Optimally Handling Commitment Issues in Online Throughput Maximization

Franziska Eberle
Department for Mathematics and Computer Science, University of Bremen, Germany
feberle@uni-bremen.de

Nicole Megow
Department for Mathematics and Computer Science, University of Bremen, Germany
nicole.megow@uni-bremen.de

Kevin Schewior
Universität zu Köln, Department of Mathematics and Computer Science, Germany
kschewior@gmail.com

─── **Abstract** ───

We consider a fundamental online scheduling problem in which jobs with processing times and deadlines arrive online over time at their release dates. The task is to determine a feasible preemptive schedule on m machines that maximizes the number of jobs that complete before their deadline. Due to strong impossibility results for competitive analysis, it is commonly required that jobs contain some *slack* $\varepsilon > 0$, which means that the feasible time window for scheduling a job is at least $1 + \varepsilon$ times its processing time. In this paper, we answer the question on how to handle commitment requirements which enforce that a scheduler has to guarantee at a certain point in time the completion of admitted jobs. This is very relevant, e.g., in providing cloud-computing services and disallows last-minute rejections of critical tasks. We present the first online algorithm for handling commitment on parallel machines for arbitrary slack ε. When the scheduler must commit upon starting a job, the algorithm is $\Theta(\frac{1}{\varepsilon})$-competitive. Somewhat surprisingly, this is the same optimal performance bound (up to constants) as for scheduling without commitment on a single machine. If commitment decisions must be made before a job's slack becomes less than a δ-fraction of its size, we prove a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon - \delta})$ for $0 < \delta < \varepsilon$. This result nicely interpolates between commitment upon starting a job and commitment upon arrival. For the latter commitment model, it is known that no (randomized) online algorithms admits any bounded competitive ratio.

1 Introduction

We consider the following fundamental online scheduling model: jobs from an unknown job set arrive online over time at their *release dates* r_j. Each job has a *processing time* $p_j \geq 0$ and a *deadline* d_j. There are m identical machines to process these jobs or a subset of them. A job is said to *complete* if it receives p_j units of processing time within the interval $[r_j, d_j)$. We allow *preemption*, i.e., the processing of a job can be interrupted at any time. We distinguish schedules *with* and *without migration*. If we allow migration, then a preempted job can resume processing on any machine whereas it must run completely on the same machine otherwise. In the three-field notation this problem is $P \,|\, \text{online } r_j, \text{pmtn} \,|\, \sum(1 - U_j)$ [16].

In a feasible schedule, two jobs are never processing at the same time on the same machine. The number of completed jobs in a feasible schedule is called *throughput*. The task is to find a feasible schedule with maximum throughput.

As jobs arrive online, we cannot hope to find an optimal schedule [12]. To assess the performance of online algorithms, we resort to standard *competitive analysis*. This means, we compare the throughput of an online algorithm with the throughput achievable by an optimal offline algorithm that knows the job set in advance.

It is well-known that "tight" jobs with $d_j - r_j \approx p_j$ prohibit competitive online decision making as jobs must start immediately and do not leave a chance for observing online arrivals [6]. Thus, it is commonly required that jobs contain some *slack* $\varepsilon > 0$, i.e., every job j satisfies $d_j - r_j \geq (1 + \varepsilon)p_j$. The competitive ratio of our online algorithm will be a function of ε; the greater the slack, the better should the performance of our algorithm be. This slackness parameter has been considered in previous work, e.g., in [2, 4, 8, 14, 15, 24, 26]. Other results for scheduling with deadlines use speed scaling, which can be viewed as another way to add slack to the schedule, e.g., [1, 3, 17, 18, 25].

In this paper, we focus on the question how to handle *commitment* requirements in online throughput maximization. Modeling commitment addresses the issue that a good-throughput schedule may abort jobs close to their deadlines in favor of many shorter and more urgent tasks [13], which may not be acceptable for the job owner. Consider a company that starts outsourcing mission-critical processes to external clouds and that needs a guarantee that jobs complete before a certain time point when they cannot be moved to another computing cluster anymore. In other situations, a commitment to complete jobs might be required even earlier just before starting the job, e.g., for a faultless copy of a database [8].

Different commitment models have been formalized [2, 8, 24]. The requirement to commit at a job's release date has been ruled out for online throughput maximization by strong impossibility results (even for randomized algorithms) [8]. We distinguish *(i) commitment upon job admission* and *(ii) δ-commitment*. In the first model, an algorithm may discard a job any time before its start, we say its admission. This reflects a situation such as the faultless copy of a database. In the second model, δ-commitment, an online algorithm must commit to complete a job when its remaining slack is not less than a δ-fraction of the job size, for $0 < \delta < \varepsilon$. The latest time for committing to job j is then $d_j - (1 + \delta)p_j$. This models an early enough commitment (parameterized by δ) for mission-critical jobs. Recently, a first unified approach has been presented for these models in [8]. Gaps in the performance bounds remained and it was left open if scheduling with commitment is even "harder" than without commitment.

In this work, we give tight results for online throughput maximization on parallel machines and answer the "hardness" question to the negative. We give an algorithm that achieves the provably best competitive ratio (up to constants) for the aforementioned commitment models. Somewhat surprisingly, we show that the same competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$ can be achieved for both, scheduling *without* commitment and *with* commitment upon admission. Further, our algorithm does not require job migration. For parallel machines, our algorithm is the first online algorithm with bounded competitive ratio for arbitrary slack parameter ε.

Previous results

Preemptive online scheduling with hard deadlines and models for admission control have been studied rigorously, see, e.g., [5, 14, 15] and references therein. Already in the 90s several impossibility results were shown for jobs without slack [6, 7, 21–23]. The only positive result independent of slack for online throughput maximization without commitment seems to be a randomized $\mathcal{O}(1)$-competitive single-machine algorithm [20]. The best possible deterministic algorithm in this setting is $\Theta(1/\varepsilon)$-competitive for instances with ε-slack [8].

Throughput maximization with commitment has attracted researchers more recently [2, 8, 24]. We summarize the state-of-the art for the particular problem of online throughput maximization with commitment. For a single machine, Chen et al. [8] presented a universal algorithmic framework, which achieved bounded competitive ratios for several commitment models and even the tight result for scheduling without commitment. More precisely, their algorithm is $\mathcal{O}(1/\varepsilon^2)$-competitive for commitment upon admission and $\mathcal{O}(\varepsilon/((\varepsilon - \delta)\delta^2))$-competitive, for $0 < \delta < \varepsilon$, in the δ-commitment model. This improved on an earlier algorithm by Azar et al. [2] for the δ-*commitment model* (in the context of truthful mechanisms for a weighted setting) that is $\mathcal{O}(1/\varepsilon^2)$-competitive if the slack ε is sufficiently large. Chen et al. showed a lower bound of $\Omega(1/\varepsilon)$ for deterministic scheduling algorithms without commitment, which is tight in that model and also holds for the more restrictive commitment models. A significant gap between lower and upper bounds remained. On parallel machines, there is a competitive algorithm for online throughput maximization with commitment if the slack ε is sufficiently large [2].

