Commitment and Slack for Online Load Maximization

Samin Jamalabadi
TU Dortmund University
Dortmund, Germany
samin.jamalabadi@tu-dortmund.de

Chris Schwiegelshohn
Sapienza University of Rome
Rome, Italy
schwiegelshohn@diag.uniroma1.it

Uwe Schwiegelshohn
TU Dortmund University
Dortmund, Germany
uwe.schwiegelshohn@tu-dortmund.de

ABSTRACT

We consider a basic admission control problem in which jobs with deadlines arrive online and our goal is to maximize the total volume of executed job processing times. We assume that the deadlines have a slack of at least ε, that is, each deadline d satisfies $d \geq (1+\varepsilon)\cdot p + r$ with processing time p and release date r. In addition, we require the admission policy to support *immediate commitment*, that is, upon a job's submission, we must immediately make the decision of if and where we schedule the job, and this decision is irreversible.

Our main contribution is a deterministic algorithm with nearly optimal competitive ratio for load maximization on multiple machines in the non-preemptive model. Previous results either only held for a single machine, did not support commitment, or required job preemption and migration.

CCS CONCEPTS

• **Theory of computation** → **Scheduling algorithms**; • **Networks** → *Cloud computing*.

KEYWORDS

online algorithms; scheduling; commitment

ACM Reference Format:
Samin Jamalabadi, Chris Schwiegelshohn, and Uwe Schwiegelshohn. 2020. Commitment and Slack for Online Load Maximization. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3350755.3400271

1 INTRODUCTION

In the Infrastructure-as-a-Service business model of cloud computing, system providers rent out infrastructure (typically but not limited to computing power) to customers. They are responsible for system management and resource allocation. The business model may comprise multiple customer service-levels. For example, some periodic routine tasks have a low urgency while time-sensitive jobs require an almost immediate completion.

Due to their real-world relevance, it has become increasingly important to model these systems theoretically and understand the possibilities and limitations in designing algorithms for them. Here, we consider online algorithms that maximize revenue while obeying certain system constraints. In its most generic form, we aim to maximize the function $\sum_{\text{Job } j \text{ accepted}} w_j$, where w_j is some arbitrary non-negative value [2, 28]. Often algorithms target special cases, such as maximization of throughput [8, 14] ($w_j = 1$ for all jobs) or load [19] ($w_j = p_j$ with processing time p_j of job J_j). We apply *competitive analysis*, that is, we aim to maximize the ratio between the revenue of an optimal offline algorithm and the revenue obtained by our online algorithm. In accordance with the cloud computing use case, we study these objectives for parallel machines.

We use the key assumption that the jobs have *slack* $\varepsilon > 0$, that is, deadlines satisfy the slack condition $d \geq (1+\varepsilon)\cdot p + r$ with release date r and processing time p. This assumption is useful both theoretically and empirically: tight deadlines are typically not of practical importance and yield impossibility results. We can consider the slack as a system parameter determined by the system provider. Therefore, we are interested in understanding how much a small slack affects the difficulty of admission problems.

Once an infrastructure has been rented out to a customer, the agreement is binding. Therefore, we require that the admission policy *immediately commits* to every decision it makes. Commitment has received substantial recent attention (see, for instance, [2, 8, 15, 28, 29] and references therein) with the following models being most commonly studied in literature.

Immediate Commitment, also called **commitment on arrival**, is arguably the strongest and most desirable variant of commitment: we must decide immediately upon submission of a job if we execute the job. Lucier et al. [28] showed that for general objective functions, any online algorithm has an unbounded competitive ratio for any slack value. Most positive results exist for load or throughput maximization, see [11, 14, 20]. In non-preemptive machine models, the commitment property typically extends to the allocation of the job with a temporal (start time) and spatial (executing machine) component. If commitment does not include job allocation then the algorithm supports **immediate notification** [9, 17], see, for instance, preemptive machine models [10, 16] that allow interrupting and reallocating the jobs on the fly.

Delayed Commitment: given a slack ε and a parameter $\delta \leq \varepsilon$, we say that an algorithm has δ-delayed commitment if it makes the decision whether to accept a job J_j before time $r_j + \delta \cdot p_j$, see, for instance, [2, 8]. When supporting **commitment on admission**, an algorithm only commits to a job upon starting it. Many early works on online admission control use this variant, see [18, 26, 27].

Commitment with Penalties requires an immediate commitment of the scheduling algorithm but allows a later revocation at the cost of a loss in the objective function, see, for instance, [15, 31]. This variant of commitment has some similarity to robust online scheduling that limits the number of decision revocations [30].

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07…$15.00
https://doi.org/10.1145/3350755.3400271

While the former variant penalizes each commitment violation, the latter budgets the number of times we can reverse a decision.

In this paper, we maximize the total load $\sum_{\text{Job } j \text{ accepted}} p_j$ on m parallel identical machines that do not allow preemption. Furthermore, all jobs have slack ε. Our goal is to design competitive algorithms with *immediate commitment*, that is, upon submission of a job, the algorithm either rejects or allocates it.

1.1 Our Results and Techniques

For our problem with slack $\varepsilon \in (0, 1]$ and m machines, we present a lower bound $c(\varepsilon, m)$ of the competitive ratio of any deterministic online algorithm and a deterministic online algorithm with a competitive ratio that matches the lower bound for sufficiently small values of ε or deviates from it by at most $(3-e)/(e-1) \approx 0.164$.

The function $c(\varepsilon, m)$ decreases with increasing values of ε and m. We conjecture that it is optimal for every pair of m and $\varepsilon \in (0, 1]$. As m tends to ∞ and ε tends to 0, $c(\varepsilon, m)$ approaches $\ln(1/\varepsilon)$. Therefore, it is the first super constant lower bound for parallel machines that simultaneously holds for both large values of m and small values of ε.

For each value of m, $c(\varepsilon, m)$ has $m-1$ continuous phase transitions from a term $O\left(\varepsilon^{-1/(k+1)}\right)$ to a term $O\left(\varepsilon^{-1/k}\right)$ with $k \in \{1, \ldots, m-1\}$. We can provide the exact terms of $c(\varepsilon, m)$ only for the last three phases, see, for instance,

$$c(\varepsilon, 2) = \begin{cases} 2 \cdot \sqrt{\frac{25}{16} + \frac{1}{\varepsilon}} + \frac{1}{2} & \text{for} \quad 0 < \varepsilon < \frac{2}{7} \\ \frac{3}{2} + \frac{1}{\varepsilon} & \text{for} \quad \frac{2}{7} \leq \varepsilon \leq 1. \end{cases} \tag{1}$$

For the other phases, we must numerically determine the function value for each pair (ε, m). To get a better feel for these bounds, we have plotted $c(\varepsilon, m)$ for $1 \leq m \leq 4$ in Fig. 1. More details on the properties of this function are given in Section 2; an intuition as how we derived it is given in Section 3.

For $m = 1$, our online algorithm matches the performance of the optimal deterministic algorithm by Goldwasser and Kerbikov [20]. Supporting immediate commitment, it also slightly improves on the previously best bound of $1 + m + m \cdot \varepsilon^{-1/m}$ by Lee [26] that requires commitment upon admission. Using a standard technique, our algorithm yields a randomized algorithm with immediate commitment for a single machine. Its competitive ratio $O\left(\log \varepsilon^{-1}\right)$, improves on the deterministic competitive ratio of $2 + \frac{1}{\varepsilon}$ [20].

Our Techniques

Lower Bounds. The starting point of our work is Goldwasser's lower bound for a single machine [18]. To obtain a $1/\varepsilon$ lower bound[1], the adversary merely submits a job with value 1 and deadline $(1+\varepsilon)$. If the algorithm does not accept this job, no further jobs arrive and the competitive ratio is unbounded. Otherwise, the adversary submits another job with processing time $p < 1/\varepsilon$ and deadline $(1+\varepsilon) \cdot p$ the moment the first job is started. The algorithm cannot accept the second job and the lower bound follows from $p \to 1/\varepsilon$.

For two machines, we cannot replicate this approach: if the adversary submits two jobs of length 1 then the algorithm is only forced to accept one of them. Accepting both jobs leads to the same competitive ratio $O(\varepsilon^{-1})$ as in the single machine case, see the second phase of $c(\varepsilon, 2)$ in (1). If the algorithm rejects one job and reserves the second machine in anticipation of a larger job then the adversary may counter this by submitting a job of length p with $1 < p < 1/\varepsilon$ depending on the value of ε. If the algorithm rejects this job then the adversary does not submit any further job and we obtain a competitive ratio of $O(1+p)$. Otherwise, the adversary submits a final job with a processing time slightly less than $1/\varepsilon$. Since the algorithm cannot accept this job, we obtain a competitive ratio of $O\left(\frac{\varepsilon^{-1}+p}{1+p}\right)$. Equalizing both terms produces a competitive ratio of $O\left(\varepsilon^{-1/2}\right)$ that matches the first phase of $c(\varepsilon, 2)$ in (1).

