The Power of Migration in Online Machine Minimization*

Lin Chen
Magyar Tudományos Akadémia (MTA SZTAKI)
Budapest, Hungary
chenlin198662@gmail.com

Nicole Megow
Technische Universität München
Zentrum Mathematik
Garching, Germany
nmegow@ma.tum.de

Kevin Schewior
Technische Universität München
Zentrum Mathematik
Garching, Germany
schewior@ma.tum.de

ABSTRACT

In this paper we investigate the power of migration in online scheduling on multiple parallel machines. The problem is to schedule preemptable jobs with release dates and deadlines on a minimum number of machines. We show that migration, that is, allowing that a preempted job is continued on a different machine, has a huge impact on the performance of a schedule. More precisely, let m be the number of machines required by a migratory solution; then the increase in the number of machines when disallowing migration is unbounded in m. This complements and strongly contrasts previous results on variants of this problem. In both the offline variant and a model allowing extra speed, the power of migration is limited as the increase of number of machines and speed, respectively, can be bounded by a small constant.

In this paper, we also derive the first non-trivial bounds on the competitive ratio for non-migratory online scheduling to minimize the number of machines without extra speed. We show that in general no online algorithm can achieve a competitive ratio of $f(m)$, for any function f, and give a lower bound of $\Omega(\log n)$. For agreeable instances and instances with "loose" jobs, we give $\mathcal{O}(1)$-competitive algorithms and, for laminar instances, we derive an $\mathcal{O}(\log m)$-competitive algorithm.

1. INTRODUCTION

It is a central problem in operating real-time systems to schedule preemptable jobs that arrive online over time with hard deadlines on multiple identical processors. When a job is preempted, it can resume processing at any time later. A schedule is *migratory* if a preempted job may continue processing on a different machine, whereas in a *non-migratory* schedule each job is processed by exactly one machine. In practice, non-migratory schedules are highly favored because migration may cause a significant overhead in communication and synchronization and it increases the risk of cache-failures. On the other hand, migratory schedules may be easier to design and have a better performance.

In this paper we investigate the power of migration in online scheduling with the objective of minimizing the number of machines. We show that migration has a huge impact on the performance of a schedule. More precisely, we construct instances with n jobs which can be processed by a migratory offline schedule on m machines, while any online non-migratory schedule requires at least $\max\{\Omega(\log n), f(m)\}$ machines for any function f. Informally speaking, preempting jobs is not worth it if we do not allow migration. Indeed, if migration is allowed then there is an online algorithm that finds a schedule on $\mathcal{O}(m \log m)$ machines [4], while otherwise not even an exponential number of machines (in m) is sufficient.

Our result complements and strongly contrasts previous results on the offline variant of our problem and on online scheduling with extra speed. Indeed, for those problems it has been shown that migration is actually of limited power [3, 7]; more details are given below. Our result also settles an open question raised in [3].

Leveraging the result on the offline variant of our problem [7], our lower-bound construction can be used to show the same bounds also on the *competitive ratio* of online non-migratory scheduling, that is, comparing an online non-migratory schedule to the optimal *non-migratory* schedule. In fact, these bounds are the first non-trivial bounds on the competitive ratio for online non-migratory scheduling to minimize the number of machines.

For relevant special cases of the problem, we show that the power of migration is much weaker. We derive an $\mathcal{O}(\log m)$-competitive non-migratory algorithm for instances with laminar time windows and even $\mathcal{O}(1)$-competitive algorithms for instances with agreeable time windows or "loose" jobs with a processing time being at most a constant fraction of the time window.

Further related results.

The online machine minimization problem under free migration has been investigated extensively by Phillips et al. [10]. For a natural greedy algorithm called *Least Laxity First* (LLF), they show a competitive ratio of $\mathcal{O}(\log \Delta)$ whereas the even simpler and classic *Earliest Deadline First* (EDF) algorithm, has a lower bound of $\Omega(\Delta)$. Both algorithms may preempt and migrate jobs many times. Only recently, a more sophisticated algorithm with competitive

*This research was supported by the German Science Foundation (DFG) under contract ME 3825/1. The third author was supported by the DFG within the research training group 'Methods for Discrete Structures' (GRK 1408).

SPAA '16, July 11-13, 2016, Pacific Grove, CA, USA

DOI: http://dx.doi.org/10.1145/2935764.2935786

ratio $\mathcal{O}(\log m)$ has been derived [4]. Despite this improvement, there remains a huge gap to the best known lower bound on the competitive ratio of 5/4 [10].

The non-preemptive (and thus non-migratory) variant of the problem is hopeless in terms of competitiveness. Saha [11] ruled out the existence of an algorithm with competitive ratio $f(m)$, for any computable function f. She gave a lower bound of $\Omega(\log \Delta)$ or n on the competitive ratio, complemented by an $\mathcal{O}(\log \Delta)$-competitive algorithm. (An upper bound of n is obvious.) However, in the special case of unit processing times, which was also studied implicitly in the context of energy minimization [1], the optimal online algorithm has the exact competitive ratio of $e \approx 2.72$ [1,5].

In the related speed-augmentation problem, an online algorithm has access to higher-speed machines instead of a larger number of machines than the offline adversary. This problem under free migration is quite well understood in the sense that online algorithms with a speed requirement of roughly 2 are known [9,10]. Also trade-offs between speed and machine augmentation were considered for this setting in [9].

For non-migratory online scheduling, there only exist results involving speed augmentation. Chan, Lam, and To [3] gave an algorithm with speed requirement 5.828 in the pure speed-augmentation model, that is, using the same number of machines as the offline migratory optimum. Concerning trade-offs, they also give an algorithm that requires $\lceil (1 + 1/\varepsilon)^2 \rceil \cdot m$ speed-$(1+\varepsilon)^2$ machines for any $\varepsilon > 0$. They state as an open question if there exists a non-migratory online algorithm using only $f(m)$ speed-1 machines, where f is a function on m, on instances for which there is a migratory optimum solution on m machines [3].

The power of migration for machine minimization has been investigated also in the offline setting, in which all jobs are known in advance [7,8]. In particular, Kalyanasundaram and Pruhs [7] have shown that any migratory solution on m machines can be transformed into a non-migratory solution on $6m - 5$ machines.

It is noteworthy that the preemptive migratory offline problem can be solved in polynomial time by modeling it as a linear program or maximum flow problem [6].

Our contribution.

Our main result is to rule out the existence of non-migratory online algorithms that can guarantee to find a schedule using a number of machines, that is independent of the number of jobs n or their processing times. More precisely, any non-migratory online algorithm may require $\Omega(\log n)$ many machines, whereas a migratory schedule on $m \geq 3$ machines exists. Note that when allowing migration, then it is possible to construct a feasible online schedule on $\mathcal{O}(m \log m)$ machines [4].

To the best of our knowledge, our lower bound is the first non-trivial result on the non-migratory scheduling problem without speed augmentation, and it answers an open question raised in [3]. We derive the lower bound by a recursive construction in which any *non-migratory* online algorithm spreads jobs over machines in such a way that releasing another job forces it to open a new machine. This is inherently different from previous lower-bound constructions [9,10] (for *migratory* algorithms), in which any algorithm (using too little resources) is forced to open a new machine because it has delayed the wrong jobs and needs to finish too much aggregate volume by a certain deadline.

Our lower-bound result does not only complement previous results on the power of migration, it also strongly contrasts them. Firstly, in the offline setting of our problem, every migratory schedule can be transformed into a non-migratory one by increasing the number of machines only by a small constant factor [7]. Secondly, a small (factor 5.83) increase in the speed is sufficient for an online algorithm to find a non-migratory schedule on the same number of machine, m, as the optimal migratory schedule [3]. Thus, migration has a rather negligible power in offline scheduling or when having extra speed, but it has a huge impact—even unbounded in m—for our online problem.