In a natural generalization of our problem, jobs have associated individual weights and we aim for a schedule with maximum weighted throughput. The special case with each job j satisfying $w_j = p_j$ (aka machine utilization) is well understood. A simple greedy algorithm achieves the best possible competitive ratio $\Theta(1/\varepsilon)$ [11, 14] on a single machine in both commitment models, even for commitment upon arrival. For scheduling with commitment on m parallel identical machines there is an $\mathcal{O}(\sqrt[m]{1/\varepsilon})$-competitive algorithm and an almost matching lower bound [26]. It is worth mentioning that machine utilization without commitment even allows for constant competitive ratios independent of slack [6, 21, 22, 27]. General weighted (and even unweighted) throughput maximization is much less tractable. For general weights, there is no bounded competitive ratio possible in any of the aforementioned commitment models [2, 8, 24]. For weighted throughput maximization without commitment requirements there is an $\mathcal{O}(1/\varepsilon^2)$-competitive online algorithm [24].

The power of migration has been investigated in several contexts: While it is typically large in online scheduling, cf. machine-utilization [26] and resource-minimization [9, 10], we show that, in our online setting, the guarantees that can be achieved by migratory and non-migratory algorithms are within a constant factor, similar to the offline problem [19].

Our results and techniques

Our main result is one algorithm that is best possible (up to constant factors) for online throughput maximization with and without commitment on parallel identical machines. Our algorithm does not migrate jobs and still achieves a competitive ratio that matches the general lower bound for migratory algorithms.

For scheduling with commitment upon admission, we give an (up to constant factors) optimal online algorithm with competitive ratio $\Theta(1/\varepsilon)$. For scheduling with δ-commitment, our result interpolates between the models commitment upon starting a job and commitment upon arrival. If $\delta \leq \varepsilon/2$, the competitive ratio is $\Theta(1/\varepsilon)$ which is best possible [8]. For $\delta \to \varepsilon$, the commitment requirements essentially implies commitment upon job arrival which has unbounded competitive ratio [8]. Note that this is the first online algorithm with bounded competitive ratio for arbitrary slackness parameter ε.

▶ **Theorem 1.** *Consider throughput maximization on parallel identical machines with or without migration. There is an $\mathcal{O}(\frac{1}{\varepsilon - \delta'})$-competitive online algorithm with commitment where $\delta' = \varepsilon/2$ in the commitment upon admission model and $\delta' = \max\{\delta, \varepsilon/2\}$ in the δ-commitment model.*

Clearly, scheduling with commitment is more restrictive than without commitment. Hence, our algorithm is also (up to constants) optimal for the problem $P \mid \text{online } r_j, \text{pmtn} \mid \sum(1 - U_j)$ without any commitment requirements as its competitive ratio matches the lower bound [8].

▶ **Theorem 2.** *There is a $\Theta(1/\varepsilon)$-competitive algorithm for online throughput maximization on parallel identical machines without commitment requirements, with and without migration.*

The challenge in online scheduling with commitment is that, once we committed to complete a job, the remaining slack of this job has to be spent very carefully. The key is a job admission scheme which is implemented by different parameters. The high-level objectives are:

(i) never start a job for the first time if its remaining slack is too small (parameter δ),

(ii) during the processing of a job, admit only significantly shorter jobs (parameter γ), and

(iii) for each admitted shorter job, block some time period (parameter β) during which no other jobs of similar size are accepted.

While the first two goals are quite natural and have been used before [8, 24], the third goal is crucial for our new tight result. The intuition is the following: suppose we committed to complete a job with processing time 1 and have only a slack of $\mathcal{O}(\varepsilon)$ left before the deadline of this job. Suppose that c substantially smaller jobs of size $1/c$ arrive where c is the competitive ratio we aim for. On the one hand, if we do not accept any of them, we cannot hope to achieve c-competitiveness. On the other hand, accepting too many of them fills up the slack and, thus, leaves no room for even smaller jobs. The idea is to keep the flexibility for future small jobs by grouping jobs of similar size (within a factor two) into classes. This gives the fine-grained classification of jobs which is crucial for our new tight result. We distinguish two time periods with different class structures that guide acceptance. During the *scheduling interval* of a job j, we have a more restrictive acceptance scheme that ensures the completion of j whereas in the *blocking period* we guarantee the completion of previously accepted jobs. In contrast, the previous algorithm in [8] uses one long region with a uniform acceptance threshold and is then too conservative in accepting jobs.

As a key contribution on the technical side, we give a strong bound on the processing volume of any feasible non-migratory schedule in terms of the accepted volume of a certain class of online algorithms. It is crucial for our analysis and might be of independent interest.

2   The blocking algorithm

In this section, we describe the *blocking algorithm* which handles scheduling with commitment. We assume that the slackness constant $\varepsilon > 0$ and, in the δ-commitment model, $0 < \delta < \varepsilon$ are given. If δ is not part of the input or if $\delta \leq \varepsilon/2$, we set $\delta = \frac{\varepsilon}{2}$.

The algorithm never migrates jobs between machines, i.e., a job is only processed by the machine that initially started to process it, we say the job has been *admitted* to this machine. Moreover, our algorithm commits to completing a job upon admission. Hence, its remaining slack has to be spent very carefully on admitting other jobs while being competitive. As our algorithm does not migrate jobs, it transfers the admission decision to the shortest admitted and not yet completed job on each machine. Then, a job only admits significantly shorter jobs and prevents the admission of too many jobs of similar size. To this end, the algorithm maintains two types of intervals for each admitted job, a *scheduling interval* and a *blocking period*. A job can only be processed in its scheduling interval. Thus, it has to complete in this interval while admitting other jobs. Job j only admits jobs that are smaller by a factor of $\gamma = \frac{\delta}{16} < 1$. For an admitted job k, job j creates a blocking period of length at most βp_k, where $\beta = \frac{16}{\delta}$, which blocks the admission of similar-length jobs (cf. Figure 1).

For **scheduling**, the algorithm follows the *Shortest Processing Time (SPT)* order for the set of uncompleted jobs assigned to a machine, which is independent of the admission scheme. SPT ensures that, in the blocking periods of any job k admitted by j, j has highest priority.
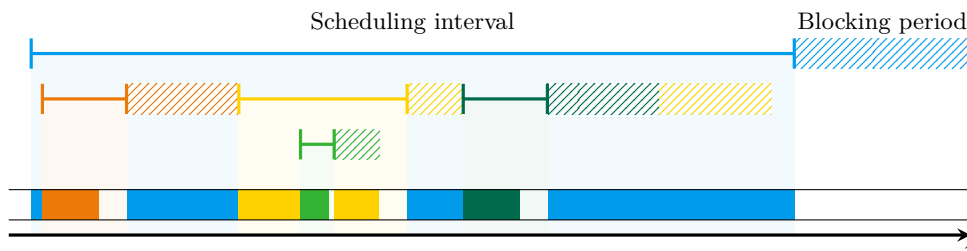
Figure 1 Scheduling interval, blocking period, and processing intervals.

For **admitting** jobs, the algorithm keeps track of *available* jobs at any time point τ. A job j with $r_j \leq \tau$ is called available if it has not yet been admitted to a machine by the algorithm and its deadline is not too close, i.e., $d_j - \tau \geq (1+\delta)p_j$.

Whenever a job j is available at a time τ when there is a machine i such that this time is not contained in the scheduling interval of any other job, the shortest such job j is immediately admitted to i, creating the scheduling interval $S(j) = [\tau, \tau + (1+\delta)p_j) := [a_j, e_j)$ and an empty blocking period $B(j) = \emptyset$. In general, however, the blocking period is a finite union of time intervals associated with job j, and its size is the sum of lengths of the intervals, denoted by $|B(j)|$. Four types of events trigger a decision of the algorithm at time τ: the release of a job, the end of a blocking period, the end of a scheduling interval, and the admission of a job. In any of these four cases, the algorithm calls the class admission routine. This subroutine iterates over all machines i and checks if j, the shortest job on i whose scheduling interval contains τ, can admit the currently shortest available job $j^\star$.

