 2005.

[13] Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. A truthful mechanism for value-based scheduling in cloud computing. In *SAGT*, pages 178–189, 2011.

[14] Ron Lavi and Chaitanya Swamy. Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. In *EC*, 2007.

[15] Eugene L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operation Research*, 26:125–133, 1991.

[16] Roger Myerson. Optimal auction design. In *Mathematics of Operations Research*, volume 6, pages 58–73, 1981.

[17] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *STOC*, 1999.

[18] Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

[19] Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

[20] Jean Charles Rochet. A necessary and sufficient condition for rationalizability in a quasi-linear context. *Journal of Mathematical Economics*, 16(2):191–200, 1987.

[21] Michael Saks and Lan Yu. Weak monotonicity suffices for truthfulness on convex domains. In *EC*, pages 286–293, 2005.

A Empirical Study

In this section, we describe some of the experiments we carried out to further evaluate the benefits of our scheduling framework. Our simulation framework utilizes empirical job traces of a large batch computing cluster.

A.1 Simulation Setup

Our simulations evaluate the performance of the mechanisms over a set of 415 jobs, taken from empirical job traces of a large batch computing cluster. The original workload consists of MapReduce jobs, comprising multiple phases with the constraint that phase $i+1$ can only start after phase i has finished. The available information includes the runtime of the job ($totTime$), the overall amount of consumed CPU hours ($totCPUHours$), the total number of servers allocated to it ($totServers$), the number of phases ($numPhases$) and the maximum number of servers allocated to a job per phase ($maxServersPerPhase$). Since our model is not a MapReduce model, we had to adjust the raw data that was available to us, while preserving the workload characteristics. We describe below the details of the simulation choices we made.

Demand D_j We took the $totCPUHours$ field to represent the demand of the job.

Parallelism bound k_j Since the cloud capacity is given in units of server hours per time slot, the parallelism bound must be given in server CPU hour units as well. The data available to us does not contain information on the actual running time per job of each of servers allocated to it. Therefore, we gave the following estimated parallelism bound: We calculated the average length of a phase ($totTime/numPhases$) and averaged the maximal servers per phase ($maxServersPerPhase$) over the average length of the phase. To translate servers into CPU hours, we took the average amount of CPU hours per server to be the total amount of CPU hours divided by the total number of servers ($totCPUHours/totServers$).

$$k_j = \frac{maxServersPerPhase}{\left(\frac{totTime}{numPhases}\right)} \cdot \frac{totCPUHours}{totServers}$$

Values v_j **and deadlines** d_j Our job traces do not contain any information regarding job deadlines nor any indication on the value of the job. Hence, we synthetically generate them as follows. The deadline is set according the effective length of the job ($len_j = \lceil \frac{D_j}{k_j} \rceil$) multiplied by the slackness parameter s. The value of the job is uniformly drawn from $[0,1]$.

Cloud parameters C, T The cloud capacity is set according to the total demand, so that the total demand would exceed the total amount of available resources. T is set according to the maximal deadline.

A.2 Resource utilization

High utilization is certainly one of the main concerns in the area of cloud computing (see, e.g., [11]). The main practical drawback of the solution given in [13] is that on average, more than half of the resources remain unallocated. Without considering incentives, utilization could be practically improved by adding unscheduled jobs to the solution in a greedy manner, whenever possible. Yet, it is unclear how to improve utilization under the framework of [13] without affecting the truthfulness of the mechanism. In our first experiment, we compare the average utilization under GreedyRTL and the allocation mechanism of [13]. Since the maximum possible utilization level generally depends on the job characteristics, we compare the algorithm to an upper on the utilization, denoted OPT^*_{Util}. Specifically, OPT^*_{Util} is obtained by solving the

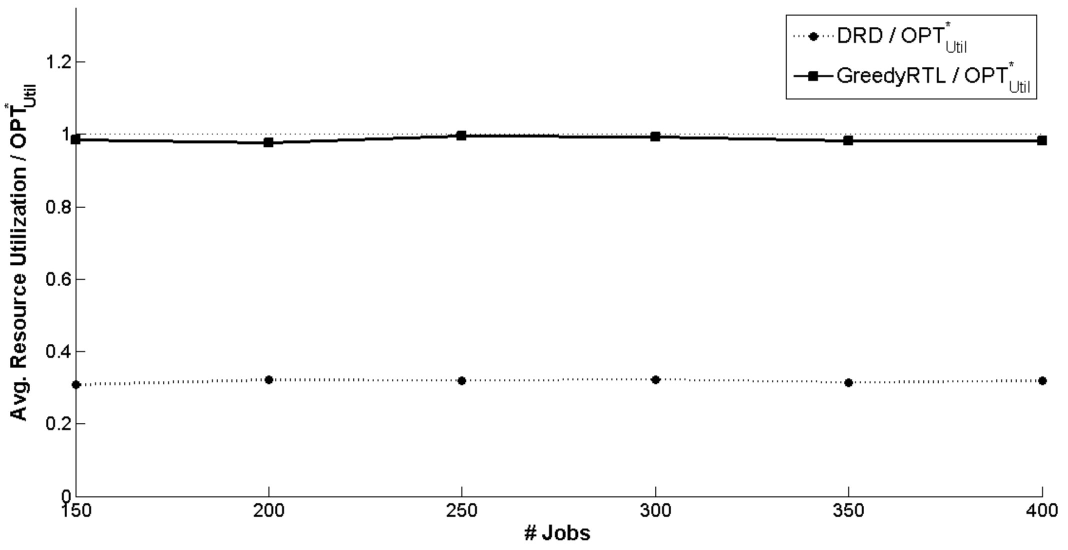

Figure 1: Average resource utilization compared to the Decompose-Ranomly-Draw (DRD) mechanism of [13]. Results show that GreedyRTL utilizes nearly all of the cloud resources.

