

Online Flexible Busy Time Scheduling on Heterogeneous Machines

Gruia Calinescu* Sami Davies† Samir Khuller‡ Shirley Zhang §

July 1, 2024

Abstract

We study the online busy time scheduling model on heterogeneous machines. In our setting, jobs with uniform length arrive online with a deadline that becomes known to the algorithm at the job's arrival time. An algorithm has access to machines, each with different associated capacities and costs. The goal is to schedule jobs on machines by their deadline, so that the total cost incurred by the scheduling algorithm is minimized. While busy time scheduling has been well-studied, relatively little is known when machines are heterogeneous (i.e., have different costs and capacities), despite this natural theoretical generalization being the most practical model for clients using cloud computing services. We make significant progress in understanding this model by designing an 8-competitive algorithm for the problem on unit-length jobs and providing a lower bound of 2 on the competitive ratio. The lower bound is tight in the setting when jobs form non-nested intervals. Our 8-competitive algorithm generalizes to one with competitive ratio $8(2p - 1)/p < 16$ when all jobs have uniform length p .

*calinescu@iit.edu. Department of Computer Science, Illinois Institute of Technology. Work performed in part while at Northwestern University.

†samidavies@berkeley.edu. Department of EECS and the Simons Institute for the Theory of Computing, UC Berkeley. Supported by an NSF Computing Innovation Fellowship while at Northwestern University.

‡samir.khuller@northwestern.edu. Department of Computer Science, Northwestern University. Work supported by (IDEAL) NSF 2216970.

§szhang2@g.harvard.edu. Paulson School of Engineering and Applied Sciences, Harvard University. Supported by an NSF Graduate Research Fellowship.

1 Introduction

Busy time scheduling is an energy minimization model that is well-studied due to its applicability to cloud computing and its relationship to energy usage [KSST10, CKM14]. In this model, there are an unlimited number of *heterogeneous* machines available and n jobs arrive online, each with a deadline. A machine of *type* k , for $k \in \mathbb{Z}_{\geq 0}$, has cost $c_k \geq 1$ and capacity $B_k \in \mathbb{Z}_{\geq 1}$, where the cost indicates how much the algorithm must pay for each time step that the machine is executing at least one job, and the capacity dictates how many jobs the machine can execute during each time step. For each job j , we assume its release time r_j and deadline d_j become known to the algorithm when the job is released. The release time is the same time that the job “arrives” online, in the sense that the algorithm learns about the existence of the job.

We consider the setting where jobs have uniform length p (so for all jobs j , $d_j - r_j \geq p - 1$), and all release times and deadlines are integral. Note that in our model, a unit-length job with $r_j = d_j$ may be feasibly executed at time slot r_j . Jobs can be assigned to a machine that is already processing jobs, provided there is spare capacity, and this may result in the machine running for a longer time. The goal is to find a non-preemptive schedule completing all jobs by their deadlines with minimum total cost.

Busy time scheduling is in part motivated by applications in cloud computing. Providers such as Amazon Web Services (AWS), Google Cloud, and Microsoft Azure rent virtual machines, each of which has some fixed cost and processing power. Customers are charged by how many working server hours they use, and therefore they can minimize their costs by making smart choices about which virtual machines they rent. Busy time scheduling captures the difficulty that customers face in deciding whether they should greedily use cheap servers that only complete the jobs imminently due, or whether they should fork up for a more expensive server to complete additional jobs that are waiting to be processed.

One interpretation of the problem with unit-length jobs is as a model of a shuttle scheduling problem. Suppose passengers arrive in airport parking, not in walking distance to the terminal, and tell a dispatcher when their flight is. The shuttle company has shuttles of varying sizes that incur different costs. The company needs to carry all passengers to the terminal, but would like to minimize its cost in doing so.

There has been a surge of theoretical interest in busy time scheduling over the past decade or so, with many variations on the model having been studied. Almost all of the work is for the case of *homogeneous* machines, i.e., when machines are identical with the same costs and capacities. We discuss the different variations of busy time scheduling on homogeneous machines in Section 1.2. Much less is known about the setting when machines are heterogeneous, despite this being (arguably) the most practical setting. Most data centers consist of machines of different types, built over time, and these in general have different costs and capacities.

In the special case of interval jobs¹, there is no flexibility as to when a job is executed; each job has to instantly be scheduled upon arrival. This lack of flexibility can make the resulting cost of any schedule extremely high—for example, if single jobs arrive one after the other and we schedule them all separately, we might pay a huge cost compared to a schedule that has the flexibility to wait and bundle a group of jobs together. Ren and Tang [RT20] studied the problem of inflexible jobs with *heights*, where each job may take up non-unit space on each machine, on heterogeneous machines when the normalized cost-per-capacity rate is either monotonically increasing or decreasing as the

¹This is also called the *inflexible* setting.

machine capacity increases. They developed a $O(1)$ -approximation algorithm (with constants 9 and 14, respectively) in the offline setting and a $\theta(\mu)$ -competitive algorithm in the online setting, for μ the max/min ratio of a job’s arrival time and deadline window length. Later, Liu and Tang [LT21] proved that one can obtain the same guarantees (up to a constant factor), but without the restrictions on the cost-per-capacity rate. They also provide matching lower bounds for both settings.

We are the first to study the online setting when jobs are *flexible*, i.e., not interval, and machines are heterogeneous. However, there is already a lower bound on the competitive ratio of $\Omega(\sqrt{\log \mu})$ [AV19]². In order to study the problem without being totally restricted by this lower bound, we must be willing to make some kind of trade-off; we choose to give up the generality of arbitrary job processing lengths and heights. By focusing on uniform-length jobs, we make the problem more tractable, while still maintaining the core theoretical difficulty of the setting. When jobs are flexible, an algorithm must make the additional decision of *when* in a job’s active interval it should be scheduled, and on heterogeneous machines, the algorithm must make the additional decision on whether to pay more to process large batches, or pay less and defer other jobs’ completions. At a high-level, the “online” and “flexible” aspects of the setting incentivize the algorithm to make decisions lazily, while the “heterogeneous” aspect punishes the algorithm for making decisions too late by forcing the algorithm to use some very expensive resource.

1.1 Our results

Our main result is the following theorem.

Theorem 1 *There is an $8(2p - 1)/p$ -competitive algorithm for online uniform-length busy time scheduling on heterogeneous machines with running time $O(n \log n)$, for n the total number of jobs and p the processing length of any job.*

Observe that the competitive ratio is at most 16 for any p . Due to space restrictions, we present the proof of Theorem 1 for the case where $p = 1$ in the main body of the paper, and the complete extension to uniform length jobs can be found in Appendix A. Overall, the setting of unit jobs captures the essence of the algorithm and proof strategy for general uniform-length jobs, but with a cleaner presentation.

Several straight-forward procedures give $O(1)$ -competitive algorithms when jobs have *agreeable* deadlines, which is when if job j arrives before job j' , then the deadline of job j' cannot be before that of j . For unit-length jobs with agreeable deadlines, we can improve our competitive ratio from 8 to 2 by using a much simpler algorithm. Note that this result does not extend to the uniform-length setting

Theorem 2 *If jobs have agreeable deadlines, then there is a 2-competitive algorithm for online unit-length busy time scheduling on heterogeneous machines with running time $O(nK + n \log n)$, for n the total number of jobs and K the number of distinct machine types.*

The above result is complemented by a matching lower bound on the competitive ratio.

Theorem 3 *The competitive ratio of any deterministic online algorithm for online unit-length busy time scheduling on heterogeneous machines is at least 2.*

Moreover, Theorem 3 shows that the algorithm proving Theorem 2 is tight in its competitive ratio, as our construction proving the lower bound is an agreeable instance.

²This result holds in the much simpler setting where jobs are inflexible and machines are homogeneous.

1.2 Related work

Busy time scheduling for arbitrary length jobs on homogeneous machines³ Even scheduling interval jobs (which is easier than scheduling flexible jobs) on homogeneous machines is NP-hard [WZ03]. Khandekar et. al. [KSST10] showed a 5-approximation for busy time scheduling of flexible jobs with arbitrary heights. When jobs have uniform heights, Chang et al. [CKM14] showed that algorithms developed by Alicherry and Bhatia [AB03] and Kumar and Rudra [KR05] are 4-approximations, and gave an improved algorithm that obtains a 3-approximation.

In the online setting when jobs are flexible, Koehler and Khuller [KK17] give a 5-approximation for the busy time problem when machines have infinite capacities and obtain a $O(\log P)$ competitive ratio (where P is the ratio of maximum to minimum processing time) when machines have finite capacities. Also in the infinite capacity setting, Ren and Tang [RT17] give an algorithm with a competitive ratio of $4 + 2\sqrt{2}$, and complement this result with a lower bound on the competitive ratio of $\frac{\sqrt{5}+1}{2}$, which holds for any deterministic online scheduler. One variant of the online busy time problem considers the setting when job lengths are unknown at arrival time [LTC14, RTLC16]. Notably, the $\theta(\mu)$ -competitive algorithm and matching lower bound of Ren and Tang [RT20] hold for this setting as well. In our work, job lengths are uniform, and therefore known at arrival.

On homogeneous machines with interval jobs, Azar et al. [AV19] developed an online algorithm with tight approximation factor $\theta(\sqrt{\log \mu})$, where again μ is the ratio of the maximum interval window length over the minimum. Improvements are known, but these involve certain assumptions about the knowledge of the profile of jobs that have not yet arrived [BFM⁺21]. It is worth noting that on homogeneous machines, one can find the optimal schedule in the online or offline setting if jobs have unit length by using the algorithm by Finke et al. [FJQS08]⁴.

Power management and batch scheduling Much work in scheduling theory has focused on energy minimization or power management, and busy time scheduling is one such model [YDS95, IP05, ISG07, CCL⁺07, BKL⁺08, LWAT12, AGKK20]. The models that are most relevant to ours study energy minimization when the machine(s) have some cost to turn on or off [CKM14, DKZ22, CW21, AIS08]. In batch scheduling problems, machines can run groups of jobs simultaneously (also called active time scheduling). A series of papers develop approximation algorithms in the pre-emptive model [CGK14, KK18, CFL⁺22]. Classic objectives include throughput maximization [BNGK⁺09] and functions of flow time and makespan [IM13], [WTFY17].

Capacitated covering problems

For $d > 1$, d -dimensional capacitated rectangle stabbing is NP-hard, although there do exist approximation algorithms [HM91, GIK02, ELR⁺08]. When $d = 1$, algorithms for capacitated rectangle stabbing solve the offline version of unit-length busy time scheduling on heterogeneous machines; rectangles are 1-dimensional x -axis aligned lines that correspond to jobs' arrival time, deadline intervals, and stabbers can be equipped with costs and capacities to correspond to a batch of jobs to be scheduled on a machine in a single time slot. This 1-dimensional version can be solved exactly via a dynamic programming algorithm [ELR⁺08], which is based on the influential work of Baptiste [Bap06].

Capacitated versions of covering problems related to vertex cover, facility location, and other scheduling problems have also been studied [CN06, SK12, BP14, CS15]. Several online problems also have a similar flavor to ours, as an algorithm has to make a choice between different options,

³Results are for machines with finite capacities, unless explicitly stated otherwise.

⁴The algorithm with a different proof of optimality appeared previously in work by Bodlaender and Jansen [BJ95].

such as the ski-rental problem with multiple discount options [ZPX11], the parking permit problem with multiple durations [Mey05], and capacitated interval coloring [EEL09].

1.3 Preliminaries

For the rest of the main body of the paper, we discuss this problem specifically for $p = 1$.

As a warm-up, in Section 2 we discuss our algorithm for unit jobs with agreeable deadlines. Then in Section 3, we consider jobs with general active interval structures. We prove a lower bound on the optimal solution’s cost in Subsection 3.2, and we prove an 8-competitive ratio for the case when jobs have unit length (i.e., Theorem 1 in the case where $p = 1$), in Subsection 3.3. A lower bound of 2 on the competitive ratio of any deterministic algorithm is provided to complement these algorithmic results in Section 4. Appendix A contains the proof of Theorem 1 for general p , with further preliminaries for that more general setting in Appendix A.1.

The formal set-up and notation throughout the main body of the paper is as follows. A set of unit-length jobs arrive online, and it is denoted by J with $|J| = n$. Job $j \in J$ is equipped with integral arrival r_j and deadline d_j . A machine of type $k \in \mathbb{Z}_{\geq 0}$ can execute at most $B_k \in \mathbb{Z}_{\geq 1}$ many jobs at once and costs $c_k \geq 1$ per time unit.

The labeling on the machine types is such that the capacity of the machines is increasing, i.e., $B_0 < B_1 < \dots$. Note we assume the ordering is strictly increasing, as if there were ever machines with the same capacity but different costs, one would always use the cheaper machine of the same capacity. Without loss of generality machines with larger capacity should cost more; e.g., if $c_1 \leq c_0$, then an optimal schedule could always use type 1 machines instead of type 0 machines. After these considerations, let K denote the number of remaining machine types.