To extend this idea for multiple machines, we use a recursive construction and assume $\varepsilon < 1$. Initially, we generate a sequence of jobs with processing time 1. Due to $\varepsilon < 1$, no two jobs can be processed on the same machine. The algorithm selects $k \in \{1, \ldots m\}$ of these jobs. For each value of k, we recursively determine a final sequence of jobs that forces the competitive ratio: for $k = m$, the adversary submits m jobs with processing time p_m slightly less than $1/\varepsilon$ and the algorithm must reject all of them. Since there is at least one idle machine for $k < m$, the adversary submits up to m jobs of length p_k until the algorithm accepts one. If the algorithm has not accepted anyone of these jobs then we have a competitive ratio of $\frac{m \cdot p_k}{k}$. Otherwise, the adversary continues as if the algorithm had initially accepted $k+1$ jobs with processing time 1. The only difference is the accepted load of $k + p_k$ instead of $k+1$. As in the case with two machines, we determine the processing times $p_k, \ldots, p_{m-1}$ by equalizing the $m-k+1$ possible competitive ratios

$$\frac{m \cdot p_k}{k} = \ldots = \frac{m \cdot p_{q+1}}{k + \sum_{i=k}^{q} p_i} = \ldots = \frac{m \cdot p_m}{k + \sum_{i=k}^{m-1} p_i} \tag{2}$$

and obtain $p_k = O\left(\varepsilon^{-1/k}\right)$. Each value of k represents a phase of the function $c(\varepsilon, m)$. The term of first (dominant) phase of function $c(\varepsilon, m)$ is $O\left(\varepsilon^{-1/m}\right)$. Identity (2) lies at the heart of our construction and proof, and is the main reason that we can give an analytic expression only for those phases of the function $c(\varepsilon, m)$ that are represented by $k \in \{m-2, m-1, m\}$.

Upper Bounds. Any algorithm for our problem consists of two phases: (a) a decision phase in which we decide whether to accept or reject a job, and (b) an allocation phase in which we assign the job to a machine and a start time. To explain the main ideas behind the algorithm, we use an easy variant of the problem in which all release dates are 0. Before considering a new job J_j, we index the machines m_i with $i \in \{1, \ldots m\}$ by decreasing load $l(m_i)$. For the decision phase, we use an admission threshold $\max_{i \geq k}\{l(m_i) \cdot f_i\}$ based on the $m-k+1$ least loaded machines. The value k corresponds to k in the lower bound description while the values $f_i > 1$ are closely related to the values p_i for $i \in \{k, \ldots, m\}$ in the lower bound description, see Section 4 for details. Note that $f_i < f_{i+1}$ holds for $i \in \{k, \ldots, m-1\}$.

Let J_j be the rejected job with the largest deadline among all rejected jobs. Then there is some machine m_h with $d_j < l(m_h) \cdot f_h$ and our competitive ratio is $O\left(d_j \cdot m / \sum_{1 \leq i \leq m} l(m_i)\right)$. To limit the competitive ratio, the allocation part of our algorithm primarily seeks machines m_1 to m_{k-1} as target machines since the loads

[1] The optimal bound is $2 + 1/\varepsilon$. We consider a worse bound to simplify the exposition.

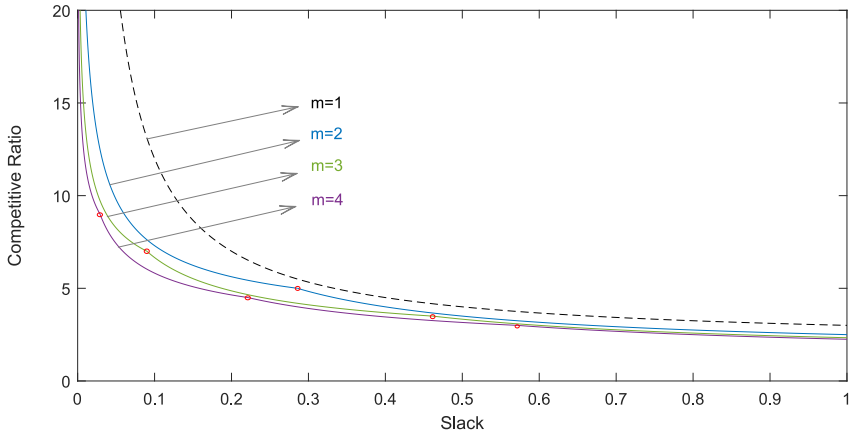

Figure 1: Functions of tight competitive ratios in slack interval $(0, 1]$ for $m = 2$ (blue line) and $m = 3$ (green line) in comparison to the tight competitive ratio for $m = 1$ (dashed line), see Goldwasser and Kerbikov [20]. The competitive ratio for $m = 1$ is identical to a greedy approach with list scheduling in a parallel environment, see Kim and Chwa [23]. For $m = 4$ (violet line), there is a very small gap of at most 0.12 between $c(\varepsilon, 4)$ and the obtained competitive ratio in the right most phase. The circles on the lines indicate transitions from one phase to the next one.

on these machines do not affect the threshold. For machines m_i with $k \leq i < h$, the algorithm tries to allocate sufficient load to them such that $l(m_i) \cdot f_i$ does not cause a larger threshold by exceeding $l(m_h) \cdot f_h$. It turns out that a good load distribution is always obtained by a simple best-fit heuristic: we allocate an accepted job to the most loaded machine that can process it prior to its deadline and start it immediately after the completion of the preceding job on this machine. The effect is two-fold: first this policy affects our ability to accept and schedule longer jobs the least. Second it turns out that this allocation policy has the smallest effect on the term $\max_{k \leq i \leq m}\{l(m_i) \cdot f_i\}$.

1.2 Related Work

Load maximization objectives have been a staple of online scheduling since the early 90s [1, 3–6, 17, 18, 23, 25, 27]. Without slack, competitive ratios are typically of the form $O(\log \Delta)$ [17, 27], where Δ is the ratio of the largest to the smallest possible processing time.

For slack scheduling, most of these early algorithms only satisfy the commitment on admission property. The state of the art for non-preemptive machines with slack seems to be an optimal $\left(2 + \frac{1}{\varepsilon}\right)$-competitive deterministic algorithm by Goldwasser [18], and an $O(1+m+m\cdot\varepsilon^{-1/m})$ competitive algorithm on m identical machines and an $O(\log 1/\varepsilon)$-competitive randomized algorithm for a single machine by Lee [26]. The former result has since been extended to algorithms with immediate commitment by Goldwasser and Kerbikov [20]. Currently, the only known immediate notification algorithms for parallel machines that improve over Goldwasser and Kerbikov [20] require some degree of preemption. DasGupta and Palis [10] and Garay et al. [16] independently proved a competitive ratio of $1 + \frac{1}{\varepsilon}$, assuming that preemption (but not migration) is allowed. DasGupta and Palis [10] also gave a lower bound of roughly 4 for small values of ε. Schwiegelshohn and Schwiegelshohn [29] studied parallel machines that support preemption and migration. For this machine model and immediate commitment, they gave a deterministic algorithm with a competitive ratio that approaches $(1 + \varepsilon) \cdot \log \frac{1+\varepsilon}{\varepsilon}$ if m is large enough.

A different line of research does not use any slack to obtain meaningful competitive ratios but instead assumes that the jobs have unit length. On a single machine, Baruah et al. [4] proposed an optimal deterministic algorithm with a competitive ratio of 2. For randomized algorithms, the state of the art is a 5/3 competitive algorithm due to Chrobak et al. [9]. The best known randomized lower bound is 4/3 [17]. On parallel machines, improvements are possible. Ding et al. [11] gave a deterministic algorithm with a competitive ratio that approaches $\frac{e}{e-1}$ if the number of machines is large enough. A matching lower bound was given by Ebenlendr and Sgall [13]. Further results on parallel machines with a more relaxed form of commitment are given in [7, 12, 21].

2 DEFINITIONS AND PRELIMINARIES

Each job J_j is specified by a tuple (r_j, p_j, d_j), corresponding to release date, processing time, and deadline, respectively. The deadline satisfies the slack condition

$$d_j \geq (1 + \varepsilon) \cdot p_j + r_j. \tag{3}$$

If (3) holds with equality, we say that the job has a *tight slack*. We assume throughout this paper that $\varepsilon \in (0, 1]$. For slack values greater than 1, it is easy to derive constant competitive algorithms with immediate commitment using previous work[2]. The system comprises m identical non-preemptive machines. The indicator variable U_j is 1 if and only if we reject job J_j, see Graham et al. [22]. We refer to our problem as $Pm|\text{online}, \varepsilon, \text{immediate}| \sum p_j \cdot (1 - U_j)$ using the standard three-field notation.

