Truthful Online Scheduling with Commitments

YOSSI AZAR, Blavatnik School of CS, Tel Aviv University, Tel Aviv, Israel
INNA KALP-SHALTIEL, Blavatnik School of CS, Tel Aviv University, Tel Aviv, Israel
BRENDAN LUCIER, Microsoft Research, Cambridge, MA
ISHAI MENACHE, Microsoft Research, Redmond, WA
JOSEPH (SEFFI) NAOR, CS Department, Technion, Haifa, Israel
JONATHAN YANIV, CS Department, Technion, Haifa, Israel

We study online mechanisms for preemptive scheduling with deadlines, with the goal of maximizing the total value of completed jobs. This problem is fundamental to deadline-aware cloud scheduling, but there are strong lower bounds even for the algorithmic problem without incentive constraints. However, these lower bounds can be circumvented under the natural assumption of deadline slackness, i.e., that there is a guaranteed lower bound $s > 1$ on the ratio between a job's size and the time window in which it can be executed.

In this paper, we construct a truthful scheduling mechanism with a constant competitive ratio, given slackness $s > 1$. Furthermore, we show that if s is large enough then we can construct a mechanism that also satisfies a *commitment* property: it can be determined whether or not a job will finish, and the requisite payment if so, well in advance of each job's deadline. This is notable because, in practice, users with strict deadlines may find it unacceptable to discover only very close to their deadline that their job has been rejected.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; K.6.2 [**Management of Computing and Information Systems**]: Installation Management—*pricing and resource allocation*

Additional Key Words and Phrases: Online scheduling; deadline scheduling; scheduling algorithms; mechanism design; truthful mechanisms; committed scheduling

1. INTRODUCTION

Modern computing applications, such as search engines and big-data processing, run on large clusters operated by either first or third parties (a.k.a., private and public clouds, respectively). Since end-users do not own the compute infrastructure, the use of cloud computation necessitates crisp contracts between them and the cloud provider on the service terms (i.e., Service Level Agreements - SLAs). The problem of designing and implementing such contracts falls within the scope of online mechanism design, which concerns the design of mechanisms for allocating resources when agents arrive and depart over time, and the mechanism must make allocation decisions online. A contract can be as simple as renting out a virtual machine for a certain price per hour. However, with the increased variety of cloud-offered services come more performance-centric contracts, such as paying per number of transactions [Azure 2015], or a guarantee to finish executing a job by a certain deadline [Curino et al. 2014; Ferguson et al. 2012].

This work is supported in part by the Technion-Microsoft Electronic Commerce Research Center, by the Israel Science Foundation (grant No. 1404/10) and by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

Author's addresses: Y. Azar, Blavatnik School of CS, Tel Aviv University, Tel Aviv, Israel; email: azar@tau.ac.il; I. Kalp-Shaltiel, Blavatnik School of CS, Tel Aviv University, Tel Aviv, Israel; email: innakalp@post.tau.ac.il; B. Lucier, Microsoft Research, Cambridge, MA; email: brlucier@microsoft.com; I. Menache, Microsoft Research, Redmond, WA; email: ishai@microsoft.com; J. Naor, Computer Science Department, Technion, Haifa, Israel; email: naor@cs.technion.ac.il; J. Yaniv, Computer Science Department, Technion, Haifa, Israel; email: jyaniv@cs.technion.ac.il.

EC'15, June 15–19, 2015, Portland, OR, USA. ACM 978-1-4503-3410-5/15/06 ...$15.00.

http://dx.doi.org/10.1145/2764468.2764535

Since the underlying physical resources are often limited, a cloud provider faces resource management challenges, such as deciding which service requests to accept in view of the required SLAs, and determining how best to schedule or allocate resources to the different users. For instance, the provider may opt to delay time-insensitive tasks when usage peaks, or prevent admission of low-priority jobs if higher-priority jobs are expected to arrive. To make these decisions in a principled manner, one wishes to design a mechanism for an online scheduling problem with deadlines, aimed at maximizing the total value of completed jobs. This social welfare objective is most relevant in the private cloud setting. It is also relevant for markets with competition between cloud providers, where each provider wishes to extend its market share by increasing user satisfaction. At a high level, the goal of this paper is to provide algorithmic foundations for scheduling jobs with different demands, values and deadlines, in a manner that is compatible with user incentives.

The problem can be abstracted as follows. Each job request j is associated with an arrival time a_j, a size (demand) D_j, a deadline d_j and a value v_j. There are C identical machines that can process jobs. Each job uses at most a single machine at a time, and jobs can be preempted and resumed. The goal is to maximize the total value of jobs completed by their deadlines. In a perfect world, a solution to this problem would achieve a good competitive ratio, would be incentive compatible, and would notify jobs whether or not they are completed as swift as possible. Unfortunately, the basic online scheduling problem, without considering incentives or commitments, is inherently difficult even when $C = 1$. From a worst-case perspective, there is a polylogarithmic lower bound on the competitive ratio of any randomized algorithm [Canetti and Irani 1998]. However, the known lower bounds only apply when all deadlines are tight (i.e., $d_j = a_j + D_j$). Recent work circumvented the lower bound by assuming *deadline slackness*, where every job j satisfies $d_j - a_j \geq s \cdot D_j$ for a slackness parameter $s > 1$ [Lucier et al. 2013]. Our aim is to continue this line of inquiry and design incentive compatible scheduling mechanisms in the presence of deadline slackness.

Truthfulness. In our online scheduling context, the incentive compatibility requirement is multi-parameter: agents must be incentivized to report their tuple of job parameters $\langle v_j, D_j, a_j, d_j \rangle$. As is standard, we assume agents cannot deviate to an arrival time earlier than a_j, nor report a deadline later than d_j. These assumptions are natural if one views the arrival time as the first time the customer is able to interact with the mechanism, and that job results are not released to a customer until the reported deadline. Furthermore, we generally assume that a job holds no value to the customer unless it is fully completed. Hence, a user cannot benefit from underreporting the job demand.

Commitments. In addition to incentive compatibility, another important feature of a practical scheduling mechanism is commitment: whether, and when, a scheduler guarantees to complete a given job. Traditionally, a preemptive scheduler is allowed to accept a job, process it partially, but then abandon it once its deadline has passed. While this behavior may be justified in terms of pure optimization, in many real-life scenarios it is not acceptable, since users might be left empty-handed at their deadline. In reality, users with business-critical jobs require an indication, well before their deadline, of whether their jobs can be processed. Since sustaining deadlines is becoming a key requirement for modern computation clusters (e.g., [Curino et al. 2014] and references therein), it is essential that schedulers provide some degree of commitment.

The question is: at what point of time should the scheduler commit to jobs? One option is to require the scheduler to commit to jobs upon arrival. Namely, once a job arrives, the scheduler immediately decides whether it accepts the job (and then it is required to complete it) or reject the job. However, [Lucier et al. 2013] proved that for general values no scheduler can commit to jobs upon arrival while providing any performance guarantees, even assuming deadline slackness. Therefore, a more plausible alternative from the user perspective is to allow the committed scheduler to delay the decision, but only up to some predetermined point.

Definition 1.1. A scheduling mechanism is called *β-responsive* (for $\beta \geq 0$) if, for every job j, by time $d_j - \beta \cdot D_j$ it either (a) rejects the job, or (b) guarantees that the job will be completed by its deadline and specifies the required payment.

Note that β-responsiveness requires deadline slackness $s \geq \beta$ for feasibility. Schedulers that do not provide advance commitment are by default 0-responsive; we often refer to them as being non-committed. Useful levels of commitments are typically obtained when $\beta \geq 1$, as this provides rejected users an opportunity to execute their job elsewhere before their deadline.

One might consider different definitions for responsiveness in online scheduling. In a sense, the definition given here is additive: for each job j, the mechanism must make its decision βD_j time units before the deadline. An alternative definition could be fractional: the decision must be made before some fraction of job execution window, e.g., $d_j - \omega(d_j - a_j)$ for $\omega \in (0, 1)$. It turns out that many of our results[1] also satisfy responsiveness under this alternative definition, as well as other useful properties[2]. We discuss this further in Section 6.

1.1. Our Results

We design the first truthful online mechanisms for preemptive scheduling with deadlines. Moreover, our mechanism can be made β-responsive as defined above.

Main Theorem (informal): For every $\beta \geq 0$, given sufficiently large slackness $s \geq s(\beta)$, there is a truthful, β-responsive, $O(1)$-competitive mechanism for online preemptive scheduling on C identical servers.

The precise competitive ratio achieved by our mechanism depends on the level of input slackness. We establish the main result in two steps. First, we build a mechanism that is truthful, but not committed. Second, we develop a reduction from the problem of scheduling with responsive commitment to the problem of scheduling without commitment. Each of these two steps may be of interest in their own right. In particular, we obtain in the first step a truthful $O(1)$-competitive mechanism for online preemptive scheduling with deadlines.

THEOREM 1.2. *There is a truthful mechanism for online scheduling on multiple identical servers that obtains a competitive ratio of* $2 + \Theta\left(\frac{1}{\sqrt[3]{s}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s}-1)^3}\right)$ *for any* $s > 1$.