Competitive analysis is the major concept for evaluating the performance of online algorithms. Instead of comparing the performance of an online non-migratory algorithm with an optimal *migratory* schedule, one compares here with an optimal *non-migratory* schedule. A close connection between both ratios is not too difficult to see, and we describe our further results using the notion of competitive ratios. Indeed, we give the first competitiveness results for our non-migratory scheduling complementing earlier results that required speed-augmentation [3].

- Our strong lower bound above, implies a general lower bound of $\Omega(\log n)$ on the competitive ratio for non-migratory scheduling.

However, we identify a number of special cases which admit $\mathcal{O}(\log m)$- or even $\mathcal{O}(1)$-competitive algorithms.

The first special case occurs if all jobs have a processing time that is at most a constant fraction, say $\alpha \in (0, 1)$, of the respective job time interval. We call these jobs α-loose.

- For instances consisting only of α-loose jobs we give an $\mathcal{O}(1)$-competitive algorithm. The high-level idea is very simple. We use a known algorithm with a performance guarantee based on speed-augmentation from [3] as a black box. To do so, we increase the processing times of our jobs by a certain factor (guided by the guaranteed speed-factor) and apply the black-box algorithm. Then we transform back the resulting schedule into a feasible schedule on unit-speed machines for the original instance. The tricky part is to relate the minimum number of machines required by the original instance to the number needed by the modified instance. Our key result is an upper bound on this increase, which might be of independent interest.

Other special cases are agreeable and laminar instances which are widely believed to be the most important complementary special cases for scheduling jobs with time intervals. Instances are *agreeable*, if no intersecting time windows are fully contained in one another. For *laminar* instances it holds that whenever the time windows of two jobs intersect then they are fully contained in each other.

- We provide an $\mathcal{O}(\log m)$-competitive algorithm for laminar instances. With the above result for α-loose jobs, we only need to consider the remaining jobs. For those we design an algorithm which is inspired by an $\mathcal{O}(\log m)$-competitive algorithm for the general preemptive problem [4] and from which we carefully eliminate migration.

- For agreeable instances, we achieve a competitive ratio of $\mathcal{O}(1)$. Our algorithm is very simple and constructs even a non-preemptive solution. One may wonder, if agreeable instances are even trivial to solve. We answer this negatively by giving the first non-trivial lower bound, $6 - 2\sqrt{6} \approx 1.10$, on the competitive ratio for agreeable instances, even for the migratory problem.

Note that these positive results for special cases also justify the complexity of our lower-bound construction; a construction as simple as the one in Saha's lower bound for the non-preemptive problem [11], in which all jobs are α-loose and form a laminar instance, is not possible in our case.

Outline.

In Section 2, we formally define the problem and give some structural results. We construct the strong lower bound in Section 3. In Section 4, we give an $\mathcal{O}(1)$-competitive algorithm for instances consisting of α-loose jobs. We derive $\mathcal{O}(\log m)$- and $\mathcal{O}(1)$-competitive algorithms for laminar and agreeable instances in Sections 5 and 6, respectively.

2. PRELIMINARIES

Problem definition.

We are given a set of n jobs where each job $j \in \{1, 2, \ldots, n\}$ has a processing time $p_j \in \mathbb{Q}$, a release date $r_j \in \mathbb{Q}$ which is the earliest possible time at which the job can be processed, and a deadline $d_j \in \mathbb{Q}$ by which it must be completed. The task is to determine a feasible preemptive schedule on a minimum number of identical parallel machines. In a feasible schedule each job j is scheduled for p_j units of time within the time window $[r_j, d_j)$. Each machine can process at most one job at the time, and no job is running on multiple machines at the same time. We allow job preemption, i.e., a job can be preempted at any moment in time and may resume processing later. We distinguish migratory and non-migratory preemptive schedules. In a migratory schedule, a preempted job may resume processing on an arbitrary machine, whereas in the non-migratory setting, each job is processed on exactly one machine.

In this paper we focus our investigations on the performance of non-migratory online algorithms compared to a migratory offline optimum. However, we note a direct connection to *competitive analysis*, the standard method to evaluate the performance of online algorithms; see, e.g., [2]. The difference is that in competitive analysis we compare the performance of an online algorithm with an optimal offline solution within the *same* problem setting with respect to migration (and or preemption). In this paper, we give first non-trivial bounds on the competitive ratio of non-migrative online algorithms. In general, we call an online algorithm ρ-*competitive* if m' machines with $m' \leq \rho \cdot \mathsf{OPT}$ suffice to guarantee a feasible solution for any instance that admits a feasible schedule on OPT machines.

Throughout this paper we assume that the optimum number of machines is known to the online algorithm. It has been shown in [4] that we can do so at the loss of a small constant factor in the competitive ratio.

Notation.

We denote the *(processing) interval* of job j by $I(j) = [r_j, d_j)$, and we say j *covers* each time point $t \in I(j)$. For a set of jobs S, we define $I(S) = \cup_{j \in S} I(j)$. For $I = \cup_{i=1}^{k} [a_i, b_i)$ where $[a_1, b_1), \ldots, [a_k, b_k)$ are pairwise disjoint, we define the *length* of I to be $|I| = \sum_{i=1}^{k} (b_i - a_i)$.

We distinguish jobs by the relation between processing volume of a job and the length of its interval. We call a job α-*loose*, for some $\alpha < 1$, if $p_j \leq \alpha(d_j - r_j)$ and α-*tight* otherwise.

The *laxity* of a job j is defined as $\ell_j = d_j - r_j - p_j$. The remaining processing time of j at time t is denoted by $p_j(t)$.

Characterization of the optimum.

The optimal number of machines can be characterized based on the work load of jobs to be processed in certain time intervals [4]. Interestingly, in [4] only the lower bound on m was used. Now, in this new work we crucially employ the exact characterization, that is, lower and upper bound.

Given a finite union of intervals $I \subset \mathbb{R}$, let the *contribution* of a job j to I be $C(j, I) \coloneqq \max\{0, |I \cap I(j)| - \ell_j\}$, i.e., the least amount of processing that j receives during I in any feasible schedule. Let the contribution of a job set S to I, denoted by $C(S, I)$, be the sum of the individual contributions of jobs in S.

THEOREM 1 ([4]). *Let m be the minimum number of machines needed to schedule a given job set S feasibly. There exists a finite union of intervals I with $\lceil C(S, I)/|I| \rceil = m$ but none with $\lceil C(S, I)/|I| \rceil > m$.*

Power of migration and competitive ratio.

The performance of non-migratory online scheduling compared to migratory offline scheduling is closely related to the competitive ratio for non-migrative scheduling. In fact, we observe that they are equal up to a constant factor.

LEMMA 1. *Let f and g be functions in m, the optimal number of machines in a migratory schedule. The following two statements are equivalent for some $g \in \Theta(f)$.*

(a) There is an online algorithm that computes a non-migratory schedule on $f(m) \cdot m$ machines for any instance, for which there is a migratory schedule on m machines.

(b) There is a $g(m)$-competitive online algorithm for non-migratory scheduling.

To see that, the following result by Kalyanasundaram and Pruhs [7] will become handy.

THEOREM 2 ([7]). *Any feasible migratory schedule on m machines can be turned (offline) into a feasible non-migratory schedule on $6m - 5$ machines.*

PROOF OF LEMMA 1. For a given instance, let m be the number of machines used in an optimal migratory schedule and let OPT be the number of machines in an optimal non-migratory solution. Clearly, $m \leq \mathsf{OPT}$. Furthermore, Theorem 2 implies $\mathsf{OPT} \leq 6m - 5$.

Suppose there is an online algorithm that computes a non-migratory schedule on $k \leq f(m) \cdot m \leq f(m) \cdot \mathsf{OPT}$ and, thus, the algorithm is $f(m)$-competitive for non-migratory scheduling. We conclude that (a) implies (b).

To see the reverse direction, assume that there is a $g(m)$-competitive algorithm for non-migratory scheduling. This algorithm finds a non-migratory schedule using at most $g(m) \cdot \mathsf{OPT} < g(m) \cdot 6m$ many machines, which is $f(m) \cdot m$ for some $g \in \Theta(f)$. Thus, (b) implies (a). □

3. A STRONG LOWER BOUND

We prove the following theorem.