A *batch* of jobs X is defined by three features: (1) $J(X)$, the set of jobs the batch contains, (2) $t(X)$, the batch’s type, which corresponds to the machine type that the jobs in $J(X)$ should be executed on, and (3) $\tau(X)$, the time at which the jobs in $J(X)$ are executed on a machine of type $t(X)$. We denote a family of batches as \mathcal{X} . A feasible schedule is a partition of jobs into batches if for all X , $|J(X)| \leq B_{t(X)}$ and every $j \in J(X)$ has $r_j \leq \tau(X) \leq d_j$.

An optimal offline solution is denoted by OPT , with cost denoted by $\text{cost}(\text{OPT})$. The time horizon is denoted $[0, T]$, with time steps $\tau \in [0, T]$, where sometimes time steps have subscripts or take an argument. Continuous intervals with interval endpoints in $[0, T]$ are denoted by I , also sometimes with subscripts or taking an argument.

2 Warm-up: Agreeable Deadlines

In Subsection 2.1, we present a 2-competitive algorithm for the setting when jobs have agreeable deadlines. Then in Subsection 2.2, we discuss extending to general job deadlines.

2.1 Proof of Theorem 2

We will show that Algorithm 1 is 2-competitive when jobs have agreeable deadlines. Algorithm 1 is a straightforward greedy algorithm using the earliest deadline first heuristic—it collects jobs until a deadline is reached, and then it creates optimal batches for scheduling the set of all the waiting jobs. We use W to represent the set of waiting jobs.

Algorithm 1 Greedy

```
Let  $W \leftarrow \emptyset, \mathcal{X} \leftarrow \emptyset$ .
for  $\tau \leftarrow 1$  to  $T$  do
   $\triangleright$  Let  $J'$  be the set of jobs with arrival time  $\tau$ .
   $\triangleright$  Update  $W \leftarrow W \cup J'$ .
  if  $\exists j^* \in W$  with  $d_{j^*} = \tau$  then
     $\triangleright \mathcal{X}_\tau \leftarrow \text{GetOptimalBatches}(W)$ .
     $\triangleright$  Execute all batches in  $\mathcal{X}_\tau$ . // Thus completing all jobs in  $W$ 
     $\triangleright$  Update  $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{X}_\tau$ .
     $\triangleright W \leftarrow \emptyset$ .
  end if
end for
return A schedule  $\mathcal{X}$ . // With cost at most  $2 \cdot \text{cost}(OPT)$ 
```

The subroutine `GetOptimalBatches(A)` takes as input a set of jobs A . It first finds the lowest cost set of machines with total capacity $|A|$, then fills these machines with the jobs in A , outputting a set of new batches. It can be implemented in time $O(n \cdot K)$ using a classic dynamic program, where recall that K is the number of distinct machine types. We can initialize the optimal cost of scheduling 0 jobs to 0. After initialization, we can then optimally compute the cost of scheduling $|A|$ jobs by solving the following recurrence:

$$\text{cost}(|A|) = \min_{k \in [K]} \text{cost}(|A| - \min(B_k, |A|)) + c_k.$$

This computation can be made once before Algorithm 1 starts, then the optimal values stored in arrays can be used whenever Algorithm 1 needs them. We sort J by release times so that J' can be found in time $O(|J'|)$. This involves simply maintaining an ordered set of jobs. To store W , we use a min-heap with keys d_j , for elements $j \in W$. We do not have to go time slot by time slot, but instead use the sorted J to find the next time where $J' \neq \emptyset$ and compare this with the smallest key in W . The overall running time is $O(nK + n \log n)$.

The following lemma crucially (though perhaps subtly) uses the agreeable deadlines structure. As in Algorithm 1, let \mathcal{X}_τ be the set of batches executed at time τ . For each set of batches \mathcal{X}_τ executed by Algorithm 1, construct an interval I_τ starting at the earliest release time of any job carried in a batch in \mathcal{X}_τ and ending at the latest deadline of any job in a batch in \mathcal{X}_τ .

Lemma 1 *At most two of the intervals I_τ as defined above overlap at any point in time.*

Proof It suffices to prove that each such interval contains at most one left endpoint of another interval. From the way the algorithm schedules all the jobs in W , we obtain that for any τ there exists at most one \mathcal{X}_τ . Assume we have \mathcal{X}_τ and $\mathcal{X}_{\tau'}$ with $\tau < \tau'$. Then the left endpoint of $I_{\tau'}$ is strictly to the right of τ , as any job released at or before τ that has not been scheduled before τ is scheduled in \mathcal{X}_τ . This also implies that the left endpoint of $I_{\tau'}$ is strictly to the right of the left endpoint of I_τ . Call two such intervals $I_\tau, I_{\tau'}$ *consecutive* if $\tau < \tau'$ and there is no τ'' with $\tau < \tau'' < \tau'$ such that the algorithm produces an $\mathcal{X}_{\tau''}$. If $I_\tau, I_{\tau'}$ are consecutive, then there is no $I_{\tau''}$ whose left endpoint is between the left endpoints of I_τ and $I_{\tau'}$.

Consider three consecutive intervals $I_{\tau_1}, I_{\tau_2}, I_{\tau_3}$. For interval I_{τ_1} , let job j_1 be a job scheduled in \mathcal{X}_{τ_1} with the earliest release time, breaking ties in favor of the job with the earliest deadline. Also

for I_{τ_1} , let job j'_1 be a job scheduled in \mathcal{X}_{τ_1} with the latest deadline, breaking ties arbitrarily. Let j_2, j'_2, j_3 , and j'_3 be defined analogously for I_{τ_2} and I_{τ_3} .

In order to show that I_1 and I_3 do not overlap, we will show that

$$d_{j'_1} \leq d_{j_2} = \tau_2 < r_{j_3}. \quad (1)$$

We first argue that $\tau_2 = d_{j_2}$. Indeed, we must have $\tau_2 \leq d_{j_2}$ since j_2 is scheduled in \mathcal{X}_{τ_2} . On the other hand, if it were the case that $\tau_2 < d_{j_2}$, then there is some job j scheduled in \mathcal{X}_{τ_2} with deadline $d_j = \tau_2$, and jobs j and j_2 break the agreeable relation because $d_j < d_{j_2}$ and $r_j > r_{j_2}$ (by the way j_2 is selected). See Figure 1.

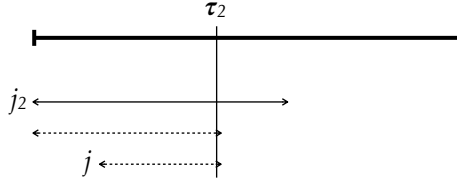


Figure 1: An illustration of an argument from the proof of Lemma 1, precisely why $d_{j_2} = \tau_2$: If $\tau_2 < d_{j_2}$, then one of the dashed jobs/intervals must end at τ_2 . However, the upper dashed job cannot exist since j_2 is chosen as having the earliest deadline among the jobs scheduled in \mathcal{X}_{τ_2} , while the lower dashed job cannot exist since it is not agreeable with j_2 .

The last inequality of 1 follows from the discussion in the first paragraph of the proof, with τ_2 instead of τ and τ_3 instead of τ' . The same reasoning gives that $\tau_1 < r_{j_2}$. We have that $r_{j'_1} \leq \tau_1$ since j'_1 is scheduled in \mathcal{X}_{τ_1} . Thus $r_{j'_1} < r_{j_2}$. If $d_{j'_1} > d_{j_2}$, then j'_1 and j_2 are not agreeable. Thus the first inequality of Equation (1) also holds. ■

If deadlines are not agreeable, many such intervals can overlap at any time point. In the next section, we use a more complicated interval construction to lower bound $\text{cost}(\text{OPT})$.

Proof [*Proof of Theorem 2*] As in the statement of Lemma 1, construct an interval I_τ for each set of batches \mathcal{X}_τ dispatched by the Greedy algorithm, where the left endpoint of the interval corresponding to \mathcal{X}_τ is the earliest release time of any job in $\cup_{X \in \mathcal{X}_\tau} J(X)$ and the right endpoint is the latest deadline of any job in $\cup_{X \in \mathcal{X}_\tau} J(X)$. Lemma 1 implies that we can partition the intervals into two sets S_1 and S_2 of disjoint intervals. Specifically for any two consecutive, overlapping intervals, it suffices to have one such interval in S_1 and the other in S_2 . Let J_1 be the set of jobs executed by batches in \mathcal{X}_τ with $I_\tau \in S_1$, and J_2 likewise for S_2 . Let $\text{OPT}(J_i)$ denote an optimal offline schedule for J_i , for $i = 1, 2$. Note that for any sets of jobs J and J' , if $J' \subseteq J$, then $\text{cost}(\text{OPT}(J')) \leq \text{cost}(\text{OPT}(J))$. Then:

$$\text{cost}(\text{Greedy}) = \text{cost}(\text{OPT}(J_1)) + \text{cost}(\text{OPT}(J_2)) \leq 2 \cdot \text{cost}(\text{OPT})$$

The equality is because the intervals in S_1 are disjoint and the intervals in S_2 are disjoint, and we have sent all jobs in each interval using the lowest cost possible with `GetOptimalBatches`. ■

2.2 Difficulty in non-agreeable deadlines

Any algorithm for this problem can execute batches only when it is forced to, i.e., when the current time slot τ is the deadline of some uncompleted job j^* , as there is nothing to be gained from executing a batch before a deadline. Additionally, jobs can always be assigned to an open batch by an earliest deadline first (EDF) rule—that is, as long as a batch has space, repeatedly select from the set of jobs released and not yet assigned to a batch the job with the earliest deadline, and assign it to the batch. It follows that the only remaining decision for an algorithm is what type of batch (and hence machine) should be used to execute job j^* at time τ ; note that another way to phrase this question is how many waiting jobs should be processed with j^* . An adversary can punish an algorithm for processing too many waiting jobs at a time slot by dispatching a huge batch of jobs soon after (as it would have been more cost efficient to execute every job together on a machine with large capacity). On the flip side, the adversary can punish an algorithm for processing small batches by not sending any new jobs (as a large batch would have been more cost efficient here). We discuss why some simple heuristics to choose the machine type fail in Appendix B.

Agreeable deadlines were easy to handle because we can charge the cost of the Greedy algorithm to simple lower bounds on $\text{cost}(\text{OPT})$, and recall that the lower bounds were due to the nice structure of the intervals corresponding to the batches created by Greedy. For general deadlines, we are able to track a more complicated set of intervals corresponding to batches in order to decide which machine type to use at the deadline of an uncompleted job. Moreover, this complicated interval structure implies we need a more careful lower bound on $\text{cost}(\text{OPT})$. Specifically, our algorithm tracks a set of nested intervals $[r_{j^*}, \tau] = I_0 \subseteq I_1 \subseteq \dots \subseteq I_k$, all with right endpoint τ , where for every $\ell \in \{0, \dots, k\}$, a batch of type ℓ was already executed in interval I_ℓ . Eventually, our method of constructing this sequence of nested intervals produces an interval I_{ℓ^*} that contains no time slot where a batch of type ℓ^* was executed. This indicates that our algorithm should create a batch of type ℓ^* at time τ .

3 Proof of Theorem 1 when $p = 1$

In the case where $p = 1$, we prove a lower bound on $\text{cost}(\text{OPT})$ in Subsection 3.2, and then use it to analyze our algorithm that proves Theorem 1 when $p = 1$ in Subsection 3.3. We begin with some additional preliminaries. The straightforward proofs of the propositions in the next subsection can be found in Appendix A.1 under Propositions 16, 18, and 19.

3.1 Additional preliminaries for Section 3

We show in the following proposition that by a standard bucketing argument, we can assume that all machine costs are powers of 2, with a loss of only 2 in the competitive ratio.