Note that, as implied by known lower bounds, this competitive ratio grows without bound as $s \to 1$. However, as s grows large, the competitive ratio we achieve approaches 2. Our approach for this result is to begin with a greedy scheduling rule that prioritizes jobs by value density (value per size), then modify this scheduler so that (a) jobs are not allowed to begin executing too close to their deadlines, and (b) one job cannot preempt another unless its value density is sufficiently greater. These modifications generate incentive issues that need to be addressed with some additional tweaking. We then analyze the competitive ratio of this scheduler using dual fitting techniques, as described in Section 2.3. This analysis appears in Section 3.

For the second step, we provide a general reduction from committed scheduler design to non-committed scheduler design. We abstract our reduction here for $\beta = s/2$. The idea behind the reduction is to employ simulation: each incoming job is slightly modified and submitted to a simulator for the first half of its execution window. The simulator uses the given non-committed scheduling to "virtually" process jobs. If the simulation completes a job, then the algorithm commits to executing the job on the physical server. See Section 4 for more details. This reduction can be applied to any scheduling algorithm, not just the truthful scheduler described above. Specifically, applying our reduction to the (non-truthful) algorithm described in [Lucier et al. 2013] generates a (non-truthful) committed scheduler with a competitive ratio that approaches 5 as s grows large.

THEOREM 1.3. *There is a* $(s/2)$*-responsive scheduler for online scheduling on multiple identical servers that obtains a competitive ratio of* $5 + \Theta\left(\frac{1}{\sqrt[3]{s}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s}-1)^2}\right)$ *for any* $s > 4$.

[1] Specifically, all of the results stated in Section 1.1, except for Theorem 1.4.

[2] Such as the *no-early-processing* property: the scheduler cannot begin to process a job without committing first to its completion. This implies that any job that begins processing is guaranteed to complete.

To obtain both truthfulness and responsiveness, we wish to compose our reduction with the truthful non-committed mechanism described above. One challenge is that our basic reduction preserves truthfulness with respect to all parameters *except* arrival time. We can therefore immediately obtain a constant competitive-ratio scheduling mechanism which is $(s/2)$-responsive, given sufficient slackness; and truthful, given that jobs do not purposely delay their arrivals. For the single server case, we obtain the same asymptotic bound as in Theorem 1.3 for $s > 4$; see Section 5.

To yield our most general result, we explicitly construct a scheduling mechanism that obtains full truthfulness based on the truthful non-committed scheduler and a general reduction from committed scheduling to non-committed scheduling. The construction is rather technical and significantly increases the competitive ratio. We obtain the following result, with constants $s_0 = 279.744$ and $c_0 = 187.496$; the constants improve for the single server case.

THEOREM 1.4. *There is a truthful, $(s/12)$-responsive mechanism for online scheduling on multiple identical servers that obtains a competitive ratio of* $c_0 + \Theta\left(\frac{1}{\sqrt[3]{s/s_0}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s/s_0}-1)^3}\right)$ *for any* $s > s_0$.

1.2. Related Work

Online preemptive scheduling models have been widely studied in the scheduling theory for various objectives, with value maximization results being of most relevance to our work. Canetti and Irani [Canetti and Irani 1998] consider the case of tight deadlines, obtaining a deterministic lower bound of κ and a randomized $\Omega\left(\sqrt{\log\kappa/\log\log\kappa}\right)$ lower bound, where κ is the max-min ratio between either job values or job demands. Several upper bounds have been constructed [Koren and Shasha 1992; 1994; Canetti and Irani 1998; Porter 2004], with the best being a randomized $O(\log\kappa)$ algorithm. In [Lucier et al. 2013], we show that by incorporating a deadline slackness constraint, a non-committed online preemptive scheduler for the general value model exists, and prove a bound[3] of $2+\Theta\left(\frac{1}{\sqrt[3]{s}-1}\right)+\Theta\left(\frac{1}{(\sqrt[3]{s}-1)^2}\right)$ on its competitive ratio, which is constant for every $s > 1$. However, [Lucier et al. 2013] do not provide any algorithmic guarantees for committed scheduling models. Other constant competitive schedulers have been known only for special cases. When all demands are identical, a 5-competitive scheduler exists, which can be improved to 2 assuming a discrete timeline [Hajiaghayi et al. 2005]. Another studied model is where the value of each job equals its demand; this model is known as the busy time maximization problem [DasGupta and Palis 2000; Garay et al. 2002; Bar-Noy et al. 1999] . These works can be combined to obtain a 1-responsive algorithm with a competitive ratio of $\min\{5.83, 1+1/s\}$; however, the algorithm cannot be extended to incorporate general values.

Much less is known about *truthful* online scheduling mechanisms. Previous works (e.g., [Lavi and Swamy 2007; Archer and Éva Tardos 2001]) focus mostly on offline settings with makespan as main objective. [Jain et al. 2011; 2012] design incentive compatible algorithms for jobs with deadlines, but restrict attention to the offline setting. Works on online truthful scheduling have largely focused on achieving the (non-constant) bounds from the algorithmic literature [Porter 2004; Hajiaghayi et al. 2005] Finally, [Lucier et al. 2013] proposes an incentive compatible heuristic for the 1-responsive case; no formal guarantees are provided for the quality of the heuristic.

2. PRELIMINARIES

In this section we present the scheduling model and necessary definitions (Sections 2.1 and 2.2). We then provide a brief overview of the dual fitting technique, which is used to analyze the proposed mechanisms (Section 2.3).

[3] The bound presented by [Lucier et al. 2013] can be generalized to this form.

2.1. Scheduling Model

We consider a system consisting of C identical servers, which are always available throughout time. The scheduler receives job requests over time. Denote by $\mathcal{J}$ the set of all job requests received by the scheduler. Each job request $j \in \mathcal{J}$ is associated with a *type* $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$. The type of each job j consists of the job value v_j, the job resource demand (size) D_j, the arrival time a_j and the deadline d_j. Write T as the space of possible types. We denote by $\rho_j = v_j / D_j$ the value-density of job j. The job requests in $\mathcal{J}$ are revealed to the scheduler only upon arrival. The scheduler can allocate resources to jobs, provided that at any point each job is processed on at most one server and each server is processing at most one job. Preemption is allowed. Specifically, jobs may be paused and resumed from the point they were preempted. If a job is allocated to servers for a total time of D_j during the interval $[a_j, d_j]$, then it is completed by the scheduler.

An input instance of the scheduling problem is represented by a type profile $\tau = \{\tau_j : j \in \mathcal{J}\}$. Given a scheduling algorithm $\mathcal{A}$, denote by $\mathcal{A}(\tau)$ the jobs that are fully completed by $\mathcal{A}$ on an instance τ, and by $v(\mathcal{A}(\tau))$ their aggregate value. The goal of the scheduler is to maximize $v(\mathcal{A}(\tau))$. Let OPT denote the optimal offline algorithm. The quality of an online scheduler is measured by its competitive ratio, which is the worst case ratio between the optimal offline value and the value gained by the algorithm. In this paper, we define the competitive ratio as a function of the input *slackness*, defined $s \triangleq s(\tau) = \min \left\{ \frac{d_j - a_j}{D_j} \mid \tau_j = \langle v_j, D_j, a_j, d_j \rangle \in \tau \right\}$. The competitive ratio of an online algorithm $\mathcal{A}$ is given by:

$$\mathrm{cr}_{\mathcal{A}}(s) = \max_{\tau : s(\tau) = s} \left\{ \frac{v(OPT(\tau))}{v(\mathcal{A}(\tau))} \right\} \in [1, \infty). \tag{1}$$

The following definitions refer to the execution of an online allocation algorithm $\mathcal{A}$ over an instance τ. We drop $\mathcal{A}$ and τ from notation when they are clear from context. Time is represented by a continuous variable t. For a scheduling algorithm $\mathcal{A}$, denote by $j^i_{\mathcal{A}}(t)$ the job running on server i at time t and by $\rho^i_{\mathcal{A}}(t)$ its value-density. We use $y^i_j(t)$ as a binary[4] variable indicating whether job j is running on server i at time t, i.e., whether $j = j^i_{\mathcal{A}}(t)$ or not. We often refer to the function y^i_j as the *allocation* of job j on server i, and to y_j as the *allocation* of job j.

2.2. Mechanisms and Incentives

Each job in $\mathcal{J}$ is owned by a rational agent (i.e., user), who submits it to the scheduling mechanism. We will be studying direct revelation mechanisms, where each user participates by announcing its type $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$ from the space T of possible types. A mechanism then consists of an allocation rule $\mathcal{A} : T^{\mathcal{J}} \to \{0,1\}^{\mathcal{J}}$ and a payment rule $p : T^{\mathcal{J}} \to \mathbb{R}^{\mathcal{J}}$. Writing $\mathcal{A}(\tau)$ as the profile of allocations returned by the mechanism given type profile τ, we interpret $\mathcal{A}_j(\tau)$ as an indicator for whether the job of customer j is fully completed by its deadline. In general mechanisms can be randomized, in which case we can interpret $\mathcal{A}_j(\tau) \in [0,1]$ as the expected allocation of agent j. However, all of the mechanisms we consider in this paper are deterministic. We will restrict our attention to online mechanisms, which are constrained to make scheduling decisions at each point in time without knowledge of jobs that arrive at future times. Agents have quasilinear utilities: given allocations x and payments p, the utility of user j is given by $u_j(\tau) = v_j \mathcal{A}_j(\tau) - p_j(\tau)$. Each user attempts to maximize its personal utility.