To this end, any admitted job j classifies available jobs k with $r_k \in S(j)$ and $p_k < \gamma p_j$ depending on their processing time. More precisely, job j maintains a *class structure* $(\mathcal{C}_c(j))_{c\in\mathbb{N}_0}$ where $\mathcal{C}_c(j)$ contains all jobs k that are available at some time during $S(j)$ and satisfy $\frac{\gamma}{2^{c+1}}p_j \leq p_k < \frac{\gamma}{2^c}p_j$. Only jobs $k \in \mathcal{C}_c(j)$ for $c \in \mathbb{N}_0$ qualify for admission by j. Upon admission by j, job $j^\star$ obtains two disjoint consecutive intervals, the *scheduling interval* $S(j^\star) = [a_{j^\star}, e_{j^\star})$ and the *blocking period* $B(j^\star)$ of size at most $\beta p_{j^\star}$. At the admission of job $j^\star$, the blocking period $B(j^\star)$ is planned to start at $e_{j^\star}$, the end of $j^\star$'s scheduling interval. During $B(j^\star)$ of job $j^\star \in \mathcal{C}_c(j)$, j only admits jobs k of *higher* classes, i.e., $k \in \mathcal{C}_{c'}(j)$ for $c' > c$. Particularly, j only admits job $j^\star \in \mathcal{C}_c(j)$ if the blocking period of the last job in $\mathcal{C}_c(j)$ admitted to the same machine has completed.

Hence, when job j decides if it admits the currently shortest available job $j^\star$ at time τ, it makes sure that $j^\star$ indeed belongs to a class $\mathcal{C}_c(j)$ and that no higher class $c' \geq c$ is blocking τ, i.e., it checks that $\tau \notin B(k)$ for all jobs $k \in \mathcal{C}_{c'}(j)$ admitted to the same machine. In this case, we say that $j^\star$ is a *child* of j and that j is the *parent* of $j^\star$, denoted by $\pi(j^\star) = j$. If job $j^\star$ is admitted at time τ by job j, the algorithm sets $a_{j^\star} = \tau$ and $e_{j^\star} = a_{j^\star} + (1+\delta)p_{j^\star}$ and assigns the scheduling interval $S(j^\star) = [a_{j^\star}, e_{j^\star})$ to $j^\star$.

If $e_{j^\star} \leq e_j$, the routine sets $f_{j^\star} = \min\{e_j, e_{j^\star} + \beta p_{j^\star}\}$ which determines $B(j^\star) = [e_{j^\star}, f_{j^\star})$. As the scheduling and blocking periods of children k of j are supposed to be disjoint, we have to **update the blocking periods**. First consider the job $k \in \mathcal{C}_{c'}(j)$ for $c' < c$ admitted to the same machine whose blocking period contains τ (if it exists), and let $[e_k', f_k')$ be the maximal interval of $B(k)$ containing τ. We set $f_k'' = \min\{e_j, f_k' + (1+\delta+\beta)p_{j^\star}\}$ and replace the interval $[e_k', f_k')$ by $[e_k', \tau) \cup [\tau + (1+\delta+\beta)p_{j^\star}, f_k'')$. For all other jobs $k \in \mathcal{C}_{c'}(j)$ with $B(k) \cap [\tau, \infty) \neq \emptyset$ admitted to the same machine, we replace the remaining part of their

blocking period $[e'_k, f'_k)$ by $[e'_k + (1 + \delta + \beta)p_{j^\star}, f''_k)$ where $f''_k := \min\{e_j, f'_k + (1 + \delta + \beta)p_{j^\star}\}$. In this update we follow the convention $[e, f) = \emptyset$ if $f \leq e$. Observe that the length of the blocking period might decrease due to such updates.

Note that $e_{j^\star} > e_j$ is also possible as j does not take the end of its own scheduling interval e_j into account when admitting jobs. Thus, the scheduling interval of $j^\star$ would end outside j's scheduling interval and inside j's blocking period. During $B(j)$, $\pi(j)$, the parent of j, did not allocate the interval $[e_j, e_{j^\star})$ for completing jobs admitted by j but for ensuring its own completion. Hence, the completion of both $j^\star$ and $\pi(j)$ is not necessarily guaranteed anymore. To prevent this, we **modify all scheduling intervals** $S(k)$ (including $S(j)$) of jobs admitted to the same machine that contain time τ and the corresponding blocking periods $B(k)$. For each job k admitted to the same machine with $\tau \in S(k)$ (i.e., including j) and $e_{j^\star} > e_k$ we set $e_k = e_{j^\star}$. We also update their blocking periods (in fact, single intervals) to reflect their new starting points. If the parent $\pi(k)$ of k does not exist, $B(k)$ remains empty; otherwise we set $B(k) := [e_k, f_k)$ where $f_k = \min\{e_{\pi(k)}, e_k + \beta p_k\}$. Note that, after this update, the blocking intervals of any but the largest such job will be empty. Moreover, the just admitted job $j^\star$ does not get a blocking period in this special case.

During the analysis of the algorithm, we show that any admitted job j still completes before $a_j + (1 + \delta)p_j$ and that $e_j \leq a_j + (1 + 2\delta)p_j$ holds in retrospective for all admitted jobs j. Thus, any job j that admits another job $j^\star$ tentatively assigns this job a scheduling interval of length $(1 + \delta)p_{j^\star}$ but, for ensuring j's completion, it is prepared for losing $(1 + 2\delta)p_{j^\star}$ time units of its scheduling interval $S(j)$. We summarize the blocking algorithm in Algorithm 1.

■ **Algorithm 1** Blocking algorithm.

```
Scheduling routine: At all time τ and on all machines i, run the job with shortest
    processing time that has been admitted to i and has not yet completed
```

```
Event: Upon release of a new job at time τ:
    Call admission routine.
```

```
Event: Upon ending of a blocking period or scheduling interval at time τ:
    Call admission routine.
```

```
Admission routine:
j* ← a shortest available job at τ, i.e.,
    j* ∈ arg min{p_j | r_j ≤ τ and d_j − τ ≥ (1 + δ)p_j}
i ← 1
while j* is not admitted and i ≤ m do
    K ← the set of jobs on machine i whose scheduling intervals contain τ
    if K = ∅
        1. admit job j* to machine i, a_j* ← τ, e_j* ← a_j* + (1 + δ)p_j*, and f_j* ← e_j*
        2. call admission routine
    else
        j   ← arg min{p_k | k ∈ K}
        if j* ∈ C_c(j) and there exists no c′ ≥ c with t ∈ B(j′) for j′ ∈ C_c′(j), then
            1. admit job j* to machine i, a_j* ← τ and e_j* ← a_j* + (1 + δ)p_j*
                if e_j* ≤ e_j, then
                    f_j* ← min{e_j, e_j* + βp_j*}
                    set S(j*) ← [a_j*, e_j*) and B(j*) ← [e_j*, f_j*)
                else
                    set e_j ← e_j* and f_j* ← e_j*
                    modify S(k) and B(k) for k ∈ K
            2. update B(k) for k ∈ C_c′(j) admitted to machine i with c′ < c and
                B(k) ∩ [τ, ∞) ≠ ∅
            3. call admission routine
        else
            i ← i + 1
```

Roadmap for the analysis

During the analysis, it is sufficient to concentrate on instances with small slack, as also noted in [8]. For $\varepsilon > 1$ we run the blocking algorithm with $\varepsilon = 1$, which only tightens the commitment requirement, and obtain constant competitive ratios. Thus, we assume $0 < \varepsilon \le 1$. Moreover, in the δ-commitment model, committing to the completion of a job j at an earlier point in time clearly satisfies committing at a remaining slack of δp_j. Therefore, we may assume $\delta \in [\frac{\varepsilon}{2}, \varepsilon)$.