THEOREM 3. *Let $m \geq 3$. For every (deterministic) non-migratory online algorithm, there is an instance on n jobs such that the algorithm requires $\Omega(\log n)$ machines.*

We inductively define an adversarial strategy that forces any non-migratory online algorithm to use an unbounded number of machines while (offline) the resulting instance has a migratory schedule on three machines which even has some idle intervals. This is made formal in the following key lemma which directly implies the theorem.

LEMMA 2. *For every non-migratory online algorithm A and $k \in \mathbb{N}$, there is an instance I_k with $\mathcal{O}(2^k)$ jobs and a critical time t_0 such that:*

(i) In the schedule computed by A, there are at time t_0 unfinished critical jobs $j_1, \dots, j_k \in I_k$ assigned to k different machines.

(ii) There is a feasible schedule of I_k on three machines with the following properties. Two machines are idle within $[t_0, t_0 + \varepsilon)$ for some $\varepsilon > 0$. The other one is continuously idle from t_0 on.

The base of our inductive proof is easy to see. The intuition for the induction step is the following. Given I_k as above, we obtain I_{k+1} by first releasing I_k, forcing the online algorithm to open k different machines. We then release I'_k, a scaled-down copy of I_k which can be handled during the idle intervals of the offline schedule, and we distinguish two cases. Firstly, if the online algorithm uses $k+1$ machines to schedule both I_k and I'_k, we will be able to select a suitable subset of the critical jobs of I_k and I'_k to obtain the critical jobs of I_{k+1}. Secondly, if the online algorithm uses the same k machines to schedule both I_k and I'_k, we release a new job. This job will be designed to be in conflict with each of the critical jobs of I'_k and will thus require the online algorithm to open a new machine for the new job. The new job and the critical jobs of I_k will then be the critical jobs of I_{k+1}.

Before we give the formal proof of the lemma, we introduce some notation: The latest time when a job j has to be assigned to a machine and the earliest time when it can be finished are denoted by $a_j = r_j + \ell_j$ and $f_j = d_j - \ell_j$, respectively.

PROOF OF LEMMA 2.. We define the instances I_k depending on constants $\alpha \in (1/2, 1)$ and $\beta \in (0, 1/2)$, which will later be chosen appropriately. For the sake of intuition, α and β can be thought of constants very close to 1 and 0, respectively.

Clearly, I_1 exists. We define I_2: At time 0, we release job j_1 with $p_{j_1} = \alpha$ and $d_{j_1} = 1$. From a_{j_1} on, we start additionally releasing *short* jobs: For the i-th short job j, we set $r_j = a_{j_1} + (i-1) \cdot \beta$, $p_j = \alpha\beta$, and $d_j = a_{j_1} + i\beta$. We ensure that our choice of α and β fulfills

$$\left\lfloor \frac{f_{j_1} - a_{j_1}}{\beta} \right\rfloor \cdot \alpha\beta > \ell_{j_1}, \tag{1}$$

implying that the total processing time of short jobs that have to be scheduled within $[a_j, f_j) \subsetneq I(j_1)$ will eventually add up to more than the laxity of of j_1. This ensures that A has to schedule some short job on a different machine than j_1. We define j_2 to be the first such job, let $t_0 := a_{j_2}$, and stop releasing jobs then.

Given Equation (1), we check (i) and (ii) for critical jobs j_1, j_2 and critical time t_0: By definition of j_1 and j_2, they are assigned to different machines by A. Using $t_0 = a_{j_2} < f_{j_1}$ (implied by Equation (1)) and $t_0 = a_{j_2} < f_{j_2}$ (implied by $\alpha > 1/2$), both jobs are not finished at time t_0, that is, we get (i). On the other hand, an offline schedule could simply run job j_1 on one machine and all short jobs on another machine. Since for all jobs j holds $p_j < d_j - r_j$, they can be preempted within some interval $[t_0, t_0 + \varepsilon)$. This shows (ii).

We check that Equation (1) can indeed be made true: Using the definition of a_j, f_j, and ℓ_j, we obtain that Equation (1) is equivalent to

$$\left\lfloor \frac{2\alpha - 1}{\beta} \right\rfloor \cdot \alpha\beta > 1 - \alpha,$$

which, for instance, is true for $\alpha = 3/4$ and $\beta = 1/4$.

For $k > 2$, we define I_k inductively. Suppose the lemma is true for $k-1$. We start by applying the lemma once, creating $k-1$ critical jobs $j_1, \dots, j_{k-1}$ that are not finished by A and assigned to $k-1$ different machines at the critical time t_0. Also, there is a feasible schedule of I_k on three machines such that machine 1 and 2 are idle within some $[t_0, t_0 + \varepsilon)$ with $\varepsilon > 0$ while machine 3 is continuously idle from t_0 on. Recall that $p_j(t)$ is defined to be the remaining processing time of a job j at time t. We define

$$\varepsilon' := \min\left\{\varepsilon, p_{j_1}(t_0), \dots, p_{j_{k-1}}(t_0)\right\} > 0 \tag{2}$$

to be the largest ε' such that within $[t_0, t_0 + \varepsilon')$ no job j_i for $i \in \{1, \dots, k-1\}$ can get finished by A and machine 1 and 2 can still be idle in a feasible schedule. We apply the lemma with $k-1$ another time in a scaled-down way such that the latest deadline (i.e., that of the job released first) occurring in this sub-instance I'_k is $t_0 + \varepsilon'/2$. We call the corresponding critical jobs and time $j'_1, \dots, j'_{k-1}$ and t'_0, respectively, and distinguish two cases:

Case 1: The jobs $j_1, \dots, j_{k-1}$ and $j'_1, \dots, j'_{k-1}$ are scheduled by A on different sets of machines. By using the induction hypothesis, there is a job j'_i such that $j_1, \dots, j_{k-1}, j'_i$ are scheduled on k different machines. By our choice of ε' and the induction hypothesis for I'_k, these jobs are not finished by the critical time t'_0, implying (i). We also show (ii): Again by our choice of ε', only I'_k has to be scheduled within $[t_0, t_0 + \varepsilon'/2)$. By the induction hypothesis regarding I'_k, there is a schedule of this sub-instance with the following property: Two machines are idle within $[t'_0, t'_0 + \varepsilon'')$ for some $\varepsilon'' > 0$ and the other machine is continuously idle from t'_0 on. Since we can again choose to leave machine 3 continuously idle from t'_0 on, we get (ii).

Case 2: The jobs $j_1, \dots, j_{k-1}$ and $j'_1, \dots, j'_{k-1}$ are scheduled by A on the same set of machines. Then we additionally release exactly one job $j^\star$ at time t'_0 with deadline $t_0 + \varepsilon'$. We choose

$$p_{j^\star} \in \Big(t_0 + \varepsilon' - t'_0 - \min_{i=1,\dots,k-1} p_{j'_i}(t'_0),\; t_0 + \varepsilon' - t'_0\Big),$$

ensuring that $j^\star$ cannot be scheduled on the same machine as any j'_i for $i \in \{1,\dots,k-1\}$ (lower bound on $p_{j^\star}$) and that it has strictly positive initial laxity (upper bound on $p_{j^\star}$). We further make sure that

$$p_{j^\star} > t_0 + \frac{\varepsilon'}{2} - t'_0,$$

meaning that $j^\star$ cannot be finished at time $t_0 + \varepsilon'/2$ (note that $p_{j^\star}$ still exists since $t_0 + \varepsilon'/2 - t'_0 < t_0 + \varepsilon' - t'_0$). We release no further jobs and claim that (i) and (ii) hold with critical jobs $j_1,\dots,j_{k-1},j^\star$ and critical time $t''_0 \coloneqq t_0 + \varepsilon'/2$.