We now introduce the function $f(\varepsilon, m)$ that is crucial for both the lower bound and the online algorithm. The function depends

[2]For example, a greedy algorithm that allocates the jobs in a non-delay fashion always achieves a competitive ratio less than 3 for $\varepsilon > 1$.

on the slack ε and the number of machines m, and is defined by recursion, that is, for each pair (ε, m), we obtain a different variant of the recursion. For each variant, $f(\varepsilon, m)$ uses $m-k+1$ parameters $f_q(\varepsilon, m)$ for $q \in \{k, \ldots, m\}$. Informally, we have already introduced integer parameter k in Section 1.1. Later in this section, we provide a formal definition. Although the anchor (4) of the recursion only depends on the slack, we use the notation $f_m(\varepsilon, m)$ for the corresponding parameter to be consistent with the notations of the other parameters of the recursion. Remember that the indexing of parameters $f_q(\varepsilon, m)$ matches the indexing of the machines based on the outstanding load in Section 4, see Section 1.1 and Equation (9).

$$f_m(\varepsilon, m) = \frac{1+\varepsilon}{\varepsilon} \tag{4}$$

$$c(\varepsilon, m) = \frac{1 + m \cdot f_q(\varepsilon, m)}{k + \sum_{h=k}^{q-1}(f_h(\varepsilon, m) - 1)} \text{ for } q \in \{k, \ldots, m\} \tag{5}$$

For a fixed value k, all $f_q(\varepsilon, m)$ are strictly decreasing with increasing slack due to (4). Similar to (2), (5) requires that the ratio $c(\varepsilon, m)$ is independent of parameter q. Therefore, we have $f_q(\varepsilon, m) < f_{q+1}(\varepsilon, m)$ for $q \in \{k, \ldots, m-1\}$.

Before explaining the role of k, we introduce another constraint

$$f_q(\varepsilon, m) \geq 2 \text{ for } q \in \{k, \ldots, m\} \tag{6}$$

that is required for technical reasons, see the details in Section 3. Due to $\varepsilon \in (0, 1]$, $f_m(\varepsilon, m)$ always satisfies (6). To prevent a violation of (6) for other values, parameter $k \in \{1, \ldots, m\}$ defines corner values $\varepsilon_{k,m}$ of the slack parameter and restricts the range of q:

$$f_k(\varepsilon_{k,m}, m) = 2 \tag{7}$$

Due to the monotony of $f_q(\varepsilon, m)$, these corner values partition the slack interval $(0, 1]$ into m slack intervals $(0 = \varepsilon_{0,m}, \varepsilon_{1,m}]$, $(\varepsilon_{1,m}, \varepsilon_{2,m}]$, …, $(\varepsilon_{m-1,m}, \varepsilon_{m,m} = 1]$, see Fig. 1. In slack interval $(\varepsilon_{k-1,m}, \varepsilon_{k,m}]$, we use the same variant of the recursion since the same value of k guarantees validity of (6). Due to

$$\frac{1 + m \cdot f_q(\varepsilon_{k,m}, m)}{k + \sum_{h=k}^{q-1}(f_h(\varepsilon_{k,m}, m) - 1)} = \frac{1 + m \cdot f_q(\varepsilon_{k,m}, m)}{k + 1 + \sum_{h=k+1}^{q-1}(f_h(\varepsilon_{k,m}, m) - 1)}$$

for all $q \in \{k+1, \ldots, m\}$, the competitive ratio is continuous at the corner values.

Finally, we remark that the competitive ratio approaches $\ln 1/\varepsilon$ for small slack values as m tends to ∞.

Proposition 1.

$$\lim_{m \to \infty} c(\varepsilon, m) = \ln \frac{1}{\varepsilon} \text{ for } \varepsilon \in (0, \varepsilon_{1,m}]$$

Proof. We use a continuous extension of parameter function $f_q \to f(x)$ and transform recursion (5)

$$\lim_{m \to \infty} c(\varepsilon, m) = \frac{\frac{1}{m} + f_q(\varepsilon, m)}{\frac{1}{m} + \frac{1}{m} \cdot \sum_{h=1}^{q-1}(f_h(\varepsilon, m) - 1)} \to \frac{f(x)}{\frac{1}{m} + \int_0^x (f(z) - 1)dz} \text{ for } x \in (0, 1] \tag{8}$$

We replace the undefined term $1/m$ by $f(0)/c(\varepsilon, m)$ to maintain independence of the right side of (8) from $c(\varepsilon, m)$. Then we obtain the claim by applying derivation to the right side of (8) and solving the resulting homogeneous differential equation. □

3 LOWER BOUNDS

Throughout this section, we will prove the following theorem.

Theorem 1. *Assume m machines and slack ε. The partitioning of interval $(0, 1]$ yields k with $\varepsilon \in (\varepsilon_{k-1,m}, \varepsilon_{k,m}]$. Then any deterministic online algorithm for the problem $Pm|online, \varepsilon, immediate| \sum p_j \cdot (1 - U_j)$ has at least competitive ratio*

$$c(\varepsilon, m) = \frac{m \cdot f_k(\varepsilon, m) + 1}{k}.$$

Proof. The high level idea is as follows. The adversary submits a sequence of jobs with the aim of permitting the algorithm to schedule no more than a single job on each machine. The entire procedure consists of three phases that we now describe. In Fig. 2, we show an illustration of the various events that trigger the phases and the adversary's actions for $m = 3$ and $\varepsilon \in [\varepsilon_{1,3}, \varepsilon_{2,3})$. Fig. 3 displays the online schedule and an optimal schedule for the path marked in red in Fig. 2.

Phase 1 is mainly a set-up in which the adversary submits a job $J_1(0, 1, d_1)$. If the algorithm rejects J_1 then the adversary stops submission and the competitive ratio is unbounded. Otherwise, the algorithm selects some start time t. All subsequent jobs will now arrive at time t. The large deadline d_1 allows an optimal schedule to complete J_1 either before t or to start it after the largest deadline of all other jobs.

Phase 2 consists of up to m subphases. , The adversary submits up to $2m$ identical jobs $J_{2,h}(t, p_{2,h}, d_{2,h} = t + 2 \cdot p_{2,h})$ in subphase $h \in \{1, \ldots, m\}$. Due to $\varepsilon \leq 1$, each job observes the slack condition (3). If the algorithm has accepted a job in a subphase then the adversary immediately terminates this subphase and starts the next one. Assume that the algorithm does not accept any job in subphase u. For $u < k$, the adversary stops submission. Otherwise, it continues with phase 3.

Phase 3 consists of up to $m - u + 1$ subphases. In subphase $h \in \{u, \ldots, m\}$, the adversary submits m identical jobs $J_{3,h}(t, p_{3,h} = (f_h(\varepsilon, m) - 1) \cdot p_{2,u}, d_{3,h} = t + p_{2,u} + p_{3,h})$. Immediately after the algorithm has accepted a job in a subphase, the adversary terminates this subphase and starts the next one. If the algorithm does not accept a job in a subphase then the adversary stops submission.

We now turn to the analysis of phases 2 and 3.

Lemma 1. *Let $\beta > 0$ be an arbitrarily small constant. The adversary can select $p_{2,h} \in (1 - \beta, 1)$ such that every machine in the online schedule executes at most one job after the completion of phase 2.*

Proof. The adversary uses an overlap interval I_{h-1} with the initial setting $I_0 = (t + 1 - \beta, t + 1)$ such that all previously allocated jobs execute during this overlap interval. The processing time $p_{2,h}$ is the mid time of the overlap interval I_{h-1} minus the submission time t. Therefore, the algorithm cannot execute any job of type $J_{2,h}$ on a machine with an already allocated job while any allocation of this job to an idle machine overlaps with either the lower or the upper half of interval I_{h-1}. Hence, we have $|I_h| \geq |I_{h-1}|/2$ and $1 - \beta < p_{2,h} < 1$. □

Due to Lemma 1, phase 2 ends after subphase m at the latest. For the remainder of the proof, we assume that β is small enough to not affect the competitive ratio and we drop β from the notation.

Lemma 2. *If phase 2 has stopped after subphase u then the resulting competitive ratio is $(2m+1)/u$.*

Proof. If the adversary stops phase 2 after subphase u then the algorithm has accepted $u-1$ jobs in phase 2 and job J_1 in phase 1. Since the optimal schedule executes the $2m$ jobs of subphase u and job J_1, the competitive ratio is $(2m+1)/u$. □

For phase 3, we establish properties similar to those of Lemma 1 and Lemma 2.