The blocking algorithm does not migrate any job. In the analysis, we first compare the throughput of our algorithm to the solution of an optimal non-migratory schedule. We then use a well-known result by Kalyanasundaram and Pruhs to compare this to an optimal solution that may exploit migration. Here, ω_m is the maximal ratio of the throughput of an optimal migratory schedule to the throughput of an optimal non-migratory schedule [19].

▶ **Theorem 3** (Theorem 1.1 in [19]). $\omega_m \le (6m - 5)/m$.

The proof of our results consists of two parts. In the first part, Section 3, we show that the blocking algorithm completes all admitted jobs on time. The second part, Section 4, is to show that the blocking algorithm admits sufficiently many jobs to be competitive.

3   Completing all admitted jobs on time

We show that the blocking algorithm finishes every admitted job on time in Theorem 5. Our choice of parameters guarantees that Inequality (1) is satisfied.

As the blocking algorithm does not migrate jobs, it suffices to consider each machine individually in this section. The proof relies on the following observations: (i) the sizes of admitted jobs belonging to different classes of job j are geometrically decreasing, (ii) the scheduling intervals of jobs are completely contained in the scheduling intervals of their parents, and (iii) scheduling in Shortest Processing Time order guarantees that job j has highest priority in the blocking periods of its children. We start by proving the following technical lemma about the length of the final scheduling interval of an admitted job j. In the proof we use that $\pi(k) = j$ for two jobs j and k implies that $p_k < \gamma p_j$.

▶ **Lemma 4.** Let $0 < \delta < \varepsilon$ be fixed. If $\gamma > 0$ satisfies $(1 + 2\delta)\gamma \le \delta$, then the length of the scheduling interval $S(j)$ of an admitted job j is upper bounded by $(1 + 2\delta)p_j$. Moreover, $S(j)$ contains the scheduling intervals of all descendants of j.

Proof. By definition of the blocking algorithm, the end point e_j of the scheduling interval of job j is only modified when j or one of j's descendants admits another job. Let us consider such a case: If job j admits a job $j^\star$ whose scheduling interval does not fit the scheduling interval of j, we set $e_j = e_{j^\star} = a_{j^\star} + (1 + \delta)p_{j^\star}$ to accommodate the scheduling interval $S(j^\star)$ within $S(j)$. The same modification is applied to any ancestor k of j with $e_k < e_{j^\star}$. This implies that, after such a modification of the scheduling interval, neither j nor any affected ancestors k of j are the smallest jobs in their scheduling intervals anymore. In particular, no job whose scheduling interval was modified in such a case at time τ is able to admit jobs after τ. Hence, any job j can only admit other jobs within the interval $[a_j, a_j + (1 + \delta)p_j)$. In particular, $a_{j^\star} \le a_j + (1 + \delta)p_j$ for any job $j^\star$ with $\pi(j^\star) = j$.

Thus, by induction, it is sufficient to show that $a_{j^\star} + (1 + 2\delta)p_{j^\star} \le a_j + (1 + 2\delta)p_j$ for admitted jobs $j^\star$ and j with $\pi(j^\star) = j$ in order to prove the lemma. Note that $\pi(j^\star) = j$ implies $p_{j^\star} < \gamma p_j$. Thus,

$$a_{j^\star} + (1 + 2\delta)p_{j^\star} \le (a_j + (1 + \delta)p_j) + (1 + 2\delta)\gamma p_j \le a_j + (1 + 2\delta)p_j,$$

where the last inequality follows from the assumption $(1 + 2\delta)\gamma \le \delta$.   ◀

▶ **Theorem 5.** *Let $0 < \delta < \varepsilon$ be fixed. If $0 < \gamma < 1$ and $\beta \geq 1$ satisfy*

$$\frac{\beta/2}{\beta/2 + (1 + 2\delta)}\left(1 + \delta - 2(1 + 2\delta)\gamma\right) \geq 1, \tag{1}$$

then the blocking algorithm completes a job j admitted at $a_j \leq d_j - (1 + \delta)p_j$ on time.

Our choice of parameters guarantees that Inequality (1) is satisfied.

Proof. Let j be a job admitted by the blocking algorithm with $a_j \leq d_j - (1 + \delta)p_j$. Hence, showing that a job j completes before time $d'_j := a_j + (1 + \delta)p_j$ proves the theorem. Due to scheduling in SPT order, each job j has highest priority in its own scheduling interval if the time point does not belong to the scheduling interval of a descendant of j. Thus, it suffices to show that at most δp_j units of time in $[a_j, d'_j)$ belong to scheduling intervals $S(k)$ of descendants of j. By Lemma 4, the scheduling intervals of any descendant k' of a child k of j is contained in $S(k)$. Hence, it is sufficient to only consider K, the set of children of j. In order to bound the contribution of each child $k \in K$, we partition K into two sets. The first set K_1 contains all children of j that where admitted as the first jobs in their class $\mathcal{C}_c(j)$. The set K_2 contains the remaining jobs.

We start with K_2. Consider a job $k \in \mathcal{C}_c(j)$ admitted by j. By Lemma 4, we know that $|S(k)| = (1 + \mu\delta)p_k$ where $1 \leq \mu \leq 2$. Let $k' \in \mathcal{C}_c(j)$ be the previous job admitted by j in class c. Then, $B(k') \subseteq [e_{k'}, e_k)$. Since scheduling and blocking periods of children of j are always disjoint, j had highest scheduling priority in $B(k')$. Hence, during $B(k') \cup S(k)$ job j was processed for at least $|B(k')|$ units of time. In other words, j was processed for at least a $\frac{|B(k')|}{|B(k') \cup S(k)|}$-fraction of $B(k') \cup S(k)$. We can rewrite this ratio by

$$\frac{|B(k')|}{|B(k') \cup S(k)|} = \frac{\beta p_{k'}}{\beta p_{k'} + (1 + \mu\delta)p_k} = \frac{\nu\beta}{\nu\beta + (1 + \mu\delta)},$$

where $\nu := \frac{p_{k'}}{p_k} \in (\frac{1}{2}, 2]$. By differentiating with respect to ν and μ, we observe that the last term is increasing in ν and decreasing in μ. Thus, we can lower bound this expression by

$$\frac{|B(k')|}{|B(k') \cup S(k)|} \geq \frac{\beta/2}{\beta/2 + (1 + 2\delta)}.$$

Therefore, j was processed for at least a $\frac{\beta/2}{\beta/2 + (1 + 2\delta)}$-fraction in $\bigcup_{k \in K} B(k) \cup \bigcup_{k \in K_2} S(k)$.

We now consider the set K_1. The total processing volume of these jobs is bounded by $\sum_{c=0}^{\infty} \frac{\gamma}{2^c}p_j = 2\gamma p_j$. By Lemma 4, we know that $|S(k)| \leq (1 + 2\delta)p_k$. Combining these two observations, we obtain $\left|\bigcup_{k \in K_1} S(k)\right| \leq 2(1 + 2\delta)\gamma p_j$. Combining the latter with the bound for K_2, we conclude that j is scheduled for at least

$$\left|[a_j, d'_j) \setminus \bigcup_{k \in K} S(k)\right| \geq \frac{\beta/2}{\beta/2 + (1 + 2\delta)}\left((1 + \delta) - 2(1 + 2\delta)\gamma\right)p_j \geq p_j$$

units of time, where the last inequality follows from Equation (1). Thus, j completes before $d'_j = a_j + (1 + \delta)p_j \leq d_j$. ◀

4   Admitting sufficiently many jobs

After proving that the blocking algorithm completes all admitted jobs on time, we show that the blocking algorithm admits enough jobs to achieve the competitive ratio of Theorem 1.