Proposition 2 *With only losing a factor of 2 in the competitive ratio, we can assume in online unit-length busy time scheduling on heterogeneous machines that for all $k \in \mathbb{Z}_{\geq 0}$, $c_k = 2^p$ for some $p \in \mathbb{Z}_{\geq 0}$, and $c_0 = 1$.*

Assumption 3 *For $k \in \mathbb{Z}_{\geq 0}$, a machine of type k has cost $c_k = 2^k$.*

This assumption follows with only a factor 2 loss in the competitive ratio because Proposition 2 allows us to assume that all machine costs are powers of 2, and if a machine of cost 2^k is not present for some $k \in \mathbb{Z}$, one could just use $2^{k-k'}$ machines of cost $2^{k'}$, for k' the largest type less than k that

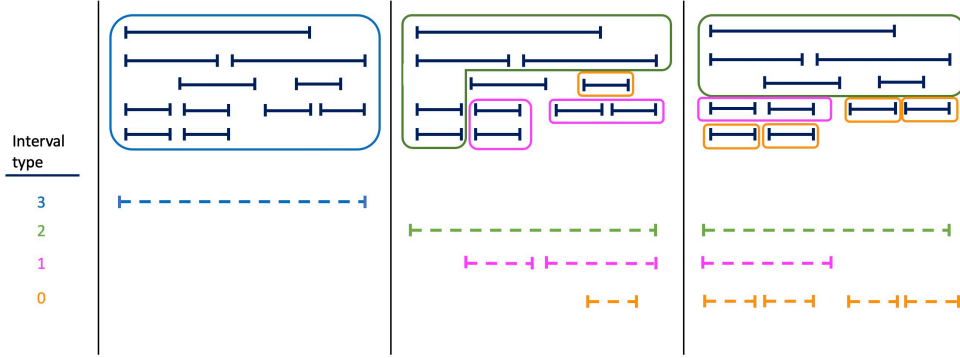


Figure 2: There are 11 jobs, each of which is depicted by a solid black line based on its release time and deadline. The three different frames depict possible valid interval assignments for the jobs. The dashed lines of different colors represent intervals of different types in each frame. There are four types of machines, with costs 2^k for $k \in \{0, 1, 2, 3\}$ and with capacities $\{2, 5, 11, 22\}$. Each interval assignment implies a different lower bound for $\text{cost}(\text{OPT})$, as in Lemma 7. We obtain better lower bounds moving from left-to-right in the frames, with bounds 2, $9/4$, and $5/2$, respectively. Note that an optimal assignment here has cost 5, with 3 batches with type 0 and 1 batch of type 1.

is present. Assumption 3 is useful in discussing the cost-per-job of the different machine types, as well as in upper bounding the number of distinct machine types.

Proposition 4 *Given Assumption 3, one can show that without loss of generality the cost-per-job is non-increasing in the machine types, i.e., $c_0/B_0 \geq c_1/B_1 \geq \dots$*

Proposition 5 *Given Assumption 3, the number of distinct machine types is bounded, with $K \leq \log n$.*

It will be helpful to track intervals $I \subseteq [0, T]$, and often we associate a type and a set of jobs with interval I . We let t_I denote the associated type and $J_I \subseteq J$ the associated set of jobs.

3.2 A lower bound on $\text{cost}(\text{OPT})$

We consider the following definition, for which Figure 2 is an accompaniment.

Definition 6 *For a set of jobs J equipped with arrival times and deadlines in $[0, T]$, let an interval assignment \mathcal{A} be a family of tuples $\{(I, t_I, J_I)\}$ of a continuous interval $I \subseteq [0, T]$, a type t_I , and a set of jobs $J_I \subseteq J$. Let \mathcal{L} denote the multi-set of all intervals I with $(I, t_I, J_I) \in \mathcal{A}$ and let $I(j)$ be the interval that job j is assigned to. We define a valid interval assignment as one such that the following holds:*

1. When $t_I \geq 1$, then $|J_I| = B_{t_I-1}$, and when $t_I = 0$, then $|J_I| = 1$.
2. If job j is in J_I , then $[r_j, d_j] \subseteq I$.
3. For any two intervals $I, I' \in \mathcal{L}$, if $t_I = t_{I'}$, then I and I' are disjoint.
4. Every job is assigned to at most one J_I . If a job j is not assigned to any J_I , then we write $I(j) = \emptyset$.

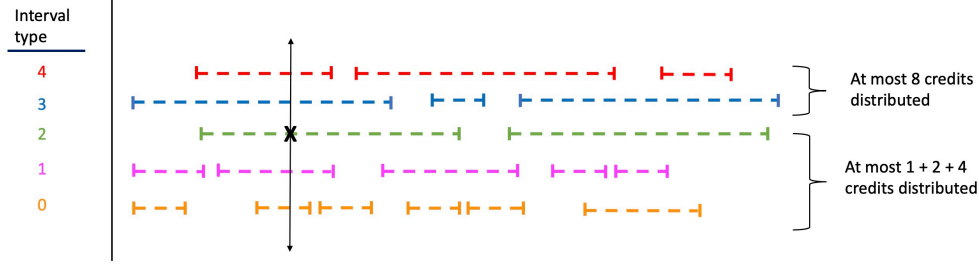


Figure 3: Distribution of credits from batch X with $k = 2$. The dashed lines of different colors represent intervals of different types in a valid interval assignment. For interval types 0, 1, and 2, X distributes $2^{t_{I(j)}}/|J_{I(j)}|$ credits per job, and there are at most $|J_{I(j)}|$ jobs in each interval type. For each job in interval types 3 and 4, X distributes $2^{k+1}/B_k$ credits, and there are at most B_k such jobs in interval types 3 and 4 combined.

A valid interval assignment represents a partition of jobs such that each job is assigned to at most one interval which fully contains the job's feasible region. Each interval is assigned a type such that no two intervals of the same type overlap. Furthermore, an interval cannot be assigned fewer jobs than its type allows. Valid interval assignments help us track how many jobs are available to be processed at any given time point. We use valid interval assignments to lower bound the cost of an optimal offline solution, as in the following lemma.

Lemma 7 *For a set of unit jobs J equipped with arrival times and deadlines in $[0, T]$ satisfying Assumption 3 and Proposition 4, let $\mathcal{A} = \{(I, t_I, J_I)\}$ be a valid interval assignment with \mathcal{L} the multi-set of all intervals I with $(I, t_I, J_I) \in \mathcal{A}$. Then,*

$$\text{cost}(\text{OPT}) \geq \frac{1}{4} \cdot \sum_{I \in \mathcal{L}} 2^{t_I}.$$

Proof At a high level, we will prove this by introducing at most $4 \cdot \text{cost}(\text{OPT})$ credits and then distributing them to all jobs j for which $I(j) \neq \emptyset$. We will do this in such a way that the sum of the credits distributed can be shown to be at least $\sum_{I \in \mathcal{L}} 2^{t_I}$, proving the inequality in the lemma statement.

Suppose that for each batch X of type k that OPT uses, we distribute credits to jobs in $J(X)$ in the following way:

- To each job j in $J(X)$ with $t_{I(j)} \leq k$ we give $2^{t_{I(j)}}/|J_{I(j)}|$ credits.
- To each job j in $J(X)$ with $t_{I(j)} > k$, we give $2^{k+1}/B_k$ credits.

Figure 3 depicts the distribution of credits.

We will first prove the following helpful claims regarding credits.

Claim 8 *When $t_{I(j)} > k$, we have that $\frac{2^{t_{I(j)}}}{|J_{I(j)}|} \leq \frac{2^{k+1}}{B_k}$.*

Proof We observe that:

$$\frac{2^{t_{I(j)}}}{|J_{I(j)}|} = 2 \cdot \frac{2^{t_{I(j)}-1}}{B_{t_{I(j)}-1}} \leq 2 \cdot \frac{2^k}{B_k} = \frac{2^{k+1}}{B_k}$$

where the inequality is because $t_{I(j)} > k$ and we assumed (based on Proposition 4) that larger machines are more cost efficient. ■

Claim 9 *The credits distributed by X of type k to jobs with $t_{I(j)} \leq k$ is at most 2^{k+1} .*

Proof We have required that all intervals with the same type must be disjoint, and therefore batch X can intersect at most one interval I of each type k' with $k' \leq k$. Moreover, since any job assigned to an interval has its arrival time and deadline inside that interval, all the jobs in $J(X)$ that are assigned to an interval of type k' are assigned to I above. From each such interval I , $J(X)$ contains at most $|J_I|$ jobs, and so the total amount of credits awarded by X to jobs associated with I is at most $2^{k'}$. The total amount of credits awarded by X to jobs with $t_{I(j)} \leq k$ is therefore upper bounded by $1 + 2 + \dots + 2^k \leq 2^{k+1}$. ■

Claim 10 *The credits distributed by X of type k to jobs with $t_{I(j)} > k$ is at most 2^{k+1} .*

Proof There are at most B_k jobs in batch X , and batch X distributes $2^{k+1}/B_k$ to each job with $t_{I(j)} > k$, so the total amount of credits awarded to jobs with $t_{I(j)} > k$ is upper bounded by 2^{k+1} . ■

We use these claims to prove the lemma statement. Let $\text{credit}(j)$ denote the amount of credit awarded to job j . Then

$$\sum_{I \in \mathcal{L}} 2^{t_I} = \sum_{I \in \mathcal{L}} \sum_{j \in J_I} \frac{2^{t_I}}{|J_I|} = \sum_{j: I(j) \neq \emptyset} \frac{2^{t_{I(j)}}}{|J_{I(j)}|} \leq \sum_{j: I(j) \neq \emptyset} \text{credit}(j) \leq 4 \cdot \text{cost}(\text{OPT}).$$

Note that the first sum is over \mathcal{L} , which is a multi-set, so in particular an interval might contribute to the sum multiple times. For the first inequality, we note that every interval I must intersect some batch used by OPT , otherwise OPT would not have picked up the jobs associated with I (recall that $|J_I| > 0$). Therefore, every job j associated with an interval must receive at least $2^{t_{I(j)}}/|J_{I(j)}|$ credits (by construction and by Claim 8).

For the second inequality, we combine Claims 9 and 10 to observe that the total credits distributed by each batch X of type k sent by OPT is at most 2^{k+2} , while the cost incurred by OPT for using batch X is 2^k . Therefore, the total amount of credit awarded for all batches used by OPT will not exceed $4 \cdot \text{cost}(\text{OPT})$. This finished the proof of Lemma 7. ■

The constant of $1/4$ given by Lemma 7 is tight, as we show in Appendix C.

3.3 Main algorithm and analysis

We use Algorithm 2 to prove Theorem 1 for when $p = 1$. The algorithm scans the time horizon until we reach a time slot τ where some job j is uncompleted and has its deadline. Algorithm 2 must decide which machine type to use, and this decision is made by the subroutine in Algorithm 3. A batch $X \in \mathcal{X}$ of type $t(X)$ executed at time $\tau(X)$ completes jobs $J(X)$, which are the (at most) $B_{t(X)}$ jobs with earliest deadline that are available at $\tau(X)$ and not yet completed by that time. As before, W is the set of jobs waiting to be scheduled.

Recall that in the proof of the lower bound, it was helpful to associate a type t_I and a set of jobs J_I to interval I . In the algorithm analysis, it will be helpful to associate a type $t(X)$, two sets

Algorithm 2 The main algorithm for unit jobs

Let $W \leftarrow \emptyset$, $\tau \leftarrow 0$, and $\mathcal{X} \leftarrow \emptyset$.

while $\tau \neq \text{NULL}$ **do**

▷ Let J' be the set of jobs with arrival time τ .

▷ Update $W \leftarrow W \cup J'$.

while $\exists j^* \in W$ with $d_{j^*} = \tau$ **do** // If many such j^* , choose arbitrarily from them

▷ Run Algorithm 3 with inputs τ , W , j^* , and \mathcal{X} , which outputs new batch X^* , equipped with $t(X^*)$, $J(X^*)$, $\tau(X^*)$, $\tilde{S}(X^*)$, $\tilde{I}(X^*)$.

▷ Update $\mathcal{X} \leftarrow \mathcal{X} \cup X^*$ and $W \leftarrow W \setminus J(X^*)$.

▷ Add $\{(\tilde{I}(X^*), t(X^*), \tilde{S}(X^*))\}$ to $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$.

end while

Increment $\tau \leftarrow \tau + 1$, or let $\tau = \text{NULL}$ if time has ended.

end while

Return \mathcal{X} .

of jobs $\tilde{S}(X)$ and $J(X)$, and an interval $\tilde{I}(X)$ to batch X . It is not surprising to see $t(X)$ as the associated type, but what might be surprising is to see two sets of associated jobs. While $J(X)$ are the jobs in batch X , $\tilde{S}(X)$ are the jobs we charge to X for our analysis, where jobs may be paying for a batch other than the one they are a part of.

Intuitively speaking, we employ a type of “pay-it-forward” philosophy. We charge the cost of one of the batches that the algorithm produces to the cost that **OPT** expends to execute one job in that batch (which we will refer to as the critical job) and the non-critical jobs in another batch. The *critical job* in batch X is the job $j^* \in J(X)$ with deadline at X 's execution time, i.e. $d_{j^*} = \tau(X)$; if there are many such jobs, we arbitrarily choose one to be the critical job of the batch. To determine $\tilde{S}(X)$, throughout the algorithm we will keep track of an interval associated with batch X , which we denote by $\tilde{I}(X)$. Note this is a slight abuse of notation from the lower bound subsection, where we let $I(j)$ denote the interval containing job j . Additionally, tracking $\tilde{S}(X)$ and $\tilde{I}(X)$ is not necessary for the algorithm (hence they are given the additional tilde in the notation), but for clarity in the analysis, we construct them in Algorithm 3 and keep them in Algorithm 2.

In order to use our lower bound from the previous subsection, we will use the batches to construct a valid interval assignment (as in Definition 6). A visualization of the intervals we construct is provided in Figure 4. Note that Algorithm 2 sets $t_{\tilde{I}(X)} = t(X)$ when it defines $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ as an interval assignment. We prove that the conditions for Definition 6 hold in Lemmas 11 and 13 and Proposition 12.