relaxed linear program (P) with the marginal value of each job set to one (equivalently, we set $v_j := D_j$). Figure 1 (see Appendix A) shows that GreedyRTL reaches a utilization level which is very close to OPT^*_{Util} (within 2% thereof), while the mechanism of [13] achieves around 35% of the upper bound on utilization. The results are consistent regardless the number of jobs that we consider. The utilization results not only provide an explanation to the social welfare improvements we obtain, but also stand on their own – given the significance of utilization in large cloud clusters.

A.3 Revenues

In the next experiment, we evaluate the potential of our approach in terms of revenue maximization. In particular, we examine the revenues of our mechanism against an idealized mechanism which we term the *Optimal Fixed Price (OFP)* mechanism. A fixed price mechanism is a mechanism that charges a fixed price q per server CPU hour, regardless of the job identity. Given that the mechanism charges a fixed price q, it would only schedule a job j with non-negative net utility, namely, $v_j \geq q \cdot D_j$. To focus the comparison solely on the revenue dimension, the fixed price mechanisms uses the same allocation algorithm as the GreedyRTL mechanism, with the value of each job set to $q \cdot D_j$ (equivalently, the marginal value of each job is q) in order to maximize revenues. The *optimal fixed price (OFP)* mechanism charges a price q^*, which is the unit price that maximizes the revenues of the allocation algorithm. Since q^* must be of the form v_j/D_j for some job j (if not, then q^* can be increased without changing the allocation of jobs, thus increasing revenues), we can effectively determine q^* by repeating the allocation algorithm for n different prices $\{v_j/D_j\}_{j=1}^n$, and setting q^* to be the revenue-maximizing price among this set.

We emphasize that OFP is an *idealistic* mechanism, since we assume it has *full* knowledge of the private values of users (and other job parameters). That is, users are assumed to be truthful, although the mechanism does not guarantee that. Recall that our GreedyRTL algorithm is guaranteed to obtain a near-optimal factor of the optimal revenues when we assume Bayesian knowledge on the user valuations (cf. Section 4). Note that the Bayesian assumption is weaker than having full knowledge on the values. To stretch-test GreedyRTL, we do not assume even that for our experiments. That is, for the revenue experiment, we take a worst case scenario in which the algorithm has no knowledge on value distributions, and simply maximizes

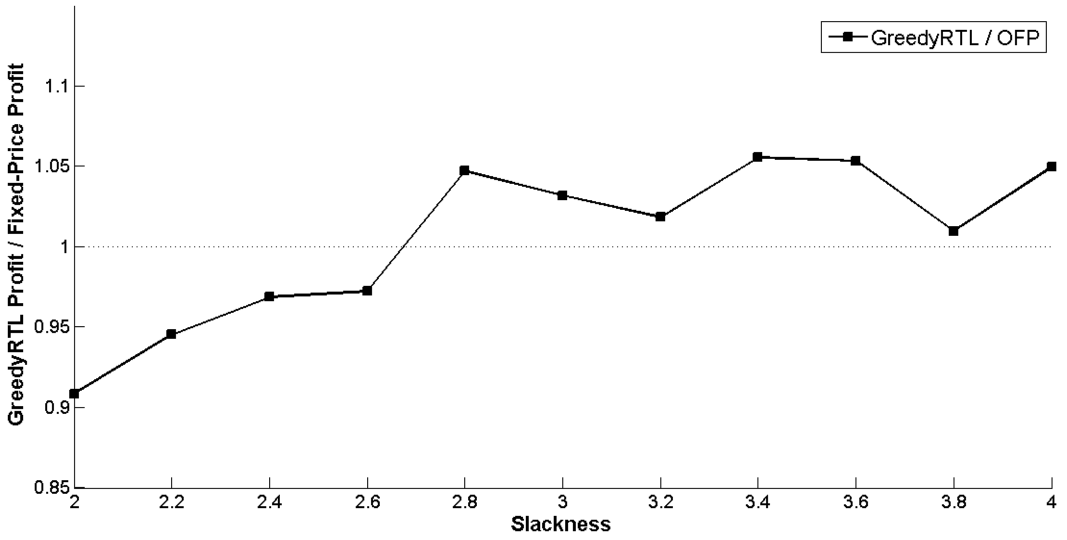

Figure 2: Revenue ratio between GreedyRTL and OFP. The truthful GreedyRTL mechanism is nearly as good as an idealistic optimal fixed-price mechanism. For this experiment, we overload the system such that the total demand exceeds the cloud capacity, so that truthful pricing is strictly positive.

social welfare by setting incentive compatible prices. We examine the revenues generated under the objective of social welfare, revenues that can be obviously improved when statistical knowledge on evaluations is available. Figure 2 (see Appendix A) depicts the ratio of revenues between GreedyRTL and OFP as a function of the slackness parameter s. Surprisingly, despite the fact that OFP has significant value information that GreedyRTL is not assume to have, that GreedyRTL achieves most of the revenues of OFP for small values of s, and outperforms it for larger values of s.

