Renting a Cloud

Barna Saha

AT&T Shannon Research Laboratory
180 Park Avenue, Florham Park, NJ, USA
barna@research.att.com

Abstract

We consider the problem of efficiently scheduling jobs on data centers to minimize the cost of renting machines from "the cloud." In the most basic cloud service model, cloud providers offer computers on demand from large pools installed in data centers. Clients pay for use at an hourly rate. In order to minimize cost, each client needs to decide on the number of machines to be rented and the duration of renting each machine. This suggests the following optimization problem, which we call Rent Minimization. There is a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs. Job j_i is released at time $r_i \geq 0$, has a deadline of d_i, and requires $p_i > 0$ contiguous processing time, $r_i, d_i, p_i \in \mathbb{R}$. The jobs need to be scheduled on identical parallel machines. Machines may be rented for any length of time; however, the cost of renting a machine for $\ell \geq 0$ time units is $\lceil \ell/D \rceil$ dollars, for some given large real D; in particular, one pays \$2 whether the machine is rented for $D+1$ or $2D$ time units. The goal is to schedule all the jobs in a way that minimizes the incurred rental cost.

In this paper, we develop offline and online algorithms for Rent Minimization problem. The algorithms achieve a constant factor approximation for the offline version and $O(\log \frac{p_{max}}{p_{min}})$ for the online version, where p_{max} and p_{min} are the maximum and minimum processing time of the jobs respectively. We also show that no deterministic online algorithm can achieve an approximation factor better than $\log_3 \frac{p_{max}}{p_{min}}$ within a constant factor. Both of these algorithms use the well-studied problem of Machine Minimization as a subroutine. Machine Minimization is a special case of Rent Minimization where $D = \max_i d_i$. In the process of solving the Rent Minimization problem, in this paper, we also develop the first online algorithm for Machine Minimization.

1998 ACM Subject Classification F.2 ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY, F.2.2 Nonnumerical Algorithms and Problems, F.1.2 Modes of Computation

Keywords and phrases Scheduling Algorithm, Online Algorithm, Approximation Algorithm

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Computing in "the cloud" has emerged as a popular way for companies to compute and to store data. The first commercial cloud service was Amazon Web Services's Elastic Compute Cloud (EC2). Since the appearance of EC2, many other companies have offered their own cloud services. Low maintenance overhead and the ability to pay only for what one uses have induced numerous clients to move their businesses onto the cloud. The clients rent machines from cloud vendors either in advance according to their projected job load or on demand. Payment is usually made at an hourly rate.

While obtaining adequate service, clients want to minimize rental cost. Motivated by this issue of reducing rental cost, we define Rent Minimization as follows. There is a set $J = \{j_1, j_2, \ldots, j_n\}$ of n jobs. Job j_i is released at time $r_i \geq 0$, has a deadline at time d_i, and requires p_i units of contiguous processing time. The jobs need to be scheduled on

Conference title on which this volume is based on.
Editors: Billy Editor, Bill Editors; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

identical parallel machines. Machines may be rented for any duration; however, the cost of renting a machine for $\ell \geq 0$ time units is $\$\lceil \ell/D \rceil$, for a given large real D; in particular, one pays \$2 whether the machine is open for $D + \epsilon$ or $2D$ time units. The goal is to schedule all the jobs in a way that minimizes the incurred rental cost. We call this problem Rent Minimization. To quote the Amazon pricing model, "*Pricing is per instance-hour consumed for each instance, from the time an instance is launched until it is terminated. Each partial instance-hour consumed will be billed as a full hour.*"–this is precisely the model used for defining Rent Minimization.

Rent Minimization is a generalization of the well-studied classical problem called Machine Minimization. The input to this problem is again a set of jobs each with a release time, deadline and processing time, and the goal is to use a minimum number of machines such that each job is allocated, between its release time and deadline, an interval (closed on the left and open on the right) of length equal to its processing time, on some machine. Garey and Johnson [9] show that for this case, deciding whether all the jobs can be scheduled on a single machine is already NP-hard. The problem admits a constant-factor approximation [5]; this paper improves upon the previous work of Chuzhoy, Guha, Khanna and Naor [6], which achieved an approximation factor of $O(\sqrt{\frac{\log n}{\log \log n}})$. (Taking an instance of Machine Minimization and setting $D = \max_{i=1}^{n} d_i$ converts the instance to one of Rent Minimization, for the cost of renting any machine, for any duration, is \$1.) There is also a discrete version of Machine Minimization, in which the discrete set of multiple intervals in which each job can be feasibly scheduled is given explicitly. The discrete version of the problem is markedly harder than the continuous version (single release time and deadline) and is known to be inapproximable beyond $\Omega(\log \log n)$ [7].

Our algorithm for the offline version of Rent Minimization is based on carefully classifying jobs based on processing times and difference between the release times and deadlines, so that by losing a constant factor either an algorithm for Machine Minimization can be used for them or there exists a combinatorial algorithm for scheduling the jobs. Therefore, we achieve an approximation ratio of $\Theta(\alpha)$ where α is the approximation factor for Machine Minimization. For the online version, the most nontrivial part is to design an online algorithm for Machine Minimization. To the best of our knowledge, no online algorithm for Machine Minimization was known before this work. Next, using the same classification of jobs, the online Rent Minimization problem can be solved.