Lemma 11 *Algorithm 2 produces $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ that satisfy Conditions (1) and (2) of being a valid interval assignment (Definition 6).*

Proof Let $X \in \mathcal{X}$ and let $\tau = \tau(X)$. If $t(X) = 0$, then indeed there is only the critical job in $\tilde{S}(X)$, so $|\tilde{S}(X)| = 1$. Further, the critical job j^* completely determines the interval, with $\tilde{I}(X) = [r_{j^*}, d_{j^*}]$. Thus, Conditions (1) and (2) are satisfied when $t(X) = 0$.

Fix $t(X) = k > 0$. Batch X is created at the right endpoint of $\tilde{I}(X)$, recall that this endpoint is time τ . Consider the sequence of nested intervals that the algorithm built up to construct $\tilde{I}(X)$, i.e., I_0, I_1, \dots, I_k , where now $\tilde{I}(X) = I_k$. These are indeed nested intervals, since for all $j = 0, \dots, k-1$, I_j contains τ_j and I_{j+1} is defined by $I_{j+1} \leftarrow I_j \cup [\tau'_j, \tau_j]$. Also note that the right endpoint of $\tilde{I}(X)$ is τ . Associated with these intervals are batches $X_0, X_1, \dots, X_{k-1} \in \mathcal{X}$ that forced interval $\tilde{I}(X)$

Algorithm 3 Deciding the next batch for unit jobs

Input: time τ , waiting jobs W , critical job j^* with $d_{j^*} = \tau$, and batches \mathcal{X} , where every $X \in \mathcal{X}$ is equipped with $t(X)$, $J(X)$, $\tau(X)$, $\tilde{S}(X)$, $\tilde{I}(X)$.

▷ Let $k \leftarrow 0$.

▷ Let $I_0 \leftarrow [r_{j^*}, \tau]$.

while I_k contains a time slot $\tau_k := \tau(X_k)$ with $t(X_k) = k$ for some $X_k \in \mathcal{X}$ **do**

// Only the latest such τ_k is stored

▷ Let j_k be a job in $J(X_k)$ with the earliest arrival time. // Tie-break arbitrarily

▷ Let τ'_k be the arrival time of j_k .

▷ Set $I_{k+1} \leftarrow I_k \cup [\tau'_k, \tau_k]$.

▷ Increment $k \leftarrow k + 1$.

end while

Set $I^* \leftarrow I_k$ and $S^* \leftarrow \{j^*\}$.

if $k > 0$ **then**

Add all the non-critical jobs of X_{k-1} to S^* .

end if

▷ Let J^* be the B_k (or $|W|$ if $|W| < B_k$) jobs in W with earliest deadline. // Tie-break arbitrarily after putting in j^*

▷ Create batch X^* with type $t(X^*) \leftarrow k$, execution time $\tau(X^*) \leftarrow \tau$, and jobs $J(X^*) \leftarrow J^*$.

▷ Define $\tilde{I}(X^*) \leftarrow I^*$ and $\tilde{S}(X^*) \leftarrow S^*$.

Return X^* .

to be created (see Algorithm 3). Recall that τ'_ℓ is the arrival time of the earliest arriving job in $J(X_\ell)$. Since $\tau \in \tilde{I}(X)$, we have that for all $\hat{\tau} \in \tilde{I}(X)$, either $\hat{\tau} \in I_0$ or there exists some $0 < \ell < k$ such that $\hat{\tau} \in [\tau'_\ell, \tau_\ell]$.

We prove by induction on $\ell < k$ that X_ℓ is a full batch (that is, $|J(X_\ell)| = B_{t(X_\ell)}$), and all jobs in $J(X_\ell)$ have deadline no later than τ .

Base case: Here, $\ell = 0$. Let j^* be the critical job of X . Note that $\tau(X_0) = \tau_0 \in I_0$, the left endpoint of I_0 is r_{j^*} and the right endpoint of I_0 is $\tau = d_{j^*}$. By construction, $j^* \in W$ and $J(X_0) \cap W = \emptyset$ (recall that $\tau_0 \leq \tau$ and therefore the jobs in $J(X_0)$ were removed from W during the construction of X_0). Hence job j^* is not in batch X_0 . Since $\tau_0 \in [r_{j^*}, d_{j^*}]$, the job j^* already belonged to W when X_0 was constructed. Since $j^* \notin J(X_0)$, we can deduce that batch X_0 is full and each job in $J(X_0)$ has deadline at latest $\tau = d_{j^*}$ due to the EDF rule.

Induction step: Fix some integer $1 \leq \ell < k$. Assume that for all $0 \leq \ell' < \ell$, $X_{\ell'}$ is filled to its capacity, and all the jobs in batch $X_{\ell'}$ have deadline no later than τ . We will show that X_ℓ is filled to its capacity, and all the jobs in batch X_ℓ have deadline no later than τ . Since X_ℓ exists, either $\tau_\ell \in I_0$ (in which case the reasoning from the base case applies), or there exists some $\mu < \ell$ with $\tau_\ell \in [\tau'_\mu, \tau_\mu]$. The latter case is the interesting one, so we fix a smallest such μ . This means that $\tau_\ell < \tau_\mu$, so batch X_ℓ was created before batch X_μ .

Let job j_μ be any job arriving at time τ'_μ that is scheduled in batch X_μ at time τ_μ . Since j_μ is not in batch X_ℓ , it must be that *batch X_ℓ is full*, as j_μ was available to be sent with $\tau_\ell \in [\tau'_\mu, \tau_\mu]$. Again since batch X_ℓ was scheduled without job j_μ , the EDF rule implies that the jobs in $J(X_\ell)$ have deadlines no later than the deadline of j_μ . Thus, *all jobs in $J(X_\ell)$ have deadline no later than τ* , since $j_\mu \in J(X_\mu)$ has deadline $\leq \tau$ by the inductive hypothesis.

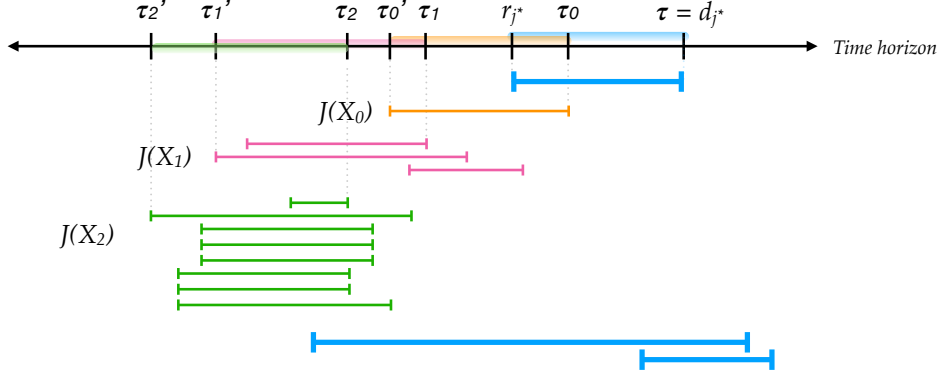


Figure 4: Constructing intervals. At time $\tau = d_{j^*}$, job j^* (highest bold interval) must be executed. Algorithm 3 decides which machine type to use for the next batch at τ by constructing intervals I_0, I_1, I_2, I_3 , which are indicated on the time horizon, where I_0 is the blue interval, I_1 is the union of the blue and orange intervals, I_2 is the union of the blue, orange, and pink intervals, and I_3 is the union of all colored intervals. These intervals are defined by taking the previous interval and extending it to the earliest arrival time of a job in the corresponding batch. Here Algorithm 3 decides to use a batch of type 3. Capacities of batches in this instance are $B_0 = 1, B_1 = 3, B_2 = 8, B_3 = 20$. Jobs with bold (arrival time, deadline) intervals are executed in the new batch, which can have less than B_3 jobs if these are all the jobs in the waiting list W .

This completes the inductive step.

Finally, let us look at the moment that the algorithm has found an interval I_k that does not contain a time executing a batch of type k ; we can assume that $k > 0$ as we already considered the case when $k = 0$. Recall that this will be the interval associated with the new batch, i.e., the algorithm sets $\tilde{I}(X) = I_k$. Note that $[\tau'_{k-1}, \tau] \subseteq I_k$ and recall that $\tilde{S}(X)$ is the critical job of batch X plus all the non-critical jobs of batch X_{k-1} . Since batch X_{k-1} is full, $|\tilde{S}(X)| = B_{k-1}$ and so Condition (1) is satisfied. Moreover, every job in $J(X_{k-1})$ has arrival time at or after τ'_{k-1} and deadline at most τ , as proven above. The critical job of batch X also has arrival time and deadline inside I_k (since I_k contains I_0), and we conclude that Condition (2) is satisfied since $\tilde{I}(X) = I_k$. ■

Proposition 12 *Algorithm 2 produces batches \mathcal{X} equipped with $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ which satisfy Condition (3) of being a valid interval assignment (Definition 6).*

Proof Intervals of the same type cannot intersect; see the while loop in Algorithm 3. ■

Lemma 13 *Algorithm 2 produces batches \mathcal{X} equipped with $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ which satisfy Condition (4) of being a valid interval assignment (Definition 6).*

Proof Fix a non-critical job j_k that is in $J(X_k)$. If j_k were in more than one batch, it must have ended up in batches $\tilde{S}(X)$ and $\tilde{S}(X')$, where both X and X' are of type $k + 1$. But then, by Condition (3), these corresponding intervals $\tilde{I}(X)$ and $\tilde{I}(X')$ must be disjoint. These intervals cannot be disjoint, since by Condition (2) they both contain $[r_{j_k}, d_{j_k}]$. ■

Proof [*Proof of Theorem 1 for unit jobs*] Combining Lemmas 11 and 13 with Proposition 12, we see that Algorithm 2 schedules batches \mathcal{X} equipped with a valid interval assignment $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$. By Assumption 3, the costs of machines are all powers of 2, and recall we make this assumption with

only a factor 2 loss in the competitive ratio. Therefore, the cost of the batches in \mathcal{X} is $\sum_{X \in \mathcal{X}} 2^{t(X)}$. Applying Lemma 7 to $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$, we see that for $\mathcal{L} = \{\tilde{I}(X)\}_{X \in \mathcal{X}}$:

$$\text{cost}(\text{OPT}) \geq \frac{1}{2} \cdot \frac{1}{4} \cdot \sum_{I \in \mathcal{L}} 2^{t_I} = \frac{1}{8} \cdot \sum_{X \in \mathcal{X}} 2^{t(X)}.$$

Note the factor of 2 in the first inequality follows from Proposition 2. Therefore, the cost of \mathcal{X} is at most $8 \cdot \text{cost}(\text{OPT})$.

The running time of this algorithm is $O(n \log n)$. As in Algorithm 1, we sort J so that J' can be found in time $|J'|$. We use a min-heap with keys d_j , for elements $j \in W$, to store W . We use the sorted J to find the next time slot where $J' \neq \emptyset$ and compare it with the smallest key in W . Recall that $K \leq \log n$ is an upper-bound on the number of types of machines by Proposition 5. We also keep for each type $\ell \in \{0, \dots, K\}$ the time $\tau(X)$ and the earliest arrival time of a job in $J(X)$, where X is the latest batch of type ℓ . Using these, each critical job requires at most $K + 1$ iterations through the while loop of Algorithm 3 to choose the type of the batch which will contain this critical job. In time $O(\log n)$ per job, we can update the data structures. ■

4 Proof of Theorem 3

Here, we prove a lower bound of 2 for the competitive ratio of any deterministic online algorithm, assuming an all-powerful (also called *adaptive-offline* [BE98]) adversary. Our construction will also result in an agreeable instance of unit jobs, which implies that Algorithm 1 is an optimal deterministic algorithm for agreeable instances.

Proof [*Proof of Theorem 3*] Suppose that there are two machine types, which we call “small” and “large”. The small machines have cost 1 and capacity 1, whereas the large machines have cost M and capacity M^3 , where M is a large even integer. We call large and small batches the corresponding batches of jobs. Here, jobs are unit length. In this instance, the release time of every job will be odd and the deadline of every job will be even. Many jobs will have the same release time, but no two jobs will have the same deadline. The adversary’s only action will be to release groups of $M^3/2$ jobs, where those $M^3/2$ jobs arrive at some time τ and each released job has as deadline the earliest even time slot that does not already contain a job deadline (thereby adding $M^3/2$ unique deadlines to the instance). Note that this release structure will result in an agreeable instance.

At time 1, the adversary releases one group of $M^3/2$ jobs. If no large machine is used to execute any of the jobs from the current group, then no more jobs are released by the adversary. Otherwise, the adversary will release another group of $M^3/2$ jobs at the first odd time step after the algorithm uses a large machine. The adversary repeats this at most $M - 1$ more times, so that we have at most M such groups. We index the groups of jobs released by the adversary as β_1, \dots, β_M .