Lemma 3. *Let $u \geq k$ be the final subphase of phase 2. We consider subphase $h \in \{u, \dots, m\}$ of phase 3. Then job $J_{3,h}$ satisfies the slack condition* (3) *and every machine executes at most one job.*

Proof. Due to the monotony of the parameters $f_q(\varepsilon, m)$ and the anchor condition (4), we have

$$d_{3,h} = t + p_{2,u} + p_{3,h} = t + \left(\frac{1}{f_h(\varepsilon,m)-1} + 1\right) \cdot p_{3,h}$$
$$\geq t + \left(\frac{1}{f_m(\varepsilon,m)-1} + 1\right) \cdot p_{3,h} \geq t + (\varepsilon+1) \cdot p_{3,h}.$$

Phase 3 requires (6). Therefore, we have $p_{3,h} \geq p_{2,u}$. Due to Lemma 1, the algorithm needs an idle machine to execute job $J_{3,h}$ if $p_{3,h} = p_{2,u}$ holds. For $p_{3,h} > p_{2,u}$, $d_{3,h}$ requires job $J_{3,h}$ to complete after time $t + p_{2,u}$ and to start at this time at the latest. Therefore, the algorithm cannot execute $J_{3,h}$ on any machine that does not permit execution of $J_{2,u}$ and Lemma 1 yields the claim. □

If phase 3 ends with subphase h then the algorithm has accepted h jobs and occupied h machines due to Lemmas 1 and 3. Therefore, phase 3 ends with subphase m at the latest.

Lemma 4. *If phases 2 and 3 end with subphases u and h, respectively, then the resulting competitive ratio is*

$$\frac{1 + m \cdot f_h(\varepsilon, m)}{u + \sum_{i=u}^{h-1} (f_i(\varepsilon, m) - 1)},$$

Proof. If phases 2 and 3 stop with subphases u and h, respectively then the algorithm has accepted a total processing time of $u + \sum_{i=u}^{h-1}(f_i(\varepsilon,m)-1)$ while the optimal schedule executes job J_1, m jobs $J_{2,u}$, and m jobs $J_{3,h}$ resulting in the claimed competitive ratio. □

Now we are ready to prove the theorem. Our deadline selection in phase 2 and Lemma 3 show that the generated instance is valid, that is, all jobs satisfy the slack condition (3).

To maximize the competitive ratio, the adversary requires the competitive ratios of Lemma 4 to be independent of the final subphase h such that none of the possible competitive ratios is smaller than the others. This selection denies us the opportunity to stop phase 3 at a favorable subphase and leads to our key recursion (5).

Next we turn to the selection of the final subphase u of phase 2 since this selection is our only remaining choice to minimize the competitive ratio. Stopping phase 2 before subphase k is not beneficial since it produces a larger competitive ratio than stopping after subphase k, see Lemma 2. Stopping phase 2 with the final subphase $u > k$ removes a subphase of phase 3 and effectively exchanges a job of phase 3 having a processing time of at least 1 with a job having processing time 1. Therefore, this exchange cannot increase the denominator of the competitive ratio in Lemma 4 and cannot decrease the competitive ratio. □

Algorithm 1 Threshold

```
1:  l(m_h) = 0 for 1 ≤ h ≤ m
2:  for the next job J_j do
3:      update l(m_h) for 1 ≤ h ≤ m
4:      determine d_lim with (9) and (10)
5:      if d_j < d_lim then
6:          reject J_j
7:      else
8:          accept J_j
9:          allocate J_j to the candidate machine with highest load
10:         start J_j after completing the load of this machine
```

4 ONLINE ALGORITHM

In this section, we present a deterministic online algorithm and determine its competitive ratio. We assume a fixed slack value ε and a fixed number of machines m and omit any dependency on ε and m in the variable notation of this section. We still distinguish the various parameters f_q. The next theorem contains our main result.

Theorem 2. *Assume m machines and slack ε. The partitioning of interval $(0,1]$ yields k with $\varepsilon \in (\varepsilon_{k-1,m}, \varepsilon_{k,m}]$. Then Algorithm 1 has a competitive ratio of at most*

$$\frac{m \cdot f_k + 1}{k} \qquad \text{for } k \leq 3$$
$$\frac{m \cdot f_k + 1}{k} + \frac{3-e}{e-1} \qquad \text{for } k > 3.$$

for the problem $Pm|online, \varepsilon, immediate| \sum p_j \cdot (1 - U_j)$.

First we describe the notation used by Algorithm 1. For acceptance, we use a machine-dependent deadline threshold $d_{lim,h}$ for each machine m_h. It is simply the product of f_h (recall the definition from Section 2) times the current (outstanding) load $l(m_h)$ on a machine m_h in addition to the current time t:

$$d_{lim,h} \quad = \quad t + l(m_h) \cdot f_h \text{ for } h \in \{k, \dots, m\} \tag{9}$$

Remember that we index the machines by decreasing loads such that $l(m_{h-1}) \geq l(m_h)$ holds for $h \in \{2, \dots, m\}$. The system deadline d_{lim} for acceptance of a new job is the maximum of the machine-dependent deadlines:

$$d_{lim} \quad = \quad \max_{h \in \{k,\dots,m\}} d_{lim,h} \tag{10}$$

Based on (9) and (10), we can provide another intuitive interpretation of variable k: for $\varepsilon \in (\varepsilon_{k-1}, \varepsilon_k]$, we use only the $m-k+1$ least loaded machines to determine the deadline threshold. Then our online schedule has a minimal total load if the k most loaded machines all have the same (balanced) load.

In Algorithm 1, there is initially no load on any machine (Line 1). After the submission of a new job J_j at time r_j, we update the

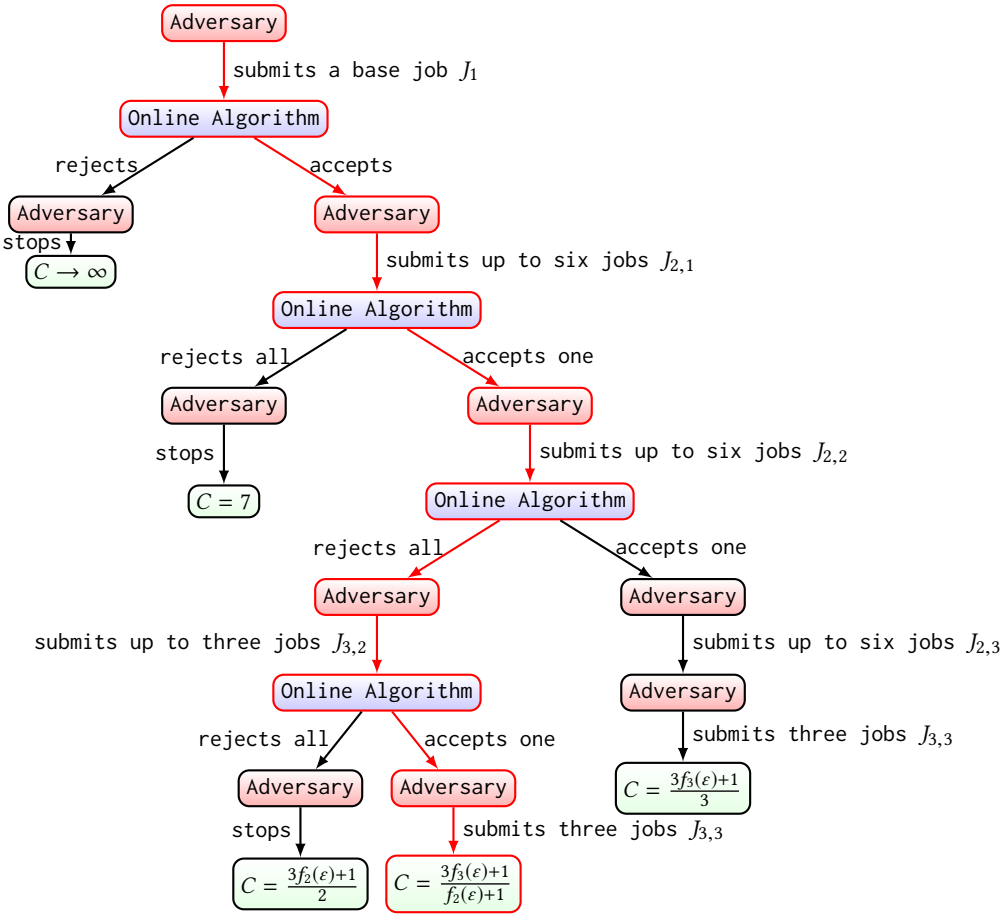

Figure 2: As example, we provide the lower bound process as a decision tree for $m = 3$ and $\varepsilon \in [\varepsilon_{1,3}, \varepsilon_{2,3})$. We have highlighted one path through this tree in red. For this path, we provide the online and the optimal schedule in Fig. 3

(outstanding) load of the machines, adapt the machine indexes, and determine the new threshold d_{lim} using (9) and (10) (Lines 3 and 4). We accept J_j only if its deadline d_j is at least as large as the deadline threshold (Lines 7 and 8). A machine m_i is a candidate machine for an accepted job J_j if it can complete J_j on time ($l(m_i) + p_j \leq d_j$). We allocate the accepted job J_j to the candidate machine with the largest load (Line 9) and start J_j as early as possible, that is, immediately after the completion of the outstanding load on this machine (Line 10).