Contributions

1. Following the pricing model of Amazon EC2, we define a new class of problems called Rent Minimization with the objective of minimizing the amount of money paid by cloud users while renting machineries from the cloud. We develop a constant factor offline algorithm for Rent Minimization. In the online version we provide an algorithm with approximation factor of $O(\log \frac{p_{max}}{p_{min}})$ which also matches the lower bound of any deterministic online algorithm within a constant factor.

2. We develop the first online algorithm for the well-studied problem of Machine Minimization. The algorithm achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$ which also matches the lower bound of any deterministic online algorithm for Machine Minimization within a constant factor.

Related Work

The *throughput maximization*, also known as *real time scheduling* problem, focuses on maximizing the number of jobs scheduled given a pre-fixed number of machines and can be thought of as the dual of Machine Minimization. In a more general setting, jobs may have weights and goal is to maximize the total weight of the scheduled jobs in given number of machines. The best known approximation factor for throughput maximization for the unweighted case is $\frac{e}{e-1} + \epsilon$, for any constant ϵ [8], whereas, for the weighted case nothing better than 2-approximation is known [2]. In the discrete setting, even for the unweighted case with only 2 feasible intervals for each job, the throughput maximization is known to be MAX-SNP hard. However, no such hardness result is known for the continuous case. The problem has also been studied in the online setting for unit length jobs and equal length jobs both in the single processor and multi-processor setting, where decision to schedule a job must be made at the time job arrives [3, 4]. The jobs may also have heights indicating the resource requirements and multiple jobs can be run in parallel in a single machine as long as the total width of the jobs at any time is not more than 1 [1, 8].

2 Offline Algorithm

Let the intervals of the form $[(r-1)D, rD)$, for positive integers r, be *machine intervals*. We assume that the earliest release time of the jobs in J is 0. We partition the job sets J into three subsets, $J = J_1 \sqcup J_2 \sqcup J_3$.

1. $J_1 = \{j \mid p(j) \geq \frac{D}{2}\}$. These are the big jobs.
2. $J_2 = \{j \mid j \notin J_1, \exists a \in \mathbb{N}, r(j) < aD, r(j) + p(j) > aD, d(j) - p(j) < aD\}$. No matter in which valid intervals we schedule these jobs, their scheduling intervals always contain the time aD.
3. $J_3 = J \setminus \{J_1 \cup J_2\}$.

Let *rent-OPT* denote the renting cost of an optimal solution for Rent Minimization (RM). We first show how to schedule jobs in J_1 and J_2 at rental cost $O(\textit{rent-OPT})$. We next consider the following special case of RM and denote it by SRM.

Each machine is open for exactly one machine interval. The entire time period, from the earliest release time to the latest deadline, can be partitioned into s machine intervals, where s is polynomial in n.

Let *Srent-OPT* denote the renting cost of an optimal solution for SRM.

For jobs in J_3, we show that any solution for RM can be reduced to a solution in SRM by losing only a constant factor. Therefore, a constant-factor approximation for SRM implies a constant-factor approximation algorithm for RM as well. We next further partition the jobs in J_3 into two categories, $J_3 = J_3^1 \sqcup J_3^2$.

▶ **Definition 1.** Let $H_r = [(r-1)D, rD)$ be the rth machine interval.

1. $J_3^1 = \{j \in J_3 \mid \exists r \in \mathbb{N}, r(j) \in H_r, d(j) \in H_r \cup H_{r+1}\}$.
2. $J_3^2 = \{j \in J_3 \mid \exists s \in \mathbb{N}, r(j) \in H_r, d(j) \in H_t, t \geq r+2\}$.

Scheduling jobs in J_1 and J_2. We simply assign a new machine to each job $j \in J_1$ and keep the machine open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals. We show by doing so the rental cost can increase at most by a factor of 3.

▶ **Lemma 2.** *The rental cost where each job $j \in J_1$ is assigned to a new machine that is open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals within a cost of $2\textit{rent-cost}$.*

Proof. We know $rent\text{-}OPT \geq \sum_{j \in J} \frac{p_j}{D} \geq \sum_{j \in J_1} \frac{p_j}{D}$. The rental cost for scheduling jobs in J_1 each to a new machine is $\sum_{j \in J_1} \lceil \frac{p_j}{D} \rceil \leq 2 \sum_{j \in J_1} \frac{p_j}{D} \leq 2(rent\text{-}OPT)$. ◀

For jobs in J_2, we again assign a new machine to each of them and rent the machine exactly for D time units. We show that by doing so the incurred rental cost for assigning jobs in J_2 is at most $rent\text{-}OPT$.

▶ **Lemma 3.** *The rental cost for assigning each job in J_2 to a new machine is at most $rent\text{-}OPT$.*

Proof. Consider any two jobs $j_1, j_2 \in J_2$ such that for some $a \in \mathbb{N}$, they have $r(j_1) < aD, r(j_1)+p(j_1) > aD, d(j_1)-p(j_1) < aD$ and $r(j_2) < aD, r(j_2)+p(j_2) > aD, d(j_2)-p(j_2) < aD$. Consider the time point aD. No matter where the jobs j_1 and j_2 are scheduled, the intervals in which they are scheduled contain the time point aD. Hence, j_1 and j_2 must be assigned to two different machines in OPT.