4.1   Key lemma on the size of non-admitted jobs

For the proof of the main result in this section, we rely on the following strong, structural lemma about the volume processed by a feasible non-migratory schedule in some time interval and the size of jobs admitted by a certain class of online algorithms in the same time interval. Let σ be a feasible non-migratory schedule. Let ALG be a non-migratory online algorithm satisfying the following two properties: (i) ALG never admits a job j later than $d_j - (1+\delta)p_j$ for $0 < \delta < \varepsilon$ and (ii) retrospectively, for each time τ, there is a threshold $u_\tau \in (0, \infty]$ such that any available job j with $d_j - \tau \geq (1+\delta)p_j$ that was not admitted by ALG at τ satisfies $p_j \geq u_\tau$. We will show that our blocking algorithm satisfies (i) and (ii) for a non-trivial u_τ that allows us to bound the volume of any feasible schedule.

Without loss of generality, we assume that σ completes all jobs on time that it started. Let X^σ be the jobs completed by σ and not admitted by ALG. For $1 \leq i \leq m$, let X_i^σ be the jobs in X^σ processed by machine i. Let C_x be the completion time of job $x \in X^\sigma$ in σ.

▶ **Lemma 6.** *Let σ and* ALG *be defined as above. Let $0 \leq \zeta_1 \leq \zeta_2$ and fix $x \in X_i^\sigma$ as well as $Y \subset X_i^\sigma \setminus \{x\}$. If*

(R) $r_x \geq \zeta_1$ as well as $r_y \geq \zeta_1$ for all $y \in Y$,
(C) $C_x \geq C_y$ for all $y \in Y$, and
(P) $\sum_{y \in Y} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1)$

hold, then $p_x \geq u_{\zeta_2}$ where u_{ζ_2} is the upper bound imposed by ALG *at time ζ_2. In particular, if $u_{\zeta_2} = \infty$, then no such job x exists.*

Proof. We show the lemma by contradiction. More precisely, we show that, if $p_x < u_{\zeta_2}$, the schedule σ cannot complete x on time and, hence, is not feasible.

Remember that $x \in X_i^\sigma$ implies that ALG did not admit job x at any point τ. At time ζ_2, there are two possible reasons why x was not admitted: $p_x \geq u_{\zeta_2}$ or $d_x - \zeta_2 < (1+\delta)p_x$. In case of the former, the statement of the lemma holds. Thus, let us assume $p_x < u_{\zeta_2}$ and, therefore, $d_x - \zeta_2 < (1+\delta)p_x$ has to hold. As job x arrived with a slack of at least εp_x at its release date r_x and $r_x \geq \zeta_1$ by assumption, we have

$$\zeta_2 - \zeta_1 \geq \zeta_2 - d_x + d_x - r_x > -(1+\delta)p_x + (1+\varepsilon)p_x = (\varepsilon - \delta)p_x. \qquad (2)$$

As all jobs in Y complete earlier than x by Assumption (C) and are only released after ζ_1 by (R), the volume processed by σ in $[\zeta_1, C_x)$ on machine i is greater than $\frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1) + p_x$ by (P). Moreover, σ can process at most a volume of $(\zeta_2 - \zeta_1)$ on machine i in $[\zeta_1, \zeta_2)$. These two bounds imply that σ has to process job parts with a processing volume of at least

$$\frac{\varepsilon}{\varepsilon - \delta}(\zeta_2 - \zeta_1) + p_x - (\zeta_2 - \zeta_1) > \frac{\delta}{\varepsilon - \delta}(\varepsilon - \delta)p_x + p_x = (1+\delta)p_x$$

in $[\zeta_2, C_x)$, where the inequality follows using Inequality (2). Thus, $C_x > \zeta_2 + (1+\delta)p_x > d_x$ which contradicts the feasibility of σ.

Observe that the online algorithm ALG admits the shortest available job that satisfies $p_j \leq u_\tau$. In particular, if $u_\tau = \infty$ for some time point τ, ALG admits the shortest job if there is one available. Hence, for $0 \leq \zeta_1 \leq \zeta_2$ with $u_{\zeta_2} = \infty$, there does not exist a job $x \in X_i^\sigma$ and a set $Y \subset X_i^\sigma \setminus \{x\}$ satisfying (R), (C), and (P) for any machine i.   ◀

4.2   Admitting sufficiently many jobs

▶ **Theorem 7.** *An optimal non-migratory (offline) algorithm can complete at most a factor $\alpha + 4$ more jobs on time than admitted by the blocking algorithm where $\alpha := \frac{\varepsilon}{\varepsilon - \delta}\left(2\beta + \frac{1+2\delta}{\gamma}\right)$.*

For proving the theorem, we fix an instance and an optimal offline algorithm OPT. Let X be the jobs that OPT scheduled and the blocking algorithm did not admit. We assume without loss of generality that OPT completes all jobs in X on time. Let J be the jobs that the blocking algorithm scheduled. Then, $X \cup J$ clearly is a superset of the jobs that OPT scheduled. Hence, to show the theorem it is sufficient to prove that $|X| \leq (\alpha + 3)|J|$. Let $\overline{X} \subseteq X$ be the jobs scheduled on the machine with the *highest* throughput and let $\underline{J} \subseteq J$ be the jobs scheduled on the machine with the *lowest* throughput. In Lemma 11 we develop a charging scheme of $\overline{X}$ to jobs in $\underline{J}$ such that no job gets charged more than $\alpha + 3$ jobs.

Without loss of generality, we assume that the union of all scheduling intervals of jobs in J, i.e., $\bigcup_{j \in J} S(j)$, forms one interval. If this assumption does not hold, we consider each maximal interval in $\bigcup_{j \in J} S(j)$ separately. Instead of directly charging the jobs in $\overline{X}$ to jobs in $\underline{J}$ we take a detour and charge jobs in $\overline{X}$ to intervals that cover $\bigcup_{j \in J} S(j)$. The idea behind our charging scheme is that OPT is not able to schedule arbitrarily many jobs during a scheduling interval or a blocking period created by the blocking algorithm. Intuitively, jobs that were released during a scheduling interval or a blocking period and not admitted by the algorithm have to satisfy certain lower bounds on their processing times. Thus, the charging scheme relies on the release date r_x and the size p_x of a job $x \in \overline{X}$ as well as on the precise structure of the intervals created by the blocking algorithm. The number of jobs we charge to one interval will depend on the relative length of the interval.

We retrospectively consider the interval structure created by the algorithm on the machine that schedules $\underline{J}$; let this w.l.o.g. be the first machine. Let T be the set of all time points corresponding to the admission of a new job, the end of a scheduling interval, and the start as well as the end of a blocking period of jobs in $\underline{J}$. Index the elements in T by their actual value, i.e., $\tau_1 < \tau_2 < \ldots < \tau_{|T|}$. Let $\mathcal{I}$ be the set of intervals of the form $I_t := [\tau_t, \tau_{t+1})$ for $1 \leq t < |T|$. The next lemma holds as the admission of a job adds at most three time points.