Since the allocation of a new job to a machine increases the load of this machine, we need a notation that specifies the load values during the run of our algorithm: $l(m_i)|_j$ is the value $l(m_i)$ when making the decision to accept or reject job J_j. We apply the same notation $|_j$ to the thresholds d_{lim} and $d_{lim,h}$. If $d_{lim}|_j = d_{lim,i}|_j$ holds then we say that machine m_i determines d_{lim} when testing job J_j for acceptance. The following claim shows that the algorithm is correct.

Claim 1. *Algorithm 1 completes any accepted job on time.*

Proof Sketch. We simply show that any accepted job will finish on time on machine m_m. □

Assume that Algorithm 1 has produced online schedule S for some problem instance $\mathcal{J}$.

Definition 1. *An interval $[t_s, t_e)$ of S is uncovered if it does not intersect with the interval $[r_j, d_j)$ of any rejected job $J_j \in \mathcal{J}$.*

We transform instance $\mathcal{J}$ into our target instance $\mathcal{J}_{max} \supseteq \mathcal{J}$ with the same schedule S by adding jobs such that Algorithm 1 rejects all added jobs and the total length of all uncovered intervals of schedule S is minimal.

Definition 2. *We obtain the set of all covered intervals of schedule S by removing all uncovered intervals of target instance $\mathcal{J}_{max}$ from interval $[0, \max_{J_j \in \mathcal{J}_{max}} d_j)$.*

Due to Definition 1, every covered interval starts with the submission time of a rejected job. Our transformation guarantees that for the end t_e of any covered interval $[t_s, t_e)$, there is a rejected job $J_j \in \mathcal{J}_{max}$ such that the difference $d_{lim}|_j - t_e$ is positive but arbitrarily small. Therefore, we use the approximation that every covered interval ends with a deadline threshold.

For our performance analysis of schedule S, we consider each covered interval separately. For covered interval $[t_s, t_e)$, $(t_e - t_s) \cdot m$ is an upper bound on its optimal load.

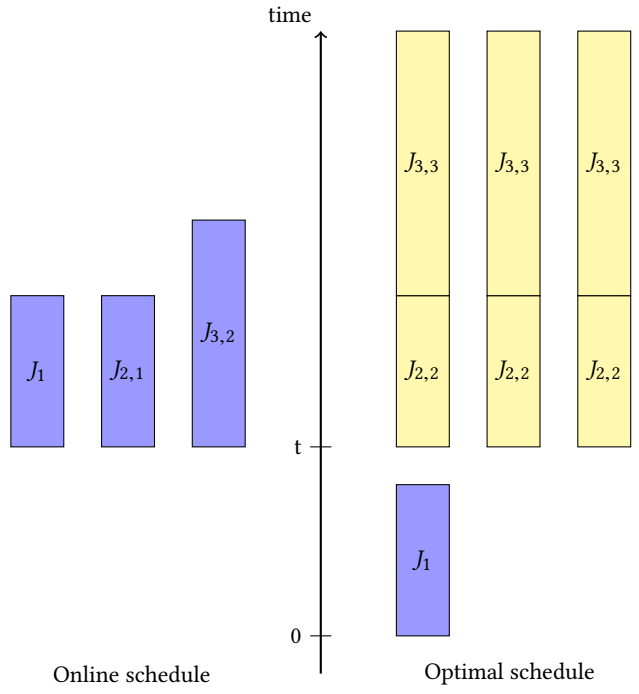

Figure 3: The online algorithm has accepted the blue jobs and rejected the orange jobs. It has started job $J_1(0, 1, d_1)$ at time $t \geq 1$ while the optimal schedule completes J_1 before starting any other job. The processing times and deadlines of jobs $J_{3,2}(t, p_{3,2}, t + p_{2,2} + p_{3,2})$ allow execution on the same machine together with $J_{2,2}$ but not together with $J_{2,1}$. Jobs $J_{3,3}(t, p_{3,3}, t + p_{2,2} + p_{3,3})$ have a tight slack and a processing time that also allows execution on the same machine together with $J_{2,2}$.

Next we introduce some notations and define the performance ratio of an interval of schedule S that follows the concept of the competitive ratio but is restricted to an interval.

$P_{m_i}[t_1, t_2)$ is the total processing time on a machine m_i in interval $[t_1, t_2)$ of schedule S with $P_{m_{i-1}}[t_1, t_2) \geq P_{m_i}[t_1, t_2)$ for $i \in \{2, \ldots, m\}$.

Note that $P_{m_i}[t_1, t_2)$ is a final value of S while $l(m_i)$ is a dynamic value that changes with progression of time and acceptance of new jobs. Machine m_i in $P_{m_i}[t_1, t_2)$ does not necessarily represent a single physical machine due to possible index changes during the execution of Algorithm 1. It merely corresponds to the (potentially changing) machine with ith largest load. Finally, $P_{m_i}[t_1, t_2)$ may only contain parts of a job[3].

$P^-[t_1, t_2)$ describes the total processing time of all jobs in schedule S that any schedule must execute in interval $[t_1, t_2)$ if it also accepts the jobs that contribute to the load in interval $[t_1, t_2)$.

Definition 3. *The performance ratio of an interval $[t_1, t_2)$ is*

$$\mathcal{R}[t_1, t_2) = \frac{m \cdot (t_2 - t_1) - P^-[t_1, t_2)}{\sum_{i=1}^{m} P_{m_i}[t_1, t_2)} + 1. \tag{11}$$

[3]These are remnants of either a job started before t_1 or a job completed after t_2.

It is perhaps instructive to note that $P^-[t_1, t_2) = \sum_{i=1}^{m} P_{m_i}[t_1, t_2)$ if there exists no flexibility in the schedule, that is, all jobs have starting times and deadlines in $[t_1, t_2]$. In this case, $\mathcal{R}[t_1, t_2)$ is merely an upper bound on the ratio between optimal load and S in interval $[t_1, t_2)$. If $P^-[t_1, t_2) = 0$ then deadlines are large, and the optimum schedule moves jobs out of interval $[t_1, t_2)$. In this case, our performance ratio increases by 1.

For our analysis, we define a special form of a covered interval.

Definition 4. *A basic covered interval is a covered interval $[t_s, t_e)$ that satisfies the following two conditions:*

submission *All jobs contributing to $\sum_{i=1}^{m} P_{m_i}[t_s, t_e)$ have at least submission time t_s.*

monotony *If $h < k$ machines are busy at some time $t \in [t_s, t_e)$ then at most h machines are busy at any moment in interval $[t, t_e)$.*

We use $t_k \in [t_s, t_e)$ to denote the first time with less than k busy machines in the basic covered interval $[t_s, t_e)$. Then we have $P_{m_k}[t_s, t_k) = t_k - t_s$ and $P_{m_i}[t_s, t_e) = p = P_{m_i}[t_s, t_s + p)$ for any $i \in \{1, \ldots, k\}$.

The proof of Theorem 2 consist of three parts:

(1) We prove our claim for a basic covered interval $[t_s, t_e)$.
(2) We extend the result to a covered interval $[t_s, t_e)$ that still satisfies the submission condition of Definition 4.
(3) We relax the submission condition of Definition 4 and show its impact on the performance ratio.

The first part is the most important one since the second part is mostly technical and the impact of the third part is minor.

Although its proof is rather simple, the next lemma describes a key property of our algorithm.

Lemma 5. *Assume that Algorithm 1 allocates job J_j to machine m_i with $i \in \{2, \ldots, m\}$.*

- *For any $t > r_j + l(m_{i-1})|_j$, J_j contributes at least processing time $\min\{p_j, t - r_j - l(m_{i-1})|_j\}$ to $P^-[r_j, t)$.*
- *For $i > k$, J_j contributes at least processing time*

$$\max_{h \in \{k, \ldots, i-1\}} \{l(m_h)|_j \cdot (f_h - 1)\}$$

to $P^-[r_j, d_{lim}|_j)$.
- *For $i > k$, we have $l(m_k)|_j < p_j$.*

Proof. If Algorithm 1 does not allocate J_j to machine $i - 1$ then we have $r_j + l(m_{i-1})|_j + p_j > d_j$. The last starting time of J_j in any schedule is $d_j - p_j < r_j + l(m_{i-1})|_j$. The first claim follows immediately.