Fix a machine. If there are exactly l jobs in J_2 scheduled on that machine, then it is possible to find positive integers $a_1 < a_2 < \cdots < a_l$ such that for all i, one job $j_i \in J_2$ is scheduled during a half-open interval containing $a_i D$. The number of half-open intervals $[x, x + D)$ of length D needed to cover $\{a_1 D, a_2 D, ..., a_l D\}$ is l, since no such interval can cover two such points.

It follows that the rental cost associated with that one machine is at least l. Since we can sum over machines, it follows that $rent\text{-}OPT \geq |J_2|$. ◀

Hence, we get the following lemma.

▶ **Lemma 4.** *There exists a schedule in which each job $j \in J_1 \cup J_2$ is assigned to a new machine that is open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals and incurs a cost at most 3rent-cost.*

Proof. Follows from Lemma 2 and Lemma 3. ◀

Reduction to SRM

Making machine opening and closing time uniform.

We now consider only the jobs in J_3. We next show that if there are jobs J_3' in an optimal schedule that are allocated intervals overlapping two consecutive machine intervals, then by blowing up the rental cost at most by a factor of 5, all these jobs can be scheduled completely inside a single machine interval. Moreover, machines can be assumed to be rented exactly for one machine interval, and opened and shut down at integral multiples of D.

We consider the first machine interval to start at the earliest release time (counting it as 0), and the last machine interval to end at the smallest multiple of D greater than or equal to the latest deadline. If a machine is open from time a to b in OPT, we instead open it at time $a' \leq a$, $a' = D\lfloor \frac{a}{D} \rfloor$, which is the largest multiple of D which is less than or equal to a. Similarly, we can shut down the machine at time $b' \geq b$, $b' = D\lceil \frac{b}{D} \rceil$. A minute's thought will confirm that overall we can increase $rent\text{-}OPT$ at most by a factor of 2 in this process. We do not change the scheduling intervals of the jobs in J_3 in this process.

Now consider the subset $J_3' \subseteq J_3$ of jobs that are scheduled in OPT in intervals that overlap two machine intervals. Since these jobs have processing time less than $\frac{D}{2}$, they overlap at most two machine intervals. For any such job j, if $(a-1)D \leq r(j) \leq a(D)$, for some $a \in \mathbb{N}$, we either have $r(j) + p(j) \leq aD$, or $d(j) - p(j) \geq aD$; otherwise, j would belong to J_2. It is possible to allocate each of these jobs a new machine such that the job is scheduled entirely within one machine interval. The machine needs to be rented for just one machine interval. If there are l jobs of J_3' scheduled in a machine, then that machine is rented at least for l machine intervals, since each of these jobs contain a different boundary point of machine intervals. Therefore, $rent\text{-}OPT \geq |J_3'|$. Since each new machine that can

be rented to allocate J_3' needs to be open exactly for D time units, the rental cost can increase at most by $|J_3'|$. Hence, we get the following lemma.

▶ **Lemma 5.** *There exists a schedule of J_3 such that each job is scheduled entirely within a single machine interval and rental cost is at most $3rent\text{-}OPT$.*

Since all jobs are scheduled within a single machine interval, without loss of generality, we can assume that each machine is open exactly for D time units.

Hence at the end, we have a bunch of machines each opened and closed at multiples of D, each remaining open for exactly D time units, and which together schedule all the jobs in J_3.

Making the number s of machine intervals polynomial in n. Can be found in the full-version of the paper[1].

Hence, we have an instance of SRM.

We now proceed with jobs in J_3^1 and J_3^2 respectively.

Scheduling jobs in J_3^1 and J_3^2.

Let $J_{3,r}^1 = J_3^1 \cap \{j \mid r(j) \in H_r\}$. For each $J_{3,r}^1, r = 1, 2, ..., s$, we define an instance of Machine Minimization and invoke an algorithm for it such as [5] to schedule the jobs.

▶ **Lemma 6.** *The rental cost for assigning the jobs in J_3^1 is at most $2\alpha Srent\text{-}OPT$ where α is the approximation ratio of the best* Machine Minimization *algorithm.*

Proof. In an optimal solution for SRM the machine intervals used for scheduling jobs in $J_{3,r}^1$ are completely disjoint from jobs in $J_{3,r'}^1$ such that $|r-r'| \geq 2$. Therefore, the total rental cost paid by $J_{3,r}^1, r = 0, 2, 4, 6, ...$ is the sum of the rental cost paid by each $J_{3,r}^1, r = 0, 2, 4, 6, ...$ which is the total optimal cost for Machine Minimization for each of $J_{3,r}^1, r = 0, 2, 4, 6, ...$ Hence using the polynomial time algorithm for Machine Minimization with the best approximation factor, the total cost paid for $J_{3,r}^1, r = 0, 2, 4, 6, ...$ is at most α $Srent\text{-}OPT$. Similarly, the total rental cost paid for $J_{3,r}^1, r = 1, 3, 5, 7, ...$ is at most α $Srent\text{-}OPT$. Therefore, the overall cost paid is 2α $Srent\text{-}OPT$. ◀

We now consider scheduling jobs in J_3^2. The following lemma shows that every job in J_3^2 can be scheduled in machine intervals that do not contain its release time or deadline by losing a factor 3 in the approximation for SRM.