We adopt a model in which we only allow late reports of arrivals, early reports of deadlines, and increased reports of job lengths. As discussed in the introduction, this assumption is justifiable in the context of allocating cloud resources. We say a mechanism is *truthful* if, subject to these restrictions on type reports, each user j maximizes expected utility by reporting his true type to the mechanism, for any possible declarations of the other agents.

[4] In Section 2.3 we extend the range of values $y^i_j(t)$ may receive. However, we will always treat it as an allocation indicator.

We will make heavy use of a characterization of truthfulness made by [Hajiaghayi et al. 2005]. We say that a type $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$ *dominates* $\tau'_j = \langle v'_j, D'_j, a'_j, d'_j \rangle$ if $v_j \geq v'_j$, $D_j \leq D'_j$, $a_j \geq a'_j$, and $d_j \leq d'_j$. We then say that an algorithm $\mathcal{A}$ is *monotone* if for any type profile τ, any j, and any τ'_j that dominates τ_j, we have that $\mathcal{A}_j(\tau_j, \tau_{-j}) \leq \mathcal{A}_j(\tau'_j, \tau_{-j})$. For deterministic algorithms, this means that if job j is allocated under input profile τ, then it will also be allocated if customer i's report changes from τ_j to a type that dominates τ_j.

THEOREM 2.1 ([HAJIAGHAYI ET AL. 2005]). *Given an allocation algorithm $\mathcal{A}$, there exists a payment rule p such that mechanism $(\mathcal{A}, p)$ is truthful if and only if $\mathcal{A}$ is monotone.*

2.3. LP and Dual Fitting

Our competitive ratio analysis relies on a relaxed formulation of the problem as a linear program (LP). The relaxed LP formulation was suggested in [Jain et al. 2011] and considered later in [Jain et al. 2012; Lucier et al. 2013]. In this paper, we do not require the LP formulation itself, but do rely on its dual. For completeness, we present below both the primal and dual programs. The primal program holds a variable $y^i_j(t)$ representing the allocation of a job j on server i at time $t \in [a_j, d_j]$.

Primal Program.

$$\max \quad \sum_{j \in \mathcal{J}} \sum_{i=1}^{C} \int_{a_j}^{d_j} \rho_j y^i_j(t) dt \tag{2}$$

$$\sum_{i=1}^{C} \int_{a_j}^{d_j} y^i_j(t) dt \leq D_j \qquad \forall j \tag{3}$$

$$\sum_{j : t \in [a_j, d_j]} y^i_j(t) \leq 1 \qquad \forall i, t \tag{4}$$

$$\sum_{i=1}^{C} y^i_j(t) - \frac{1}{D_j} \cdot \sum_{i=1}^{C} \int_{a_j}^{d_j} y^i_j(t) dt \leq 0 \qquad \forall j, t \in [a_j, d_j] \tag{5}$$

$$y^i_j(t) \geq 0 \qquad \forall j, i, t \in [a_j, d_j]$$

The first two sets of constraints (3),(4) are standard demand and capacity constraints. The constraints (5) are gap-reducing constraints; see [Jain et al. 2011] for an interpretation of these constraints. Note that for the single server case, the constraints (5) are redundant, since they follow from (4). The primal objective (2) is to maximize the total (fractional) value.

The dual linear program of an instance τ is given as follows.

Dual Program.

$$\min \quad \sum_{j \in \mathcal{J}} D_j \alpha_j + \sum_{i=1}^{C} \int_{0}^{\infty} \beta_i(t) dt \tag{6}$$

$$\text{s.t.} \quad \alpha_j + \beta_i(t) + \pi_j(t) - \frac{1}{D_j} \int_{a_j}^{d_j} \pi_j(t') dt' \geq \rho_j \qquad \forall j \in \mathcal{J},\, i,\, t \in [a_j, d_j] \tag{7}$$

$$\alpha_j,\, \beta_i(t),\, \pi_j(t) \geq 0 \qquad \forall j \in \mathcal{J},\, i,\, t \in [a_j, d_j] \tag{8}$$

We provide the intuition behind the dual formulation. The dual program holds a constraint (7) for every tuple (j, i, t), where j is an input job, i is a server index, and $t \in [a_j, d_j]$ is a specific time. Note that since time is continuous, there are an infinite number of constraints. However, this does not impose an issue, since we do not solve the dual program explicitly. There are three types of dual variables. We typically set $\pi_j(t) = 0$, since these variables are not required throughout this paper. The second variable α_j is associated with each job j and appears in all of the constraints of job j. Setting $\alpha_j = \rho_j$ allows us to satisfy all of the constraints associated with job j. As a result, the dual objective function (6) increases by $D_j\alpha_j = D_j\rho_j = v_j$. The α_j variables are typically used to cover all the constraints of a completed job j, since the cost of covering their constraints is equal to their value. The last variables $\beta_i(t)$ appear in all constraints associated with a server i and time t. These variables are typically used to cover the dual constraints associated with incomplete jobs, since these variables are shared across the constraints of all jobs.

We denote by $OPT^*(\tau)$ the optimal fractional solution of the dual program for an instance τ. Define IG$(s) = \max_{\tau : s(\tau)=s} \{v(OPT^*(\tau))/v(OPT(\tau))\}$ as the integrality gap for instances with slackness s. We are interested in online scheduling algorithms that induce upper bounds on the integrality gap.

Definition 2.2. An online scheduling algorithm $\mathcal{A}$ induces an upper bound on the integrality gap for a given slackness s if IG$(s) \leq$ cr$_{\mathcal{A}}(s)$.

The *dual fitting* technique bounds both the competitive ratio cr$_{\mathcal{A}}(s)$ of an online algorithm $\mathcal{A}$ and the integrality gap IG(s) by constructing a feasible solution to the dual program and bounding its dual cost. Every feasible dual solution induces an upper bound on the optimal fractional solution, and the well-known weak duality theorem implies that $v(OPT(\tau)) \leq v(OPT^*(\tau))$. Moreover, $v(\mathcal{A}(\tau)) \leq v(OPT(\tau))$. Therefore, we can obtain bounds on the integrality gap and the competitive ratio of $\mathcal{A}$. This is summarized in the following theorem.

THEOREM 2.3 (DUAL FITTING [VAZIRANI 2001]). *Let $\mathcal{A}$ be an online scheduling algorithm. If for every instance τ with slackness $s = s(\tau)$ there exists a feasible dual solution (α, β, π) with a dual cost of at most $r(s) \cdot v(\mathcal{A}(\tau))$, then* cr$_{\mathcal{A}}(s) \leq r(s)$ *and* IG$(s) \leq r(s)$.

3. TRUTHFUL NON-COMMITTED SCHEDULING

Our first goal is to design a truthful online scheduling mechanism under the deadline slackness assumption, without regard for commitments. The algorithmic version of this problem was studied in [Lucier et al. 2013]. [Lucier et al. 2013] presents a modified greedy scheduling algorithm, and shows that it obtains a constant competitive ratio for any $s > 1$. However, the algorithm in [Lucier et al. 2013] is not monotone. We refer the reader to the extended version of the paper for a counterexample, in which a job that would not be completed can manipulate the algorithm by reporting a lower value and consequently be completed by its deadline.

In this section, we develop a new *truthful* mechanism $\mathcal{A}_T$, which also obtains a constant competitive ratio for any $s > 1$. The mechanism will be parameterized by constants $\gamma > 1$ and $\mu > 1$, which will be specified below. A key element in $\mathcal{A}_T$ is dividing the jobs into buckets (classes), differentiated by their value densities. Precisely, the job classes are $\mathcal{C}_\ell = \left\{ j \mid \rho_j \in \left[\gamma^\ell, \gamma^{\ell+1}\right) \right\}$. Notice that job j belongs to class $\mathcal{C}_\ell$ for $\ell = \lfloor \log_\gamma(\rho_j) \rfloor$. We think of a job j' as dominating another job j if j' is in a "higher" bucket than j. More formally, we use the following notation throughout the section:

Definition 3.1. Given jobs j and j', we say that $j' \succ j$ if $\lfloor \log_\gamma(\rho_{j'}) \rfloor > \lfloor \log_\gamma(\rho_j) \rfloor$.