We first show (i): Consider the schedule computed by A. By the induction hypotheses for I_{k-1} and our choice of ε', at time t''_0 the jobs $j_1,\dots,j_{k-1}$ are not finished and scheduled on $k-1$ different machines. Using the assumption that $j'_1,\dots,j'_{k-1}$ are scheduled on exactly the same set of machines as $j_1,\dots,j_{k-1}$ and the fact that $j^\star$ cannot be scheduled on the same machine as any j'_i for $i \in \{1,\dots,k-1\}$, the jobs $j_1,\dots,j_{k-1},j^\star$ are scheduled on k different machines. By construction of $j^\star$, it is not finished by time t''_0 either.

To create a schedule obeying the condition in (ii), we schedule the whole instance except for $j^\star$ according to Part (ii) of the induction hypothesis (for both I_{k-1} and I'_{k-1}). Using that the latest deadline in I'_{k-1} is $t_0 + \varepsilon'/2$, in this schedule, machine 3 is continuously idle from t'_0 on and machine 1 (and 2) within $[t_0+\varepsilon'/2, t_0+\varepsilon')$. We schedule $j^\star$ within $[t'_0, t_0+\varepsilon'/2)$ on machine 3 and afterwards on machine 1, starting it on machine 1 as late as possible. Since $j^\star$ has positive initial laxity, this again leaves an idle period $[t''_0, t''_0+\varepsilon'')$ for $\varepsilon'' > 0$ on machines 1 and 2. Also, machine 3 is continuously idle from time t''_0 on, meeting the requirements of (ii). We illustrate this schedule in Figure 1.

We finally note that we have indeed $\mathcal{O}(2^k)$ jobs in the resulting instance: I_2 has a constant number of jobs, and for I_k we release I_{k-1} twice plus at most one additional job, showing that I_k has $\mathcal{O}(2^k)$ jobs. □

Using Lemma 1, we obtain a similar theorem concerning competitive ratio. We get the following more explicit statement by directly applying Theorem 2.

THEOREM 4. *Consider instances with n jobs that have a feasible non-migratory offline schedule on* OPT *machines, where* OPT ≥ 13 *is fixed. Every (deterministic) non-migratory online algorithm requires $\Omega(\log n)$ machines.*

PROOF. Suppose there is a non-migratory online algorithm A on $o(\log n)$ machines. Using Lemma 2, we obtain instances with n jobs on which A uses more than $\Omega(\log n)$ machines and which have feasible (migratory) schedules on 3 machines. Using Theorem 2, we can turn these schedules into a feasible non-migratory schedules on $6 \cdot 3 - 5 = 13$ machines. We obtain a contradiction to the assumption that A only requires $o(\log n)$ machines. □

4. A CONSTANT-COMPETITIVE ALGORITHM FOR LOOSE JOBS

In this section, we consider instances consisting of loose jobs, i.e., jobs j with $p_j \leq \alpha \cdot (d_j - r_j)$ for arbitrary constant $\alpha \in (0,1)$. The main theorem of this section is the following.

THEOREM 5. *For any fixed $\alpha \in (0,1)$, there is a non-migratory online algorithm on $\mathcal{O}(m)$ machines for instances consisting only of α-loose jobs.*

The key to proving the result is a reduction to a speed-augmentation problem on m machines.

THEOREM 6. *Suppose there is a non-migratory online algorithm* A *on $f(m)$ speed-s machines for general instances. Then, for every fixed $\alpha \in (0, 1/s)$, there is a non-migratory online algorithm on $f(\mathcal{O}(m))$ (unit-speed) machines for instances consisting only of α-loose jobs.*

The proof idea is as follows. To schedule α-loose jobs (for $\alpha < 1/s$) without additional speed, we increase the processing time of each job from the input instance by a factor of s. We denote the set of jobs with modified processing times by J^s. Note that the resulting instance is feasible in the sense that $p_j \leq d_j - r_j$ for all jobs j. We apply the speed-s algorithm to this instance. Whenever a job is scheduled in the resulting schedule, our algorithm schedules the corresponding job from the original instance on the respective machine.

To analyze our algorithm, we have to relate the minimum numbers of machines needed for scheduling the original instance J and for scheduling the instance with increased processing times J^s, respectively. To do so, we introduce two auxiliary job types. For some $\gamma \in (0,1)$ and every job j, we define two jobs $j^\triangleleft_\gamma, j^\triangleright_\gamma$ with

$$I(j^\triangleleft_\gamma) \coloneqq [r_j, d_j - \gamma\ell_j),\ p_{j^\triangleleft_\gamma} \coloneqq p_j$$
$$\text{and } I(j^\triangleright_\gamma) \coloneqq [r_j + \gamma\ell_j, d_j),\ p_{j^\triangleright_\gamma} \coloneqq p_j,$$

that is, we remove a γ-fraction of the laxity from either side of j's feasible time window. We further define $J^\triangleleft_\gamma \coloneqq \{j^\triangleleft_\gamma \mid j \in J\}$ and $J^\triangleright_\gamma \coloneqq \{j^\triangleright_\gamma \mid j \in J\}$. The following lemma relates the number of machines needed for scheduling J to that needed for scheduling $J^\triangleleft_\gamma$ and $J^\triangleright_\gamma$, and it may be of independent interest. Here we denote by $m(S)$ the minimum number of machines needed to schedule an instance S.

LEMMA 3. *For every instance J and $\gamma \in (0,1)$, we have*

$$m(J^\triangleleft_\gamma) \leq \frac{1}{1-\gamma} \cdot m(J) + 1$$
$$\text{and } \ m(J^\triangleright_\gamma) \leq \frac{1}{1-\gamma} \cdot m(J) + 1. \tag{3}$$

PROOF. We show the statement for $J^\triangleright_\gamma$, and the proof for $J^\triangleleft_\gamma$ is symmetric. According to Theorem 1, there exists a finite union of intervals I such that $m(J^\triangleright_\gamma) = \lceil C(J^\triangleright_\gamma, I)/|I| \rceil$. Without loss of generality, we can assume $I = [g_1,h_1) \cup \dots \cup [g_k,h_k)$ with $h_i < g_{i+1}$ for all $i = 1,\dots,k-1$. To derive the relationship between $m(J^\triangleright_\gamma)$ and $m(J)$, we expand I to a superset (and also finite union of intervals) $\operatorname{ex}(I)$ such that $|\operatorname{ex}(I)| = |I|/(1-\gamma)$. We will show for each job $j \in J$ with $C(j^\triangleright_\gamma, I) > 0$ that $C(j, \operatorname{ex}(I)) \geq C(j^\triangleright_\gamma, I)$. This will imply

$$\begin{aligned} m(J^\triangleright_\gamma) - 1 &\leq \frac{C(J^\triangleright_\gamma, I)}{|I|} = \frac{\sum_{j \in J} C(j^\triangleright_\gamma, I)}{|I|} \\ &\leq \frac{\sum_{j \in J} C(j, \operatorname{ex}(I))}{(1-\gamma) \cdot |\operatorname{ex}(I)|} = \frac{1}{1-\gamma} \cdot \frac{C(J, \operatorname{ex}(I))}{|\operatorname{ex}(I)|} \\ &\leq \frac{1}{1-\gamma} \cdot m(J), \end{aligned}$$

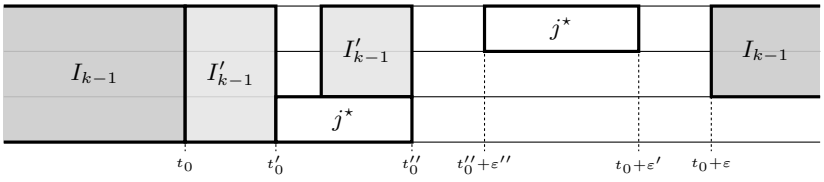

Figure 1: The optimal schedule constructed in Case 2 of the proof of Lemma 2. Recall $t''_0 = t_0 + \varepsilon'/2$.

where we use the other direction of Theorem 1 in the last step. Hence, Inequality (3) will follow.