▶ **Lemma 7.** *There exists a schedule of jobs in J_3^2 with total rental cost at most $3Srent\text{-}OPT$ such that no job is scheduled in machine intervals containing its release time or deadline.*

Proof. Consider an optimal schedule. We perturb the optimal schedule for SRM in a way so that by paying at most $3Srent\text{-}OPT$ every job is scheduled in a machine interval that neither contains it release time nor deadline. Let H_r be one such interval where some jobs $J' \subseteq J_3^2$ are both released and scheduled and some jobs $J'' \subseteq J_3^2$ have their deadlines and are scheduled. There must be at least one such interval H_r for which either J' or J'' is non-empty, otherwise, the lemma trivially holds. If J' are scheduled in h_1 machines in H_r, then rent h_1 extra machines each for D time units in the following machine interval H_{r+1} and schedule the jobs in J' in these new h_1 machines in H_{r+1} exactly the way they were scheduled in H_r. That is if a job $j' \in J'$ was scheduled from time $[a, b)$ in H_r then schedule it at time $[a + D, b + D)$ in H_{r+1}. This is a feasible schedule for J' as the earliest of their deadlines can occur in machine interval H_{r+2} or higher. We charge the h_1 machines to machine interval H_{r+1}. If J'' are scheduled in h_2 machines in H_r, then rent h_2 extra

[1] `http://www.cs.umd.edu/~barna/renting-cloud.pdf`

machines each for D time units in the previous machine interval H_{r-1} and schedule the jobs in J'' in these new h_2 machines in H_{r-1} exactly the way they were scheduled in H_r. That is if a job $j'' \in J''$ is scheduled from time $[a, b)$ in H_r then schedule it at time $[a - D, b - D)$ in H_{r-1}. This is a feasible schedule for J'' as the latest release time can occur in machine interval H_{r-2} or lower. We charge h_2 machines to machine interval H_{r-1}.

Each machine interval is charged at most twice with at most the total number of machines in the machine intervals immediately before and after it. Therefore, the total rental cost can go up only by a factor of 3. ◀

We will schedule each job in J_3^2 only in machine intervals that do not contain its release time or deadline as follows. We treat each machine interval as a point on a line. Each job $j \in J_3^2$ can be scheduled in some consecutive machine intervals, denoted by an interval I_j. We pick minimum number of points to cover all the intervals and if I_j is covered by a point p_k then we assign job j to machine interval k. This problem of interval covering can be solved optimally, in fact by a single scan.

Algorithm A.

1. Scan the right end point of the intervals in non-decreasing order. If the first right end point is p_k, insert p_k in the solution and let it cover all the intervals that contain it. Remove all the intervals that are covered. Continue.
2. If machine intervals indexed $i_1, i_2, ..., i_l$ are selected in this procedure to cover jobs $J_{i_1}, J_{i_2}, ..., J_{i_l}$, then pack jobs J_{i_j} greedily using minimum number of machines each opened for D time units in machine interval H_{i_j}.

It can easily be verified that step 1 of the above algorithm returns minimum number of points. Both step 1 and 2 of Algorithm A can also be implemented easily in an online model where jobs arrive online.

▶ **Lemma 8.** *The rental cost for assigning the jobs in J_3^2 is at most $9Srent\text{-}OPT$.*

Proof. Let $Srent\text{-}OPT'$ denote the optimal rental cost for jobs in J_3^2 where no job is scheduled in machine interval containing its release time or deadline. Then from Lemma 7 $Srent\text{-}OPT' \leq 3Srent\text{-}OPT$.

The total rental cost paid is $\sum_{j=1}^{l} \lceil \frac{2p(J_{i_j})}{D} \rceil$ where $p(J_{i_j}) = \sum_{j \in J_{i_j}} p(j)$. The factor 2 comes from the fact that each opened machine may have at most $D/2$ empty slots. Now,

$$\begin{aligned} &\textstyle\sum_{j=1}^{l} \lceil \frac{2p(J_{i_j})}{D} \rceil \leq l + \sum_{j:p(J_{i_j}) \geq D} \lceil \frac{2p(J_{i_j})}{D} \rceil \\ &\leq Srent\text{-}OPT' + 2Srent\text{-}OPT' \leq 9Srent\text{-}OPT, \end{aligned}$$

where the second inequality comes from the fact that both l and $\sum_{j:p(J_{i_j}) \geq D} \lceil \frac{p(J_{i_j})}{D} \rceil$ serve as a lower bound on $Srent\text{-}OPT'$. ◀

▶ **Theorem 9.** *There exists a polynomial-time approximation algorithm for* RM *that incurs a rental cost of $(30 + 6\alpha)rent\text{-}OPT$, where $rent\text{-}OPT$ is the optimal rental cost and α is the approximation factor of the algorithm from [5] for* Machine Minimization.

Proof. From Lemma 4, Lemma 6 and Lemma 8, we have the total cost to be at most $3rent\text{-}OPT + (2\alpha + 9)Srent\text{-}OPT$ which is at most $30 + 6\alpha rent\text{-}OPT$ from Lemma 5. ◀

Note that we have not tried to optimize the constants which we believe can be reduced further.

