

Online scheduling with preemption or non-completion penalties

Stanley P. Y. Fung

Published online: 10 April 2013
© Springer Science+Business Media New York 2013

Abstract We consider online preemptive scheduling problems where jobs have deadlines and the objective is to maximize the total weight of jobs completed before their deadlines. In the first problem, preemptions are not free but incur a penalty. In the second problem, a job has to be accepted or rejected immediately upon arrival, and may need to be immediately allocated a fixed scheduling interval as well; if these accepted jobs are not eventually completed, the job is lost, and a penalty is incurred. We give an algorithm with the optimal competitive ratio for the first problem, and new and improved algorithms for the second problem, under different models of preemptions and job weights.

Keywords Online algorithms · Competitive analysis · Immediate notification · Immediate decision · Preemption penalty

1 Introduction

1.1 Penalties

In most work on preemptive scheduling, preemption is assumed to be free. However, preemptions may actually be quite costly or have otherwise negative impact, e.g., customer dissatisfaction, so that we should aim at minimizing preemption if possible. One way to model this is to introduce *preemption penalties*, i.e., a penalty has to be paid for each preemption. Recently, several papers (Zheng et al. 2005; Fung 2008; Zheng et al. 2009; Thibault and Laforest 2009) studied algorithms for preemptive scheduling with penalties

where the objective is to maximize the total weight of completed jobs, minus penalties.

In this paper we introduce a related notion, perhaps more suitable from a customer's perspective, which we call *non-completion penalties*. Here preemption by itself does not cause penalty, as all the customer cares is that the job is completed on time. If a committed job is not delivered at the end, however, the algorithm needs to pay a penalty. This is akin to, for example, a customer giving a job to a company with a specific deadline, which the company promises will be completed on time, only to find out later that it cannot be completed—clearly this creates dissatisfaction to the customer. Such non-completion penalties are related to preemption penalties because, in the special case where a job once preempted is lost forever (for example if the job has a tight deadline so there is no room to process some other jobs in the middle of its execution), the two types of penalties are equivalent.

With non-completion penalties, it is necessary to give algorithms the ability to refuse jobs on arrival. As soon as a job arrives, the algorithm has to decide whether to accept or reject the job. If it is rejected, no profit is obtained (the customer goes away) but no penalty is imposed either. But if the job is accepted but cannot be finished by its deadline (for example if there are other “more important” jobs arriving later), not just the profit is lost but a penalty has to be paid.

We can further distinguish between *immediate notification* and *immediate decision* models: in immediate notification, the algorithm informs the customer whether the job is accepted, but not the time interval during which it will be processed, and the algorithm is free to reassign an accepted job to another time interval or to preempt a partially executed job. Immediate decision is a stronger notion where additionally the algorithm has to fix (tell the customer) during which time interval the job will be scheduled (for example a

S. P. Y. Fung (✉)
Department of Computer Science, University of Leicester,
Leicester, UK
e-mail: pyfung@mcs.le.ac.uk

company may want to advise its customers on when a delivery will take place, at the time the order is accepted). The algorithms in this paper can be used for both notification and decision models, but the analysis is done on the notification model which implies the same performance for the decision model (as we will explain later).

To be more precise, here are the models of penalties that we consider. In all cases, the penalty is proportional to the weight of the job.

(Preemption penalty) Jobs in execution can be preempted, and a penalty has to be paid for the preemption. The job can be restarted or resumed again later, depending on the model (this will be addressed shortly). Note that a job could be preempted (and restarted/resumed) multiple times, and it will then pay multiple penalties. No commitment to completing the job need to be made at a job's arrival.

(Immediate notification with non-completion penalty) At a job's arrival it must be decided immediately whether to accept or reject a job, but not the time interval to execute it. A rejected job will not be penalized (but also will not gain its weight). A penalty is incurred if an accepted job is not completed, but not for each individual preemption.

(Immediate decision with non-completion penalty) At a job's arrival it must be decided immediately whether to accept or reject a job and, if it is accepted, to allocate a time interval (with length equal to the length of the job) for its execution. This allocated time interval cannot be changed. However the job can be removed from the schedule (thus freeing the allocated time interval), either by discarding before it is started, or by preempting it during its execution. In either case the evicted job cannot be run again later (hence the job is forever lost). A penalty is incurred for such a non-completed job. This is the exact model in [Thibault and Laforest \(2009\)](#).