B SimpleGreedy Algorithm

```
   Input: $n$ jobs with $type_j = (v_j, d_j, D_j, k_j)$.
   Output: A feasible allocation of resources to jobs.
1  begin
2      initialize: $y \leftarrow 0$, $\alpha \leftarrow 0$, $\beta \leftarrow 0$, $\pi \leftarrow 0$, $charge \leftarrow 0$
3      sort jobs in non-increasing order of value/demand ratio: $\frac{v_1}{D_1} \geq \frac{v_2}{D_2} \geq \cdots \geq \frac{v_n}{D_n}$
4      for $(j = 1 \ldots n)$ do
5          if $\left(\sum_{t \leq d_j} \min\left\{\bar{W}(t), k_j\right\} \geq D_j\right)$ then          //If job $j$ can be allocated
6              Allocate($j$)
7          else
8              if $(\beta(d_j) = 0)$ then
9                  $\beta$-cover($j$)

10
11 Allocate($j$)
12 begin
13     set $\{y_j(t)\}$ arbitrarily to complete job $j$ without violating capacity/parallelism constraints.
14     $\alpha_j \leftarrow v_j / D_j$

15
16 $\beta$-cover($j$)
17 begin
18     $t_{cov} \leftarrow \min\{t : \beta(t) = 0\}$
19     for $(t = t_{cov} \ldots R(d_j))$ do
20         $\beta(t) \leftarrow v_j / D_j$
21     for $(t = 1 \ldots R(d_j))$ do
22         for $(i = 1 \ldots n)$ do
23             if $(y_i(t) > 0 \wedge charge_i(t) = 0)$ then
24                 $charge_i(t) \leftarrow \left[\frac{C}{C-k} \cdot \frac{s}{s-1}\right] \cdot \frac{v_j}{D_j} \cdot y_i(t)$
```

Algorithm 1: SimpleGreedy

C GreedyRTL Algorithm

```
**Input**: $n$ jobs with $type_j = (v_j, d_j, D_j, k_j)$.
**Output**: A feasible allocation of resources to jobs.
begin
    execute SimpleGreedy, replacing lines 13 (Allocate($j$)) with: AllocateRTL($j$)
    foreach charged job $j$ do
        call $\alpha$-correct($j$)



AllocateRTL($j$)
begin
    $t \leftarrow d_j$
    while $j$ has not been fully allocated do
        $\Delta \leftarrow \min\left\{k_j, D_j - \sum_{t'=t+1}^{d_j} y_j(t')\right\}$
        while $\left(\bar{W}(t) < \Delta\right)$ do
            let $t'$ be the closest unsaturated time slot earlier than $t$
            if $(\beta(t') > 0$ or no such $t'$ exists$)$ then
                jump to line 20
            let $j'$ be a job such that $y_{j'}(t) > y_{j'}(t')$
            increase $y_{j'}(t)$ and decrease $y_{j'}(t')$ simultaneously until $\bar{W}(t) = \Delta$ or $y_{j'}(t) = y_{j'}(t')$
        $y_j(t) \leftarrow \Delta$
        $t \leftarrow t - 1$
    while $j$ has not been fully allocated do          //Allocate $j$ from right to left in a greedy manner
        $y_j(t) \leftarrow \min\left\{k_j, \bar{W}(t), D_j - \sum_{t'=t+1}^{d_j} y_j(t')\right\}$
        $t \leftarrow t - 1$



$\alpha$-correct($j$)
begin
    $\alpha(j) \leftarrow \frac{v_j}{D_j} - \beta(bp(y_j))$
    for $(t = (bp(y_j) + 1) \ldots d_j)$ do
        $\pi_j(t) \leftarrow \beta(bp(y_j)) - \beta(t)$
        $\alpha_j \leftarrow \alpha_j + \frac{k_j}{D_j} \cdot (\beta(bp(y_j)) - \beta(t))$

```

Algorithm 2: GreedyRTL

D Related Work

Scheduling Problems. Scheduling problems have been extensively studied in operations research and computer science (see [7] for an extensive study). Of specific relevance to our work is [15], which considers the problem of preemptively allocating jobs on a single server to maximize the social welfare. Lawler gives an optimal solution in pseudo-polynomial time via dynamic programming to this problem. However, his algorithm cannot be extended to the case where jobs have parallelization limits. Our model significantly extends the basic job interval scheduling problem studied by [4, 5]. The best known approximation factor for this problem is 2. A more general version, in which every interval is given with a width, has also been studied by [19, 4].

Mechanism Design. Mechanism design is a subfield of economic theory which has received much recent attention from computer scientists, commencing with the seminal paper of Nisan and Ronen [17] (see also [18] for a survey book). In its algorithmic aspect, the goal is to design computationally efficient choice mechanisms, such as resource allocation, while optimizing an objective function (e.g., social welfare, total profit). The difficulty of algorithmic mechanism design is that unlike classic algorithmic design, participants act *rationally* in a game theoretic sense and may deviate in order to maximize their personal utilities. Since participants' preferences are usually kept private from the mechanism, we search for efficient mechanisms that implement certain strategic properties to deal with participants' *incentives*, e.g., incentivize users to truthfully report their preferences, while attempting to optimize an objective function.