3 Online Algorithm for Machine Minimization

In this section, we give an online algorithm for Machine Minimization problem. We first consider the case where jobs have nearly uniform processing time. We assume, each job has processing time in $[p, 2p]$, $p > 0$. We decompose the jobs into two sets based on the length of their time window, that is the duration between their release time and deadline.

▶ **Definition 10.** We call a job j "*slack*" if its release time $r(j)$ and deadline $d(j)$ satisfy $d(j) - r(j) \geq 8p$. Else, we call a job "*non-slack*".

▶ **Definition 11.** For a job j, any interval of length $p(j)$ in $[r(j), d(j))$ is referred to as a valid interval. The valid interval in which the job j is scheduled is referred to as its scheduling interval.

Suppose there exists a schedule of all the jobs in t machines. Clearly, using $2t$ machines, we can schedule the slack jobs in t machines and non-slack jobs separately in another t machines. Hence, with a loss of a factor of 2 in the approximation, we can consider scheduling of slack jobs and non-slack jobs separately.

3.1 Scheduling Slack Jobs

Let J_s denote the set of slack jobs. First, we show that any optimal schedule of J_s can be transformed into another schedule by blowing up the number of machines by at most a factor of 3 such that the scheduling interval of no job $j \in J_s$ intersects $[r(j), r(j) + 2p)$, or $[d(j) - 2p, d(j))$, that is, we wait at least $2p$ time units from the release time of each job before scheduling it and the job finishes $2p$ time units before its deadline.

▶ **Lemma 12.** *If t denotes the minimum number of machines in which all slack jobs can be scheduled, then there exists a schedule of all slack jobs using $3t$ machines, such that the scheduling interval of no job intersects the first $2p$ time units or the last $2p$ time units of its time window.*

Proof. Consider $J'_s \subseteq J_s$ be the set of jobs with their scheduling intervals intersecting the first $2p$ time units from their release time in an optimal solution. Let $J''_s \subseteq J_s \setminus J'_s$ be the set of jobs with their scheduling window intersecting the last $2p$ time units. We first schedule all the jobs in $J_s \setminus J'_s \cup J''_s$ in t machines as in the optimal solution. For each job $j \in J'_s$ that was initially scheduled on the rth, $r \in \{1, 2, \ldots, t\}$, machine in the optimal solution from time $[a, a + p(j))$, we schedule it in the $(t + r)$th machine from time $[a + 2p, a + 2p + p(j))$. First, clearly, $[a + 2p, a + 2p + p(j)) \cap [r(j), r(j) + 2p) = \emptyset$, since $a \geq r(j)$. Second, since $a < r(j) + 2p$ (by definition of J'_s), $p(j) \leq 2p$, we have $a + 2p + p(j) < r(j) + 6p$ but $d(j) \geq r(j) + 8p$. Hence, $a + 2p + p(j) \leq d(j) - 2p$. Therefore, $[a + 2p, a + 2p + p(j))$ is a valid interval for job j, and we have $r(j) + 2p \leq a + 2p < a + 2p + p(j) \leq d(j) - 2p$. Similarly, For each job $j \in J''_s$ that was initially scheduled on the rth, $r \in \{1, 2, \ldots, t\}$, machine in the optimal solution from time $[b, b + p(j))$, we schedule it in the $(t + 2r)$th machine from time $[b - 2p, b2p + p(j))$. Since $d(j) \geq b + p(j) \geq d(j) - 2p$, we have $d(j) - 2p \geq b - 2p + p(j) > b - 2p \geq d(j) - 6p \geq r(j) + 2p$. Therefore, $[b - 2p, b - 2p + p(j)]$ is a valid interval for job j, and we have $r(j) + 2p \leq b - 2p < b - 2p + p(j) \leq d(j) - 2p$. ◀

Define $r(j)_{new} = 2p\lceil r(j)/2p \rceil$, and $d(j)_{new} = 2p\lfloor d(j)/2p \rfloor$. From Lemma 12, we get the following corollary.

▶ **Corollary 13.** *If t denotes the minimum number of machines in which all slack jobs J_s can be scheduled, then there exists a schedule of J_s in $3t$ machines such that the scheduling window of any job $j \in J_s$ is contained in $[r(j)_{new}, d(j)_{new})$.*

Proof. From Lemma 12, we know there exists a schedule of jobs J_s in $3t$ machines such that, each job is scheduled after $2p$ time units of its release time $r(j)$, and hence after $r(j)_{new}$. Also, these jobs finish before $2p$ time units from their deadline and hence before $d(j)_{new}$, since $d(j) \geq 8p$. ◀

We can therefore assume without loss of generality that jobs are released and have deadlines at an integral multiples of $2p$.

We can further assume, with a loss of a factor of 2 in the approximation that the starting point of the scheduling interval of each job is an integral multiple of $2p$.