At a high level, algorithm $\mathcal{A}_T$ proceeds as follows. At each point in time, $\mathcal{A}_T$ will process the job with highest priority according to the ordering $\succ$. That is, a pending job j' can preempt a running job j only if $j' \succ j$. However, there is an important exception: if a job j has not begun its execution by time $d_j - \mu D_j$, then the scheduler will discard that job and will not schedule it thereafter (i.e., it can be rejected immediately). The following intuition motivates these principles. The preemption rule guarantees that the running jobs belong to the highest classes out of all available jobs (proven later, see Claim 3.2). This prevents users from benefiting from a misreport of their values. The decision to not execute a job that has not begun by time $d_j - \mu D_j$ is used to bound the competitive ratio; note that this condition implies that there is slackness in the time interval from the first time the job is executed, to the job's deadline.

We now formally describe our truthful algorithm for the single server case (see Algorithm 1 for pseudo-code). The extension to multiple servers can be found in the full version of the paper.

ALGORITHM 1: Truthful Non-Committed Algorithm $\mathcal{A}_T$ for a Single Server

```
$\forall t,$  $J^P(t) = \{\, j \in \mathcal{J} \mid j$ partially processed by $\mathcal{A}_T$ at time $t \,\wedge\, t \in [a_j, d_j]\}$.
     $J^E(t) = \{\, j \in \mathcal{J} \mid j$ unallocated by $\mathcal{A}_T$ at time $t \,\wedge\, t \in [a_j, d_j - \mu D_j]\}$.

Event: On arrival of job $j$ at time $t = a_j$:
  1. call ClassPreemptionRule($t$).

Event: On completion of job $j$ at time $t$:
  1. resume execution of job $j' = \arg\max \{\rho_{j'} \mid j' \in J^P(t)\}$.
  2. call ClassPreemptionRule($t$).
  3. delay the output response of $j$ until time $d_j$.

ClassPreemptionRule ($t$):
  1. $j$ $\leftarrow$ job currently being processed.
  2. $j^* \leftarrow \arg\max \{\rho_{j^*} \mid j^* \in J^E(t)\}$.
  3. if $(j^* \succ j)$ :
        3.1. preempt $j$ and run $j^*$.
```

Note that the algorithm maintains two job sets. The first set $J^P(t)$ represents jobs j that have been partially processed by time t and can still be executed. The second set $J^E(t)$ represents all jobs j that have not been allocated by time t, where $t \leq d_j - \mu D_j$.

The algorithm's decisions are triggered by one of the following two events: either when a new job arrives, or when a processed job is completed. The algorithm handles both events similarly. When a new job j arrives, the algorithm invokes a *class preemption rule*, which decides which job to process. In this case, the arriving job j preempts the running job only if it belongs to a higher class. The second type of event occurs when the running job is completed. As mentioned earlier, the algorithm delays the output of the job until its respective deadline (line 3). When a job is completed, the algorithm resumes the best job j' among the preempted jobs in $J^P(t)$ (line 1) and calls the class preemption rule (line 2). The class preemption rule would override the decision to resume j' if there exists an unallocated job j^* in $J^E(t)$ belonging to a higher class. In that case, j^* is processed and j' remains preempted. Notice that in both cases, the algorithm favors jobs belonging to higher classes. Formally,

CLAIM 3.2. *Let $j = j_{\mathcal{A}_T}(t)$ be the job processed at time t by $\mathcal{A}_T$. Let $j' \in J^P(t) \cup J^E(t)$. That is, j has either been allocated by time t and $t \in [a_{j'}, d_{j'}]$, or j has not been allocated by time t and $t \in [a_{j'}, d_{j'} - \mu D_j]$. Then, $j' \not\succ j$.*

PROOF. Assume towards contradiction that $j' \succ j$. Let t^* denote the earliest time job inside the interval $\big[a_{j'}, t\big]$ during which j is allocated. Note that t^* must exist, since the claim assumes that j has being processed at time t. At time t^*, the algorithm $\mathcal{A}$ either started processing j or resumed the execution of j. For $\mathcal{A}$ to start j, the threshold preemption rule must have preferred j over j', which is impossible. The second case where $\mathcal{A}$ resumed the execution of job j is also impossible, since either j' would have been resumed instead of j, or the threshold preemption rule would have immediately preempted j. We conclude that $j' \not\succ j$. □

Claim 3.2 implies that at any point in time, the job allocated by $\mathcal{A}_T$ belongs to the highest class among the jobs that can be processed, i.e., either an unallocated job j such that $t \in [a_j, d_j - \mu D_j]$ or a partially processed job j such that $t \in [a_j, d_j]$. Notice further that equalities in job classes are broken in favor of partially processed jobs. This feature is crucial for proving the truthfulness and the performance guarantees of our algorithm. Using Claim 3.2 we prove an additional property, which is also required for establishing truthfulness.

CLAIM 3.3. *At any time t, the set $J^P(t)$ contains at most one job from each class.*

PROOF. By induction. Assume the claim holds and consider one of the possible events. Upon arrival of a new job j^* at time t, the threshold preemption rule allocates j^* only if $j^* \succ j$. Since j is the maximal job in $J^P(t)$, with respect to $\succ$, if j^* is allocated then it is the single job in $J^P(t)$ from its class. Upon completion of job j, it is removed from $J^P(t)$ and the threshold preemption rule is invoked. As before, if a new job is allocated, it belongs to a unique class. □

We now prove that $\mathcal{A}_T$ is truthful, i.e., $\mathcal{A}_T$ can be used to design a truthful online scheduling mechanism.

CLAIM 3.4. *The algorithm $\mathcal{A}_T$ (single server) is monotone.*

The proof of Claim 3.4 is deferred to the extended version. The intuition behind the result is as follows. The algorithm is defined so that the processing of higher-class jobs is independent of the presence of lower-class jobs in the system. As a result, a job j is completed if precisely two conditions hold: first, that there is some time in $[a_j, d_j - \mu D_j]$ in which no job of equal or higher class is executing (so that job j can start), and second, there are at least D_j units of time after the earliest such start time, but before d_j, in which higher class jobs are not executing. These conditions are well-defined because the processing of job j does not impact the times in which jobs of higher class are processed. One can then note, however, that each of these two conditions are monotone with respect to the job's class, length, arrival time, and deadline. One can therefore conclude that the algorithm is monotone, and hence truthfulness follows from Theorem 2.1.

The competitive-ratio analysis of $\mathcal{A}_T$ is similar to the analysis of the non-truthful algorithm $\mathcal{A}$ [Lucier et al. 2013], and proceeds via the dual fitting methodology. Therefore, we defer the full proof to the full paper. Our result is the following.

THEOREM 3.5. *The mechanism $\mathcal{A}_T$ (single-server) is truthful and obtains a competitive ratio*

$$\mathrm{cr}_{\mathcal{A}_T}(s) \;=\; 2 + \Theta\left(\frac{1}{\sqrt[3]{s}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s}-1)^3}\right), \quad s > 1.$$

3.1. Extension to Multiple Servers

We next extend our algorithm to handle multiple servers. We provide a high level description of the algorithm; full details can be found in the extended version. The multiple server algorithm runs a local copy of the single server algorithm on each of the C servers. The algorithm allows a job to use different servers throughout time (equivalently, we use say that a job is allowed to *migrate* between servers), yet with some restrictions: a preempted job j can migrate to any other server before time $d_j - \mu D_j$. After that time, the job may only use the subset of servers which were allocated to it before time $d_j - \mu D_j$. We obtain the following competitive-ratio result.

THEOREM 3.6. *The algorithm $\mathcal{A}_T$ (multiple-servers) obtains a competitive ratio of:*

$$\mathrm{cr}_{\mathcal{A}_T}(s) \;=\; 2 + \Theta\left(\frac{1}{\sqrt[3]{s}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s}-1)^3}\right), \quad s > 1.$$

Observe that the competitive ratio for the multiple server case is (asymptotically) identical to the bound obtained for a single server. However, we note that the constants hidden inside Θ are slightly larger for the multiple-server case.

4. COMMITTED SCHEDULING

In this section we develop the first committed (i.e., responsive) scheduler for online scheduling with general job types, assuming deadline slackness. Our solution is based on a novel reduction of the problem to the "familiar territory" of non-committed scheduling. We introduce a parameter $\omega \in (0,1)$ that affects the time by which the scheduler commits. Specifically, the scheduler we propose decides whether to admit jobs during the first $(1-\omega)$-fraction of their availability window, i.e., by time $d_j - \omega(d_j - a_j)$ for each job j. The deadline slackness assumption $(d_j - a_j \geq sD_j)$ then implies that our scheduler is (ωs)-responsive (cf. Definition 1.1 for $\beta = \omega s$).

We start with the single server case (Section 4.1), where we highlight the main mechanism design principles. We then extend our solution to accommodate multiple servers, which requires some subtle changes in our proof methodology (Section 4.2).

Our competitive-ratio results hold for slackness values greater than some threshold (e.g., $s > 4$ for the single-server case). In Section 4.3, we provide an indication that high slackness is indeed required, by obtaining a related impossibility result for inputs with small slackness.