Truthful mechanisms in single-value domains have been completely characterized by [2]. An allocation rule can implement a truthful mechanism if and only if it is monotone. For allocation rules implementing truthful mechanisms, there is a unique pricing rule implementing it in which unallocated users are charged 0. Myerson, in his celebrated paper [16], first shown this result for single item auctions under Bayesian settings. Compared to single-parameter domains, much less is known about the characterization of implementable allocation rules for multi-parameter problems. Rochet [20] gave an equivalent property to monotonicity called *cyclic monotonicity*, which is a necessary and sufficient condition for truthfulness. Yet, it is unclear how to use this property to easily construct truthful mechanisms from it and only few successful efforts are known (for example, [14]). Saks and Yu [21] showed that for deterministic settings, cyclic monotonicity is equivalent to a simpler property called *weak monotonicity*, which conditions only on cycles of length 2 (see also [3]). However, this result is not valid for randomized mechanisms [6].

Our work is much related to research on algorithmic mechanism design for scheduling problems. We note that most papers in the area mainly focus on minimizing the makespan (see, e.g., [14, 2]). Of specific relevance to our work is a recent paper by Feige and Tennenholtz [10] that provides an impossibility result for designing constant-factor approximation mechanisms when users' demands are uncertain. Our paper confronts demand uncertainties by restricting the deadlines that users may request as a function of certain job characteristics.

E Proofs Omitted from Section 3

E.1 SimpleGreedy

Proof of Lemma 3.1: By induction. Initially, $\beta = 0$ and the claim trivially holds. Consider a call to β-cover(j) and let j' be the last unallocated job for which β-cover(j') was called. By the order through which the greedy algorithm considers jobs, $j' < j$, and thus $v_{j'}/D_{j'} \geq v_j/D_j$. The claim holds since we set $\beta(t)$ to be v_j/D_j in the range $t \in [t_{cov}, R(d_j)]$ and since $d_j \leq R(d_j)$ by the definition of $R(\cdot)$. □

Proof of Lemma 3.3: By Induction. Initially, both sides are 0, thus the claim trivially holds. Let j' be the last unallocated job for which β-cover(j') was called and assume that the claim holds after the call to β-cover(j'). Note that saturated time slots cannot become unsaturated. Between the two calls, the left hand side (LHS) of the inequality is updated as follows:

- $R(d_j) - R(d_{j'}) - |E_j \setminus E_{j'}|$ new saturated time slots in the interval $\big(R(d_{j'}), R(d_j)\big]$ are included in the LHS. Since every active job i in such a time slot t is either being charged or has been charged before, $charge_i(t) \geq \left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right) \cdot y_i(t) \cdot \frac{v_j}{D_j}$. Therefore, for each such t the leftmost expression increases by at least:

$$\frac{C}{C-k} \cdot \frac{s}{s-1} \cdot \sum_{i=1}^{j-1} y_i(t) \cdot \frac{v_j}{D_j} \geq C \cdot \frac{s}{s-1} \cdot \frac{v_j}{D_j},$$

  where the inequality follows since t is saturated. The cost of setting $\beta(t) = \frac{v_j}{D_j}$ for such a time slot t is $C \cdot \frac{v_j}{D_j}$, thus the total gain to the LHS is at least: $\big(R(d_j) - R(d_{j'}) - |E_j \setminus E_{j'}|\big) \cdot C \cdot \frac{v_j}{D_j}$.

- $|E_{j'} \setminus E_j|$ time slots have been saturated in the interval $\big[1, R(d_{j'})\big]$. Since $\beta(t)$ has been already set for such time slots t, by arguments similar to before, the LHS increases by at least: $C \cdot \frac{s}{s-1} \cdot \frac{v_j}{D_j} \cdot |E_{j'} \setminus E_j|$.

- $|E_j \setminus E_{j'}|$ unsaturated time slots have been covered at cost: $C \cdot \frac{v_j}{D_j} \cdot |E_j \setminus E_{j'}|$.

Thus, the left hand side of the inequality increases by at least:

$$C \cdot \frac{v_j}{D_j} \cdot \left[\frac{R(d_j) - R(d_{j'})}{s-1} - \frac{|E_j \setminus E_{j'}|}{s-1} + \frac{s}{s-1} \cdot |E_{j'} \setminus E_j| - |E_j \setminus E_{j'}|\right]. \tag{17}$$

Therefore, by applying the inductive assumption we have:

$$\begin{aligned}
LHS \quad &\geq \quad C \cdot \frac{v_{j'}}{D_{j'}} \cdot \frac{s}{s-1} \cdot \left[\frac{R(d_{j'})}{s} - |E_{j'}|\right] + \\
&\quad + \; C \cdot \frac{v_j}{D_j} \cdot \left[\frac{R(d_j) - R(d_{j'})}{s-1} - \frac{|E_j \setminus E_{j'}|}{s-1} + \frac{s}{s-1} \cdot |E_{j'} \setminus E_j| - |E_j \setminus E_{j'}|\right] \\
&\geq \quad C \cdot \frac{v_j}{D_j} \cdot \left[\frac{R(d_j)}{s-1} - \frac{s}{s-1}|E_j|\right] + C \cdot \frac{v_j}{D_j} \cdot \frac{s}{s-1} \cdot \big[|E_j| - |E_{j'}| - |E_j \setminus E_{j'}| + |E_{j'} \setminus E_j|\big] \\
&= \quad C \cdot \frac{v_j}{D_j} \cdot \frac{s}{s-1} \cdot \left[\frac{R(d_j)}{s} - |E_j|\right],
\end{aligned}$$

since $|E_j| - |E_{j'}| = |E_j \setminus E_{j'}| - |E_{j'} \setminus E_j|$. □

E.2 GreedyRTL

Proof of Claim 3.5: The only stage of the algorithm in which we decrease the total workload $W(t)$ for some time slot t is when we cannot allocate Δ resource units during a call to AllocateRTL. Since we decrease $W(t)$ up to the point where $W(t) = C - \Delta$ and then allocate Δ resource units to j, time slot t becomes full. Specifically, saturated time slots remain saturated throughout the algorithm. $\square$