We now show why this instance implies a lower bound of 2 on the competitive ratio. First, consider the case where the algorithm uses at least $2M^2$ small machines in a row in some group. These batches would incur a cost of at least $2M^2$. One feasible schedule across all groups of jobs would be to have instead executed all of the jobs from each group in a large batch at the first deadline of a job from each group, which would have incurred a cost of at most M^2 total. This implies a competitive ratio of at least 2.

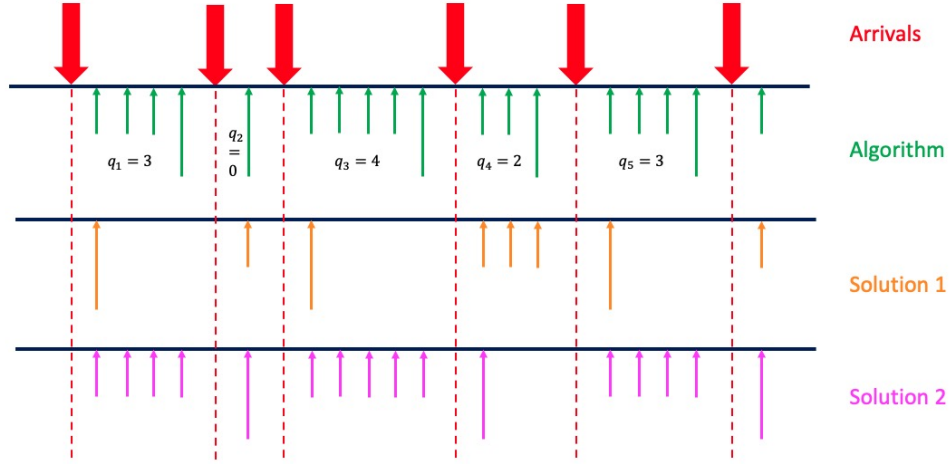


Figure 5: An illustration for the proof of Theorem 3. Each red thick arrow represents the arrival of $M^3/2$ jobs. The short thin arrows represent the scheduling of a batch on a small machine, and the long thin arrows represent the scheduling of a batch on a large machine.

See Figure 5 for an illustration of the rest of the proof. If an algorithm does not send $2M^2$ small batches in a row in any group, then it executes at least one large batch for every group. Note that in this case all M groups are released by the adversary. Let q_i represent the number of small batches scheduled by the algorithm between the releases of groups β_i and β_{i+1} for $i \in [M-1]$, and let q_M be the number of small batches scheduled by the algorithm after β_M was released. Then the cost incurred by the algorithm is $M^2 + \sum_{i=1}^M q_i$.

We consider two other possible solutions to this instance (see Solution 1 and Solution 2 in Figure 5). One possible solution is to schedule a large batch at the first deadline, then small batches at the next $q_2 + 1$ deadlines, then a large batch at the next deadline, then small batches at the next $q_4 + 1$ deadlines, and so on, ending with a large batch to clear away any remaining jobs. This solution is feasible as we can map any job executed by the algorithm to a batch in the solution, as follows. Any job executed in a small batch in an even group in the algorithm is also executed by a small batch in this solution. The job with the earliest deadline that is in a large batch in every even batch is also in a small batch. Every other job executed by the algorithm is in a large batch. Since the algorithm scheduled all jobs, we know this solution scheduled all jobs as well. The cost of this solution is at most $(M/2 + 1)M + \sum_{i=1}^{M/2} (1 + q_{2i}) = (M + 3)(M/2) + \sum_{i=1}^{M/2} q_{2i}$.

Another possible solution is to use small batches at the first $q_1 + 1$ deadlines, then a large batch at the next deadline, then small batches at the next $q_3 + 1$ deadlines, and so on, ending with a large batch to clear away any remaining jobs. We can again show this solution is feasible by mapping jobs carried by the algorithm to jobs carried by this solution in a similar fashion. The cost of this solution is at most $(M/2 + 1)M + \sum_{i=1}^{M/2} (1 + q_{2i-1}) = (M + 3)(M/2) + \sum_{i=1}^{M/2} q_{2i-1}$. The sum of the costs for these two possible solutions is at most $(1 + 3/M)$ times the cost of the online algorithm. This then implies that the solution with the lower cost of the two is at most $(1 + 3/M)/2$ times the cost of the online algorithm. Letting M go to infinity completes the proof of a lower bound of 2 for the competitive ratio of any deterministic online algorithm in the all-powerful adversary model. ■

5 Conclusion and Future Directions

We study online busy time scheduling with flexible, uniform-length jobs on heterogeneous machines. For the general case, we design an $8(2p - 1)/p$ -competitive algorithm. When jobs have unit length and agreeable deadlines, a simpler algorithm is 2-competitive, which is tight. We focused on jobs that have uniform processing time and resource requirement⁵, which could be relaxed in future work. Indeed, Ren and Tang [RT20] consider inflexible jobs with different heights. Sarpatwar et al. [SSS23] consider preemptively scheduling flexible jobs with arbitrary processing times and heights on homogeneous machines; in the online setting, there is a $3/2$ lower bound on the competitive ratio of any online algorithm that knows a job’s characteristics at arrival time, even assuming agreeable deadlines.

Generalizing our setting to allow non-uniform processing lengths or non-uniform heights introduces significant complexity into the problem. If jobs have different processing times, then we are no longer able to only schedule jobs $p - 1$ time steps before deadlines, and would need to reexamine our earliest deadline first heuristic. For heterogeneous machines, if jobs have unit length and agreeable deadlines but are allowed to have a resource requirement, then we can still apply our Greedy algorithm (Algorithm 1) and get a competitive ratio of 2, but this may not be a polynomial-time algorithm—for example, if jobs have heights and a machine can only process a set of jobs of total height at most 1 (homogeneous machines), then `GetOptimalBatches()` must solve a bin-packing problem. If `GetOptimalBatches()` is an α -approximation algorithm, we get a competitive ratio of 2α . When deadlines are not agreeable and jobs have heights, we do not know how to extend our 8-competitive algorithm (Algorithm 2), even when jobs are unit-length. We can no longer assume that the machine with the largest capacity is the most cost-efficient, as jobs may not fit on smaller machines.

Acknowledgements

We would like to thank an anonymous referee for suggesting that our Algorithm 2 can be adapted to jobs of uniform processing time instead of just unit; indeed this was true.

⁵Recall that in the literature, the resource requirement is also known as the height of a job.

References

- [AB03] Mansoor Alicherry and Randeep Bhatia. Line system design and a generalized coloring problem. In *European Symposium on Algorithms*, pages 19–30. Springer, 2003.
- [AGKK20] Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, pages 2758–2769. SIAM, 2020.
- [AIS08] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. *SIAM Journal on Computing*, 37(5):1499–1516, 2008.
- [AV19] Yossi Azar and Danny Vainstein. Tight bounds for clairvoyant dynamic bin packing. *ACM Transactions on Parallel Computing (TOPC)*, 6(3):1–21, 2019.
- [Bap06] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA*, volume 6, pages 364–367, 2006.
- [BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- [BFM⁺21] Niv Buchbinder, Yaron Fairstein, Konstantina Mellou, Ishai Menache, and Joseph Naor. Online virtual machine allocation with lifetime and load predictions. *ACM SIGMETRICS Performance Evaluation Review*, 49(1):9–10, 2021.
- [BJ95] Hans L. Bodlaender and Klaus Jansen. Restrictions of graph partition problems. part i. *Theoretical Computer Science*, 148(1):93–109, 1995.
- [BKL⁺08] Niv Buchbinder, Tracy Kimbrelt, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 952–961, 2008.
- [BNGK⁺09] Amotz Bar-Noy, Sudipto Guha, Yoav Katz, Joseph Naor, Baruch Schieber, and Hadas Shachnai. Throughput maximization of real-time scheduling with batching. *ACM Transactions on Algorithms (TALG)*, 5(2):1–17, 2009.
- [BP14] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. *SIAM Journal on Computing*, 43(5):1684–1698, 2014.
- [CCL⁺07] Ho-Leung Chan, Wun-Tat Chan, Tak Wah Lam, Lap-Kei Lee, Kin-Sum Mak, and Prudence WH Wong. Energy efficient online deadline scheduling. In *SODA*, volume 7, pages 795–804. Citeseer, 2007.
- [CFL⁺22] Nairen Cao, Jeremy T. Fineman, Shi Li, Julián Mestre, Katina Russell, and Seeun William Umboh. Brief announcement: Nested active-time scheduling. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 381–383. ACM, 2022.
- [CGK14] Jessica Chang, Harold N. Gabow, and Samir Khuller. A model for minimizing active processor time. *Algorithmica*, 70(3):368–405, 2014.

- [CKM14] Jessica Chang, Samir Khuller, and Koyel Mukherjee. LP rounding and combinatorial algorithms for minimizing active and busy time. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 118–127, 2014.
- [CN06] Julia Chuzhoy and Joseph Naor. Covering problems with hard capacities. *SIAM Journal on Computing*, 36(2):498–515, 2006.
- [CS15] Tim Carnes and David B Shmoys. Primal-dual schema for capacitated covering problems. *Mathematical Programming*, 153:289–308, 2015.
- [CW21] Gruia Călinescu and Kai Wang. A new LP rounding algorithm for the active time problem. *J. Sched.*, 24(5):543–552, 2021.
- [DKZ22] Sami Davies, Samir Khuller, and Shirley Zhang. Balancing flow time and energy consumption. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 369–380, 2022.
- [EEL09] Leah Epstein, Thomas Erlebach, and Asaf Levin. Online capacitated interval coloring. *SIAM Journal on Discrete Mathematics*, 23(2):822–841, 2009.
- [ELR⁺08] Guy Even, Retsef Levi, Dror Rawitz, Baruch Schieber, Shimon Shahar, and Maxim Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Transactions on Algorithms (TALG)*, 4(3):1–17, 2008.
- [FJQS08] Gerd Finke, Vincent Jost, Maurice Queyranne, and András Sebő. Batch processing with interval graph compatibilities between tasks. *Discrete Applied Mathematics*, 156(5):556–568, 2008.
- [GIK02] Daya Ram Gaur, Toshihide Ibaraki, and Ramesh Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *Journal of Algorithms*, 43(1):138–152, 2002.
- [HM91] Refael Hassin and Nimrod Megiddo. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30(1):29–42, 1991.
- [IM13] Sungjin Im and Benjamin Moseley. Online batch scheduling for flow objectives. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 102–104, 2013.
- [IP05] Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM Sigact News*, 36(2):63–76, 2005.
- [ISG07] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41–es, 2007.
- [KK17] Frederic Koehler and Samir Khuller. Busy time scheduling on a bounded number of machines. In *Workshop on Algorithms and Data Structures*, pages 521–532. Springer, 2017.
- [KK18] Saurabh Kumar and Samir Khuller. Brief announcement: A greedy 2 approximation for the active time problem. In Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 347–349. ACM, 2018.

- [KR05] Vijay Kumar and Atri Rudra. Approximation algorithms for wavelength assignment. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science: 25th International Conference, Hyderabad, India, December 15-18, 2005. Proceedings 25*, pages 152–163. Springer, 2005.
- [KSST10] Rohit Khandekar, Baruch Schieber, Hadas Shachnai, and Tami Tamir. Minimizing busy time in multiple machine real-time scheduling. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.
- [LT21] Mozhengfu Liu and Xueyan Tang. Analysis of busy-time scheduling on heterogeneous machines. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 340–350, 2021.
- [LTC14] Yusen Li, Xueyan Tang, and Wentong Cai. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 2–11, 2014.
- [LWAT12] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. *IEEE/ACM Transactions on Networking*, 21(5):1378–1391, 2012.
- [Mey05] Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*, pages 274–282. IEEE, 2005.
- [RT17] Runtian Ren and Xueyan Tang. Online flexible job scheduling for minimum span. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 55–66, 2017.
- [RT20] Runtian Ren and Xueyan Tang. Busy-time scheduling on heterogeneous machines. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 306–315, 2020.
- [RTL16] Runtian Ren, Xueyan Tang, Yusen Li, and Wentong Cai. Competitiveness of dynamic bin packing for online cloud server allocation. *IEEE/ACM Transactions on Networking*, 25(3):1324–1331, 2016.
- [SK12] Barna Saha and Samir Khuller. Set cover revisited: Hypergraph cover with hard capacities. In *International Colloquium on Automata, Languages, and Programming*, pages 762–773. Springer, 2012.
- [SSS23] Kanthi Sarpatwar, Baruch Schieber, and Hadas Shachnai. The preemptive resource allocation problem. *Journal of Scheduling*, pages 1–16, 2023.
- [WTFY17] Qian Wang, Ji Tian, Ruyan Fu, and Xiangjuan Yao. Online algorithms for scheduling on batch processing machines with interval graph compatibilities between jobs. *Theoretical Computer Science*, 700:37–44, 2017.
- [WZ03] Peter Winkler and Lisa Zhang. Wavelength assignment and generalized interval graph coloring. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 830–831, 2003.