4.1. Reduction for a Single Server

Our reduction consists of two key components: (1) *simulator*: a virtual server used to simulate an execution of a non-committed algorithm $\mathcal{A}$; and (2) *server*: the real server used to process jobs. The speeds of the simulator and server are the same. We emphasize that the simulator does not utilize actual job resources. It is only used to determine which jobs to admit. We use the simulator to simulate an execution of the non-committed algorithm. Upon arrival of a new job, we submit the job to the simulator with a *virtual type*, defined below. If a job is completed on the simulator, then the committed scheduler admits it to the system and processes it on the server (physical machine). We argue later that the overall value gained by the algorithm is relatively high, compared to the value guaranteed by $\mathcal{A}$.

We pause briefly to highlight the challenges in such simulation-based approach. The underlying idea is to admit and process jobs on the server only after they are "virtually" completed by $\mathcal{A}$ on the simulator. If the simulator completes all jobs near their actual deadlines, the scheduler might not be able to meet its commitments. This motivates us to restrict the latest time in which a job can be admitted. The challenge is to guarantee that all admitted jobs are completed, while still guaranteeing relatively high value.

We now provide more details on how the simulator and server are handled by the committed scheduler throughout execution.

Simulator. The simulator runs an online non-committed scheduling algorithm $\mathcal{A}$. Every arriving job j is automatically sent to the simulator with a virtual type $\tau_j^{(v)} = \langle v_j, D_j^{(v)}, a_j, d_j^{(v)} \rangle$, where $d_j^{(v)} = d_j - \omega(d_j - a_j)$ is the virtual deadline of j, and $D_j^{(v)} = D_j/\omega$ is the virtual demand of j. If $\mathcal{A}$ completes the virtual request of job j by its virtual deadline, then j is admitted and sent to the server.

Server. The server receives admitted jobs once they have been completed by the simulator, and processes them according to the Earliest Deadline First (EDF) allocation rule. That is, at any time t the server processes the job with the earliest deadline out of all admitted jobs that have not been completed.

The reduction effectively splits the availability window to two subintervals. The first $(1-\omega)$ fraction is the first subinterval and the remainder is the second. The virtual deadline $d_j^{(v)}$ serves as the breakpoint between the two intervals. During the first subinterval, the algorithm uses the simulator to decide whether to admit j or not. Then, at time $d_j^{(v)}$, it communicates the decision to the job. In practical settings, this may allow a rejected job to seek other processing alternatives during the remainder of the time. Furthermore, if j is admitted, the scheduler is left with at least $\omega(d_j - a_j)$ time to process the admitted job on the server.

The virtual demand of each job j is increased to D_j/ω. We use this in our analysis to guarantee that the server meets the deadlines of admitted jobs. Note that we must require $D_j/\omega \leq (1-\omega)sD_j$, otherwise j could not be completed on the simulator. By rearranging terms, we get a constraint on the values of s for which our algorithm is feasible: $s \geq \frac{1}{\omega(1-\omega)}$.

4.1.1. Correctness. We now prove that when the reduction is applied, each accepted job is guaranteed to finish by its deadline. Note that the simulator can complete a job before its virtual deadline, hence it may be admitted earlier. However, in the analysis below, we assume without loss of generality that jobs are admitted at their virtual deadline. Accordingly, We define the *admitted type* of job j as $\tau_j^{(a)} = \langle v_j, D_j, d_j^{(v)}, d_j \rangle$.

Recall that $\mathcal{A}_C(\tau)$ represents the jobs completed by the committed algorithm. Equivalently, these are the jobs completed by the non-committed algorithm $\mathcal{A}$ on the simulator. To prove that $\mathcal{A}_C$ can meet its guarantees, we must show that the EDF rule deployed by the server completes all jobs in $\mathcal{A}_C(\tau)$, when submitted with their admitted types. It is well known that for every set of jobs S, if S can be feasibly allocated on a single server (i.e., before their deadline), then EDF produces a feasible schedule of S. Hence, it suffices to prove that there exists a feasible schedule of $\mathcal{A}_C(\tau)$. We prove the following general claim, which implies the correctness of our algorithm.

THEOREM 4.1. *Let S be a set of jobs. For each job $j \in S$, define the virtual deadline of j as $d_j^{(v)} = d_j - \omega(d_j - a_j)$. If there exists a feasible schedule of S on a single server with respect to the virtual types $\tau_j^{(v)} = \langle v_j, D_j/\omega, a_j, d_j^{(v)} \rangle$ for each $j \in S$, then there exists a feasible schedule of S on a single server with respect to the admitted types $\tau_j^{(a)} = \langle v_j, D_j, d_j^{(v)}, d_j \rangle$ for each $j \in S$.*

PROOF. We describe an allocation algorithm that generates a feasible schedule of S with respect to admitted types. That is, the algorithm produces a schedule where a each job $j \in S$ is processed for D_j time units inside the time interval $[d_j^{(v)}, d_j]$. The algorithm we describe allocates jobs in decreasing order of their virtual deadlines. For two jobs $j, j' \in S$, we write $j' \succ j$ when $d_{j'}^{(v)} > d_j^{(v)}$. In each iteration, the algorithm considers some job $j \in S$ by the order induced by $\succ$, breaking ties arbitrarily. We say that time t is *used* when considering j if the algorithm has allocated some job j' at time t; otherwise, we say that t is *free*. We denote by $\mathcal{U}_j$ and $\mathcal{F}_j$ the set of used and free times when the algorithm considers j, respectively. The algorithm works as follows. Consider an initially empty schedule. We iterate over jobs in S in decreasing order of their virtual deadlines, breaking ties arbitrarily; this order is induced by $\succ$. Each job j in this order is allocated during the latest possible D_j free time units. Formally, define $t' = \arg\max\{t : |[t, d_j] \cap \mathcal{F}_j| = D_j\}$ as the latest time such that there are exactly D_j free time units during $[t', d_j]$. The algorithm allocates j during those free D_j time units $[t', d_j] \cap \mathcal{F}_j$.

We now prove that the algorithm returns a feasible schedule of S, with respect to the admitted job types. It is enough to show that when a job $j \in S$ is considered by the algorithm, there is enough free time to process it; namely, there should be at least D_j free time units during $[d_j^{(v)}, d_j]$. Consider the point where the algorithm allocates a job $j \in S$. Define $\ell_R = \max\{\ell \mid [d_j, d_j + \ell] \subseteq \mathcal{U}_j\}$ and denote $t_R = d_j + \ell_R$. By definition, the time interval $[d_j, t_R]$ is the longest continuous block that starts at d_j in which all times $t \in [d_j, t_R]$ are used. Define $t_L = a_j - \ell_R \cdot (1-\omega)/\omega$. We claim that any job $j' \succ j$ allocated in the interval $[d_j^{(v)}, t_R]$ must satisfy $[a_{j'}, d_{j'}] \subseteq [t_L, t_R]$. Assume the claim holds. We show how the claim leads to the theorem. Denote by J_{LR} all jobs $j' \succ j$ that have been allocated sometime during the interval $[d_j^{(v)}, t_R]$. Obviously, we also have $[a_j, d_j] \subseteq [t_L, t_R]$. Now, since we know there exists a feasible schedule of S with respect to the virtual types, we can conclude that the total virtual demand of jobs in $J_{LR} \cup \{j\}$ is at most $t_R - t_L$, since the interval

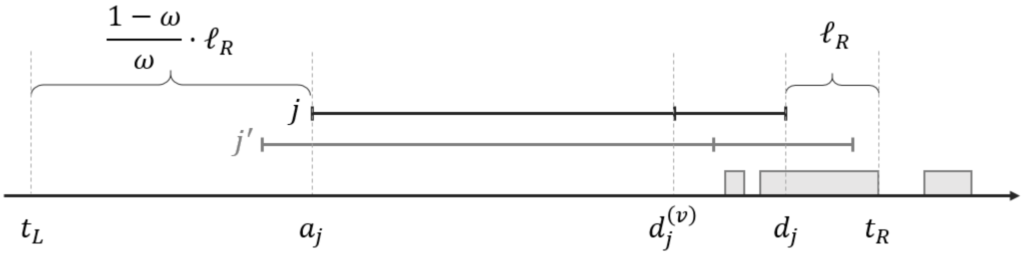

Fig. 1. Illustration of the proof to Theorem 4.1.