▶ **Fact 8.** *The set $\mathcal{I}$ contains at most $3|\underline{J}|$ intervals.*

For analyzing the competitive ratio of our algorithm, we first charge jobs $x \in \overline{X}$ to intervals $I_t \in \mathcal{I}$ and then assign this subset to the job that was "responsible" for not admitting other jobs during I_t because of its scheduling interval or because of its blocking period. In Lemma 11, we show that a job j is assigned at most $\frac{\varepsilon}{\varepsilon - \delta}(2\beta + \frac{1+2\delta}{\gamma}) = \alpha$ jobs and that each interval I_t gets at most one job. Fact 8 bounds the number of intervals in $\mathcal{I}$. Combining these observations then proves that $|\overline{X}| \leq (\alpha + 3)|\underline{J}|$ and, thus, $|X| \leq (\alpha + 3)|J|$.

Consider a time point $\tau \in \bigcup_{j \in J} S(j)$. Let $j \in J$ be the shortest job on machine i such that $\tau \in S(j) \cup B(j)$. The blocking algorithm only admits an available job k to machine i in two cases: (i) $\tau \in S(j)$ and $p_k < \gamma p_j$ or (ii) $\tau \in B(j)$ and k belongs to a smaller class of the parent of j. Condition (ii) clearly is satisfied if $p_k < p_j/2$. This implies that at any time τ the blocking algorithm maintains a threshold $u_{\tau,i}$ for each machine i so that only available jobs smaller than this threshold qualify for admission to machine i. Note that the admission of a job k at time τ to machine i decreases the threshold $u_{\tau,i}$. If τ does not belong to a scheduling interval of a job on machine i, we set $u_{\tau,i} = \infty$. By taking the maximum of these upper bounds, we obtain a time-dependent threshold u_τ that guides the admission decisions of the blocking algorithm. Hence, the conditions of Lemma 6 are met by the our algorithm.

Note that these upper bounds only change when a scheduling interval starts or ends, or when an interval belonging to a blocking period starts or ends. For an interval $I_t \in \mathcal{I}$ we define $\underline{u_t}$ as the threshold on the machine with the lowest throughput, i.e., $\underline{u_t} := u_{\tau_t,1} \in (0, \infty]$. If $\underline{u_t} = \infty$, then the interval $I_t \in \mathcal{I}$ does not belong to the scheduling interval of a job in J and $u_\tau = \infty$ for all $\tau \in I_t$. Then the next lemma holds.

▶ **Fact 9.** *In every interval $I_t = [\tau_t, \tau_{t+1}) \in \mathcal{I}$ the upper bound u_τ created by the blocking algorithm is lower bounded by $u_{\tau_t,1}$, i.e., $u_\tau \geq \underline{u_t}$ for all $\tau \in I_t$.*

The charging scheme developed in Lemma 11 is based on a careful modification of the following partition $(F_t)_{1 \leq t < |T|}$ of the set $\overline{X}$. Fix an interval $I_t \in \mathcal{I}$ and define the set $F_t \subset \overline{X}$ as the set that contains all jobs $x \in \overline{X}$ released during I_t, i.e., $F_t := \{x \in \overline{X} : r_x \in I_t\}$. As, upon release, each job is available, the next corollary directly follows from Fact 9.

▶ **Fact 10.** *For all jobs $x \in F_t$ it holds $p_x \geq \underline{u_t}$. In particular, if $\underline{u_t} = \infty$, then $F_t = \emptyset$.*

In fact, the charging scheme maintains this property and only assigns jobs in $\overline{X}$ to intervals I_t if $p_x \geq \underline{u_t}$. In particular, no job will be assigned to an interval with $\underline{u_t} = \infty$.

We now formalize how many jobs in $\overline{X}$ we will assign to a specific interval I_t. Let $\varphi_t := \left\lfloor \frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t+1}-\tau_t}{\underline{u_t}} \right\rfloor + 1$ be the *target number* of I_t if $\underline{u_t} < \infty$ and $\varphi_t = 0$ if $\underline{u_t} = \infty$.

If $\underline{u_t} < \infty$, let $j_t \in \underline{J}$ be the *smallest* job with $\tau_t \in S(j) \cup B(j)$. Except for one job per interval $I_t \in \mathcal{I}$ which remains assigned to I_t, the jobs assigned to I_t will be accounted for by j_t. Suppose that each of the sets F_t satisfies $|F_t| \leq \varphi_t$. Then, at most $\frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t+1}-\tau_t}{\underline{u_t}}$ will be charged to job j_t because of interval I_t. By definition of $\underline{u_t}$, we have $\underline{u_t} \geq \gamma p_{j_t}$ if $I_t \subseteq S(j_t)$ and, if $I_t \subseteq B(j_t)$, we have $\underline{u_t} \geq p_{j_t}/2$. The total length of intervals I_t for which $j = j_t$ holds sums up to at most $(1+2\delta)p_j$ for $I_t \subseteq S(j)$ and to at most $2\beta p_j$ for $I_t \subseteq B(j)$. Hence, in total, the charging scheme assigns at most $\frac{\varepsilon}{\varepsilon-\delta}(2\beta + \frac{1+2\delta}{\gamma}) = \alpha$ jobs in $\overline{X}$ to job $j \in \underline{J}$. In combination with Fact 8, that bounds the number of intervals in $\mathcal{I}$, this would imply Theorem 7. In general, $|F_t| \leq \varphi_t$ does not have to be true as OPT may preempt jobs and process the parts during several intervals I_t. In the remainder of this section, we show that there exists another partition $(G_t)_{1 \leq t < |T|}$ of the jobs in $\overline{X}$ such that $|G_t| \leq \varphi_t$ holds.

▶ **Lemma 11.** $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|.$

Proof. As observed before it suffices to show that there is a partition $\mathcal{G} = (G_t)_{1 \leq t < |T|}$ such that $|G_t| \leq \varphi_t$ and $\bigcup_{1 \leq t < |T|} G_t = \overline{X}$ in order to prove the lemma. The high-level idea of this proof is the following: Consider an interval $I_t = [\tau_t, \tau_{t+1})$. If F_t does not contain too many jobs, i.e., $|F_t| \leq \varphi_t$, we would like to set $G_t = F_t$. Otherwise, we find a later interval $I_{t'}$ with $|F_{t'}| < \varphi_{t'}$ such that we can assign the excess jobs in F_t to $I_{t'}$.

In order to repeatedly apply Lemma 6, we only assign such excess jobs $x \in F_t$ to $G_{t'}$ if their processing time is at least the threshold of $I_{t'}$, i.e., $p_x \geq \underline{u_{t'}}$. Then, by our choice of parameters, a set $G_{t'}$ with $\varphi_{t'}$ many jobs of size at least $\underline{u_{t'}}$ "covers" the interval $I_{t'} = [\tau_{t'}, \tau_{t'+1})$ as often as required by (P) in Lemma 6, i.e.,

$$\sum_{x \in G_{t'}} p_x \geq \varphi_{t'} \cdot \underline{u_{t'}} = \left(\left\lfloor \frac{\varepsilon}{\varepsilon-\delta} \frac{\tau_{t'+1}-\tau_{t'}}{\underline{u_{t'}}} \right\rfloor + 1 \right) \cdot \underline{u_{t'}} \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_{t'+1}-\tau_{t'}). \tag{3}$$

The proof consists of two parts: the first one is to inductively (on t) construct the partition $\mathcal{G} = (G_t)_{1 \leq t < |T|}$ of $\overline{X}$ with $|G_t| \leq \varphi_t$. The second one is the proof that a job $x \in G_t$ satisfies $p_x \geq \underline{u_t}$. During the construction of $\mathcal{G}$ we define temporary sets $A_t \subset \overline{X}$ for intervals I_t. The set G_t is chosen as a subset of $F_t \cup A_t$ of appropriate size. In order to apply Lemma 6 to each job in A_t individually, alongside A_t, we construct a set $Y_{x,t}$ and a time $\tau_{x,t} \leq r_x$ for each job $x \in \overline{X}$ that is added to A_t. Let C_x^* be the completion time of some job $x \in \overline{X}$ in the optimal schedule OPT. The second part of the proof is to show that x, $\tau_{x,t}$, and $Y_{x,t}$ satisfy

(R) $r_y \geq \tau_{x,t}$ for all $y \in Y_{x,t}$,
(C) $C_x^* \geq C_y^*$ for all $y \in Y_{x,t}$, and
(P) $\sum_{y \in Y_{x,t}} p_y \geq \frac{\varepsilon}{\varepsilon-\delta}(\tau_t - \tau_{x,t})$.