The expanding works as follows. Given I as above, we expand each of the intervals $[g_i, h_i)$ to $[g'_i, h_i)$ with $g'_i \leq g_i$ and obtain $\operatorname{ex}(I) \coloneqq [g'_1, h_1) \cup \cdots \cup [g'_k, h_k)$. We start at the rightmost interval $[g'_k, h_k)$ and try to set $g'_k \coloneqq h_k - (h_k - g_k)/(1-\gamma)$. If this would, however, produce an overlap between $[g'_k, h_k)$ and $[g_{k-1}, h_{k-1})$, we set $g'_k \coloneqq h_{k-1}$ as well as $\delta_k \coloneqq h_{k-1} - (h_k - (h_k - g_k)/(1-\gamma))$ and try to additionally expand $[g_{k-1}, h_{k-1})$ by δ_k instead. After that, we continue this procedure until $i = 1$. More formally, we let $h_0 = -\infty$ as well as $\delta_{k+1} = 0$, and for all $i = k, \dots, 1$ we set

$$g'_i \coloneqq \max\left\{h_{i-1}, h_i - \left(\frac{h_i - g_i}{1-\gamma} + \delta_{i-1}\right)\right\}$$

$$\text{and } \delta_i \coloneqq \max\left\{0, h_{i-1} - \left(h_i - \left(\frac{h_i - g_i}{1-\gamma} + \delta_{i-1}\right)\right)\right\}.$$

Obviously, indeed $|\operatorname{ex}(I)| = |I|/(1-\gamma)$.

We show $C(j, \operatorname{ex}(I)) \geq C(j^\triangleright_\gamma, I)$ for all $j \in J$ with $C(j^\triangleright_\gamma, I) > 0$. By the definition of contribution, we have

$$\begin{aligned} C(j, \operatorname{ex}(I)) &= \max\{0, |\operatorname{ex}(I) \cap I(j)| - \ell_j\} \\ \text{and} \quad C(j^\triangleright_\gamma, I) &= \max\{0, |I \cap I(j^\triangleright_\gamma)| - \ell_{j^\triangleright_\gamma}\} \\ &= \max\{0, |I \cap I(j^\triangleright_\gamma)| - (1-\gamma)\cdot\ell_j\}, \end{aligned}$$

that is, we can restrict to showing $|\operatorname{ex}(I) \cap I(j)| \geq |I \cap I(j^\triangleright_\gamma)| + \gamma\ell_j$. Next, we define $I' \coloneqq I \cap I(j^\triangleright_\gamma)$. Using the fact that $I' \subseteq I$ implies $\operatorname{ex}(I') \subseteq \operatorname{ex}(I)$, it even suffices to show

$$|\operatorname{ex}(I') \cap I(j)| \geq |I'| + \gamma\ell_j. \tag{4}$$

To see this, we distinguish two cases:

Case 1: The leftmost point of $\operatorname{ex}(I')$ is left of r_j. Since the leftmost point of I' was not left of $r_j + \gamma\ell_j$, $[r_j, r_j + \gamma\ell_j)$ is a subset of $\operatorname{ex}(I')$. Consequently, we have $|\operatorname{ex}(I') \cap I(j)| - |I' \cap I(j)| \geq \gamma\ell_j$. Using $|I' \cap I(j)| = |I'|$, the claim follows.

Case 2: The leftmost point of $\operatorname{ex}(I')$ is not left of r_j. By the way our expanding works, we have that $\operatorname{ex}(I')$ takes up a length of $|I' \cap I(j)|/(1-\gamma)$ within $I(j)$. Plugging $|I'| > \ell_{j^\triangleright_\gamma} = (1-\gamma)\cdot\ell_j$, which follows since $C(j^\triangleright_\gamma, I) > 0$, into that, we also obtain $|\operatorname{ex}(I') \cap I(j)| - |I' \cap I(j)| \geq \gamma\ell_j$. Again using $|I' \cap I(j)| = |I'|$, the claim follows.

This completes the proof. □

We apply this lemma to get a similar statement about jobs with an in increased processing time, J^s, which might be of independent interest. Formally, for $j \in J$, let j^s denote a copy of job j with a processing time increased by a factor s, and $J^s \coloneqq \{j^s \mid j \in J\}$.

LEMMA 4. *Let $s \geq 1$. For every instance J consisting only of α-loose jobs for some fixed $\alpha \in (0, 1/s)$, we have $m(J^s) = \mathcal{O}(m(J))$.*

PROOF. We investigate the number of machines required to schedule J^s: To do so, we define

$$\delta \coloneqq \frac{1-\alpha s}{\lceil s \rceil} \cdot (d_j - r_j).$$

Note that $\ell_{j^s} \geq (1-\alpha s)\cdot(d_j - r_j) > 0$ and thus

$$0 < \delta \leq \frac{\ell_{j^s}}{\lceil s \rceil}. \tag{5}$$

Further, for every job $j \in J$, we define $\lceil s \rceil$ different jobs $j_1, \dots, j_{\lceil s \rceil}$ with

$$\begin{aligned} I(j_i) &\coloneqq [r_j + (i-1)\cdot(p_j + \delta), r_j + i\cdot(p_j + \delta)), \\ p_{j_i} &\coloneqq p_j \\ &\quad \text{for all } i < \lceil s \rceil \\ \text{and } I(j_{\lceil s \rceil}) &\coloneqq [r_j + (\lceil s \rceil - 1)\cdot(p_j + \delta), r_j + s\cdot p_j + \lceil s \rceil \delta)), \\ p_{j_{\lceil s \rceil}} &\coloneqq (s - \lceil s \rceil + 1)\cdot p_j. \end{aligned}$$

Furthermore let $J_i \coloneqq \{j_i \mid j \in J\}$ for all i. Now note that, for showing $m(J^s) = \mathcal{O}(m(J))$, it suffices to show that $m(J_i) = \mathcal{O}(m(J))$ for all i. This is because we have $I(j_i) \subseteq I(j^s)$ for all i, $I(j_i) \cap I(j_{i'}) \neq \emptyset$ for all i, i', and $\sum_i p_{j_i} = p_{j^s}$, which can be easily checked. Thus any feasible schedule of $J_1, \dots, J_{\lceil s \rceil}$ on $\mathcal{O}(m(J))$ machines each can be transformed into a feasible schedule of J^s on $\lceil s \rceil \cdot \mathcal{O}(m(J)) = \mathcal{O}(m(J))$ by scheduling each j^s whenever any j_i is scheduled in the respective schedule.

We show that indeed $m(J_i) = \mathcal{O}(m(J))$ for all i. We first show this for $i < \lceil s \rceil$: Note that, using Inequality 5, we have $\ell_{j_i} = \delta = \beta\ell_j > 0$ for all $j \in J$ and constant $\beta > 0$. Thus, by applying Lemma 3 up to two times to shorten the time window from both sides, $m(J_i) = \mathcal{O}(m(J))$. For $i = \lceil s \rceil$, first note that the above reasoning also works for $J'_{\lceil s \rceil} \coloneqq \{j'_{\lceil s \rceil} \mid j \in J\}$ with

$$\begin{aligned} I(j'_{\lceil s \rceil}) &\coloneqq [r_j + (s-1)\cdot p_j + (\lceil s \rceil - 1)\cdot\delta, \\ &\qquad\qquad r_j + s\cdot p_j + \lceil s \rceil \delta)), \\ p_{j'_{\lceil s \rceil}} &\coloneqq p_j, \end{aligned}$$

that is, $m(J'_{\lceil s \rceil}) = \mathcal{O}(m(J))$. Now clearly any feasible schedule of $J'_{\lceil s \rceil}$ can be transformed into one of $J_{\lceil s \rceil}$ without increasing the number of machines by scheduling each $j_{\lceil s \rceil}$ whenever $j'_{\lceil s \rceil}$ is scheduled unless $j_{\lceil s \rceil}$ is finished, implying $m(J_{\lceil s \rceil}) \leq m(J'_{\lceil s \rceil}) = \mathcal{O}(m(J))$. □

We now apply the preceding lemma in the proof of Theorem 6.