Proof of Claim 3.6: Let j' be the first (unallocated) job for which the call to β-cover(j') charges j. In order for the allocation y_j to be changed by the algorithm, the following must hold: there must be two time slots $t' < t$, as observed by the AllocateRTL algorithm, such that: (1) $y_j(t) > y_j(t')$ (2) $\beta(t') = 0$. By the monotonicity of β we also know that $\beta(t) = 0$.

It suffices to show that after the call to β-cover(j') it holds that $\beta(bp(y_j)) > 0$. If so, by the monotonicity of β, $bp(t') < t' < t$ implying $y_j(t) = y_j(t') = k_j$, therefore AllocateRTL would not have changed y_j. At the end of the call to β-cover(j') we must have $s(y_j) \leq R(d_{j'})$, otherwise j wouldn't have been charged. Moreover, by the definition of $R(\cdot)$ and since y_j is β-consistent, we have $bp(y_j) \leq R(d_{j'})$. Therefore, after the call to β-cover(j'), $\beta(bp(y_j))$ is at least $v_{j'}/D_{j'} > 0$, as desired. $\square$

Proof of Claim 3.7: By Induction. Initially, the claim trivially holds. Assume that all existing allocations are β-consistent and consider a call to AllocateRTL(j). Recall that by Claim 3.5 saturated time slots remain saturated, and that variables $\beta(t) > 0$ are never unset. If at the end of the call $bp(y_j) = s(y_j)$ then the allocation is trivially β-consistent. Otherwise, consider the point where $t = bp(y_j)$. From this point on, AllocateRTL may allocate job j arbitrarily (specifically, we jump to line 20) since we cannot find an unsaturated time slot t' to the left of t with $\beta(t') = 0$. Therefore, y_j is β-consistent.

Now, consider an allocation $y_{j'}$ of an allocated job j' modified by the AllocateRTL rule, and denote by $\tilde{y}_{j'}$ the resulting modified allocation. As in claim 3.6, there must be two time slots $t' < t$ such that: (1) $y_{j'}(t) > y_{j'}(t')$ (2) $\beta(t') = \beta(t) = 0$. Notice first that $t' \leq bp(y_{j'})$, otherwise $y_{j'}(t') = k_j$ and then we wouldn't have modified $y_{j'}$. Second, by the choice of t', all time slots in the interval $(t', t]$ are either saturated or have a non-zero β value. By the β-consistency property of y_j, the same condition holds for the interval $(s(y_j), bp(y_j)]$. Thus, we have (1) $s(\tilde{y}_{j'}) = \min\{s(y_j), t'\}$ (2) $bp(\tilde{y}_{j'}) = \max\{bp(y_{j'}), t\}$, since if $t > bp(y_{j'})$ we decrease $y_{j'}(t)$ below k_j. Combining (1),(2) with the previous observations prove that $\tilde{y}_{j'}$ is β-consistent, since $\big(s(y_{j'}), bp(y_{j'})\big] = (s(y_j), bp(y_j)] \cup (t', t]$. $\square$

Proof of Theorem 3.8: First we show that the dual solution (α, β, π) constructed by GreedyRTL is feasible, and then we bound its cost. Recall the dual constraint:

$$\alpha_j + \beta(t) + \pi_j(t) - \frac{k_j}{D_j} \cdot \sum_{t' \leq d_j} \pi_j(t') \;\geq\; \frac{v_j}{D_j} \qquad \forall j, t \leq d_j \tag{18}$$

For an unallocated job, by the way β-cover sets the $\beta(t)$ variables and since $\alpha_j = \pi_j(t) = 0$ for every time slot t, all of the dual constraints associated with j are satisfied. Now consider an uncharged job j. Here, we set $\alpha_j = v_j/D_j$ and $\pi_j(t) = 0$ for every time slot t, thus feasibility in this case follows since $\beta(t) \geq 0$ for every time slot t.

Finally, consider a charged job j. To satisfy the dual constraints of j, we follow the routine α-correct (Algorithm 2, lines 25 – 31). Initially, we set α_j $(v_j/D_j - \beta(bp(y_j)))$ to cover all of the constraints up to $bp(y_j)$ (by the monotonicity of β). To cover the remaining constraints, we use the $\pi_j(t)$ variables. First notice that whenever a variable $\pi_j(t)$ is set to some value ε, every time slot (including t) incurs a "punishment" of $-k_j/D_j \cdot \varepsilon$. To balance this, the routine α-correct increases α_j by $k_j/D_j \cdot \varepsilon$ (line 30). To conclude, for each $t \in (bp(y_j), d_j]$ we set $\pi_j(t)$ such that $\beta(t) + \pi_j(t) = \beta(bp(y_j))$ and correct α_j accordingly. By arguments similar to ones used in the previous case, we cover the remaining dual constraints.