$[t_L, t_R]$ contains the availability windows of all these jobs. Notice that $t_R - t_L = (t_R - d_j^{(v)})/\omega$. Since the virtual demand is $1/\omega$ times larger than the admitted demand, we can conclude that the total amount of used time slots during $[d_j^{(v)}, t_R]$ is at most $(t_R - d_j^{(v)}) - D_j$. Thus, there have to be D_j free time units during $[d_j^{(v)}, d_j]$ since $[d_j, t_R]$ is completely full. It remains to prove the claim. Let $j' \in J_{LR}$. Notice that $d_{j'} \leq t_R$; otherwise, the allocation algorithm could have allocated j' after time t_R, and since we assume j' has been allocated sometime between $[d_j^{(v)}, d_j]$, this would contradict the definition of t_R. Also, $j' \succ j$ means $d_{j'}^{(v)} \geq d_j^{(v)}$. Therefore:

$$\begin{aligned} a_{j'} &= \frac{1}{\omega} \cdot d_{j'}^{(v)} - \frac{1-\omega}{\omega} \cdot d_{j'} \quad \geq \quad \frac{1}{\omega} \cdot d_j^{(v)} - \frac{1-\omega}{\omega} \cdot t_R \\ &= \frac{1}{\omega} \cdot d_j - (d_j - a_j) - \frac{1-\omega}{\omega} \cdot (d_j + \ell_R) \quad = \quad a_j - \frac{1-\omega}{\omega} \cdot \ell_R \quad = \quad t_L \end{aligned}$$

which completes the proof. □

4.1.2. Competitive Ratio. We now analyze the competitive ratio obtained via the single server reduction. The competitive ratio is bounded using dual fitting arguments. Specifically, for every instance τ with slackness $s = s(\tau)$, we construct a feasible dual solution (α, β) with dual cost proportional to $v(\mathcal{A}_C(\tau))$, the total value gained by $\mathcal{A}_C$ on τ. Recall the dual constraints (7) corresponding to types $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$. For the single server case, we make two simplifications. First, we denote $\beta(t) = \beta_1(t)$ to simplify notation. Second, we assume that $\pi = 0$ without loss of generality[5]. The dual constraints corresponding to τ reduce to:

$$\alpha_j + \beta(t) \;\geq\; \rho_j \qquad \forall j \in \mathcal{J},\, t \in \left[a_j, d_j\right]. \tag{9}$$

Our goal is to construct a dual solution which satisfies (9) and has a dual cost of at most $r \cdot v(\mathcal{A}_C(\tau))$ for some r. Note that $v(\mathcal{A}_C(\tau)) = v(\mathcal{A}(\tau^{(v)}))$. To do so, we transform a dual solution corresponding to virtual types $\tau^{(v)}$ to a dual solution satisfying (9). The dual constraints corresponding to the virtual types are:

$$\alpha_j + \beta(t) \;\geq\; \omega\rho_j \qquad \forall j \in \mathcal{J},\, t \in \left[a_j, d_j^{(v)}\right] \tag{10}$$

Assume that the non-committed algorithm $\mathcal{A}$ induces an upper bound on $\mathrm{IG}(s^{(v)})$, where $s^{(v)} = s \cdot \omega(1-\omega)$ is the slackness of the virtual types $\tau^{(v)}$. This implies that the optimal dual solution (α^*, β^*) satisfying (10) has a dual cost of at most $\mathrm{cr}_{\mathcal{A}}(s^{(v)}) \cdot v(\mathcal{A}(\tau^{(v)})) = \mathrm{cr}_{\mathcal{A}}(s^{(v)}) \cdot v(\mathcal{A}_C(\tau))$. Yet, (α^*, β^*) satisfies (10), while we require a solution that satisfies (9). To construct a feasible dual solution corresponding to the original job types τ, we perform two transformations on (α^*, β^*) called *stretching* and *resizing*.

[5] This assumption is valid due to the redundancy of the primal constraints corresponding to π for a single server.

LEMMA 4.2 (RESIZING LEMMA). *Let (α, β) be a feasible solution for the dual program corresponding to a type profile $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$. There exists a feasible solution (α', β') for the dual program with demands $D'_j = f \cdot D_j$ for some $f > 0$, with a dual cost of:*

$$\sum_{j \in \mathcal{J}} D'_j \alpha'_j \, + \int\limits_0^\infty \beta'(t) dt \, = \, \sum_{j \in \mathcal{J}} D_j \alpha_j \, + \, \frac{1}{f} \cdot \int\limits_0^\infty \beta(t) dt.$$

PROOF. Notice that the value density corresponding to $D'_j = f \cdot D_j$ is $\rho'_j = \rho_j / f$. Hence, by setting $\alpha'_j = \alpha_j / f$ for every job $j \in \mathcal{J}$ and $\beta(t) = \beta(t)/f$ for every time t, we obtain a feasible dual solution corresponding to resized demands D'_j. The dual cost is as stated since $D'_j \alpha'_j = D_j \alpha_j$ for every job j. □

LEMMA 4.3 (STRETCHING LEMMA, [LUCIER ET AL. 2013]). *Let (α, β) be a feasible solution for the dual program corresponding to a type profile $\tau_j = \langle v_j, D_j, a_j, d_j \rangle$. There exists a feasible solution (α', β') for the dual program with deadlines $d'_j = d_j + f \cdot (d_j - a_j)$ for some f, with a dual cost of:*

$$\sum_{j \in \mathcal{J}} D_j \alpha'_j \, + \int\limits_0^\infty \beta'(t) dt \, = \, \sum_{j \in \mathcal{J}} D_j \alpha_j \, + \, (1 + f) \cdot \int\limits_0^\infty \beta(t) dt.$$

These two lemmas allow us to bound the competitive ratio of $\mathcal{A}_C$.

THEOREM 4.4. *Let $\mathcal{A}$ be a single server scheduling algorithm that induces an upper bound on the integrality gap $\mathrm{IG}(s^{(v)})$ for $s^{(v)} = s \cdot \omega(1 - \omega)$ and $\omega \in (0, 1)$. Let $\mathcal{A}_C$ be the committed algorithm obtained by the single server reduction. Then $\mathcal{A}_C$ is ωs-responsive and*

$$\mathrm{cr}_{\mathcal{A}_C}(s) \, \leq \, \frac{\mathrm{cr}_{\mathcal{A}}\Big(s \cdot \omega(1 - \omega)\Big)}{\omega(1 - \omega)} \quad , \quad s > \frac{1}{\omega(1 - \omega)}.$$

PROOF. We first prove that the scheduler is ωs-responsive. Note that each job j is either committed or rejected by its virtual deadline $d_j^{(v)} = d_j - \omega(d_j - a_j)$. The deadline slackness assumption states that $d_j - a_j \geq s D_j$ for every job j. Hence, each job is notified by time $d_j - \omega s D_j$, as required.

We now bound the competitive ratio. Consider an input instance τ and denote its slackness by $s = s(\tau)$. Let $\tau^{(v)}$ denote the virtual types corresponding to τ, and let $s^{(v)} = s \cdot \omega(1 - \omega)$ denote their slackness. We prove the theorem by constructing a feasible dual solution (α, β) satisfying (9) and bounding its total cost. By the assumption on $\mathcal{A}$, the optimal fractional solution (α^*, β^*) corresponding to $\tau^{(v)}$ has a dual cost of at most $\mathrm{cr}_{\mathcal{A}}(s^{(v)}) \cdot v(\mathcal{A}(\tau^{(v)})) = \mathrm{cr}_{\mathcal{A}}(s^{(v)}) \cdot v(\mathcal{A}_C(\tau))$. We transform (α^*, β^*) into a feasible solution (α, β) corresponding to τ by applying the resizing lemma and the stretching lemma, as follows.

— We first apply the resizing lemma for $f = \frac{1}{\omega}$ to cover the increased job demands during simulation. The dual cost increases by a multiplicative factor of $\frac{1}{\omega}$.

— We then apply the stretching lemma to cover the remaining constraints; that is, the times in the jobs' execution windows not covered by the execution windows of the virtual types. We choose f such that $d_j = d_j^{(v)} + f \cdot \big(d_j^{(v)} - a_j\big)$; hence, $f = \frac{\omega}{1 - \omega}$. As a result, the competitive ratio is multiplied by an additional factor of $1 + f = \frac{1}{1 - \omega}$.

After applying both lemmas, we obtain a feasible dual solution that satisfies the dual constraints (9). The dual cost of the solution is at most $\frac{1}{\omega(1 - \omega)} \cdot \mathrm{cr}_{\mathcal{A}}\big(s \cdot \omega(1 - \omega)\big) \cdot v(\mathcal{A}_C(\tau))$. The theorem follows through the correctness of the dual fitting technique, Theorem 2.3. □

4.2. Reductions for Multiple Servers

We extend our single server reduction to incorporate multiple servers. We distinguish between two cases based on the following definition.

Definition 4.5. A scheduler is called *non-migratory* if it does not allow preempted jobs to resume their execution on different servers. Specifically, a job is allocated at most one server throughout its execution.

Constant-competitive non-migratory schedulers are known to exist in the presence of deadline slackness [Lucier et al. 2013]. Given such a scheduler, we can easily construct a committed algorithm for multiple servers by extending the single server reduction; see full paper for details. However, we do not know how to use this reduction to obtain a committed scheduler which is truthful, since it requires that the non-committed scheduler is both truthful and non-migratory; unfortunately, we are not aware of such schedulers.

Therefore, we construct below a second reduction, which does not require a non-migratory non-committed scheduler. This is essential for Section 5, where we design a truthful committed scheduler. We note that the first reduction leads to better competitive-ratio guarantees, hence should be preferred in domains where users are not strategic.