Then, x, $Y = Y_{x,t}$, $\zeta_1 = \tau_{x,t}$, and $\zeta_2 = \tau_t$ satisfy the conditions of Lemma 6 and we can deduce that the processing time of x is at least the threshold at time τ_t, i.e., $p_x \geq u_{\tau_t} \geq \underline{u_t}$.

Constructing $\mathcal{G} = (\mathbf{G_t})_{1 \leq t \leq |\mathbf{T}|}$.   We inductively construct the sets G_t in the order defined by their indices. For simplicity, we add a singleton as last interval, i.e., $I_{|T|} = \{\tau_{|T|}\}$ with $\varphi_{|T|} = 0$. We start by setting $A_t = \emptyset$ for all intervals $1 \leq t \leq |T|$. We define $Y_{x,t} = \emptyset$ for each job $x \in \overline{X}$ and each interval I_t. The preliminary value of the time $\tau_{x,t}$ is the minimum of the start point τ_t of the interval I_t and the release date r_x of x, i.e., $\tau_{x,t} := \min\{\tau_t, r_x\}$. We refer by *step t* to the step in the construction where G_t was defined.

Starting with $t = 1$, let I_t be the next interval to consider during the construction. Depending on the cardinality of $F_t \cup A_t$, we have to distinguish two cases. If $|F_t \cup A_t| \leq \varphi_t$, we set $G_t = F_t \cup A_t$.

If $|F_t \cup A_t| > \varphi_t$, we order the jobs in $F_t \cup A_t$ in increasing order of completion times in OPT. The first φ_t jobs are assigned to G_t while the remaining $|F_t \cup A_t| - \varphi_t$ jobs are added to A_{t+1}. In this case, we might have to redefine the times $\tau_{x,t+1}$ and the sets $Y_{x,t+1}$ for the jobs x that were newly added to A_{t+1}. Fix such a job x. If there is no job z in the just defined set G_t that has a smaller release date than $\tau_{x,t}$, we set $\tau_{x,t+1} = \tau_{x,t}$ and $Y_{x,t+1} = Y_{x,t} \cup G_t$. Otherwise let $z \in G_t$ be a job with $r_z < \tau_{x,t}$ that has the smallest time $\tau_{z,t}$. We set $\tau_{x,t+1} = \tau_{z,t}$ and $Y_{x,t+1} = Y_{z,t} \cup G_t$.

Finally, we also construct $G_{|T|}$ this way. As we will show that $p_x \geq \underline{u_{|T|}}$ for all $x \in G_{|T|}$, we will get that $G_{|T|} = \emptyset$ (since $\underline{u_{|T|}} = \infty$) and therefore $G_{|T|} \leq \varphi_{|T|} = 0$.

Bounding the size of the jobs in $\mathbf{G_t}$.   We consider the intervals again in increasing order of their indices and show by induction that any job x in G_t satisfies $p_x \geq \underline{u_t}$ which implies $G_t = \emptyset$ if $\underline{u_t} = \infty$. Clearly, if $x \in F_t \cap G_t$, Fact 10 guarantees $p_x \geq \underline{u_t}$. Hence, in order to show the lower bound on the processing time of $x \in G_t$, it is sufficient to consider jobs in $G_t \setminus F_t \subset A_t$. To this end, we show that for such jobs (R), (C), and (P) are satisfied. Then, Lemma 6 guarantees that $p_x \geq u_{\tau_t} \geq \underline{u_t}$. Therefore, $A_t = \emptyset$ if $\underline{u_t} = \infty$ as the global bound is also unbounded, i.e., $u_{\tau_t} \geq \underline{u_t} = \infty$, by Fact 9.

By construction, $A_1 = \emptyset$. Hence, (R), (C), and (P) are satisfied for each job $x \in A_1$.

Assume that the conditions (R), (C), and (P) are satisfied for all $x \in A_t$ for all $1 \leq t < s$. Hence, for $t < s$, the set G_t only contains jobs x with $p_x \geq \underline{u_t}$. Let $t \geq s$ be the first index with $A_t \neq \emptyset$ and fix $x \in A_t$. We want to show that $p_x \geq \underline{u_t}$. By induction and by Fact 10, $p_y \geq \underline{u_{t-1}}$ holds for all $y \in G_{t-1}$. Because x did not fit in G_{t-1}, $|G_{t-1}| = \varphi_{t-1}$.

We distinguish two cases based on the jobs in G_{t-1}. If there is no $z \in G_{t-1}$ with $r_z < \tau_{x,t-1}$, then $\tau_{x,t} = \tau_{x,t-1}$, and (R) and (C) are satisfied by construction and by induction. For (P), consider

$$\sum_{y \in Y_{x,t}} p_y = \sum_{y \in Y_{x,t-1}} p_y + \sum_{y \in G_{t-1}} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + \underline{u_{t-1}} \cdot \varphi_{t-1}$$

$$> \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{t-1}) = \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{x,t}),$$

where the first inequality holds by induction.

If there is a job $z \in G_{t-1}$ with $r_z < \tau_{x,t-1} \leq \tau_{t-1}$, then $z \in A_{t-1}$. In step t, we chose z with minimal $\tau_{z,t-1}$. Thus, $r_y \geq \tau_{y,t-1} \geq \tau_{z,t-1}$ for all $y \in G_{t-1}$ and $r_x \geq \tau_{x,t-1} > r_z \geq \tau_{z,t-1}$. Moreover, by induction, $r_y \geq \tau_{z,t-1}$ holds for all $y \in Y_{z,t-1}$. Thus, $\tau_{x,t}$ and $Y_{x,t}$ satisfy (R). For (C), consider that $C_x^* \geq C_y^*$ for all $y \in G_{t-1}$ by construction and, thus, $C_x^* \geq C_z^* \geq C_y^*$ also holds for all $y \in Y_{z,t-1}$. For (P), observe that

$$\sum_{y \in Y_{x,t}} p_y = \sum_{y \in Y_{z,t-1}} p_y + \sum_{y \in G_{t-1}} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{z,t-1}) + \underline{u_{t-1}} \cdot \varphi_{t-1}$$

$$\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{z,t-1}) + \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{t-1}) \geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{x,t}).$$

Here, the first inequality follows by induction and the second by definition of $\underline{u_{t-1}}$ and φ_{t-1}. Hence, Lemma 6 implies $p_x \geq u_{\tau_t} \geq \underline{u_t}$.