For $i > k$ and any $h \in \{k, \ldots, i - 1\}$, we have

$$\begin{aligned} r_j + l(m_h)|_j \cdot f_h &\overset{(9)}{=} d_{lim,h}|_j \\ &\overset{(10)}{\leq} d_{lim}|_j \leq d_j < r_j + l(m_h)|_j + p_j \qquad (12) \\ \Rightarrow\ l(m_h)|_j &\overset{(6)}{\leq} (f_h - 1) \cdot l(m_h)|_j < p_j \qquad (13) \end{aligned}$$

The second claim follows from (12) and the first claim. The third claim follows directly from (13) for $h = k$. □

The third claim of Lemma 5 states that any allocation of a job to a machine m_i with $i > k$ guarantees that this machine afterwards

receives an index smaller than k. Therefore, the machine cannot immediately cause a large threshold deadline by using a large factor f_h. Intuitively, this approach guarantees that the growth of a covered interval due to a larger threshold deadline goes along with an increase of the load of the online schedule in the covered interval and therefore limits the increase of the performance ratio.

Since it is difficult determining $P^-[t_s, t_e)$ of a basic covered interval $[t_s, t_e)$ with dynamic load parameters for the first k machines, we use $P_{m_i}[t_s, t_e)$ instead.

Lemma 6. *Assume a basic covered interval $[t_s, t_e)$ and some time $t \in [t_s, t_e]$. Then we have*

$$\sum_{i=2}^{k} \min\{P_{m_i}[t_s, t), t - t_s - P_{m_{i-1}}[t_s, t)\} \leq P^-[t_s, t). \tag{14}$$

Proof. Since $P_{m_h}[t_s, t)$ always refers to interval $[t_s, t)$ in this proof, we omit the interval when using this notation. First, we establish some relationship between $l(m_h)$ and P_{m_h}.

Let J_j be a job with $r_j \in [t_s, t)$. Assume that $l(m_h)|_j > 0$ holds for some $h \in \{1, \ldots, k\}$. Then we have

$$P_{m_h} \begin{cases} = t - t_s & \text{for } t \leq r_j + l(m_h)|_j \\ \geq r_j + l(m_h)|_j - t_s & \text{else} \end{cases}$$

since at least h machines are always busy in interval $[t_s, r_j)$.

Assume that Algorithm 1 has accepted J_j and allocated it to machine m_i with $i \in \{1, \ldots, m\}$. Then J_j completes at $r_j + p_j + l(m_i)|_j$, and we define $c_j = \min\{t - r_j, p_j + l(m_i)|_j\}$. For $h \in \{2, \ldots, i\}$, J_j increases P_{m_h} by

$$\Delta_j(h) = \begin{cases} \min\{0, c_j - l(m_h)|_j\} & \text{for } c_j < l(m_{h-1})|_j \\ l(m_{h-1})|_j - l(m_h)|_j & \text{for } c_j \geq l(m_{h-1})|_j \end{cases} \tag{15}$$

while we have $\Delta_j(h) = 0$ for $h \in \{i+1, \ldots, m\}$.

Assume $h \in \{2, \ldots, k\}$. For $P_{m_h} = r_j + l(m_h)|_j - t_s \geq (t - t_s)/2$, the term $\min\{P_{m_h}, t - t_s - P_{m_{h-1}}\}$ on the left hand side of (14) does not increase. For $P_{m_h} < (t - t_s)/2 \leq P_{m_{h-1}}$, the above term increases by at most $t - t_s - P_{m_{h-1}} - P_{m_h} \leq (t - t_s)/2 - P_{m_h}$. For $P_{m_{h-1}} < (t - t_s)/2$, the increase is at most $\Delta_j(h)$.

We prove (14) with an inductive approach in the sequence of jobs contributing to interval $[t_s, t)$. Clearly, (14) holds before the first job contributes to interval $[t_s, t)$. We assume that (14) holds before the acceptance of J_j and discuss the impact of the allocation of J_j.

Since (15) yields $\sum_{h=2}^{k} \Delta_j(h) \leq p_j$, the first claim of Lemma 5 guarantees (14) for $p_j \leq t - r_j - l(m_{i-1})|_j$.

We consider $p_j > t - r_j - l(m_{i-1})|_j$ and determine the largest value $q \in \{1, \ldots, m\}\}$ with $r_j + l(m_q)|_j - t_s \geq (t - t_s)/2$. Since (14) holds for $q \geq i$ and $q \geq k$, we assume $q < \min\{i, k\}$.

Assume $t - r_j - l(m_{i-1})|_j > (t - t_s)/2$. If there is no q then $\sum_{h=2}^{k} \Delta_j(h) \leq l(m_1)|_j - l(m_k)|_j \leq (t - t_s)/2 < t - r_j - l(m_{i-1})|_j$ and (14) holds. Otherwise, term $\min\{P_{m_{q+1}}, t - t_s - P_{m_q}\}$ increases by at most $t - t_s - P_{m_q} - P_{m_{q+1}} \leq (t - t_s)/2 - P_{m_{q+1}}$. Then the left hand side of (14) increases at most by $(t - t_s)/2 - P_{m_{q+1}} + l(m_{q+1})|_j - l(m_k)|_j \leq (t - t_s)/2$ and (14) is valid.

For $t - r_j - l(m_{i-1})|_j \leq (t - t_s)/2$, we have $r_j + l(m_{i-1})|_j - t_s \geq t - t_s - (t - t_s)/2 = (t - t_s)/2$ and $q = i - 1$. Then the increase of term $\min\{P_{m_i}, t - t_s - P_{m_{i-1}}\}$ is limited by $t - t_s - P_{m_{i-1}} = t - r_j - l(m_{i-1})|_j$ and (14) is valid. □

Now we analyze the performance ratio of a basic covered interval $[t_s, t_e)$. In particular, the next lemma considers the case $t_k - t_s < t_e - t_s \leq f_k \cdot (t_k - t_s)$. This case occurs if machine m_k has caused the largest deadline threshold in $[t_s, t_e)$. The other cases are useful if another machine is responsible for the largest deadline threshold.

Lemma 7. *Assume a basic covered interval $[t_s, t_e)$ with $t \in [t_s, t_e]$. Then we have*

$$\mathcal{R}[t_s, t) \leq \begin{cases} \frac{m \cdot \frac{t - t_s}{t_k - t_s} + 1}{k} & \text{for } t \geq 2t_k - t_s \\ \frac{(m - k + 1) \cdot \frac{t - t_s}{t_k - t_s} + 2k - 1}{k} & \text{for } t_k \leq t < 2t_k - t_s \\ \frac{m \cdot f_k + 1}{k} & \text{for } \frac{t - t_s}{t_k - t_s} \leq f_k \end{cases}$$

Proof. Since we always refer to interval $[t_s, t)$ in this proof, we omit all references to this interval in the notation P_{m_i}.

First we consider $t \geq t_k$: remember that we have $P_{m_k} = t_k - t_s$ in a basic covered interval. Due to Lemma 6, any machine m_i with $i \in \{2, \ldots, k\}$ contributes at least $t - t_s - P_{m_{i-1}}$ to $P^-[t_s, t)$ if $P_{m_{i-1}} + P_{m_i} \geq t - t_s$ holds. If there are $h < k - 1$ such machines, then any machine m_i with $i \in \{h + 2 \ldots, k\}$ contributes P_{m_i} to $P^-[t_s, t)$. Then we obtain

$$\begin{aligned} \mathcal{R} &\leq \frac{(m - h) \cdot (t - t_s) + \sum_{i=1}^{h} P_{m_i} - \sum_{i=h+2}^{k} P_{m_i}}{\sum_{i=1}^{k} P_{m_i}} + 1 \\ &\leq \frac{(m - h) \cdot (t - t_s) + 2 \cdot \sum_{i=1}^{h} P_{m_i} + P_{m_{h+1}}}{\sum_{i=1}^{k} P_{m_i}} \\ &\leq \frac{(m - h) \cdot \frac{t - t_s}{t_k - t_s} \cdot P_{m_k} + 2 \cdot \sum_{i=1}^{h} P_{m_i} + P_{m_{h+1}}}{\sum_{i=1}^{k} P_{m_i}} \\ &\leq \frac{(m - h) \cdot \frac{t - t_s}{t_k - t_s} \cdot P_{m_k} + (2 \cdot h + 1) \cdot P_{m_k}}{k \cdot P_{m_k}} \\ &= \frac{(m - h) \cdot \frac{t - t_s}{t_k - t_s} + 2 \cdot h + 1}{k}. \end{aligned}$$

For $t - t_s \geq 2 \cdot (t_k - t_s)$, we select $h = 0$ and obtain the first claim while the second claim results from $h = k - 1$ for $t - t_s < 2 \cdot (t_k - t_s)$. The first claim and $f_k \geq 2$ lead to the third claim. □

To conclude our analysis of basic covered intervals, we must extend Lemma 7 to consider the case that a machine m_i with $i \in \{k + 1, \ldots, m\}$ determines the largest threshold of the basic covered interval $[t_s, t_e)$. Using Lemma 5, we argue that only a specific sequence of accepted jobs causes such situation.