▶ **Lemma 14.** *If there exists a schedule of jobs in J_s with release time $r(j)_{new}$, deadline $d(j)_{new}$, and processing time $p(j) \in [p, 2p]$ in t' machines, then there exists a schedule of J_s in $2t'$ machines such that the starting point of the scheduling window of each job is an integral multiple of $2p$. Furthermore, for any $a \in \mathbb{Z}^+$, in the time window $[a * 2p, (a+1)2p)$ at most one job is scheduled in each machine.*

Proof. Consider an optimal schedule. We proceed in increasing order of time and modify the optimal schedule. Since the processing time of each job is in $[p, 2p]$, for each machine, the starting points of the scheduling intervals of at most two jobs j_1, j_2 can be contained in the time window $[0, 2p)$. If there are such j_1 and j_2, then, since "new" release times are integral multiples of $2p$, we must have $r(j_1)_{new} = r(j_2)_{new} = 0$. If j_1 and j_2 are scheduled on the rth machine, $r \in \{1, 2, \ldots, t'\}$, we schedule j_1 on the rth machine and j_2 on the $(r + t')$th machine respectively in the intervals $[0, p(j_1))$ and $[0, p(j_2))$. Clearly, the schedule in feasible and only one job is scheduled in each machine in the time window $[0, 2p)$. Now, by induction hypothesis, suppose, we can obtain a modified schedule for the jobs scheduled in $[0, a * 2p)$ in the optimal solution. Now, consider the time window $[a * 2p, (a + 1) * 2p))$. For the rth machine, again the starting points of the scheduling intervals of at most two jobs can be contained in $[a*2p, (a+1)*2p))$ in the optimal solution. We can safely schedule them respectively in the rth and $(r + t')$th machine starting from $a * 2p$, since the "new" release times of these two jobs cannot be after $a * 2p$. We thus have the proof by induction. ◀

Therefore, we strive to obtain a schedule with the following property: *We have a set of jobs J_s, with each job j having processing time in $[p, 2p]$, release time $r(j)_{new}$ that is an integral multiple of $2p$ and deadline $d(j)_{new}$. The goal is to use minimum number of machine and schedule all the jobs such that the scheduling interval of each job starts at integral multiple of $2p$.*

If we can obtain an online algorithm for the above stated scheduling problem with approximation factor γ, then from Lemma 12, Corollary 13 and Lemma 14, we get a schedule of all the slack jobs on 6γ times optimal number of machines. Indeed, we show, if OPT denotes the optimal number of machine for such schedules and we know OPT, then the following algorithm (EDF(OPT)) which schedules jobs based on the earliest deadline among all the released but unscheduled jobs at any time is optimal.

▶ **Lemma 15.** *Given there exists a schedule of jobs J_s in OPT machines where each job has release time and deadline at an integral multiple of $2p$, processing time in $[p, 2p]$ and must be scheduled at an integral multiple of $2p$, EDF(OPT) returns a feasible schedule in OPT machines.*

Algorithm 1 Earliest Deadline First Among Released yet Unscheduled Jobs (EDF)(s)

time $t = 0$,
while t is less than the latest deadline of the jobs in J_s **do**
  $a = 1$
  while there exists an unscheduled job with release time $r(j) \leq t$ and $a \leq s$ **do**
    Pick among these remaining jobs the one with the earliest deadline (breaking ties arbitrarily) and schedule it on the ath machine in the time window $[t, t + p(j))$.
    $a = (a + 1)$;
  end while
  $t = t + 2p$;
end while

Proof. An EDF schedule implies for any two jobs j_1 and j_2 scheduled on the same machine, if j_1 is scheduled earlier than j_2, then either j_1 has deadline before j_2, or j_2 is not released until j_1 finishes. We can convert any given optimal schedule to an EDF schedule. Given an optimal schedule, if it is not EDF, then consider the jobs in increasing order of the starting points of their scheduling intervals, and suppose the first violation from EDF happens at time $a * 2p$, for some $a \in \mathbb{Z}^+$. Note, since jobs are scheduled always at an integral multiples of $2p$, violations can also happen only at integral multiples of $2p$. Let j_1 be scheduled at $a * 2p$ and there exists a job j_2, which is scheduled at $a' * 2p$, $a' \in \mathbb{Z}^+, a' \geq a$, but $d_{new}(j_2) < d_{new}(j_1)$. If j_2 is not released at the time j_1 starts being scheduled, then since release times are integral multiples of $2p$, j_2 is not released until time $(a + 1)2p$. But, since processing time of j_1 is at most 2p, j_1 finishes by time $(a + 1)2p$. Therefore, this does not result in a violation. Hence, j_2 must have been released at or before time $a * 2p$. In that case, we can simply swap the positions of the scheduling intervals of j_1 and j_2, that is, we schedule j_2 starting from $a * 2p$ and j_1 starting from $a' * 2p$. This schedule is clearly valid for j_2. The schedule is also valid for j_1, since $d_{new}(j_1) > d_{new}(j_2)$, $d_{new}(j_2) \geq (a' + 1) * 2p$ and $p(j) \leq 2p$. We can continue doing such swaps until there is no violation from EDF at time $a * 2p$. This swaps do not affect jobs that are scheduled before $a * 2p$, and the maximum number of swaps that may be required is bounded by $|J_s|$. Therefore, at the end, we have an EDF schedule. ◀