We can also view the penalty as imposed on the eviction of the job; in this sense this is a stronger model than preemption penalty (because of the additional commitment, and the fact that the evicted job cannot be restarted or resumed).

1.2 Definitions and notations

A job j is specified by its release time $r(j)$, deadline $d(j)$, processing time (length) $p(j)$ and weight $w(j)$. The expiry time of a job is defined as $d(j) - p(j)$. For a job that is released but not yet completed, if it has not past its expiry time, it is called *pending*, otherwise it has *expired*. The laxity $\alpha(j)$ of a job j is $(d(j) - r(j))/p(j)$. A job is *tight* if it has laxity 1, which means it must be scheduled immediately (or rejected) on arrival, i.e., there is no flexibility on when to run

the job. Each job of weight $w(j)$ gives rise to a penalty of $\rho w(j)$ if it is preempted or not completed (depending on the model), for some constant $\rho \geq 0$.

Jobs arrive *online*, meaning that they are not known before they arrive. The profit of a schedule is the total weight of jobs completed before their deadlines, minus all penalties paid. Our objective is to maximize the profit of the schedule. We will always use $\mathcal{OPT}$ to denote the optimal offline algorithm, and $\mathcal{ONL}$ to denote the online algorithm in question. When it is clear from context, they also denote the schedules produced by these algorithms on a particular input instance. $|\mathcal{OPT}|$ and $|\mathcal{ONL}|$ denote the profits of the schedules $\mathcal{OPT}$ and $\mathcal{ONL}$.

Note that $\mathcal{OPT}$ never needs to pay any non-completion penalties since it would know in advance whether a job can be completed before accepting it. However it may need to pay penalties in the preemption model because it may be beneficial to execute a job over several time intervals.

To measure the quality of online algorithms we use the standard definition of *competitive ratio*: an algorithm $\mathcal{ONL}$ is c -competitive if the profit returned by $\mathcal{ONL}$ on any instance I is at least $1/c$ that of the profit returned by the optimal offline algorithm $\mathcal{OPT}$ on the same instance.

1.3 Weight models

The general model defined above does not admit constant competitive algorithms, even without any preemption or non-completion penalties. Hence we focus on the following two special cases regarding the weights of jobs, commonly studied in the literature:

- (1) The *proportional weight* case (PW), where the weight of a job is proportional (or simply equal) to its length, i.e., $w(j) = p(j)$ for all j . In this case the objective is equivalent to maximizing the utilization of the processing resource.
- (2) The *unit length* case (UL), where jobs can have arbitrary weights, but all jobs have the same length. Without loss of generality this length can be assumed to be 1. Thus $p(j) = 1$ for all j and $w(j)$ is independent of other parameters of the job.

1.4 Preemption models

Two different preemption models have been considered in the literature:

- (1) *Preemption restart* (e.g., [Fung et al. 2008](#); [Hoogeveen et al. 2000](#)): a job can be preempted, but it can only restart from the beginning if executed again later, i.e., all current progress will be lost. Note that this is only useful for the online algorithm; $\mathcal{OPT}$ would not get any advantage

from restarts and therefore for OPT it is equivalent to a nonpreemptive model.

- (2) *Preemption resume* (e.g., Sanjoy et al. 1992; Koren and Shasha 1995): a job can be preempted, and it can resume execution at the point where it is preempted.

Both preemption penalties and immediate notification with non-completion penalties can be considered together with a restart or resume model. In the case of immediate notification, the restart or resume also refers to the relocation of a job that is accepted but not yet started. But for the case of immediate decision with non-completion penalties, there is no issue of restart or resume as the jobs are forever lost once evicted.

1.5 Previous work

The problem studied in Thibault and Laforest (2009) is closest to the models we study here. It gave an algorithm for immediate decision in the PW case, although it did not explicitly mention immediate decision. We are not aware of other results that address immediate decision/notification and penalties together in the preemptive setting, but there are other related work, which we detail below:

- (1) *Preemption penalty, proportional weight*: without penalties, a 4-competitive optimal bound was given in Sanjoy et al. (1992) for the resume model. For the case with penalties, Fung (2008) gave a lower bound $4(1 + \rho)$. If we generalize PW so that the value $w(j)/p(j)$ (sometimes called density, i.e., weight per unit length) of any two jobs differ by at most a factor of k (so PW is the case when $k = 1$), a $(1 + \sqrt{k})^2$ -competitive algorithm was given in Koren and Shasha (1995) for the case without penalties in the resume model. This bound is optimal as a matching lower bound was given in Sanjoy et al. (1992). This lower bound was generalized to $(\sqrt{\rho + k} + \sqrt{\rho + 1})^2$ for the case with penalties (Fung 2008).
- (2) *Preemption penalty, unit length*: without penalties, a 4.24 upper bound was given in Nguyen (2011) that works for both resume and restart models. For tight jobs an optimal upper bound of 4 was known (Woeginger 1994). With preemption penalties, a $(2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 1})$ -competitive algorithm was given in Zheng et al. (2005) for the restart model. A lower bound of $4(1 + \rho)$ was proved in Fung (2008). For the more general case where the ratio between maximum and minimum job length is Δ , tight bounds $\Theta(\Delta / \log \Delta)$ were given in Ting (2006) for the restart model without penalties, and an analogous upper bound for the resume model was given in Dürr et al. (2009).

(The work in Ting (2006) concerns a slightly different *broadcast scheduling* problem but the results apply here.) With restart and penalties, a $((1 + \rho)\Delta + o(\Delta))$ -competitive algorithm was given in Zheng et al. (2009), and a lower bound of $((1 + 1/\rho)^{1/\Delta} - 1)^{-1} + 1 \approx \Delta / \ln(1 + 1/\rho)$ was given in Fung (2008).

- (3) *Immediate notification or decision*: results in (1) and (2) above did not consider immediate decision or notification. The idea of immediate notification and immediate decision was proposed and studied in earlier papers (Goldwasser 2003; Ding and Zhang 2006; Ding et al. 2007; Ebenlender and Sgall 2008), but all these papers were concerned with nonpreemptive scheduling. The PW case was considered in Goldwasser (2003), while Ding and Zhang (2006) and Ding et al. (2007) considered the UL+PW case (jobs are of equal length and equal weight). Goldwasser (2003) also gave results when the laxity of jobs is a parameter.

1.6 Our results

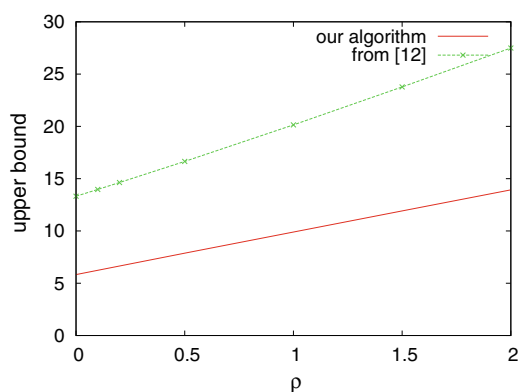
In this paper we give new algorithms for the preemption or non-completion penalty models defined above.

- For preemption penalty, we give a $4(1 + \rho)$ -competitive algorithm for the PW case. The algorithm and the analysis works for both restart or resume models. This matches the lower bound in Fung (2008).
- For immediate notification or decision in the PW case, we give an algorithm with competitive ratio $2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 2}$. It works for both restart and resume models. This bound is better than the one in Thibault and Laforest (2009). For example when $\rho = 0, 0.2, 1$ this gives $r = 5.828, 6.650, 9.899$ respectively whereas Thibault and Laforest (2009) gives $r = 13.32, 14.63, 20.14$ respectively; see Fig. 1. (Although Thibault and Laforest (2009) considers multiple processors, our algorithm can also work for the multiple processors case, with the same competitive ratio and a straightforward extension of the analysis.)
- For the UL case with immediate decision, or immediate notification with restart, we give an upper bound of $4\rho + 5 + 4\sqrt{\rho^2 + 5\rho/2 + 3/2}$ on the competitive ratio. For UL with immediate notification and resume we give an upper bound of $6\rho + 7 + 6\sqrt{\rho^2 + 7\rho/3 + 4/3}$.
- We also show a lower bound for immediate decision in the PW case even if jobs have arbitrarily large laxity.

Table 1 summarizes the results (those marked * are new results in this paper).

Table 1 Summary of previous and new results

	Upper bound	Lower bound
Preemption penalty		
PW	$4(1 + \rho) *$	
Restart UL	$2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 1}$ (Zheng et al. 2005), 4.24 when $\rho = 0$ (Nguyen 2011)	
Resume UL	4.24 when $\rho = 0$ (Nguyen 2011)	$4(1 + \rho)$ (Fung 2008)
Non-completion penalty		
PW	$2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 2} *$	
Restart UL	$4\rho + 5 + 4\sqrt{\rho^2 + 5\rho/2 + 3/2} *$	
Resume UL	$6\rho + 7 + 6\sqrt{\rho^2 + 7\rho/3 + 4/3} *$	

**Fig. 1** Comparison of upper bounds.

2 Preemption penalties

In this section we give an algorithm that gives $4(1 + \rho)$ -competitiveness for preemption penalty in the PW case.