We now bound the cost of the dual solution (α, β, π). Since by Claim 3.6, the allocation of a charged job is not modified by the AllocateRTL rule, we can apply Lemma 3.3 and bound this cost by at most:

$$\sum_j D_j \alpha_j \;+\; \sum_{j=1}^{n} \sum_{t \le d_j \bar{W}(t) < k} charge_i(t) \;\le\; \sum_j D_j \alpha_j \;+\; \sum_{j=1}^{n} \sum_{t \le d_j} charge_i(t) \tag{19}$$

Notice that unallocated jobs do not contribute to the dual objective function, since they are not charged and their α value is 0. For an uncharged job j, we only pay $D_j \alpha_j = v_j$. Now, consider a charged job j. Let j' be the first job for which β-cover(j') charges j and let y_j be the allocation of j at that point. According to Claim 3.6, y_j is the final allocation of resources to j at the end of the algorithm. By arguments similar to ones used in the proof of Claim 3.7, we have $bp(y_j) \le R(d_{j'})$. Thus, $\beta(bp(y_j))$ must be at least $v_{j'}/D_{j'}$. Moreover, since at this point j is charged for the first time, we have:

$$charge_j(t) \le \left(\frac{C}{C-k} \cdot \frac{s}{s-1}\right) \cdot \beta(bp(y_j)) \cdot y_j(t) \qquad\qquad \forall t \le d_j$$

For the charged job j, we have:

$$\begin{aligned} D_j \alpha_j &= D_j \cdot \left[\frac{v_j}{D_j} - \beta(bp(y_j)) + \sum_{t=bp(y_j)+1}^{d_j} \frac{k_j}{D_j} \cdot (\beta(bp(y_j)) - \beta(t))\right] && (20) \\ &= v_j \;-\; \left(D_j\, \beta(bp(y_j)) \;-\; \sum_{t=bp(y_j)+1}^{d_j} k_j\, \beta(bp(y_j))\right) \;-\; \sum_{t=bp(y_j)+1}^{d_j} k_j\, \beta(t) && (21) \\ &= v_j \;-\; \sum_{t \le bp(y_j)} y_j(t)\, \beta(bp(y_j)) \;-\; \sum_{t=bp(y_j)+1}^{d_j} k_j\, \beta(t). && (22) \end{aligned}$$

The last inequality follows since y_j is β-consistent by claim 3.7. Now, consider the sum of the charges incurred on j. Consider some call to β-cover(j'') that charges $charge_j(t)$ from j. Specifically, job j is charged according to a price-per-unit of $v_{j''}/D_{j''}$. Since j'' is covered after the call to β-cover(j'') and since j ¡ j'' by the order through which GreedyRTL considers jobs, we have $v_{j''}/D_{j''} \le \beta(t)$ and $v_{j''}/D_{j''} \le v_j/D_j$. Recalling that j is not charged in any time slot according to more than $v_{j'}/D_{j'}$ and since this is at most $\beta(bp(y_j))$ as shown earlier, we get that:

$$\begin{aligned} \sum_{t \le d_j} charge_j(t) \;&\le\; \left(\frac{C}{C-k} \cdot \frac{s}{s-1} - 1\right) \cdot \sum_{t \le d_j} \frac{v_j}{D_j} y_j(t) && (23) \\ &\quad+\; \sum_{t \le bp(y_j)} \beta(bp(y_j))\, y_j(t) \;+\; \sum_{t=bp(y_j)+1}^{d_j} \beta(t)\, y_j(t) && (24) \\ &=\; \left(\frac{C}{C-k} \cdot \frac{s}{s-1} - 1\right) \cdot v_j \;+\; \sum_{t \le bp(y_j)} \beta(bp(y_j))\, y_j(t) \;+\; \sum_{t=bp(y_j)+1}^{d_j} \beta(t)\, k_j. && (25) \end{aligned}$$

By combining (22) and (25) we get that:

$$D_j\alpha_j + \sum_{t\le d_j} charge_j(t) \quad \le \quad \left(\frac{C}{C-k}\cdot\frac{s}{s-1}\right)\cdot v_j.$$

We complete the proof by summing up over all jobs allocated by GreedyRTL. □

F Proofs Omitted from Section 4

Proof of Theorem 4.1: Let j be a user with $\tau_j = \langle v_j, \mathcal{P}_j\rangle$ and let $\tau_j = \langle v'_j, \mathcal{P}'_j\rangle$ be an alternative type bid. To prove that $\mathcal{M}$ is truthful, we must show that $u_j(v_j,\mathcal{P}_j) \ge u_j(v'_j,\mathcal{P}'_j)$. First, notice that by the value-monotonicity of f, the payment set in (16) is always non-negative, since $f_j(s,\mathcal{P}'_j \mid \mathcal{P}'_j) \le f_j(v'_j,\mathcal{P}'_j \mid \mathcal{P}'_j)$ for every $s \le v'_j$. By the definition of u_j and since payments are set according to (16):

$$u_j\left(v_j,\mathcal{P}_j\right) \;=\; \int_0^{v_j} f_j\big(s,\mathcal{P}_j \mid \mathcal{P}_j\big)ds, \tag{26}$$

$$u_j\left(v'_j,\mathcal{P}'_j\right) \;=\; v_j f_j\big(v'_j,\mathcal{P}'_j \mid \mathcal{P}_j\big) - \left[v'_j f_j\big(v'_j,\mathcal{P}'_j \mid \mathcal{P}'_j\big) - \int_0^{v'_j} f_j\big(s,\mathcal{P}'_j \mid \mathcal{P}'_j\big)ds\right]. \tag{27}$$

Note that (26) implies that $\mathcal{M}$ is individually rational, since the utility of a truthful user is non-negative. To conclude, we must show that $u_j(v_j,\mathcal{P}_j) \ge u_j(v'_j,\mathcal{P}'_j)$. Consider two cases. If $f_j\big(v'_j,\mathcal{P}'_j \mid \mathcal{P}_j\big) = 0$, then since payments are non-negative, $u_j(v'_j,\mathcal{P}'_j) \le 0$ and the required condition holds since $\mathcal{M}$ is individually rational.