PROOF OF THEOREM 6. To schedule J only having α-loose jobs with constant $\alpha < 1/s$, we first transform J into J^s. Using Lemma 4, by applying A to J^s, we obtain a non-migratory schedule of J^s on $f(m(J^s)) = f(\mathcal{O}(m(J)))$ speed-s processors. We transform this schedule into a non-migratory schedule of J on $f(\mathcal{O}(m(J)))$ speed-1 machines by replacing each j^s by j for all j, which does not violate feasibility by the definition of j^s. □

We are now ready to utilize an algorithm due to Chan, Lam, and To [3] for which they prove an upper bound on the required number of machines that scales with the given speed—even for speeds arbitrarily close to 1.

THEOREM 7 (CHAN, LAM, AND TO [3]). *For every $\varepsilon > 0$, there is a non-migratory online algorithm on $\lceil (1+1/\varepsilon)^2 \rceil$ speed-$(1+\varepsilon)^2$ machines.*

Indeed, it suffices to plug Theorem 7 as a black box into Theorem 6 to obtain Theorem 5. By applying Lemma 1, we also get the following result for the competitive ratio.

THEOREM 8. *For any fixed $\alpha \in (0,1)$, there there is an $\mathcal{O}(1)$-competitive non-migratory online algorithm for instances consisting only of α-loose jobs.*

5. AN O(log m)-COMPETITIVE ALGORITHM FOR LAMINAR INSTANCES

In this section, we consider *laminar* instances in which for any two jobs j, j' with $I(j) \cap I(j') \neq \emptyset$, it holds that $I(j) \subseteq I(j')$ or $I(j') \supseteq I(j)$. We prove the following result.

THEOREM 9. *There exists an online algorithm that produces a non-migratory solution on $\mathcal{O}(m \log m)$ machines for laminar instances.*

For ease of presentation, we assume throughout this section that jobs are indexed from 1 to n in accordance with their release dates: We assume $j < j'$ implies $r_j \leq r_{j'}$. If $r_j = r_{j'}$, we assume that $j < j'$ implies $d_j \geq d_{j'}$. We say that j *dominates* j' (denoted as $j \succ j'$) if $j < j'$ and $I(j) \supseteq I(j')$. We also say j' *is dominated* by j and denote this as $j' \prec j$.

5.1 Description of the Algorithm

It suffices to consider α-tight jobs for some fixed $\alpha \in (0,1)$. The remaining α-loose jobs are scheduled on a separate set of $\mathcal{O}(m)$ machines as described in the previous section (cf. Theorem 5).

Job Assignment.

We open m' machines and will later see that we can choose $m' = \mathcal{O}(m \log m)$. At every release date, the arriving jobs are immediately assigned to their machines in order of job indices. Each job is assigned to exactly one machine, that is, we obtain a non-migratory schedule. Consider some job j. If there is a machine that has no job j' with $I(j) \cap I(j') \neq \emptyset$ assigned to it, we assign j to any such machine. Otherwise we execute the following procedure.

Consider any machine and the jobs j' assigned to it with $I(j) \cap I(j') \neq \emptyset$. Based on the laminarity of the instance and the order in which we assign jobs, all these j' dominate j and are ordered linearly by $\prec$. Consequently, there exists a unique $\prec$-minimal job among them, which is said to be currently *responsible* on the considered machine. Consider the set of currently responsible jobs of *all* machines. Again, by laminarity of the instance, these jobs form a chain $c_1(j) \prec \cdots \prec c_{m'}(j)$. We call $c_i(j)$ the i-th *candidate* of j.

We now want to select a candidate $c_i(j)$ and assign j to the same machine as $c_i(j)$. Consider the laxity of a candidate, which can be viewed as a *budget* for delaying the jobs. Since $I(j) \subseteq I(c_i(j))$, it is a necessary criterion that the budget of $c_i(j)$ suffices to schedule both j and the jobs j' with $I(j') \subseteq I(c_i(j))$ that have been assigned to the same machine earlier. Intuitively, we would also like to minimize the candidate that we pick w.r.t. $\prec$ so as to save the budget of jobs with larger time windows. However, it fails to greedily assign jobs to the machine of their $\prec$-minimal candidate that fulfills the above necessary criterion. This can be shown using a well-known class of difficult (laminar) instances [10, Theorem 2.13].

Instead we use a more sophisticated balancing scheme similar to that in [4]: We partition the budget of each candidate into m' equally-sized (sub-)budgets. For every i, we only consider the i-th budget of $c_i(j)$ when assigning j. When picking $c_i(j)$ and assigning j to the same machine, we charge not just p_j but $|I(j)| \geq p_j$ to the i-th budget of $c_i(j)$. This policy ensures that, as long as no budget becomes negative, there is a feasible schedule under the current assignment (cf. Lemma 5).

We make this more formal: We call a job j' that has been assigned to the machine of its h-th candidate $c_h(j')$ the h-th *user* of $c_h(j')$. We denote the set of all h-th users of a (candidate) job j' by $U_h(j')$, for all h. Note that at any time the h-th budget of j' has been charged exactly $|I(j'')|$ for all its users j''. To assign j, we select the smallest i such that the i-th budget of $c_i(j)$ can still pay for $|I(j)|$, that is,

$$\frac{\ell_{c_i(j)}}{m'} - \sum_{j' \in U_i(c_i(j))} |I(j')| \geq |I(j)|. \tag{6}$$

If we find such an i, we assign j to $c_i(j)$ (and thereby add j to $U_i(c_i(j))$ and charge $|I(j)|$ to the i-th budget of $c_i(j)$). If we do not find such an i, the assignment of j *fails*. In the analysis, we will show that the latter case will never happen if m' is chosen large enough.

Scheduling.

At any time and on each machine, we process an arbitrary unfinished job assigned to it with minimum deadline. It will later turn out (cf. Lemma 5) that there is always a *unique* such job.

5.2 Analysis of the Algorithm

We first show that our algorithm obtains a feasible schedule if no job assignment has failed. Then we give a proof of the fact that the job assignment never fails on instances that admit a feasible schedule. As a byproduct, we get a simplification of our scheduling rule.

LEMMA 5. *Suppose the job assignment does not fail for any job. We have the following two properties.*

(i) Our algorithm produces a feasible schedule.

(ii) At any time and on each machine, there are no two unfinished jobs with the same deadline.

PROOF. Consider some machine. We first show (ii) by induction on the jobs assigned to this machine in order of their indices. The induction base is clear. So assume the statement holds until some job j gets assigned to the considered machine.

The statement is obviously fulfilled if no other unfinished job on the considered machine has deadline d_j. So suppose $j^\star$ with $d_{j^\star} = d_j$ exists. We note that then the candidate j' of j must (also) have deadline d_j: Otherwise, using the laminarity of the instance, we have $d_{j^\star} < d_{j'}$ and thus $j' \succ j^\star$ and j' cannot be the $\prec$-minimal job whose interval contains r_j. Since j is assigned to the machine of its candidate j', one sub-budget and thus the total budget of j' must be at least $|I(j)|$ right before j is assigned. Consider the state of j' and its users at this time. Using Part (ii) of the induction hypothesis, j' was so far only preempted at some time t when there was a user j'' of it with $t \in I(j'')$. Thus, it was processed for at least

$$\begin{aligned}&\Big(|I(j')| - (d_{j'} - r_j)\Big) - \sum_{i=1}^{m'} |I(U_i(j'))| \\ \geq &\Big(|I(j')| - (d_{j'} - r_j)\Big) - \big(\ell_{j'} - |I(j)|\big) \\ = &|I(j')| - \ell_{j'} = p_{j'},\end{aligned}$$

using $d_j = d_{j'}$ in the second step. Hence j' is finished at time r_j, a contradiction.