4.2.1. Non-Migratory Case. In the following, let $\mathcal{A}$ be a non-committed scheduler for multiple servers which is non-migratory. We extend our single server reduction to obtain a committed scheduler $\mathcal{A}_C$ for multiple servers. The reduction remains essentially the same: the simulator runs the non-committed scheduler on a system with C virtual servers. When a job is completed on virtual server i, it is admitted and processed on server i. Each server runs the EDF rule on the jobs admitted to it. To prove correctness (i.e., the scheduler meets all commitments), we simply apply Theorem 4.1 on each server independently. The bound on the competitive ratio obtained in Theorem 4.4 can be extended directly to the non-migratory model.

COROLLARY 4.6. *Let $\mathcal{A}$ be a multiple server, non-migratory scheduling algorithm that induces an upper bound on the integrality gap $\mathrm{IG}(s^{(v)})$ for $s^{(v)} = s \cdot \omega(1-\omega)$ and $\omega \in (0,1)$. Let $\mathcal{A}_C$ be the committed algorithm obtained by the multiple server reduction for non-migratory schedulers. Then $\mathcal{A}_C$ is ωs-responsive and*

$$\mathrm{cr}_{\mathcal{A}_C}(s) \;\leq\; \frac{\mathrm{cr}_{\mathcal{A}}\Big(s \cdot \omega(1-\omega)\Big)}{\omega(1-\omega)} \quad , \quad s > \frac{1}{\omega(1-\omega)}.$$

We note that an improved bound can be obtained by applying the reduction on the specific non-migratory multiple-server algorithm of [Lucier et al. 2013].

4.2.2. Migratory Case. We now assume that $\mathcal{A}$ allows migrations. Unfortunately, the reduction proposed for the non-migratory case does not work here. We explain why: consider some job j that is admitted after being completed on the simulator; note that j may have been processed on more than one virtual server. Our goal is to process j by time d_j. Assume each server runs the EDF rule on the jobs assigned to it, as suggested in Section 4.2.1. Since j has been processed on more than one virtual server, it is unclear how to assign j to a server in a way that guarantees the completion of all admitted jobs. One might suggest to assign each server i the portion of j that was processed on virtual server i. However, this does not necessarily generate a legal schedule. If each server runs EDF independently, a job might be allocated simultaneously on more than one server.

We propose the following modification. Instead of running the EDF rule on each server independently, we run a global EDF rule. That is, at each time t the system processes the (at most) C admitted jobs with earliest deadlines. This is known as the EDF rule for multiple servers (also known as f-EDF [Funk 2004]). It is well known that the EDF rule is not optimal on multiple servers; formally, for a set $\mathcal{S}$ of jobs that can be feasibly scheduled on C servers with migration, EDF does not necessarily produce a feasible schedule on input $\mathcal{S}$ [Hong and Leung 1989]. Nevertheless, it is known that EDF produces a feasible schedule of $\mathcal{S}$ when the servers are twice as fast [Phillips et al. 1997]. Server speedup is directly linked with demand inflation. Formally, when processing $\mathcal{S}$ on C servers, processing demands D_j on servers with α-speedup is equivalent to processing demands αD_j on servers with 1-speedup. Thus, we suggest a second adjustment. We modify the virtual demand of each job submitted to the simulator. That is, the virtual demand of each job j is increased to $2(3+2\sqrt{2}) \cdot D_j/\omega$. The additional factor of $3+2\sqrt{2} \approx 5.828$ is necessary for correctness, which is established in the following theorem.

THEOREM 4.7. *Let $\mathcal{A}$ be a multiple server scheduling algorithm that induces an upper bound on the integrality gap $\mathrm{IG}(s^{(v)})$ for $s^{(v)} = s \cdot \omega(1-\omega)$ and $\omega \in (0,1)$. Let $\mathcal{A}_C$ be the committed algorithm $\mathcal{A}_C$ obtained by the multiple server reduction. Then $\mathcal{A}_C$ is ωs-responsive and*

$$\mathrm{cr}_{\mathcal{A}_C}(s) \leq \frac{11.656}{\omega(1-\omega)} \cdot \mathrm{cr}_{\mathcal{A}}\left(s \cdot \frac{\omega(1-\omega)}{11.656}\right) \quad , \quad s > \frac{11.656}{\omega(1-\omega)}.$$

PROOF. Let $\mathcal{S}$ denote the set of jobs admitted by the committed algorithm $\mathcal{A}_C$ on an instance τ. To prove correctness, we must show that there exists a feasible schedule in which each job $j \in \mathcal{S}$ is allocated $2D_j$ demand during $[d_j^{(v)}, d_j]$. If so, then [Phillips et al. 1997] implies that EDF completes all admitted jobs by their deadline. This follows since:

(1) There exists a feasible schedule of $\mathcal{S}$ with types $\langle v_j, \frac{11.656}{\omega} \cdot D_j, a_j, d_j^{(v)} \rangle$ on C servers with migration. This is the schedule produced by the non-committed algorithm $\mathcal{A}$.
(2) [Chan et al. 2005] proved that any set $\mathcal{S}$ of jobs that can be scheduled with migration on C servers can also be scheduled without migration on C servers with 5.828-speedup. As a result, there exists a feasible non-migratory schedule of $\mathcal{S}$ with types $\langle v_j, \frac{2}{\omega} \cdot D_j, a_j, d_j^{(v)} \rangle$ on C servers.
(3) By applying Theorem 4.1 on each server separately, we obtain a feasible non-migratory schedule of $\mathcal{S}$ with types $\langle v_j, 2D_j, d_j^{(v)}, d_j \rangle$ on C servers, as desired.
(4) Therefore, EDF produces a feasible schedule of the admitted jobs $\mathcal{S}$ with types $\langle v_j, D_j, d_j^{(v)}, d_j \rangle$.

We note that step 4 (i.e., using EDF) is necessary. Even though Step 3 establishes that a feasible schedule of $\mathcal{S}$ exists, it cannot necessarily be generated online, unlike EDF. The competitive ratio can be bounded by following the same steps as in the single server case (Theorem 4.4), however the resizing lemma must be applied with $f = 11.656\omega$. Finally, note that the slackness s must satisfy $s(1-\omega) \geq 11.656/\omega$, otherwise jobs could not be completed on the simulator. □

We use this reduction in Section 5 to design a truthful committed scheduler for multiple servers.

4.3. Impossibility Result

The committed schedulers we construct guarantee a constant competitive ratio, provided that the deadline slackness s is sufficiently large. For example, s has to be at least $(\omega(1-\omega))^{-1}$ for the single server case, implying that $s > 4$ (since $\omega = 1/2$ minimizes the expression). A valid question is whether these conditions on s are merely a consequence of our choice of construction, or an inherent property of any possible committed scheduler. In this subsection, we provide some indication that the latter is more likely, by provider an impossibility result. In particular, we prove a lower bound for committed schedulers that satisfy an additional requirement, termed *no early processing*. A no early processing scheduler is a scheduler that may not process jobs before committing to their execution. We note that the schedulers we have designed in this section satisfy this requirement. It is also worth mentioning that although we did not include no-early processing as part of our β-responsive commitment definition, this is a natural property to require in many practical settings; e.g., when there is a cost (of data transmission, etc.) associated with beginning the execution of a job. Our result is the following.

THEOREM 4.8. *Consider a cluster with $C < 4$ machines. Then any committed scheduler that satisfies the no-early processing requirement has an unbounded competitive ratio for $s < 4/C$.*

In view of Theorem 4.4, note that this bound is tight for the single server case (under the no early processing requirement). It remains an open question whether removing the no-early processing requirement could lead to bounded competitive ratio for a larger range of s. More generally, obtaining tighter lower bounds for multiple servers is a direction that is still unresolved.

5. TRUTHFUL COMMITTED SCHEDULING

Our main goal is to construct a scheduling mechanism which is both truthful and committed. As it turns out, the reductions presented in the previous section preserve monotonicity with respect to values, deadlines, and demands, but not necessarily with respect to arrival times. Therefore, by plugging in an existing truthful non-committed scheduler (Section 3), we can obtain a committed mechanism that is truthful assuming all arrival times are publicly known. We first prove that the reductions preserve monotonicity for all parameters, except arrival times. Then, we describe a generalized reduction which is fully truthful, yet with a slight loss in competitive ratio.

5.1. Public Arrival Times

In this subsection, we consider the case where job arrival times are common knowledge, i.e., users cannot misreport the arrival times of their jobs. To construct the partially truthful mechanism, we apply one of the reductions from committed scheduling to non-committed scheduling (Section 4) on a truthful non-committed mechanism, which we denote by $\mathcal{A}_T$. We denote by $\mathcal{A}_{\tilde{T}C}$ the resulting mechanism. In the following, we prove that $\mathcal{A}_{\tilde{T}C}$ is *almost* truthful: it is monotone with respect to values, deadlines, and demands, but not with respect to arrival times.