The second case happens when $f_j(v'_j,\mathcal{P}'_j \mid \mathcal{P}_j) = 1$, implying the following. First, by property-monotonicity we have:

$$\forall s \le v'_j, \quad f_j\big(s,\mathcal{P}'_j \mid \mathcal{P}'_j\big) \;\le\; f_j\big(s,\mathcal{P}_j \mid \mathcal{P}_j\big). \tag{28}$$

Second, since f is rational, it implies that $f_j(v'_j,\mathcal{P}'_j \mid \mathcal{P}'_j) = 1$. By the previous claim and since f is value-monotone, for every value $s \ge v'_j$ we have $f_j(s,\mathcal{P}_j \mid \mathcal{P}_j) = 1$. From here we conclude that for every s, $f_j(s,\mathcal{P}'_j \mid \mathcal{P}'_j) \;\le\; f_j(s,\mathcal{P}_j \mid \mathcal{P}_j)$. After applying this inequality to (26) and combining it with (27) we get:

$$u_j\left(v_j,\mathcal{P}_j\right) - u_j\left(v'_j,\mathcal{P}'_j\right) \;=\; \int_{v'_j}^{v_j} f_j\big(s,\mathcal{P}'_j \mid \mathcal{P}'_j\big)ds \;-\; (v_j - v'_j) f_j\big(v'_j,\mathcal{P}'_j \mid \mathcal{P}'_j\big). \tag{29}$$

Consider the case where $v'_j \le v_j$. By value-monotonicity, for every $s \in [v'_j, v_j]$ we have$f_j(s,\mathcal{P}'_j \mid \mathcal{P}'_j) \ge f_j(v'_j,\mathcal{P}_j \mid \mathcal{P}_j)$. Therefore, (29) is non-negative as required. The case where $v'_j \ge v_j$ is symmetric, interchanging the roles of v'_j and v_j. □

Proof of Theorem 4.3: Consider the set properties $\mathcal{P}_j$ as fixed, making f a single-value allocation rule. By the characterization theorem of single-value allocation functions [2], since f is value-monotone, the mechanism $\mathcal{M} = (f,p)$ with p set as in (16) is truthful. By Theorem 4.2, the mechanism $\mathcal{M}_{\bar{\phi}}$ gives an α-approximation to the optimal expected profit.

It remains to show that $f_{\bar{\phi}}$ admits a truthful mechanism. Notice that since $\bar{\phi}_j$ is monotone for every j, $f_{\bar{\phi}}$ is value-monotone. Moreover, the property-monotonicity and rationality of f directly implies the property-monotonicity and rationality of $f_{\bar{\phi}}$ (since these properties are defined over any value v_j, specifically $\bar{\phi}_j(v_j)$), and therefore we can apply Theorem 4.1. □

Proof of Claim 4.4: Let $\mathcal{P}_j = \langle d_j, D_j, k_j \rangle$ be the property set a user j. Fix the types τ_{-j} of all users apart from j and let $v'_j \le v''_j$ be two values. It is enough to show that if $f_j(v''_j, \mathcal{P}_j \mid \mathcal{P}_j) = 0$ then $f_j(v'_j, \mathcal{P}_j \mid \mathcal{P}_j) = 0$. By the order GreedyRTL goes over the jobs, user j will be considered earlier when reporting v''_j. Consider the two executions of GreedyRTL matching the two values. Notice that both executions are identical up to the point where j is handled by GreedyRTL when reporting v''_j. If j cannot be allocated when reporting v''_j, by Claim 3.5 the amount of available resources in every time slot will only keep decreasing, thus j will not be allocated when reporting v'_j. □

Proof of Claim 4.5: For a user j, fix the types τ_{-j} reported by other users and let $\mathcal{P}_j = \langle d_j, D_j, k_j \rangle$, $\mathcal{P}'_j = \langle d'_j, D'_j, k'_j \rangle$ be two property sets for user j. Assume that there is a value v'_j for which $f_j(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j) = 1$. Now, let $s \le v'_j$. We need to show that if $f_j(s, \mathcal{P}'_j \mid \mathcal{P}'_j) = 1$ then $f_j(s, \mathcal{P}_j \mid \mathcal{P}_j) = 1$. Since the job of user j is fully completed under $\mathcal{P}'_j$, we have $D_j \le D'_j$. Specifically, user j will have a higher priority in the sorted list of jobs when reporting $\mathcal{P}_j$. By Claim 3.5, the are more available resources in every time slot t for allocating user j when he reports $\mathcal{P}_j$ instead of $\mathcal{P}'_j$.