To see (i), consider some job j and note that, by applying (ii), whenever it is preempted at some time t, there must be a user j' of j with $t \in I(j')$. According to Inequality (6), j is thus preempted for no longer than

$$\sum_{i=1}^{m'} |I(U_i(j))| \leq \sum_{i=1}^{m'} \sum_{j' \in U_i(j)} |I(j')| \leq m' \cdot \frac{\ell_j}{m'} = \ell_j.$$

Therefore, j can be processed for p_j time units. □

It remains to show that the job assignment never fails for some $m' = \mathcal{O}(m \log m)$. The proof idea is as follows: We assume the algorithm fails to assign a job $j^\star$. Then we select a critical job set and, by a load argument on this set, derive a contradiction to the existence of a feasible (migratory) schedule on m machines. Intuitively, this set is a minimal subset of the instance that still causes the failure. Note that this construction resembles the one from [4]. Indeed, the careful choice of our set allows us to use a lower bound introduced in [4] which is based on such a critical set.

We initialize $G_{m'} \coloneqq \{j^\star\}$. Given G_i, we construct F_i and G_{i-1} in the following way. First, F_i is defined to be the set of all $\prec$-maximal i-th candidates of jobs in G_i. Subsequently, G_{i-1} is constructed by adding all the i-th users of jobs in F_i to G_i. Formally,

$$\begin{aligned}F_i &\coloneqq M_\prec\big(\{c_i(j) \mid j \in G_i\}\big) \\ \text{and } G_{i-1} &\coloneqq G_i \cup \bigcup_{j \in F_i} U_i(j),\end{aligned}$$

where $M_\prec$ is the operator that picks out the $\prec$-maximal elements from a set of jobs: $M_\prec(S) \coloneqq \{j \in S \mid \nexists j' : j \prec j'\}$. After m' many such iterations, that is, when $G_0, F_1, \dots, F_{m'}$ have been computed, we set $F_0 \coloneqq M_\prec(G_0)$.

We define

$$F \coloneqq F_0 \cup F_1 \cup \dots \cup F_{m'} \text{ and } T \coloneqq \bigcup_{j \in F_0} I(j),$$

where we call (F, T) a *witness set*. The idea is to show that (F, T) obeys the following definition for suitably chosen parameters. This is the basis for applying a lower bound from [4] which we state below.

DEFINITION 1 ([4]). *Let F' be a set of α-tight jobs and let T' be a non-empty finite union of disjoint intervals. For some $\mu \in \mathbb{N}$ and $\beta \in (0, 1)$, a pair (F', T') is called (μ, β)-*critical *if*

(i) each $t \in T'$ is covered by at least μ distinct jobs in F',
(ii) and $|T' \cap I(j)| \geq \beta \ell_j$ for any $j \in F'$.

This will allows us to use the following result as a lower bound on the optimum number of machines m.

THEOREM 10 ([4]). *If there exists a (μ, β)-critical pair, then $m = \Omega\left(\frac{\mu}{\log 1/\beta}\right)$.*

To do so, we first show some structural properties of (F, T) that will be useful later. Towards this, we define the following notation. For two job sets S_1 and S_2, we write $S_1 \prec S_2$ if, for each $j_1 \in S_1$, there is a $j_2 \in S_2$ with $j_1 \prec j_2$.

LEMMA 6. *For any witness set (F, T), we have the following structural properties:*

(i) We have $F_0 \prec \dots \prec F_{m'}$.
(ii) The sets $F_0, \dots, F_{m'}$ are pairwise disjoint.

PROOF. To see (i), we define $C_i \coloneqq \{c_i(j) \mid j \in G_i\}$, for every $1 \leq i \leq m'$, and $C_0 \coloneqq G_0$. We first show $C_{i-1} \prec C_i$ for every $1 \leq i \leq m'$. For $i = 1$, this directly follows from the construction. Consider $2 \leq i \leq m'$. Let j be an arbitrary job in C_{i-1}, which is then an $(i-1)$-th candidate of some job in G_{i-1}, say, job j'. According to the construction, we have $j' \in G_i$, or j' is the i-th user of some job in $F_i \subseteq C_i$. In both cases, C_i contains the i-th candidate of job j', which dominates job j. Hence, $C_{i-1} \prec C_i$. Since F_{i-1} and F_i are obtained from C_{i-1} and C_i only by deleting dominated jobs, $F_{i-1} \prec F_i$ follows.

Consider property (ii) and suppose there is some $j \in F_i \cap F_{i'}$ for some $i < i'$. Then, by (i), there is also a job $j' \in F_{i'}$ with $j \prec j'$, contradicting the fact that $F_{i'}$ only contains $\prec$-maximal elements. □

We are ready to show that (F, T) is indeed a critical pair.

LEMMA 7. *The witness set (F, T) is a (μ, β)-critical pair for $\mu = m'$, $\beta = 1/m'$.*

PROOF OF LEMMA 7. We first consider Property (1) of a (μ, β)-critical pair. Given Lemma 6, it is obvious that every $t \in I(j)$ for $j \in F_0$ is covered by at least m' distinct jobs.

Concerning Property 1 of a (μ, β)-critical pair, we have

$$T = \bigcup_{j \in F_0} I(j) = \bigcup_{j \in M_\prec(G_0)} I(j) = \bigcup_{j \in G_0} I(j)$$

according to the definition of T. Furthermore, G_0 contains all the the i-th users of jobs in F_i, hence $|T \cap I(j)| \geq \ell_j / m'$ for any $j \in F_i$. □

We now use Lemma 7 and Theorem 10 to get a lower bound on m and thereby an upper bound on m'.

PROOF OF THEOREM 9. We show that, for a sufficiently large constant c and $m' = cm \log m$, our algorithm always produces feasible schedules. According to Lemma 5, it suffices to show that the job assignment procedure never fails. Suppose it does fail. Then we derive the witness set (F, T), which is a $(m', 1/m')$-critical pair (G, T) by Lemma 7. Using Theorem 10, we get $m = \Omega\left(\mu/\log 1/\beta\right) = \Omega(m'/\log m')$, which is a contradiction if $m' = cm \log m$ for sufficiently large c, that is, for some $m' = \mathcal{O}(m \log m)$ the algorithm never fails and thus always produces feasible schedules. □

Lemma 1 implies the following result for the competitive ratio.

THEOREM 11. *There is an $\mathcal{O}(\log m)$-competitive non-migratory online algorithm for laminar instances.*

6. AGREEABLE INSTANCES

Consider an instance J in which any two jobs $j, j' \in J$ are *agreeable*, that is, $r_j < r_{j'}$ implies $d_j \leq d_{j'}$. We also call J agreeable.

6.1 A constant-competitive non-preemptive algorithm

For agreeable instances, we derive an online algorithm that only uses $\mathcal{O}(m)$ machines. In fact, we show an even a stronger result in the context of non-*preemptive* scheduling.

THEOREM 12. *There exists an online algorithm that produces a non-preemptive solution on $32.70 \cdot m$ machines for agreeable instances.*

The idea is to again treat α-loose and -tight jobs separately for fixed $\alpha \in (0, 1)$. To obtain a non-migratory (but not necessarily non-preemptive) algorithm for α-loose jobs, note that we could simply use Theorem 6. To derive a non-preemptive algorithm, we will utilize the following theorem that has been shown for arbitrary (not necessarily agreeable) instances in the context of migratory scheduling. Given m' machines, the algorithm EDF schedules at any time m' unfinished jobs with the smallest deadlines.

THEOREM 13 ([4]). *Let $\alpha \in (0, 1)$. EDF is an online algorithm that produces a feasible (possibly migratory) solution on $m/(1-\alpha)^2$ machines for any instance that consists only of α-loose jobs.*

Now notice that EDF on agreeable instances never preempts jobs that have already started because the jobs released later have a larger deadline. This directly implies the following corollary.

COROLLARY 1. *Let $\alpha \in (0, 1)$. EDF is an online algorithm that produces a non-preemptive schedule on $m/(1-\alpha)^2$ machines for any agreeable instance that consists only of α-loose jobs.*

Consider α-tight jobs. We use the following simple algorithm MediumFit: Any job j runs exactly in the interval $[r_j + \ell_j/2, d_j - \ell_j/2)$, independently of all other jobs. Note that this algorithm is meaningful in the sense that running j in $[r_j + \ell_j, d_j)$ or $[r_j, d_j - \ell_j)$ does *not* yield a schedule on $\mathcal{O}(m)$ machines.