CLAIM 5.1. *Let $\mathcal{A}_T$ be a truthful scheduling algorithm, and let $\mathcal{A}_{\tilde{T}C}$ be a committed mechanism obtained by applying one of the reductions from committed scheduling to non-committed scheduling (assume all required preconditions apply). Then, $\mathcal{A}_{\tilde{T}C}$ is monotone with respect to values, demands and deadlines.*

PROOF. Recall that upon an arrival of a new job j, the job is submitted to $\mathcal{A}_T$ with a virtual type of $\tau_j^{(v)} = \langle v_j, \alpha D_j, a_j, d_j^{(v)} \rangle$ for some constant $\alpha \geq 1$ (the constant differs between the reductions for a single server and for multiple servers). Also recall that $d_j^{(v)} = d_j - \omega(d_j - a_j)$ is the virtual deadline of job j, which is a monotone function of d_j. Moreover, $\mathcal{A}_{\tilde{T}C}$ then completes job j on input τ precisely if $\mathcal{A}_T$ completes job j on input $\tau^{(v)}$. But since $\mathcal{A}_T$ is monotone, and since v_j, αD_j, and $d_j^{(v)}$ are appropriately monotone functions of v_j, D_j, and d_j (respectively), it follows that $\mathcal{A}_{\tilde{T}C}$ is monotone with respect to v_j, D_j, and d_j. □

Hence, the reductions from committed to non-committed scheduling (Theorems 4.4 and 4.7) can be extended to guarantee truthfulness (public arrival times) given that the reductions are applied to a monotone non-committed scheduler.

Recall that the definition of β-responsiveness for mechanisms requires not only that allocation decisions be made sufficiently early, but also that requisite payments be calculated in a timely fashion as well. To obtain a β-responsive mechanism we must therefore establish that it is possible to compute payments at the time of commitment, for each job j. Fortunately, because the time of commitment is independent of a job's reported value, this is straightforward. At the time of commitment, it is possible to determine the lowest value at which the job would have been accepted (i.e., scheduled by the simulator). This critical value is the appropriate payment to guarantee truthfulness (see, e.g., [Hajiaghayi et al. 2005]), so it can be offered promptly.

It is important to understand why $\mathcal{A}_{\tilde{T}C}$ may give incentive to misreport arrival times. Consider the single server case, and suppose there are two jobs $\tau_1 = \langle v_1, D_1, a_1, d_1 \rangle = \langle 1, 1, 0, 8 \rangle$ and $\tau_2 = \langle v_2, D_2, a_2, d_2 \rangle = \langle 10, 2, 0, 100 \rangle$. In this instance, job 1 would not be accepted: the simulator will process job 2 throughout the interval $[0, 4]$ (recall that demands are doubled in the simulation), blocking the execution of job 1. Since time 4 is the virtual deadline of job 1 (half of its execution window), the job will be rejected at that time. However, if job 1 instead declared an arrival time of 4, then the simulator would successfully complete the job by its virtual deadline of 6, and the job would be accepted.

5.2. Full Truthfulness

In this subsection, we explicitly construct a truthful, committed scheduling mechanism. The issue in the last example is that misreporting a later arrival time can lead to a later virtual deadline being used by the simulator. This ability to delay the virtual deadline can incentivize non-truthful reporting. We address this issue by imposing additional structure on the time intervals used for simulation. Given a reported job demand D_j, we determine an appropriate time duration w_j to be used in simulation, independent of the reported interval $[a_j, d_j]$. We then run multiple simulations over all intervals of the form $[i \cdot w_j, (i+1) \cdot w_j]$ contained in $[a_j, d_j]$. If the simulator accepts the job in any of these intervals, we accept the job and process it in the subsequent interval; otherwise the job is rejected.

Truthfulness follows from the fact that the simulation parameters cannot be influenced by the reported arrival time. The main technical challenge is to establish a competitive ratio bound for this modified solution; it turns out that the dual-fitting argument used to bound the competitive ratio of $\mathcal{A}_T$ in Section 3 can be modified to provide the necessary bounds. We end up with the following result. A full proof, and a more formal description of the reduction, appears in the extended version of the paper.

THEOREM 5.2. *There exists a truthful, β-responsive scheduling algorithm $\mathcal{A}_{TC}$ such that:*

$$\mathrm{cr}_{\mathcal{A}_{TC}}(s) \;=\; c_0 + \Theta\left(\frac{1}{\sqrt[3]{s/s_0}-1}\right) + \Theta\left(\frac{1}{(\sqrt[3]{s/s_0}-1)^3}\right), \qquad s > \max\{s_0, 12\beta\},$$

where $c_0 = 187.496$ and $s_0 = 279.744$. For the single server case, $c_0 = 17$ and $s_0 = 24$.

6. CONCLUSION

This paper designs and analyzes truthful online scheduling mechanisms. Although the model studied herein is clearly a theoretical abstraction of the full complexity faced by scheduling of tasks in the cloud, we believe that the principles developed here can carry over to more complex settings.

The β-responsive mechanisms described in Section 4 and Section 5.1 actually satisfy two stronger properties. First, they satisfy an alternate responsiveness property: there exists a constant $\omega \in (0, 1)$ such that the scheduler makes a commitment for each job after a $(1-\omega)$ fraction of the job execution window has passed, i.e., by time $d_j - \omega(d_j - a_j)$. Second, they satisfy no early processing, i.e., the mechanisms may process jobs only once they have committed to their completion. In contrast, the truthful scheduling mechanism from Section 5.2 does not necessarily satisfy the two properties. An interesting open question is whether there exists a (fully) truthful scheduling mechanism with constant competitive ratio that commits to scheduling each job before a constant fraction of its execution window has elapsed.

The most obvious problem left open by our work is to improve the constants in our results. The mechanisms constructed for our most general results involve large constants that can potentially be improved. One particularly interesting question along these lines is whether one can obtain an approximation factor that approaches 1 as the number of servers C grows large. An additional avenue of future work is to extend our results to more sophisticated scheduling problems. One might investigate jobs with parallelism, or jobs made up of many interdependent tasks (see, e.g., [Bodík et al. 2014]), or the impact of non-uniform machines or time-varying capacity, and so on. The primary question is then to determine to what extent deadline slackness helps to construct constant-competitive mechanisms for variations of the online scheduling problem.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their useful feedback.

REFERENCES

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

Azure. 2015. Azure Machine Learning Pricing. http://azure.microsoft.com/en-us/pricing/details/machine-learning/. (2015).

Amotz Bar-Noy, Ran Canetti, Shay Kutten, Yishay Mansour, and Baruch Schieber. 1999. Bandwidth Allocation with Preemption. *SIAM Journal of Computing* 28, 5 (1999), 1806–1828.

Peter Bodík, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. 2014. Brief announcement: deadline-aware scheduling of big-data processing jobs. In *SPAA*. 211–213.

Ran Canetti and Sandy Irani. 1998. Bounding the Power of Preemption in Randomized Scheduling. *SIAM Journal of Computing* 27, 4 (1998), 993–1015.

Ho-Leung Chan, Tak Wah Lam, and Kar-Keung To. 2005. Nonmigratory Online Deadline Scheduling on Multiprocessors. *SIAM Journal of Computing* 34, 3 (2005), 669–682.

Carlo Curino, Djellel E Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. 2014. Reservation-based Scheduling: If You're Late Don't Blame Us!. In *SoCC*. 1–14.

Bhaskar DasGupta and Michael A. Palis. 2000. Online real-time preemptive scheduling of jobs with deadlines. In *APPROX*. 96–107.

Andrew D. Ferguson, Peter Bodík, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: guaranteed job latency in data parallel clusters. In *EuroSys*. 99–112.

Shelby Hyatt Funk. 2004. *EDF Scheduling on Heterogeneous Multiprocessors*. Ph.D. Dissertation. University of North Carolina.

Juan A. Garay, Joseph Naor, Bülent Yener, and Peng Zhao. 2002. On-line Admission Control and Packet Scheduling with Interleaving. In *INFOCOM*.

Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Mohammad Mahdian, and David C. Parkes. 2005. Online auctions with re-usable goods. In *EC*. 165–174.

Kwang Soo Hong and Joseph Y.-T. Leung. 1989. Preemptive Scheduling with Release Times and Deadlines. *Real-Time Systems* 1, 3 (1989), 265–281.

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

Navendu Jain, Ishai Menache, Joseph Naor, and Jonathan Yaniv. 2012. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. In *SPAA*. 255–266.

Gilad Koren and Dennis Shasha. 1992. $\mathrm{D}^{\mathrm{over}}$; an optimal on-line scheduling algorithm for overloaded real-time systems. In *RTSS*. 290–299.

Gilad Koren and Dennis Shasha. 1994. MOCA: A Multiprocessor On-Line Competitive Algorithm for Real-Time System Scheduling. *Theoretical Computer Science* 128, 1&2 (1994), 75–97.

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

Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. 2013. Efficient online scheduling for deadline-sensitive jobs. In *SPAA*. 305–314.

Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. 1997. Optimal Time-Critical Scheduling via Resource Augmentation. In *STOC*. 140–149.

Ryan Porter. 2004. Mechanism Design for Online Real-Time Scheduling. In *EC*. 61–70.

Vijay V. Vazirani. 2001. *Approximation algorithms*. Springer.