- [YDS95] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, pages 374–382. IEEE, 1995.
- [ZPX11] Guiqing Zhang, Chung Keung Poon, and Yinfeng Xu. The ski-rental problem with multiple discount options. *Information Processing Letters*, 111(18):903–906, 2011.

A Proof of Theorem 1

We prove a lower bound on $\text{cost}(\text{OPT})$ in Subsection A.2, and then use it to analyze our algorithm that proves Theorem 1 in Subsection A.3.

First, we begin with some additional preliminaries. Otherwise, we use the same notation as in the main body.

A.1 Additional preliminaries for Section A

When we say that a machine is running in the interval $[\alpha, \beta]$, we mean that the machine is running at all time steps in the inclusive interval. For example, in $[\alpha, \alpha]$, the time slot α is active, and in $[\alpha, \alpha + 1]$ time slots α and $\alpha + 1$ are active. Thus the cost of running a machine of type k for all time steps in the interval $[\alpha, \beta]$ is $c_k \cdot (\beta - \alpha + 1)$. This cost is incurred regardless of how many jobs are running on the machine. Also, jobs can be assigned to a machine that is already processing other jobs, as long as there is spare capacity, though this may result in the machine running longer.

We first show that we can assume that all machines run for exactly $2p - 1$ time steps, and only lose a factor of $(2p - 1)/p$ in the competitive ratio. Intuitively, it is helpful to think about machines running for $2p - 1$ time steps because any two jobs which are batched together at any point must have a total processing time of at most $2p - 1$ timeslots.

Proposition 14 *When all jobs have uniform length p , we can assume that machines run for exactly $2p - 1$ time steps (thus each batch has $2p - 1$ time steps), while only losing a factor of $(2p - 1)/p$ in the competitive ratio.*

Proof

Indeed, suppose that a feasible solution runs a machine of type k at all time steps in the interval $[\alpha, \beta]$, for a cost of $c_k \cdot (\beta - \alpha + 1)$. Note it must be that $\beta \geq \alpha + p - 1$, or else no job could be scheduled on this machine. Let $\gamma = \lfloor (\beta - \alpha + 1)/p \rfloor$. For each $i = 0, \dots, \gamma - 1$, another feasible schedule runs γ machines of type k for the intervals $[\alpha + i \cdot p, \alpha + (i + 2)p - 2]$, for a total cost of $c_k \cdot \gamma \cdot (2p - 1)$. One can easily check that $\gamma \cdot (2p - 1) \leq \frac{2p-1}{p}(\beta - \alpha + 1)$ and that every job scheduled on the original machine fits on exactly one of the new machines. Moreover, at no time is a new machine over capacity, as otherwise the original schedule must have been over capacity at the midpoint of the processing time of that machine. ■

Based on the proposition, throughout this section, we make the following assumption.

Assumption 15 *All machines are run exactly for $2p - 1$ time slots. For a machine of type k , we only count c_k in the objective function, ignoring a multiplicative factor of $(2p - 1)$.*

The rest of this subsection is restating propositions and assumptions from the main body, though here we also include the proofs.

We show in the following proposition that we can assume that all machine costs are powers of 2, while losing only a factor of 2 in the competitive ratio.

Proposition 16 (Restating Proposition 2) *With only losing a factor of 2 in the competitive ratio, we can assume that for all $k \in \mathbb{Z}_{\geq 0}$, $c_k = 2^q$ for some $q \in \mathbb{Z}_{\geq 0}$, and $c_0 = 1$.*

Proof First, if $c_0 \neq 1$, scale all of the machine costs by $1/c_0$. Note this does not affect the optimality of any schedules. For each k , let q_k be the smallest integer such that $c_k \leq 2^{q_k}$. Then take $c'_k = 2^{q_k}$ to be the new machine cost for each k . Note that the cost of a schedule \mathcal{S} with machine costs c' is at most 2 times the cost of schedule \mathcal{S} with machine costs c . ■

We also make the following assumption with only an additional factor 2 loss.

Assumption 17 (Restating Assumption 3) For $k \in \mathbb{Z}_{\geq 0}$, a machine of type k has cost $c_k = 2^k$.

This assumption follows with only a factor 2 loss in the competitive ratio because Proposition 16 allows us to assume that all machine costs are powers of 2, and if a machine of cost 2^k is not present for some $k \in \mathbb{Z}$, one could just use $2^{k-k'}$ machines of cost $2^{k'}$, for k' the largest type less than k that is present.

This assumption is useful in discussing the cost-per-job of the different machine types, as seen in the following lemma.

Proposition 18 (Restating Proposition 4) Given Assumption 17, one can show that without loss of generality the cost-per-job is non-increasing in the machine types, i.e., $c_0/B_0 \geq c_1/B_1 \geq \dots$

Proof Suppose we are given machine types satisfying Assumption 17. If ever there existed k with $c_k/B_k < c_{k+1}/B_{k+1}$, then there always exists an optimal schedule that does not use any machines of type $k+1$. This is because any schedule that used machines of type $k+1$ can replace each machine of type $k+1$ with 2 machines of type k , since by Assumption 3 machine costs are powers of 2. ■

Assumption 17 allows us to upper-bound K , the number of distinct machine types.

Proposition 19 (Restating Proposition 5) Given Assumption 17, the number of distinct machine types is bounded, with $K \leq \log n$.

Proof By Assumption 17, a machine of type k has cost $c_k = 2^k$, and by Proposition 18, $c_k/B_k \geq c_{k+1}/B_{k+1}$. Substituting in for the costs and rearranging, we see that $B_{k+1} \geq 2B_k$. The batch sizes are geometrically increasing by at least a factor of 2. Since there are at most n jobs, the highest capacity machine that is relevant to the instance processes a batch of size at most n . It follows that $K \leq \log n$. ■

A.2 A lower bound on cost(OPT)

We consider the following definition, for which Figure 2 is an accompaniment.

Definition 20 For a set of jobs J equipped with arrival times and deadlines in $[0, T]$, let an interval assignment \mathcal{A} be a family of tuples $\{(I, t_I, J_I)\}$ of a continuous interval $I \subseteq [0, T]$, a type t_I , and a set of jobs $J_I \subseteq J$. Let \mathcal{L} denote the multi-set of all intervals I with $(I, t_I, J_I) \in \mathcal{A}$ and let $I(j)$ be the interval that job j is assigned to. We define a valid interval assignment as one such that the following holds:

1. When $t_I \geq 1$, then $|J_I| = B_{t_I-1}$, and when $t_I = 0$, then $|J_I| = 1$.
2. If job j is in J_I , then $[r_j, d_j] \subseteq I$.

3. For any two intervals $I, I' \in \mathcal{L}$, if $t_I = t_{I'}$, then I and I' are disjoint.
4. Every job is assigned to at most one J_I . If a job j is not assigned to any J_I , then we write $I(j) = \emptyset$.

We use valid interval assignments to lower bound the cost of an optimal offline solution, as in the following lemma. We note that the only differences from the main body in the proof for the following lemma are in Claim 23.

Lemma 21 *For a set of jobs J equipped with arrival times and deadlines in $[0, T]$ satisfying Assumption 17 and Proposition 18, let $\mathcal{A} = \{(I, t_I, J_I)\}$ be a valid interval assignment with \mathcal{L} the multi-set of all intervals I with $(I, t_I, J_I) \in \mathcal{A}$. Then,*

$$\text{cost}(\text{OPT}) \geq \frac{1}{4} \cdot \sum_{I \in \mathcal{L}} 2^{t_I}.$$

Proof At a high level, we will prove this by introducing at most $4 \cdot \text{cost}(\text{OPT})$ credits and then distributing them to all jobs j for which $I(j) \neq \emptyset$. We will do this in such a way that the sum of the credits distributed can be shown to be at least $\sum_{I \in \mathcal{L}} 2^{t_I}$, proving the inequality in the lemma statement.

Suppose that for each batch X of type k that OPT uses, we distribute credits to jobs in $J(X)$ in the following way:

- To each job j in $J(X)$ with $t_{I(j)} \leq k$ we give $2^{t_{I(j)}}/|J_{I(j)}|$ credits.
- To each job j in $J(X)$ with $t_{I(j)} > k$, we give $2^{k+1}/B_k$ credits.

Figure 3 depicts the distribution of credits.

We will first prove the following helpful claims regarding credits.

Claim 22 *When $t_{I(j)} > k$, the following holds:*

$$\frac{2^{t_{I(j)}}}{|J_{I(j)}|} \leq \frac{2^{k+1}}{B_k}.$$

Proof We observe that:

$$\frac{2^{t_{I(j)}}}{|J_{I(j)}|} = 2 \cdot \frac{2^{t_{I(j)}-1}}{B_{t_{I(j)}-1}} \leq 2 \cdot \frac{2^k}{B_k} = \frac{2^{k+1}}{B_k}$$

where the inequality is because $t_{I(j)} > k$ and we assumed (based on Proposition 4) that larger machines are more cost efficient. ■

Claim 23 *The credits distributed by X of type k to jobs with $t_{I(j)} \leq k$ is at most 2^{k+1} .*

Proof Let batch X use the time slots in the interval $[\alpha, \beta]$ and let $\tau = \alpha + (p - 1) = \beta - (p - 1)$ (note this equality is using Assumption 15). Any job j in $J(X)$ has $\tau \in [r_j, d_j]$. We have required that all intervals with the same type must be disjoint, and therefore τ can intersect at most one interval I of each type k' with $k' \leq k$. Moreover, since any job assigned to an interval has its arrival time and deadline inside that interval, all the jobs in $J(X)$ that are assigned to an interval of type k' are assigned to I above. From each such interval I , $J(X)$ contains at most $|J_I|$ jobs, and so the total amount of credits awarded by X to jobs associated with I is at most $2^{k'}$. The total amount of

credits awarded by X to jobs with $t_{I(j)} \leq k$ is therefore upper bounded by $1 + 2 + \dots + 2^k \leq 2^{k+1}$. ■

Claim 24 *The credits distributed by X of type k to jobs with $t_{I(j)} > k$ is at most 2^{k+1} .*

Proof There are at most B_k jobs in batch X , and batch X distributes $2^{k+1}/B_k$ to each job with $t_{I(j)} > k$, so the total amount of credits awarded to jobs with $t_{I(j)} > k$ is upper bounded by 2^{k+1} . ■

We use these claims to prove the lemma statement. Let $\text{credit}(j)$ denote the amount of credit awarded to job j . Then

$$\sum_{I \in \mathcal{L}} 2^{t_I} = \sum_{I \in \mathcal{L}} \sum_{j \in J_I} \frac{2^{t_I}}{|J_I|} = \sum_{j: I(j) \neq \emptyset} \frac{2^{t_{I(j)}}}{|J_{I(j)}|} \leq \sum_{j: I(j) \neq \emptyset} \text{credit}(j) \leq 4 \cdot \text{cost}(\text{OPT}).$$

Note that the first sum is over \mathcal{L} , which is a multi-set, so in particular an interval might contribute to the sum multiple times. For the first inequality, we note that every interval I must intersect some batch used by OPT , otherwise OPT would not have picked up the jobs associated with I (recall that $|J_I| > 0$). Therefore, every job j associated with an interval must receive at least $2^{t_{I(j)}}/|J_{I(j)}|$ credits (by construction and by Claim 22).

For the second inequality, we combine Claims 23 and 24 to observe that the total credits distributed by each batch X of type k sent by OPT is at most 2^{k+2} , while the cost incurred by OPT for using batch X is 2^k . Therefore, the total amount of credit awarded for all batches used by OPT will not exceed $4 \cdot \text{cost}(\text{OPT})$. This finished the proof of Lemma 21. ■

The constant of $1/4$ given by Lemma 21 is tight even for $p = 1$. See Appendix C.

A.3 Main algorithm and analysis

We use Algorithm 4 to prove Theorem 1. At any time during the algorithm's execution, there is at most one batch X^* (and hence at most one machine) that is open to processing more jobs if it has excess capacity. The algorithm scans the time horizon until we reach a time slot τ where either some job j^* is not assigned and has reached $\tau = d_{j^*} - p + 1$, or the batch X^* reaches the mid-point of its active interval. If there is a job j^* as above and X^* is not full (in other words, $|J(X^*)| < B_{t(X^*)}$, where $t(X)$ is the type of batch X), then assign j^* to batch X^* . If there is a job j^* as above and X^* is full, Algorithm 4 must decide which machine type to open and use for j^* , and this decision is made by the subroutine in Algorithm 5. If the batch X^* reaches the mid-point of its active interval, then we fill it up to capacity (or we fill it up with all the waiting jobs) using the EDF rule. As before, in the pseudocode, W is the set of unassigned jobs (jobs waiting to be scheduled).

Recall that in the proof of the lower bound, it was helpful to associate a type t_I and a set of jobs J_I to interval I . In the algorithm analysis, it will be helpful to associate a type $t(X)$, two sets of jobs $\tilde{S}(X)$ and $J(X)$, and an interval $\tilde{I}(X)$ to batch X . It is not surprising to see $t(X)$ as the associated type, but what might be surprising is to see two sets of associated jobs. While $J(X)$ are the jobs in batch X , $\tilde{S}(X)$ are the jobs we charge to X for our analysis, where jobs may be paying for a batch other than the one they are a part of.