Showing $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|$. By construction, we know that $\bigcup_{t=1}^{|T|} G_t = \overline{X}$. We start with considering intervals I_t with $\underline{u_t} = \infty$. Then, I_t is not covered by a scheduling interval on machine i. Thus, $u_\tau = \infty$ for all $\tau \in I_t$ and $F_t = \emptyset$ by Fact 10. In the previous part we have seen that the conditions for Lemma 6 are satisfied. Hence, $G_t = \emptyset$ if $\underline{u_t} = \infty$. For t with $\underline{u_t} < \infty$, we have $|G_t| = \varphi_t = \left\lfloor \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{\underline{u_t}} \right\rfloor + 1$. As explained before, we assign these jobs (except the additive one) to j_t, the shortest job in $\underline{J}$ with $I_t \subseteq S(j) \cup B(j)$, without assigning more than α jobs in $\overline{X}$ to a particular job in $\underline{J}$. Hence, the number of jobs in $\overline{X}$ is indeed bounded by $\alpha|\underline{J}| + |\mathcal{I}|$. ◀

Proof of Theorem 7. As discussed before, the union $X \cup J$ of X, the jobs only scheduled by OPT, and J, the jobs admitted by the blocking algorithm, is a superset of the jobs that OPT completed. Lemma 11 shows that $|\overline{X}| \leq \alpha|\underline{J}| + |\mathcal{I}|$.Combining this with the bound on $\mathcal{I}$ given in Fact 8, we conclude

$$\text{OPT} \leq m \cdot |\overline{X}| + |J| \leq m\,(\alpha + 3)\,|\underline{J}| + |J| \leq (\alpha + 4)\,|J|. \quad ◀$$

Proof of Theorem 1. In Theorem 5 we show that the blocking algorithm completes all admitted jobs J on time. This implies that the blocking algorithm is feasible for the model commitment upon admission. As no job $j \in J$ is admitted later than $d_j - (1+\delta)p_j$, this shows that the blocking algorithm also solves scheduling with δ-commitment. Theorem 1.1 in [19] (Theorem 3) gives a bound on the optimal migratory schedule in terms of an optimal non-migratory solution. In Theorem 7, we bound an optimal non-migratory solution OPT by $|J|$, the throughput of the blocking algorithm. Combining these theorems shows that the blocking algorithm achieves a competitive ratio of $c = 6(\alpha + 4) = 6\left(\frac{\varepsilon}{\varepsilon - \delta}\left(2\beta + \frac{1+2\delta}{\gamma}\right) + 4\right)$. Our choice of parameters $\beta = \frac{16}{\delta}$ and $\gamma = \frac{\delta}{16}$ implies $c \in \mathcal{O}\left(\frac{\varepsilon}{(\varepsilon - \delta)\delta}\right)$. In the case where $\delta \leq \varepsilon/2$, we run the algorithm with parameter $\delta' = \varepsilon/2$. Hence, $c \in \mathcal{O}\left(\frac{1}{\varepsilon - \delta'}\right) = \mathcal{O}\left(\frac{1}{\varepsilon}\right)$. If $\delta > \varepsilon/2$, then we set $\delta' = \delta$ in our algorithm. Thus, $\frac{\varepsilon}{\delta'} \in \mathcal{O}(1)$ and, again, $c \in \mathcal{O}\left(\frac{1}{\varepsilon - \delta'}\right)$. ◀

Conclusion

We close the major questions regarding online throughput maximization with and without commitment requirements and give an optimal online algorithm on identical parallel machines for the problem $P \,|\, \text{online } r_j, \text{pmtn} \,|\, \sum(1 - U_j)$. It remains open whether the problem, where m is not part of the input, admits an online algorithm with a better competitive ratio as is the case for $Pm \,|\, \text{online } r_j, \text{pmtn} \,|\, \sum p_j(1 - U_j)$ [26].

Another interesting question asks whether randomization allows for improved results. On a single machine, there is indeed an $\mathcal{O}(1)$-competitive randomized algorithm for scheduling without commitment, even without any slack assumption [20]. We are not aware of any lower bound that rules out a similar result on multiple machines. Further research directions include generalizations such as weighted throughput maximization. While strong lower bounds exist for handling weighted throughput with commitment [8], there remains a gap for the problem without commitment.

References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling parallelizable jobs online to maximize throughput. In *Proceedings of the Latin American Theoretical Informatics Symposium (LATIN)*, pages 755–776, 2018. `doi:10.1007/978-3-319-77404-6_55`.

2 Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In *Proceedings of the ACM Symposium on Economics and Computations (EC)*, pages 715–732, 2015. `doi:10.1145/2764468.2764535`.

3 Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Competitive algorithms for due date scheduling. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 28–39, 2007. `doi:10.1007/978-3-540-73420-8_5`.

4 Sanjoy K. Baruah and Jayant R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997. `doi:10.1109/12.620484`.

5 Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 228–236, 1994. `doi:10.1109/REAL.1994.342713`.

6 Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992. `doi:10.1007/BF00365406`.

7 Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998. `doi:10.1137/S0097539795283292`.

8 Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. A general framework for handling commitment in online throughput maximization. *Math. Prog.*, 2020. `doi:10.1007/s10107-020-01469-2`.

9 Lin Chen, Nicole Megow, and Kevin Schewior. The power of migration in online machine minimization. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 175–184, 2016. `doi:10.1145/2935764.2935786`.

10 Lin Chen, Nicole Megow, and Kevin Schewior. An $\mathcal{O}(\log m)$-competitive algorithm for online machine minimization. *SIAM J. Comput.*, 47(6):2057–2077, 2018. `doi:10.1137/17M116032X`.

11 Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 96–107, 2000. `doi:10.1007/3-540-44436-X_11`.

12 Michael L. Dertouzos and Aloysius K. Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Trans. Software Eng.*, 15(12):1497–1506, 1989. `doi:10.1109/32.58762`.

13 Andrew D. Ferguson, Peter Bodík, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: guaranteed job latency in data parallel clusters. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 99–112, 2012. `doi:10.1145/2168836.2168847`.

14 Juan A. Garay, Joseph Naor, Bülent Yener, and Peng Zhao. On-line admission control and packet scheduling with interleaving. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 94–103, 2002. `doi:10.1109/INFCOM.2002.1019250`.

15 Michael H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 396–405, 1999. URL: `http://dl.acm.org/citation.cfm?id=314500.314592`.

16 Ronald L. Graham, Eugene L. Lawler, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979. `doi:10.1016/S0167-5060(08)70356-X`.

17 Sungjin Im and Benjamin Moseley. General profit scheduling and the power of migration on heterogeneous machines. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 165–173, 2016. `doi:10.1145/2935764.2935771`.

18   Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. `doi:10.1145/347476.347479`.

19   Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001. `doi:10.1006/jagm.2000.1128`.

20   Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. `doi:10.1016/S0196-6774(03)00074-9`.

21   Gilad Koren and Dennis E. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1–2):75–97, 1994. `doi:10.1016/0304-3975(94)90165-1`.

22   Gilad Koren and Dennis E. Shasha. D$^{\text{over}}$: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995. `doi:10.1137/S0097539792236882`.

23   Richard J. Lipton and Andrew Tomkins. Online interval scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 302–311, 1994. URL: `http://dl.acm.org/citation.cfm?id=314464.314506`.

24   Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs: extended abstract. In *Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 305–314, 2013. `doi:10.1145/2486159.2486187`.

25   Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 352–365, 2010. `doi:10.1007/978-3-642-15369-3_27`.

26   Chris Schwiegelshohn and Uwe Schwiegelshohn. The power of migration for online slack scheduling. In *Proceedings of the European Symposium of Algorithms (ESA)*, volume 57, pages 75:1–75:17, 2016. `doi:10.4230/LIPIcs.ESA.2016.75`.

27   Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994. `doi:10.1016/0304-3975(94)90150-3`.