Let us assume that $t_e = d_{lim,i}|_j$ caused the rejection of job J_j. At time r_j, there must be a load on machines m_1 to m_i. Remember that after Algorithm 1 has allocated a new job to a machine m_h with $h > k$, the new index of this machine is less than k, see Lemma 5, and the indexes of machines m_k to m_{h-1} increase by 1 each. Therefore, the physical machine with index i at time r_j once had index k, and a sequence of jobs $J_{j_k}, \ldots, J_{j_{i-1}}$ has increased its index step by step. More precisely, this machine had index h when Algorithm 1 has accepted job J_{j_h} and allocated it to a machine with a larger index than h. During time interval $[r_{j_k}, r_j)$, there was no new load allocation to this machine, see Lemma 5.

The following properties hold for this sequence:

(1) Algorithm 1 does not allocate any two jobs of this sequence to the same machine since any index increase is only an increment and the machine that receives job J_h had a previous index larger than h and has a new index smaller than k.
(2) $l(m_h)|_{j_h} = l(m_i)|_j + r_j - r_{j_h}$.
(3) Due to Lemma 5 and $d_{lim}|_{j_h} \leq d_{lim}|_j$, J_{j_h} contributes at least $l(m_h)|_{j_h} \cdot (f_h - 1)$ to $P^-[r_{j_h}, d_{lim}|_j)$.

Lemma 8. *Let $[t_s, t_e)$ be a basic covered interval and J_j be a rejected job with $t_e = d_{lim,i}|_j = d_{lim}|_j$ and $i \in \{k+1, \ldots, m\}$. Then we have*

$$\mathcal{R}[t_s, t_e) \leq \frac{m \cdot f_i + 1}{\sum_{h=k}^{i-1}(f_h - 1) + k} = \frac{m \cdot f_k + 1}{k}.$$

Proof. We use the previously described sequence $J_{j_k}, \ldots, J_{j_{i-1}}$. Whenever Algorithm 1 allocates job J_{j_h} with $h \in \{k, \ldots, i-1\}$, it cannot allocate it to machine m_h. Therefore, the contribution of this job sequence to $P^-[r_j, d_{lim}|_j)$ is at least

$$\begin{aligned} l(m_i)|_j \cdot \sum_{h=k}^{i-1}(f_h - 1) &= \sum_{h=k}^{i-1}\left(l(m_h)|_{j_h} - r_j + r_{j_h}\right) \cdot (f_h - 1) \\ &\overset{(6)}{\leq} \sum_{h=k}^{i-1} l(m_h)|_{j_h} \cdot (f_h - 1). \end{aligned}$$

For $t_e - t_s \leq f_k \cdot (r_j + l(m_i)|_j - t_s) \leq f_k \cdot (t_k - t_s)$, we use the third claim of Lemma 7 and obtain $\mathcal{R}[t_s, t_e) \leq (f_k \cdot m + 1)/k$.

Therefore, we assume $t_e - t_s > f_k \cdot (r_j + l(m_i)|_j - t_s)$. There are at least k machines that are always busy in interval $[t_s, r_j)$. We temporarily remove all the parts of jobs from the sequence $J_{j_k}, \ldots, J_{j_{i-1}}$ that are scheduled at r_j or later and all jobs scheduled on the same physical machine after a job of the sequence. Since a part $r_j - r_{j_h}$ of job J_{j_h} contributes to $P^-[r_{j_h}, d_{lim}|_j)$ but not to $P^-[r_j, d_{lim}|_j)$, we can apply Lemma 6 to the remaining schedule and obtain a partial performance ratio

$$\mathcal{R}_p[t_s, t_e) \leq \frac{m \cdot \frac{t_e - t_s}{r_j + l(m_i)|_j - t_s} + 1}{k} \leq \frac{m \cdot \frac{f_i \cdot l(m_i)|_j}{l(m_i)|_j} + 1}{k}$$

due to the first claim of Lemma 7. We re-introduce the removed parts of the jobs of the sequence and obtain our final result

$$\begin{aligned} \mathcal{R}[t_s, t_e) &\leq \frac{m \cdot f_i \cdot l(m_i)|_j + l(m_i)|_j}{(\sum_{h=k}^{i-1}(f_h - 1) + k) \cdot l(m_i)|_j} \\ &\leq \frac{m \cdot f_i + 1}{\sum_{h=k}^{i-1}(f_h - 1) + k} \overset{(5)}{=} \frac{m \cdot f_k + 1}{k}. \end{aligned}$$

□

To generalize our results to all covered intervals, we define a subinterval $[t_h, t_{h+1})$ of a covered interval with some time $t_{h,h+1} \in (t_h, t_{h+1})$ such that in the online schedule, at least k machines are always busy in interval $[t_h, t_{h,h+1})$ and less than k machines are busy at any time instance in interval $[t_{h,h+1}, t_{h+1})$, respectively. Further, at the submission of some job $J_{h,h+1}$ with submission time $r_{h,h+1} \in [t_h, t_{h,h+1}]$, some machine determines the largest deadline threshold $d_{h,h+1}$ generated in this subinterval.

Since at the end of a covered interval $[t_s, t_e)$, less than k machines are busy, we can partition any covered interval into a sequence of such subintervals. We prove the generalization of our results by applying induction in the sequence of subintervals in the partition of a covered interval.

Lemma 9. *Let $[t_s, t_e)$ be a covered interval such that every job contributing to $[t_s, t_e)$ has at least submission time t_s. Then we have*

$$\mathcal{R}[t_s, t_e) \leq \frac{m \cdot f_k + 1}{k}.$$

Proof. Let $[t_1, t_2)$ be a subinterval in the partition of covered interval $[t_s, t_e)$. For the induction, we assume that the claim holds for interval $[t_s, d_{1,2})$ and that the validity of the claim does not require the consideration of any job submitted after $r_{1,2}$.

For $t_s = t_1$, interval $[t_s, t_2)$ only contains a single subinterval. If we ignore all jobs submitted after $J_{1,2}$ then $[t_s, d_{1,2})$ is a basic covered interval and Lemmas 7 and 8 yield the claim.

Let $[t_2, t_3)$ be the next subinterval in the partition of $[t_s, t_e)$. For $d_{1,2} \geq d_{2,3}$, the claim holds due to our assumption and we proceed with the next subinterval of the partition until we find a subinterval $[t_3, t_4)$ with $d_{1,2} < d_{3,4}$. Note that $t_2 = t_3$ is possible.

We must show that we can apply Lemmas 7 and 8 to subinterval $[t_3, t_4)$. Since less than k machines are busy just before t_3, we must only show the validity of Lemma 6 for subinterval $[t_3, d_{3,4})$ when ignoring all jobs submitted after $J_{3,4}$.

The validity of Lemma 6 is obvious for all jobs unless they have a release date less than t_3 and complete after t_3. Consider a job J_j with completion time $C_j > t_3 > r_j$ in the online schedule. We use Δ_j to denote the difference between its contribution to $P^-[t_s, d_{3,4})$ and to $P^-[t_s, t_3)$. Note that we have $P^-[t_s, t_3) + P^-[t_3, d_{3,4}) < P^-[t_s, d_{3,4})$ unless d_j is sufficiently large such that J_j does not contribute to $P^-[t_s, d_{3,4})$. We increase $P^-[t_3, d_{3,4})$ by allocating the whole amount Δ_j to $P^-[t_3, d_{3,4})$. This allocation does not change $P^-[t_s, d_{1,2})$ and the validity of the claim for interval $[t_s, d_{1,2})$. Next, we assume a job J'_j with release date t_3, processing time $\min\{p_j, C_j - t_3\}$, and deadline d_j. Job J'_j replaces job J_j in interval $[t_3, \max\{C_j, d_{3,4}\})$ of the online schedule. We consider its contribution Δ'_j to $P^-[t_3, d_{3,4})$ and compare it to Δ_j. Clearly, $\Delta'_j \leq \Delta_j$ holds. Therefore, Lemma 6 is valid for subinterval $[t_3, d_{3,4})$ due to the allocation of Δ_j. □

Next, we relax the submission time restriction of a basic covered interval, that is, Algorithm 1 can execute jobs with a submission time earlier than the start time of a covered interval within this covered interval although some machines are free earlier to execute these jobs. We call this property of Algorithm 1 a *delayed* execution. Such delayed execution does not contribute to $P^-[t_s, t_e)$ of covered interval $[t_s, t_e)$ and therefore leads to a larger performance ratio of the covered interval. Clearly, any delayed execution can only occur on machines that are busy for some time before the start of the covered interval. However, the impact of a delayed execution is only minor.