Lemma 15 guarantees that when the number of machines required in an optimal solution is known, EDF(OPT) gives a constant competitive online algorithm for slack jobs with almost uniform processing time. Now suppose that the number of machines in an optimal schedule is not known. In that case, we proceed as follows. We make a guess of the optimal number of machines s'; suppose we set $s' = 1$. We start with $2s'$ machines. We follow the EDF algorithm using the first set of s' machines. But, at each time t, we also check whether the jobs that are released at time t or before but have not been scheduled yet, can be scheduled in s' machines using an offline algorithm for Machine Minimization. If so, in the second set of s' machines, we allocate the jobs that are scheduled in the time window $[t, t+2p)$ in the computed offline solution. Otherwise, we consider the jobs that are released strictly before time t (hence at or before time $t-2p$) and schedule all of them in s' machines using the offline solution computed at time $t - 2p$. We next consider the jobs that are released exactly at time t. We know they can be scheduled in at most OPT machines. We run an offline algorithm taking only these jobs and if the optimal solution uses s_1 machines, we allocate them in s_1 new machines and update our guessed optimal number of machines to $s' = max(s_1, 2s')$ and proceed to time $t + 2p$. We follow the same algorithm from time

$t + 2p$ with new guessed value of s'.

We need a few notations to describe the algorithm.

Let $J(t) = \{\text{jobs released before time } t\}$. Let $S(t) = \{\text{jobs scheduled before time } t\}$. Let $R(t) = \{\text{jobs released at time } t\}$. We set $s' = 1$ and $t = 0$ and call the following algorithm EDF-Online(s', t).

Algorithm 2 EDF-Online(s', t)

1: Open $2s'$ new machines;
2: Schedule $\min(|J(t) \cup R(t) \setminus S(t)|, s')$ jobs with earliest deadline from $J(t) \cup R(t) \setminus S(t)$ in the first s' machines. Let $A(t)$ be the set of jobs scheduled. {this is according to EDF}
3: **if** Offline Machine Minimization algorithm can schedule $([J(t) \cup R(t)] \setminus [S(t) \cup A(t)])$ in s' machines **then**
4: Let $B(t)$ be the set of jobs scheduled by Offline Machine Minimization at time t on at most s' machines.
5: Schedule $B(t)$ in the second set of s' machines.
6: $J(t + 2p) = J(t) \cup R(t)$, $S(t + 2p) = S(t) \cup A(t) \cup B(t)$, $t = t + 2p$, GOTO 2.
7: **else**
8: Schedule $J(t) \setminus S(t)$ using offline Machine Minimization in the already available machines–the second set of s' machines.
9: Schedule $R(t)$ in new machines using offline Machine Minimization. Let it uses s_1 new machines.
10: $J(t + 2p) = J(t) \cup R(t)$, $S(t + 2p) = J(t + 2p)$;
11: EDF-Online$(\max(2s', s_1), t + 2p)$
12: **end if**

▶ **Lemma 16.** *If there exists a schedule of jobs J_s in OPT machines where each job has release time at an integral multiple of $2p$, processing time in $[p, 2p]$ and must be scheduled at an integral multiple of $2p$,* EDF-Online *returns a feasible schedule in $O(OPT)$ machines.*

Proof. First, at step 8, offline Machine Minimization can use at most s' machines. This is because, if $t = 0$ or if at time $(t - 2p)$ the event under **Else** happened, then $J(t) \setminus S(t) = \phi$. If at time $t = 2p$, event under **if** happened, then we know at time $t - 2p$, Machine Minimization returned a feasible schedule for jobs in $([J(t - 2p) \cup R(t - 2p)] \setminus [S(t - 2p) \cup A(t - 2p)])$ in s' machines and the schedule has jobs in $B(t - 2p)$ scheduled in the interval $[t, t + 2p)$. Since $J(t) \setminus S(t) \subseteq [J(t - 2p) \cup R(t - 2p)] \setminus [S(t - 2p) \cup A(t - 2p) \cup B(t - 2p)]$, Machine Minimization will use at most s' machines to schedule them. Second, at step 9 offline Machine Minimization can use at most OPT machines. This is because here we are considering a set of jobs which are just released and therefore, they can be allocated exactly as in the optimal original offline schedule.

Once our guessed value s' becomes at least OPT, it will never be increased (follows from Lemma 16). Therefore, at the end $s' < 2OPT$. Hence, the number of machines used is at most $2(s' + s'/2 + s'/4 + ... + 1) \leq 8OPT$. ◀

3.2 Scheduling Non-slack Jobs

We now consider the non-slack jobs, that is, the length of the time window in which each of them can be scheduled is at most $8p$. We consider time in blocks of length p. We know any job that is released in between $[a * p, (a + 1) * p)$, for some $a \in \mathbb{Z}^+$, it must be scheduled by

$(a+9)*p$. We now divide the jobs based on their release time into 10 classes, $P_0, P_1, \ldots, P_9$: $P_k = \{j | \lceil r(j)/p \rceil \bmod 10 = k$. We schedule jobs of each class separately in new machines, thus blowing up the approximation factor at most by 10. For the jobs of the same class, we have the following properties.

Property 1. For any two jobs $j_1, j_2 \in P_k$, $\forall k \in [0, 9]$, if $r(j_1) < r(j_2)+p$, then $d(j_1) < r(j_2)$.

Property 2. For all jobs in P_k, $k \in [0, 9]$, which have release times in a time window not more than p, at most 9 of them can be scheduled in a single machine.

Therefore, jobs of P_k, $k \in [0, 9]$, which have release times contained in some time window of length at most p, can be scheduled on a separate machine by using at most 9 times the optimal number of machines.