The analysis MediumFit works via a load argument.

LEMMA 8. *Let $\alpha \in (0, 1)$. MediumFit is an online algorithm that produces a non-preemptive solution on $16m/\alpha$ machines for any agreeable instance that consists only of α-tight jobs.*

PROOF. Consider an arbitrary time t and all jobs j' that are run by MediumFit at this time, among which we let j be a job with minimum laxity. We estimate their contributions to the interval(s)

$$I = [r_j - 2\ell_j, r_j + 2\ell_j) \cup [d_j - 2\ell_j, d_j + 2\ell_j),$$

which has a total length of at most $8\ell_j$. Distinguish two cases for each j' that is run at t.

Case 1: We have $|I(j')| \geq 2\ell_j$. As $I(j')$ contains r_j or d_j, we have $|I \cap I(j')| \geq 2\ell_j$. Given that $\ell_{j'} \leq \ell_j$, j' contributes at least $2\ell_j - \ell_{j'} \geq \ell_j$ to I.

Case 2: We have $|I(j')| < 2\ell_j$. As a consequence $I(j') \subseteq I$. Observe that $|I(j')| \geq \ell_j/2$ holds because both j and j' are run at time t by MediumFit. As j' is α-tight, its contribution to I is at least $\alpha\ell_j/2$.

Let n_1 and n_2 be the number of jobs corresponding to the above two cases, respectively. Then the contribution of the $n_1 + n_2$ jobs to I is at least

$$(n_1 + \alpha n_2/2) \cdot \ell_j \geq (n_1 + n_2) \cdot \alpha\ell_j/2.$$

Using Theorem 1, the total contribution is upper bounded by $m|I| \leq 8m\ell_j$, implying $n_1 + n_2 \leq 16m/\alpha$. □

Corollary 1 and Lemma 8 imply a non-preemptive solution on $m/(1-\alpha)^2 + 16m/\alpha$ machines, which attains its minimum at approximately $32.70 \cdot m$ for $\alpha \approx 0.63$. Theorem 12 follows. For the competitive ratio, we get the following result by Lemma 1.

THEOREM 14. *There is an $\mathcal{O}(1)$-competitive non-migratory online algorithm for agreeable instances.*

6.2 A non-trivial lower bound

We complement the upper-bound result with the first non-trivial lower bound for agreeable instances. We can even restrict to instances where all jobs have the same processing time.

THEOREM 15. *For any $\varepsilon > 0$, there is no online algorithm that produces a migratory solution on $(6 - 2\sqrt{6} - \varepsilon) \cdot m \approx 1.10 \cdot m$ machines for agreeable instances where all jobs have identical processing times.*

The lower-bound instances have a structure similar to existing ones for general instances [9, 10]. We say that an algorithm is *behind* by $w \geq 0$ at some time t if the following three properties hold:

(i) An optimal schedule could have finished all jobs released so far at time t.
(ii) The work unfinished by the algorithm is at least w.
(iii) We have $d_j \leq t + 1$ for all jobs released so far.

To prove the theorem, we will iteratively apply the following lemma.

LEMMA 9. *Consider an online algorithm* A *that uses* $(6-2\sqrt{6}-\varepsilon)m$ *machines for some* $\varepsilon > 0$. *There is a* $\delta > 0$ *with the following property.*

Let t *be a time at which* A *is behind by* w. *Then jobs can be released (in an agreeable way) such that there is a time* $t' > t$ *at which* A *is behind by* $w + \delta$.

PROOF. Define $\beta := (6 - 2\sqrt{6} - \varepsilon) - 1$. We choose some $\alpha \in [0,1] \cap \mathbb{Q}$ that is yet to be optimized, and we assume that $\alpha m \in \mathbb{N}$. At time t, we release m jobs j with $p_j = 1$ and $d_j = t + 1 + \alpha$ (type-1 jobs) and αm jobs j with $p_j = 1$ and $d_j = t + 2$ (type-2 jobs). We will now show that there is a $\delta > 0$ such that A is behind by $w+\delta$ at time $t' = t+1+\alpha$. Observe that it suffices to give a lower bound of $w + \delta$ on the work that is left of type-1 jobs at time $t + 1 + \alpha$.

First note that $(1-\alpha)m$ jobs j with $p_j = 1$ and $d_j = t+2$ could be released at time $t + 1$ without violating feasibility of the instance. This leaves at most $(\alpha + \beta)m$ machines for processing the type-1 jobs within $[t + 1, t + 1 + \alpha)$, i.e., a total capacity of $(\alpha^2 + \alpha\beta)m$. Thus, the type-2 jobs can receive $(\alpha^2 + \alpha\beta)m + \beta m - w$ of processing within $[t, t + 1)$, amounting to $(\alpha^2+\alpha\beta)m+\beta m-w+\alpha^2 m$ within $[t, t+1+\alpha)$. Consequently, at time $t' = t + 1 + \alpha$, the work left of type-1 jobs is at least

$$\alpha m - ((\alpha^2 + \alpha\beta)m + \beta m - w + \alpha^2 m) = w + \delta.$$

First note that δ is independent of w. Now the requirement $\delta > 0$ is equivalent to

$$\beta < \frac{\alpha - 2\alpha^2}{1 + \alpha}.$$

Maximizing the right-hand side of this inequality over the real numbers yields a maximum of $5 - 2\sqrt{6}$ for $\alpha = (\sqrt{6} - 2)/2 \approx 0.25$. By continuity of the right-hand side and definition of β, we can choose a rational α that makes the inequality true. This proves the lemma. □

We are ready to prove the theorem.

PROOF OF THEOREM 15. Suppose there exists some online algorithm A that uses $(6-2\sqrt{6}-\varepsilon)m$ machines. Clearly, for the empty instance and at time 0, A is behind by 0. Now applying Lemma 9 sufficiently often forces A to be behind by w at some time t where w exceeds the available capacity of $6 - 2\sqrt{6} - \varepsilon$ in $[t, t + 1)$. □

7. CONCLUSION

We have investigated the power of migration in online machine minimization. Surprisingly, it has turned out that, without allowing migration, the online machine requirement is unbounded in m even when $m = 3$ whereas with migration $\mathcal{O}(m \log m)$ machines suffice for any m [4]. A tight upper bound remains open. The currently best upper bound we are aware of, is $\mathcal{O}(\log \Delta)$, with Δ being the largest ratio of processing times, and it is obtained by a non-preemptive algorithm [11].

We remark that our lower bound leaves open if for $m = 2$ there is an online non-migratory algorithm using $\mathcal{O}(1)$ machines. Note that such an algorithm exists if we allow migration [4], but it is ruled out if we do not allow to preempt [11].

We also presented online algorithms for restricted instances, namely loose, agreeable, and laminar instances, that that construct non-migratory schedules using only $\mathcal{O}(m \log m)$ (or even $\mathcal{O}(m)$) machines. It remains an open question whether $o(m \log m)$ machines suffice for laminar instances when migration is not allowed.

8. REFERENCES

[1] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1), 2007.

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[3] H. Chan, T. W. Lam, and K. To. Nonmigratory online deadline scheduling on multiprocessors. *SIAM J. Comput.*, 34(3):669–682, 2005.

[4] L. Chen, N. Megow, and K. Schewior. An $\mathcal{O}(\log m)$-competitive algorithm for online machine minimization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 155–163, 2016.

[5] N. R. Devanur, K. Makarychev, D. Panigrahi, and G. Yaroslavtsev. Online algorithms for machine minimization. *CoRR*, abs/1403.0486, 2014.

[6] W. A. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21(1):177–185, 1974.

[7] B. Kalyanasundaram and K. Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001.

[8] G. Koren, E. Dar, and A. Amir. The power of migration in multiprocessor scheduling of real-time systems. *SIAM Journal on Computing*, 30(2):511–527, 2000.

[9] T. W. Lam and K.-K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 623–632, 1999.

[10] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200, 2002.

[11] B. Saha. Renting a cloud. In *IARCS Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 437–448, 2013.