Lemma 10. *For $k \in \{1, 2, 3\}$, delayed execution does not increase the performance ratio of Lemmas 7 and 8.*

Proof. Due to Lemma 5, only machines m_i with $i \in \{2, \ldots, m\}$ can contribute to $P^-[t_s, t_e)$ for a covered interval $[t_s, t_e)$. Since at most $k - 1$ machines are busy at any time in an uncovered interval, any delayed execution cannot occur on machine k. Therefore, there is no impact on the performance ratio for $k = 1$ and $k = 2$. Since

the case $k = 3$ requires more work without providing more insight, we skip it. □

Lemma 11. *Delayed execution caused by Algorithm 1 can increase the performance ratio of Lemma 7 by at most* $(3 - e)/(e - 1) \approx 0.164$.

Proof Sketch. We ignore that machines m_1 and m_k cannot support delayed execution and apply a continuous extension assuming $m \to \infty$. Then we optimize the resulting continuous optimization problem. □

We conclude the proof of Theorem 2 by combining our previous results.

Proof of Theorem 2. Lemma 9 specifies the performance ratio for a covered interval if for every job contributing to this interval, its submission time is at least the start time of the covered interval. For $k \leq 3$, this result also holds if we remove the submission time restriction, see Lemma 10. For $k > 3$, Lemma 11 shows that removing this restriction adds at most 0.164 to the performance ratio.

The competitive ratio of the problem cannot exceed the maximum performance ratio over all covered intervals. □

Finally, we remark that applying the *static-classification-and-select* technique, see, for instance, [1, 24, 26, 27], we can use the algorithm for parallel machines to obtain a randomized algorithm for a single machine. Our general idea is the simulation of m parallel machines followed by scheduling the jobs of a randomly selected machine. Then the load of a parallel schedule will always be within a constant factor of the load of an optimal single-machine schedule if m is large enough. Since the technique is nowadays fairly standard, we only state the result.

Corollary 1. *There exists a randomized* $O\left(\log \frac{1}{\varepsilon}\right)$ *competitive algorithm for* $1|online, \varepsilon, immediate| \sum p_j \cdot (1 - U_j)$.

REFERENCES

[1] B. Awerbuch, Y. Azar, and S.A. Plotkin. 1993. Throughput-Competitive On-Line Routing. In *Proc. of the 34th Annual Symposium on Foundations of Computer Science (FOCS)*. 32–40.

[2] Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. Naor, and J. Yaniv. 2015. Truthful Online Scheduling with Commitments. In *Proc. of the Sixteenth ACM Conference on Economics and Computation (EC)*. 715–732.

[3] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. 1999. Bandwidth Allocation with Preemption. *SIAM J. Comput.* 28, 5 (1999), 1806–1828.

[4] S.K. Baruah, J.R. Haritsa, and N. Sharma. 1994. On-Line Scheduling to Maximize Task Completions. In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS '94), San Juan, Puerto Rico, December 7-9, 1994*. 228–236. https://doi.org/10.1109/REAL.1994.342713

[5] S.K. Baruah, G. Koren, B. Mishra, A. Raghunathan, L.E. Rosier, and D.E. Shasha. 1991. On-line Scheduling in the Presence of Overload. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*. 100–110.

[6] S. K. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. E. Rosier, D. Shasha, and F. Wang. 1992. On the Competitiveness of On-Line Real-Time Task Scheduling. *Real-Time Systems* 4, 2 (1992), 125–144.

[7] D.P. Bunde and M.H. Goldwasser. 2010. Dispatching Equal-Length Jobs to Parallel Machines to Maximize Throughput. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*. 346–358.

[8] L. Chen, F. Eberle, N. Megow, K. Schewior, and C. Stein. 2019. A general framework for handling commitment in online throughput maximization. In *Proccedings of the 20th IPCO 2019, Ann Arbor, MI, USA*. 141–154.

[9] M. Chrobak, W. Jawor, J. Sgall, and T. Tichý. 2007. Online Scheduling of Equal-Length Jobs: Randomization and Restarts Help. *SIAM J. Comput.* 36, 6 (2007), 1709–1728.

[10] B. DasGupta and M.A. Palis. 2001. Online real-time preemptive scheduling of jobs with deadlines on multiple machines. *Journal of Scheduling* 4, 6 (2001), 297–312.

[11] J. Ding, T. Ebenlendr, J. Sgall, and G. Zhang. 2007. Online Scheduling of Equal-Length Jobs on Parallel Machines. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*. 427–438.

[12] J. Ding and G. Zhang. 2006. Online Scheduling with Hard Deadlines on Parallel Machines. In *Algorithmic Aspects in Information and Management, Second International Conference, AAIM 2006, Hong Kong, China, June 20-22, 2006, Proceedings*. 32–42.

[13] T. Ebenlendr and J. Sgall. 2008. A Lower Bound for Scheduling of Unit Jobs with Immediate Decision on Parallel Machines. In *Approximation and Online Algorithms, 6th International Workshop, WAOA 2008, Karlsruhe, Germany, September 18-19, 2008. Revised Papers*. 43–52.

[14] F. Eberle, N. Megow, and K. Schewior. 2019. Optimally handling commitment issues in online throughput maximization. *CoRR* abs/1912.10769 (2019). arXiv:1912.10769 http://arxiv.org/abs/1912.10769

[15] S.P.Y. Fung. 2014. Online scheduling with preemption or non-completion penalties. *J. Scheduling* 17, 2 (2014), 173–183.

[16] J.A. Garay, J. Naor, B. Yener, and P. Zhao. 2002. On-line Admission Control and Packet Scheduling with Interleaving. In *Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. 94–103.

[17] S.A. Goldman, J. Parwatikar, and S. Suri. 2000. Online Scheduling with Hard Deadlines. *Journal of Algorithms* 34, 2 (2000), 370 – 389.

[18] M.H. Goldwasser. 1999. Patience is a Virtue: The Effect of Slack on Competitiveness for Admission Control. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 396–405.

[19] M.H. Goldwasser. 2003. Patience is a virtue: the effect of slack on competitiveness for admission control. *Journal of Scheduling* 6, 2 (2003), 183–211.

[20] M.H. Goldwasser and B. Kerbikov. 2003. Admission Control with Immediate Notification. *J. Scheduling* 6, 3 (2003), 269–285.

[21] M.H. Goldwasser and M. Pedigo. 2008. Online nonpreemptive scheduling of equal-length jobs on two identical machines. *ACM Trans. Algorithms* 5, 1 (2008), 2:1–2:18.

[22] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In *Discrete Optimization II*, P.L. Hammer, E.L. Johnson, and B.H. Korte (Eds.). Annals of Discrete Mathematics, Vol. 5. Elsevier, 287 – 326.

[23] J.H. Kim and K.-Y. Chwa. 2001. On-Line Deadline Scheduling on Multiple Resources. In *Proc. of the 7th Annual International Conference of Computing and Combinatorics (COCOON)*. 443–452.

[24] C.-Y. Koo, T. W. Lam, T.-W. Ngan, and K.-K. To. 2002. Extra processors versus future information in optimal deadline scheduling. In *SPAA*. 133–142.

[25] G. Koren and D.E. Shasha. 1995. D^over: An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems. *SIAM J. Comput.* 24, 2 (1995), 318–339.

[26] J. Lee. 2003. Online deadline scheduling: multiple machines and randomization. In *Proc. of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 19–23.

[27] R.J. Lipton and A. Tomkins. 1994. Online Interval Scheduling. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia*. 302–311.

[28] B. Lucier, I. Menache, J. Naor, and J. Yaniv. 2013. Efficient online scheduling for deadline-sensitive jobs: extended abstract. In *Proc. of the 25th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 305–314.

[29] C. Schwiegelshohn and U. Schwiegelshohn. 2016. The Power of Migration for Online Slack Scheduling. In *24th Annual European Symposium on Algorithms (ESA)*. 75:1–75:17.

[30] M. Skutella and J. Verschae. 2016. Robust Polynomial-Time Approximation Schemes for Parallel Machine Scheduling with Job Arrivals and Departures. *Math. Oper. Res.* 41, 3 (2016), 991–1021.

[31] N. Thibault and C. Laforest. 2009. Online time constrained scheduling with penalties. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*. 1–8.