Overall, by blowing up the approximation factor by a constant, we can schedule all the non-slack jobs, which altogether using Lemma 15 implies a constant factor online algorithm for Machine Minimization for jobs with processing time in $[p, 2p]$. This immediately implies a $O(\log \frac{p_{max}}{p_{min}})$ online algorithm for the general Machine Minimization problem, where the bound can be achieved by simply classifying the jobs into $\lceil \log \frac{p_{max}}{p_{min}} \rceil$ classed based on their processing times, $[p = p_{min}, 2p], (2p, 4p], ...$, and applying the algorithm separately for each class.

▶ **Theorem 17.** *There exists an online algorithm for* Machine Minimization *with constant approximation factor where all the jobs have processing time in $[p, 2p]$, for some $p > 0$.*

▶ **Corollary 18.** *There exists an online algorithm for* Machine Minimization *that achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$.*

3.3 Online Algorithm for Rent Minimization

Recall the classification of jobs J into J_1, J_2, J_3^1, J_3^2 from Section 2. We consider each of these collection of jobs separately. We follow the same algorithm as in Section 2 for jobs in J_1, J_2 and J_3^2. We follow the online machine minimization algorithm separately for each $J_{3,r}^1$, $r = 0, 1, ..., s$.

- $J_1 = \{j \mid p(j) \geq \frac{D}{2}\}$. These are the big jobs. *Assign new machine to each job $j \in J_1$ and keep the machine open exactly for $\lceil \frac{p(j)}{D} \rceil$ machine intervals.*
- $J_2 = \{j \mid j \notin J_1, \exists a \in \mathbb{N}, r(j) < aD, r(j) + p(j) > aD, d(j) - p(j) < aD\}$. No matter in which valid intervals we schedule these jobs, their scheduling intervals always contain the time aD. *Assign a new machine to each job $j \in J_2$ and keep the machine open exactly for 1 machine interval.*
- $J_3^1 = \{j \in J_3 \mid \exists r \in \mathbb{N}, r(j) \in H_r, d(j) \in H_r \cup H_{r+1}\}$. $J_{3,r}^1 = J_3^1 \cap \{j \mid r(j) \in H_r\}$. *Apply online algorithm for* Machine Minimization *separately to each $J_{3,r}^1$.*
- $J_3^2 = \{j \in J_3 \mid \exists s \in \mathbb{N}, r(j) \in H_r, d(j) \in H_t, t \geq r + 2\}$. *Apply Algorithm A.*

▶ **Theorem 19.** *There exists an online algorithm for* Rent Minimization *that achieves an approximation factor of $O(\log \frac{p_{max}}{p_{min}})$.*

Proof. Follows from Lemma 4, Lemma 6, Lemma 8 and Corollary 18. ◀

3.4 Lower Bounds

The following establishes the lower bound of any deterministic online algorithm for Machine Minimization.

▶ **Theorem 20.** *No deterministic online algorithm for* Machine Minimization *can achieve an approximation factor better than* $\log_3 \frac{p_{max}}{p_{min}}$.

Proof. Consider any online algorithm O. At time $t = 0$, a job j_1 is released with processing time p_{max} and deadline $3p_{max}$. If O decides to schedule it in the window $[a, a + p_{max})$, then at time $t = a$, j_2 is released with processing time $p_{max}/3$ and deadline $a + p$. Clearly, algorithm O needs to open a new machine to schedule j_2. If O decides to schedule j_2, in the window $[b, b + p_{max}/3) \subset [a, a + p_{max})$, then at time $t = b$, job j_3 arrives with processing time $p_{max}/3^2$ and deadline $b + p_{max}$. Clearly, O cannot schedule j_3 in either of the first two machines and need a third machine to schedule it. We can proceed in this fashion for $\log_3 \frac{p_{max}}{p_{min}}$ steps. Algorithm O needs to open a new machine to schedule each of these jobs, while all of them can be scheduled in a single machine in an optimal offline solution. ◀

The following corollary is immediate.

▶ **Corollary 21.** *No deterministic online algorithm for* Rent Minimization *can achieve an approximation factor better than* $\log_3 \frac{p_{max}}{p_{min}}$.

Acknowledgements The author would like to thank Howard Karloff for asking the question of Rent Minimization, for many helpful discussions throughout the work and helping to improve the writing.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, September 2001.

2 Amotz Bar-Noy and Sudipto Guha. Approximating the throughput of multiple machines in real-time scheduling. *SIAM J. Comput.*, pages 331–352.

3 Yair Bartal, Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, and Ron Lavi. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *In Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS*, pages 187–198. Springer, 2004.

4 Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichỳ. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM J. Comput.*, 36(6):1709–1728, February 2007.

5 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX-RANDOM*, pages 70–83, 2009.

6 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS*, pages 81–90, 2004.

7 Julia Chuzhoy and Joseph (Seffi) Naor. New hardness results for congestion minimization and machine scheduling. *J. ACM*, 53(5):707–721, September 2006.

8 Julia Chuzhoy, Rafail Ostrovsky, and Yuval Rabani. Approximation algorithms for the job interval selection problem and related scheduling problems. *Math. Oper. Res.*, pages 730–738, 2006.

9 Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.