Algorithm 4 The main algorithm

Let $W \leftarrow \emptyset$, $\tau \leftarrow 0$, $\mathcal{X} \leftarrow \emptyset$, and $X^* \leftarrow \text{undefined}$.

while $\tau \neq \text{NULL}$ **do**

- ▷ Let J' be the set of jobs with arrival time τ .
- ▷ Update $W \leftarrow W \cup J'$.
- while** $\exists j^* \in W$ with $d_{j^*} = \tau + p - 1$ **do** // If many such j^* , choose one arbitrarily
 - if** X^* is defined and $|J(X^*)| < B_{t(X^*)}$ **then**
 - ▷ $J(X^*) \leftarrow J(X^*) \cup \{j^*\}$, $W \leftarrow W \setminus \{j^*\}$.
 - else**
 - if** X^* is defined **then** // Here, the current batch X^* is full
 - ▷ Update $\mathcal{X} \leftarrow \mathcal{X} \cup X^*$.
 - ▷ Add $\{(\tilde{I}(X^*), t(X^*), \tilde{S}(X^*))\}$ to $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$.
 - end if**
 - ▷ Run Algorithm 5 with inputs τ , W , j^* , and \mathcal{X} , which outputs new batch X^* , equipped with $t(X^*)$, $J(X^*)$, $\tau(X^*)$, $\tilde{S}(X^*)$, $\tilde{I}(X^*)$.
- end if**
- end while**
- if** X^* is defined and $\tau = \tau(X^*) + p - 1$ **then**
 - while** $|J(X^*)| < B_{t(X^*)}$ and $|W| > 0$ **do**
 - ▷ Find the job $j \in W$ with earliest deadline.
 - ▷ Update $J(X^*) \leftarrow J(X^*) \cup \{j\}$, $W \leftarrow W \setminus \{j\}$.
 - end while**
 - ▷ Update $\mathcal{X} \leftarrow \mathcal{X} \cup X^*$
 - ▷ Add $\{(\tilde{I}(X^*), t(X^*), \tilde{S}(X^*))\}$ to $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$.
 - ▷ $X^* \leftarrow \text{Undefined}$.
- end if**
- ▷ Increment $\tau \leftarrow \tau + 1$, or let $\tau = \text{NULL}$ if time has ended.

end while

▷ **Return** \mathcal{X} .

Intuitively speaking, we employ a type of “pay-it-forward” philosophy. We charge the cost of one of the batches that the algorithm produces to the cost that OPT expends to execute one job in that batch (which we will refer to as the critical job) and the non-critical jobs in another batch. The *critical job* in batch X^* is the job that was first introduced in $J(X^*)$ when batch X^* is created in Algorithm 5. To determine $\tilde{S}(X^*)$, throughout the algorithm we will keep track of an interval associated with batch X^* , which we denote by $\tilde{I}(X^*)$. Note this is a slight abuse of notation from the lower bound subsection, where we let $I(j)$ denote the interval containing job j . Additionally, tracking $\tilde{S}(X^*)$ and $\tilde{I}(X^*)$ is not necessary for the algorithm (hence they are given the additional tilde in the notation), but for clarity in the analysis, we construct them in Algorithm 5 and keep them in Algorithm 4.

In order to use our lower bound from the previous subsection, we will use the batches to construct a valid interval assignment (as in Definition 20). Note that Algorithm 4 sets $t_{\tilde{I}(X)} = t(X)$ when it defines $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ as an interval assignment. We prove that the conditions for Definition 20 hold in Lemmas 25 and 27 and Proposition 26.

Lemma 25 *Algorithm 4 produces $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ that satisfy Conditions (1) and (2) of*

Algorithm 5 Deciding the next batch

Input: time τ , waiting jobs W , critical job j^* with $d_{j^*} = \tau + p - 1$, and batches \mathcal{X} , where every $X \in \mathcal{X}$ is equipped with $t(X)$, $J(X)$, $\tau(X)$, $\tilde{S}(X)$, $\tilde{I}(X)$.

- ▷ Let $k \leftarrow 0$.
- ▷ Let $I_0 \leftarrow [r_{j^*}, \tau + p - 1]$. // Note that $d_{j^*} = \tau + p - 1$
- while** I_k contains a time slot $\tau_k := \tau(X_k) + p - 1$ with $t(X_k) = k$ for some $X_k \in \mathcal{X}$ **do**
 - // Only the latest such τ_k is stored
 - ▷ Let j_k be a job in $J(X_k)$ with the earliest arrival time. // Tie-break arbitrarily
 - ▷ Let τ'_k be the arrival time of j_k .
 - ▷ Set $I_{k+1} \leftarrow I_k \cup [\tau'_k, \tau_k]$.
 - ▷ Increment $k \leftarrow k + 1$.
- end while**
- ▷ and $S^* \leftarrow \{j^*\}$.
- if** $k > 0$ **then**
 - Add all the non-critical jobs of X_{k-1} to S^* .
- end if**
- ▷ Create batch X^* with type $t(X^*) \leftarrow k$, start time $\tau(X^*) \leftarrow \tau$, and jobs $J(X^*) \leftarrow \{j^*\}$.
- ▷ Define $\tilde{I}(X^*) \leftarrow I_k$ and $\tilde{S}(X^*) \leftarrow S^*$.
- Return** X^* .

being a valid interval assignment (Definition 20).

Proof Let $X \in \mathcal{X}$ and let $\tau = \tau(X)$. If $t(X) = 0$, then indeed there is only the critical job in $\tilde{S}(X)$, so $|\tilde{S}(X)| = 1$. Further, the critical job j^* completely determines the interval, with $\tilde{I}(X) = [r_{j^*}, d_{j^*}]$. Thus, Conditions (1) and (2) are satisfied when $t(X) = 0$.

Fix $t(X) = k > 0$. The midpoint of batch X is the right endpoint of $\tilde{I}(X)$, and recall that this endpoint is time $\tau + p - 1$. Consider the sequence of nested intervals that the algorithm built up to construct $\tilde{I}(X)$, i.e., I_0, I_1, \dots, I_k , where now $\tilde{I}(X) = I_k$. These are indeed nested intervals, since for all $j = 0, \dots, k - 1$, I_j contains τ_j and I_{j+1} is defined by $I_{j+1} \leftarrow I_j \cup [\tau'_j, \tau_j]$. Associated with these intervals are batches $X_0, X_1, \dots, X_{k-1} \in \mathcal{X}$ that forced interval $\tilde{I}(X)$ to be created (see Algorithm 5). Let τ'_ℓ denote the arrival time of the earliest arriving job in $J(X_\ell)$. Since $\tau \in \tilde{I}(X)$, we have that for all $\hat{\tau} \in \tilde{I}(X)$, either $\hat{\tau} \in I_0$ or there exists some $0 < \ell < k$ such that $\hat{\tau} \in [\tau'_\ell, \tau_\ell]$.

We prove by induction on $\ell < k$ that X_ℓ is a full batch (that is, $|J(X_\ell)| = B_{t(X_\ell)}$), and all jobs in $J(X_\ell)$ have deadline no later than $\tau + p - 1$.

Base case: Here, $\ell = 0$. Let j^* be the critical job of X . Note that $\tau(X_0) + p - 1 = \tau_0 \in I_0$, the left endpoint of I_0 is r_{j^*} and the right endpoint of I_0 is $\tau + p - 1 = d_{j^*}$. The existence of X_0 implies that $\tau(X_0) \leq \tau$, and furthermore the construction of X_0 is completed at or before time τ . Also, $j^* \notin J(X_0)$. The jobs j added to $J(X_0)$ because they are due, i.e. the jobs j such that $\tau = d_j - p + 1$, certainly have deadline at most $\tau + p - 1$. The jobs added to $J(X_0)$ by EDF at time $\tau(X_0) + p - 1 = \tau_0$ have deadline at most d_{j^*} since otherwise j^* would be added to $J(X_0)$, as $\tau_0 \in [r_{j^*}, d_{j^*}]$. $J(X_0)$ must be full, as otherwise j^* would have been in batch X_0 .

Induction step: Fix some integer $1 \leq \ell < k$. Assume that for all $0 \leq \ell' < \ell$, $X_{\ell'}$ is filled to its capacity, and all the jobs in batch $X_{\ell'}$ have deadline no later than $\tau + p - 1$. We will show that X_ℓ is filled to its capacity, and all the jobs in batch X_ℓ have deadline no later than $\tau + p - 1$. Since X_ℓ

exists, either $\tau_\ell \in I_0$ (in which case the reasoning from the base case applies), or there exists some $\mu < \ell$ with $\tau_\ell \in [\tau'_\mu, \tau_\mu]$. The latter case is the interesting one, so we fix the smallest such μ . Note that this implies that $\tau_\ell < \tau_\mu$, which implies that batch X_ℓ was created and completely constructed before batch X_μ is created.

Let job j_μ be any job arriving at time τ'_μ that is scheduled in batch X_μ . Let j_μ^* be the critical job of batch X_μ (it is possible that $j_\mu = j_\mu^*$). Note that $j_\mu^* \notin J(X_\ell)$ and $j_\mu \notin J(X_\ell)$. The jobs added to $J(X_\ell)$ because they were due, i.e. $j \in J(X_\ell)$ with $\tau = d_j - p + 1$, have deadline at most $\tau(X_\mu) + p - 1$, since otherwise j_μ^* would have been added to $J(X_\ell)$. Note that $\tau_u = \tau(X_\mu) + p - 1 = d_{j_\mu^*}$ and that the induction hypothesis gives $d_{j_\mu^*} \leq \tau + p - 1$. The jobs added to $J(X_\ell)$ to fill the batch have deadline at most d_{j_μ} , as $\tau_\ell \in [r_{j_\mu}, d_{j_\mu}]$ and otherwise j_μ would have been in $J(X_\ell)$. The induction hypothesis gives us that $d_{j_\mu} \leq \tau + p - 1$, and if $J(X_\ell)$ were not full, then at time $\tau_\ell = \tau(X_\ell) + p - 1$ there are no jobs in W . This cannot happen since $\tau_\ell \geq \tau'_\mu = r_{j_\mu}$ and thus j_μ would be in W when the construction of X_ℓ is completed. This completes the inductive step.

Finally, let us look at the moment that the algorithm has found an interval I_k that does not contain a time τ_k ; we can assume that $k > 0$ as we already considered the case when $k = 0$. Recall that this will be the interval associated with the new batch, i.e., the algorithm sets $\tilde{I}(X) = I_k$. Note that $[\tau'_{k-1}, \tau + p - 1] \subseteq I_k$ and recall that $\tilde{S}(X)$ is the critical job of batch X plus all the non-critical jobs of batch X_{k-1} . Since batch X_{k-1} is full, $|\tilde{S}(X)| = B_{k-1}$ and so Condition (1) is satisfied. Moreover, every job in $J(X_{k-1})$ has arrival time at or after τ'_{k-1} and deadline at most $\tau + p - 1$, as proven above. The critical job of batch X also has arrival time and deadline inside I_k (since I_k contains I_0), and we conclude that Condition (2) is satisfied since $\tilde{I}(X) = I_k$. ■

Proposition 26 *Algorithm 4 produces batches \mathcal{X} equipped with $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ which satisfy Condition (3) of being a valid interval assignment (Definition 20).*

Proof Note that the right endpoint of $\tilde{I}(X)$ is $\tau(X) + p - 1$. By construction, intervals of the same type cannot intersect; see the while loop condition in Algorithm 5. ■

Lemma 27 *Algorithm 4 produces batches \mathcal{X} equipped with $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$ which satisfy Condition (4) of being a valid interval assignment (Definition 6).*

Proof Fix a non-critical job j_k that is in $J(X_k)$. If j_k were in more than one batch, it must have ended up in batches $\tilde{S}(X)$ and $\tilde{S}(X')$, where both X and X' are of type $k + 1$. But then, by Condition (3), these corresponding intervals $\tilde{I}(X)$ and $\tilde{I}(X')$ must be disjoint. These intervals cannot be disjoint, since by Condition (2) they are both required to contain $[r_{j_k}, d_{j_k}]$. ■

Proof [Proof of Theorem 1] Combining Lemmas 25 and 27 with Proposition 26, we see that Algorithm 4 schedules batches \mathcal{X} with a valid interval assignment $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$. By Assumption 17, the costs of machines are all powers of 2, and recall we make this assumption with only a factor 2 loss in the competitive ratio. The cost of the batches in \mathcal{X} is

$$\sum_{X \in \mathcal{X}} 2^{t(X)}.$$

Applying Lemma 21 to $\{(\tilde{I}(X), t(X), \tilde{S}(X))\}_{X \in \mathcal{X}}$, we see that for $\mathcal{L} = \{\tilde{I}(X)\}_{X \in \mathcal{X}}$:

$$\text{cost}(\text{OPT}) \geq \frac{p}{2p-1} \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \sum_{I \in \mathcal{L}} 2^{t_I} = \frac{p}{8(2p-1)} \cdot \sum_{X \in \mathcal{X}} 2^{t(X)}.$$

Note the factor of 2 in the first inequality follows from Proposition 16 and the factor of $(2p - 1)/p$ follows from Proposition 14. Therefore, the cost of \mathcal{X} is at most $(8(2p - 1)/p) \cdot \text{cost}(\text{OPT})$.