Out of all 4 possibilities, the most complicated one to prove is the case where $d'_j \ge d_j$ and $k'_j \ge k_j$. If j reports an earlier deadline or a smaller parallelism bound, it only makes it more difficult for GreedyRTL to allocate j. Thus, we prove the complicated case (the three other cases will hold by similar arguments). Notice the following: once a job j is allocated by GreedyRTL, any later call to RTLAllocate(j') will not change the completion time of job j. This is true since in case $y_j(t)$ is decreased for some t, at the same time $y_j(t')$ is increased for some t', stopping once they are equal (if not earlier). Therefore, for us to have $f_j(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j) = 1$, job j must not have been allocated after d_j when reporting a type of $\langle v'_j, \mathcal{P}'_j \rangle$. By Claim 3.5, this is also true when reporting $s \le v'_j$. Thus, we can assume without loss of generality that $d'_j = d_j$.

Now, denote by y'_j the allocation set to job j at the end of the call to GreedyRTL when reporting $\langle v'_j, \mathcal{P}'_j \rangle$. Under the assumption that $f_j(v'_j, \mathcal{P}'_j \mid \mathcal{P}_j) = 1$, the maximal entry in y'_j is at most k_j. By the way RTLAllocate modifies allocations, every resource unit occupied by y'_j was available when j was initially allocated by GreedyRTL. Thus, when reporting $\langle v_j, \mathcal{P}_j \rangle$, since j will be considered earlier by GreedyRTL, it will also be possible to allocate j. This concludes the proof. □

G Proofs Omitted from Section 6

Proof of Theorem 6.1: Denote by (α', β', π') the dual solution constructed by GreedyRTL matching reported deadlines d'_j. To bound the gap between RTL' and OPT, we construct a feasible solution (α, β, π) to the dual program matching the original deadlines d_j. Notice that the difference between the two dual programs are the additional cover inequalities $t \in (d'_j, d_j]$ that need to be covered. First, notice that for every j:

$$\frac{d_j}{d'_j} \le \frac{1}{1 - \frac{\delta}{s} - \frac{1}{d_j}} \le \frac{1}{1 - \frac{\delta}{s} - \frac{1}{T}} \tag{30}$$

Denote by κ the last expression to the right of the inequality above. Intuitively, we would like to "stretch" the dual vectors β, π_j towards later time slots, by a stretch factor of κ, and fix the variables α_j to cover all dual constraints. Since the vectors β', π'_j are defined over a discrete domain, we must convert them to continuous functions, to allow the easy stretch. Let $f_{\beta'}, f_{\pi'_j} : [0,T] \to \mathbb{R}^+$ for every j be continuous functions defined as:

$$f_{\beta'}(x) = \beta'\left(\lceil \frac{x}{\kappa} \rceil\right) \quad ; \quad f_{\pi'_j}(x) = \pi'_j\left(\lceil \frac{x}{\kappa} \rceil\right) \tag{31}$$

Now, define the new dual solution to be:

$$\alpha_j = \frac{\alpha'_j}{\kappa} \quad ; \quad \beta(t) = \int_{t-1}^{t} f_{\beta'}(x)dx \quad ; \quad \pi_j(t) = \int_{t-1}^{t} f_{\pi'_j}(x)dx \tag{32}$$

Notice that $\sum_j D_j \alpha_j = \frac{1}{\kappa}\sum_j D_j \alpha'_j$ and $\sum_t C\beta(t) = \frac{1}{\kappa}\sum_t C\beta(t)$, thus the cost of the dual solution increases by a multiplicative factor of $1/\kappa$. To prove the claim, it remains to show that the solution we constructed to the original dual program (with deadlines d_j) is feasible. Since (α', β', π') is a feasible solution to the dual program matching the deadlines d'_j and since for every $t \leq d_j$, $\lceil t/\kappa \leq d'_j$ by (30), we have for every dual constraint (5) matching $j, t \leq d_j$:

$$\begin{aligned}
(5) &= \alpha_j + \beta(t) + \pi_j(t) - \frac{k_j}{D_j}\sum_{t' \leq d_j} \pi_j(t') = \\
&= \alpha_j + \int_{t-1}^{t} \left[\beta'\left(\lceil \frac{x}{\kappa} \rceil\right) + \pi'_j\left(\lceil \frac{x}{\kappa} \rceil\right)\right] dx - \frac{k_j}{D_j}\sum_{t' \leq d_j} \int_{t-1}^{t} f_{\pi'_j}(x)dx \geq \\
&\geq \frac{\alpha'_j}{\kappa} + \int_{t-1}^{t} \left[\frac{v_j}{D_j} - \alpha'_j + \frac{k_j}{D_j}\sum_{t' \leq d'_j} \pi'_j(t')\right] dx - \frac{k_j}{D_j}\sum_{t' \leq d_j} \int_{t-1}^{t} f_{\pi'_j}(x)dx \geq \\
&\geq \frac{v_j}{D_j} + \left(1 - \frac{1}{\kappa}\right)\left(\alpha'_j - \frac{k_j}{D_j}\sum_{t' \leq d'_j} \pi'_j(t)\right) \geq \frac{v_j}{D_j}
\end{aligned}$$

Where the last inequality follows since for an unallocated job j we set $\pi_j(t) = 0$ for every t, and for an allocated job j, α-correct(j) sets α' to be at least $\frac{k_j}{D_j}\sum_{t' \leq d_j} \pi'_j(t)$.

□