The running time of this algorithm is $O(n \log n)$. As in Algorithm 1, we sort J so that J' can be found in time $|J'|$. We use a min-heap with keys $d_j - p + 1$, for elements $j \in W$, to store W . We use the sorted J to find the next time slot where $J' \neq \emptyset$ and compare it with the smallest key in W and with $\tau(X^*) + p - 1$. Recall that $K \leq \log n$ is an upper-bound on the number of types of machines by Proposition 19. We also keep for each type $\ell \in \{0, \dots, K\}$ the time $\tau(X)$ and the earliest arrival time of a job in $J(X)$, where X is the latest batch of type ℓ . Using these, each critical job requires at most $K + 1$ iterations through the while loop of Algorithm 5 to choose the type of the batch which will contain this critical job. In time $O(\log n)$ per job, we can update the data structures. ■

B Discussion on Other Algorithms

We investigated several simpler algorithmic approaches for the general (non-agreeable) model, and observe that many natural algorithms do not obtain constant-factor competitive ratios. In this section, we highlight several of these algorithms and give intuition for why they fail, even just for unit jobs. For each of the below examples, we assume that we have machine types $\{0, \dots, K\}$ with the capacity of each machine type ℓ being 10^ℓ and the cost of each machine ℓ being 2^ℓ . We also assume all jobs have unit length. Here, K is an even integer that we will make large to obtain large competitive ratios.

Greedy Consider Algorithm 1, which schedules all jobs that are currently present at each time step where some job has a deadline. We showed in Section 2 that this algorithm works well when the input jobs have agreeable deadlines. In the general setting, the main problem with this algorithm is that it might have been better to send the majority of the jobs at or nearer to their deadlines. Suppose that at each time slot $10^{K/2}$ jobs arrive, of which one has an immediate deadline (i.e., at the next time slot) and all others have the same late deadline. Suppose $10^{K/2}$ sets of these jobs arrive. Greedy would send every job at its arrival time, in batches of $10^{K/2}$. Meanwhile, OPT could use $10^{K/2}$ batches of type 0 to send each job with an immediate deadline and one batch of cost 2^K to clear all jobs with the late deadline. Then our algorithm incurs cost $10^{K/2} \cdot 2^{K/2}$, while OPT incurs cost $10^{K/2} + 2^K$. The competitive ratio would be $\Omega(2^{K/2})$.

Most cost-efficient Another algorithm is to use the machine with the best cost-per-job scheduled (after waiting for a deadline and using EDF). The same setup used to show that Greedy can perform poorly also suffices to show that this cost-efficient algorithm performs poorly. Using the same logic, we find that the cost-efficient algorithm incurs cost $10^{K/2} \cdot 2^{K/2}$, while OPT incurs cost $10^{K/2} + 2^K$. The competitive ratio would again be $\Omega(2^{K/2})$.

Lazy Since the Greedy algorithm was sending jobs too early, one might instead try a lazy approach in which the algorithm does the following: schedule the minimum number of jobs that have an immediate deadline, until there are enough jobs to fill the largest capacity machine, at which point use the largest capacity machine. One problem with this algorithm occurs when not enough jobs arrive to fill the largest machine. Suppose that $10^{K/2}$ jobs arrive at time 1, each with a different deadline. Lazy would send each job separately on a machine of type 0, while OPT would send all

jobs at the beginning on a machine of type $K/2$. Then our algorithm incurs cost $10^{K/2}$, while OPT incurs cost $2^{K/2}$. The competitive ratio would be $\Omega(5^{K/2})$.

Ramp-up We present one final simple algorithm we call the *ramp-up algorithm*. The ramp-up algorithm keeps track of an accumulated cost and always uses the largest machine type whose cost is at most the accumulated cost, or a machine of type 0 if no such machine exists. When the ramp-up algorithm clears all jobs, then it resets the accumulated cost to 0. One problem is when the algorithm keeps ramping up and clearing too soon, it fails in the manner of the Greedy algorithm.

The following example shows that the ramp-up algorithm cannot obtain a constant factor competitive ratio. Suppose that 2 jobs arrive at time 0 and groups of 10^ℓ jobs arrive at each succeeding time slot $2\ell+1$, for $1 \leq \ell < K/2$. When $\ell = K/2$, a group of $10^{K/2} - 1$ jobs arrive. In each group, one job has an immediate deadline and the rest have the same very late deadline. Then our algorithm clears all waiting jobs in $2 + K/2$ machines for a total cost of $1 + 1 + \dots + 2^{K/2} = 2^{1+K/2}$, and resets the counter. Suppose that this set of job arrivals happens $10^{(K/2)-1}$ times. Then our algorithm would have a total cost of $10^{(K/2)-1} \cdot 2^{1+K/2}$, while a solution of total cost of $(2 + K/2) \cdot 10^{(K/2)-1} + 2^K$ exists: use batches of type 0 for the “immediate” jobs from each group, and one batch of type K to schedule all the other jobs at a very late moment. The competitive ratio would be $\Omega(2^{K/2}/K)$.

Our Algorithm 2 does “ramp up” to use bigger and bigger machines if many jobs are waiting, but manages to avoid the issue of ramping up too fast; for example, for the kind of adversary provided in the previous paragraph, Algorithm 2 only uses machines of type 0 for each group.

C Tight Example for the Lower Bound

We show that the constant of $1/4$ given by Lemma 7 (and Lemma 21) is tight. Suppose we are given a set of unit jobs J equipped with arrival times and deadlines between $[0, T]$. Let $\{(I, t_I, J_I)\}$ be a valid interval assignment \mathcal{A} with \mathcal{L} the multi-set of all intervals I with $(I, t_I, J_I) \in \mathcal{A}$. For convenience, let

$$\sigma(\mathcal{A}) := \sum_{I \in \mathcal{L}} 2^{t_I}.$$

Lemma 28 *For any $\epsilon > 0$ there exists an instance and a valid interval assignment \mathcal{A} such that $\text{cost}(\text{OPT}) < (\frac{1}{4} + \epsilon) \sigma(\mathcal{A})$.*

Proof Let q be a large integer. Suppose that batches of type $k \in \{0, 1, \dots, q\}$ each have capacity 2^k and batches of type $k \in \{q+1, q+2, \dots\}$ each have capacity 2^{q+k} . All batches have cost 2^k .

For this example, for every I and every $j \in J_I$, $[r_j, d_j] = I$. We construct the following family of tuples. Figure 6 provides an illustration for $q = 3$. For $r \in \{1, 2, \dots, 2^q\}$, and $s \in \{0, 1, \dots, q+1\}$, we construct the tuple $\tau_r^s = (I, t_I, J_I)$ where $I = [2r, 2r+1]$, $t_I = s$, and J_I is a set of “new” jobs (a set disjoint from the other sets constructed so far) of the size required by t_I . That is, if $s = 0$, then $|J_I| = 1$, and if $s \in \{1, 2, \dots, q+1\}$, then $|J_I| = 2^{s-1}$. We also have a tuple $\tau' = (I, t_I, J_I)$ where $I = [0, 2^{q+1} + 2]$, $t_I = 2q + 2$, and J_I is a set of new jobs of the required size, that is $|J_I| = 2^{3q+1}$. These are all the jobs of this instance, and this is \mathcal{A} , the family of tuples of this example.

Then

$$\sigma(\mathcal{A}) = 2^q(2^0 + 2^1 + \dots + 2^{q+1}) + 2^{2q+2} = 2^q(2^{q+2} - 1) + 2^{q+2} = 2^{2q+3} - 2^q. \quad (2)$$

Now we construct a feasible schedule for this instance. For $r \in \{1, 2, \dots, 2^q\}$, we use a batch of type $q+1$ at time $2r+1$ to schedule all the jobs assigned to τ_r^s , for all $s = 0, 1, \dots, q+1$. There are

exactly $1 + 1 + 2 + 4 + \dots + 2^q = 2^{q+1}$ such jobs. We fill this batch with $2^{2q+1} - 2^{q+1}$ jobs assigned to τ' . After doing this for all $r \in \{1, 2, \dots, 2^q\}$, we have scheduled all the jobs assigned to τ_r^s for all r and all s , and we have scheduled $2^q(2^{2q+1} - 2^{q+1}) = 2^{3q+1} - 2^{2q+1}$ of the jobs assigned to τ' . We use another batch of type $q + 1$ to schedule the remaining 2^{2q+1} jobs that were assigned to τ' . The cost of this solution is:

$$(2^q + 1)2^{q+1} = 2^{2q+1} + 2^{q+1}.$$

By making q large enough we finish the proof of the lemma. ■

Therefore, in order to get a competitive ratio better than 4 in the setting where the jobs have arbitrary intervals and costs powers of 2, it would be necessary to come up with a new lower bound.

type k	c_k	B_k	$ J_I $ for $t_I = k$
0	$2^0 = 1$	$2^0 = 1$	1
1	$2^1 = 2$	$2^1 = 2$	$B_{1-1} = 2^{1-1} = 1$
2	$2^2 = 4$	$2^2 = 4$	$B_{2-1} = 2^{2-1} = 2$
3	$2^3 = 8$	$2^3 = 8$	$B_{3-1} = 2^{3-1} = 4$
4	$2^4 = 16$	$2^7 = 128$	$B_{4-1} = 2^{4-1} = 8$
5	$2^5 = 32$	$2^8 = 256$	$B_{5-1} = 2^{8-1} = 128$
6	$2^6 = 64$	$2^9 = 512$	$B_{6-1} = 2^{9-1} = 256$
7	$2^7 = 128$	$2^{10} = 1024$	$B_{7-1} = 2^{10-1} = 512$
8	$2^8 = 256$	$2^{11} = 2048$	$B_{8-1} = 2^{11-1} = 1024$

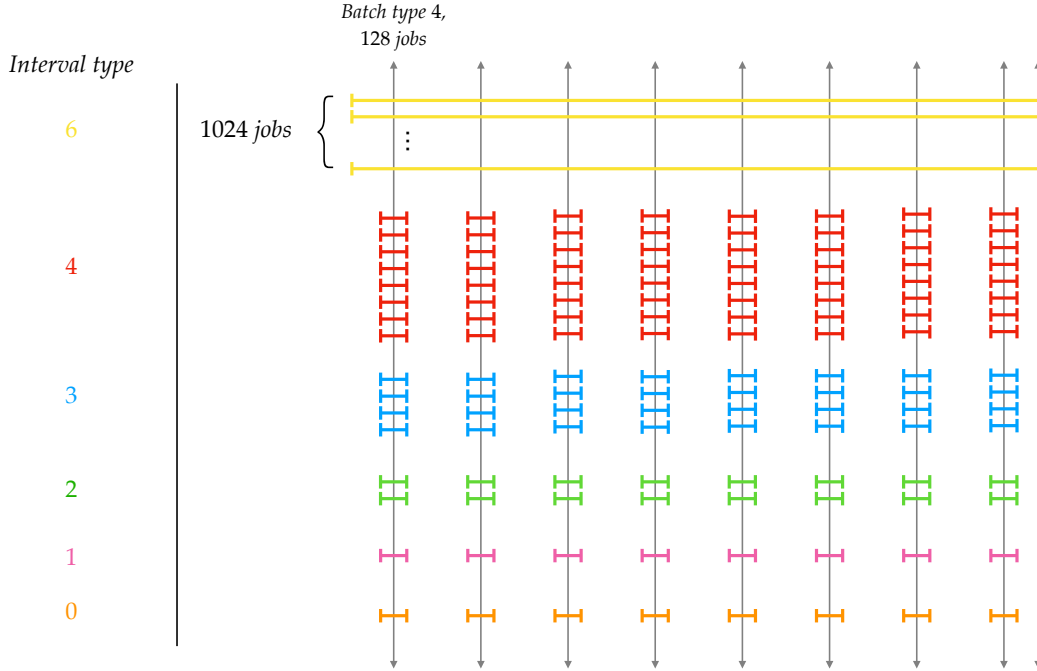


Figure 6: The construction used in Lemma 28 for $q = 3$. The multiset \mathcal{A} has one interval I of type 6, with associated J_I the 1024 yellow jobs, and for each $i = 0, \dots, 4$, eight intervals I of type i with J_I as depicted in the figure (8 red jobs for type 4, and so on). Here a solution of cost $(2^3+1)2^4 = 2^7+2^4$ exists: the 9 vertical lines. On the other hand, $\sigma(\mathcal{A}) = 2^3(1 + 2 + 4 + 8 + 16) + 2^8 = 2^9 - 2^3$.