Algorithm 1 Let $\beta = 2(1 + \rho)$. At any time the algorithm maintains a pool of all pending jobs. Newly released jobs are added to the pool. If the algorithm is idle, start the pending job with the earliest expiry time. If while executing a job j , another pending job k reaches its expiry time, then preempt j and start k if (i) $w(k) > \beta w(j)$ and j has preempted other jobs before, or (ii) $w(k) > \alpha w(j)$ and j has not preempted other jobs before, where $\alpha = 2(\beta - 1) \geq \beta$. In either case the preempted job j is never started again. Otherwise, k will not be started (and will therefore expire).

The algorithm borrows ideas from the one in Sanjoy et al. (1992), and essentially follows a simple strategy of preempting a job if another expiring one is β times more valuable than it. The exception is the first preemption after a job completion (or at the very beginning), in which case this preemption ratio is higher.

Note that the algorithm never reschedules a preempted job even though the model allows it (in fact in our algorithm this is simply not possible for those jobs that have themselves preempted other jobs, as by the definition of the algorithm a preemption only happens when the preempting job is about to expire). Thus the algorithm can be used for both resume and restart models.

In the following analysis we ignore any penalties OPT may need to pay; such penalties can only improve the competitive ratio. In effect we are comparing against a more powerful OPT that is free to preempt and resume. We analyze the competitive ratio of the algorithm in the resume model, which implies the same ratio for the restart model because ONL is the same while OPT is more powerful in the resume model (with our ‘gift’ that OPT does not need to pay penalties).

In the resume model, OPT may divide a job into several parts and run them in different time intervals. We refer to each of these as a *job fragment*.

We first define the concept of *basic subschedules*, which will be used throughout the paper. The schedule produced by ONL can be viewed as a sequence of preempted or completed jobs. The maximal sequence of jobs $j_1, j_2, \dots, j_k$, such that j_i is preempted by j_{i+1} for all $i < k$ and only j_k is completed, is called a basic subschedule. We denote by $B = (w_1, \dots, w_k)$ the weights of the jobs in the basic subschedule.

Lemma 1 For any basic subschedule $B = (w_1, \dots, w_k)$,
 (i) If $k \geq 2$ then $w_i < w_k / \beta^{k-i}$ for $2 \leq i < k$ and $w_1 < w_k / (\alpha \beta^{k-2})$,
 (ii) If a job j expires (and is not executed by ONL) while ONL is running some job in B , then $w(j) \leq \beta w_k$ if $k \geq 2$, and $w(j) \leq \alpha w_1$ if $k = 1$.

Proof (i) This is because $w_2 > \alpha w_1$ and $w_i > \beta w_{i-1}$ for $i > 2$ since this is how the algorithm preempts jobs.

(ii) Suppose $\mathcal{ONL}$ is running the job with weight w_p when j expires. If $p \geq 2$ then we have $w(j) \leq \beta w_p \leq \beta w_k$, since otherwise $\mathcal{ONL}$ would have preempted this job to start j . Similarly if $p = k = 1$ then $w(j) \leq \alpha w_1$. If $p = 1$ and $k \geq 2$ then $w(j) \leq \alpha w_1 < w_2 \leq w_k \leq \beta w_k$. $\square$

Theorem 1 Algorithm 1 is $4(1 + \rho)$ -competitive.

Proof Recall that we waive any penalties $\mathcal{OPT}$ may need to pay. The profit of $\mathcal{OPT}$ is therefore equal to its total execution time, which can be divided into two parts: those while $\mathcal{ONL}$ is busy and those while $\mathcal{ONL}$ is idle. Consider each job fragment in $\mathcal{OPT}$. If the execution time of a job fragment overlaps both busy times and idle times in $\mathcal{ONL}$, we divide it into further parts so that each part is wholly contained in either $\mathcal{ONL}$'s busy times or idle times. Furthermore a job fragment overlapping two or more basic subschedules in $\mathcal{ONL}$ can be similarly partitioned so that each fragment overlaps only one basic subschedule. So from now on we can assume each job fragment falls wholly into a basic subschedule in $\mathcal{ONL}$ or into $\mathcal{ONL}$'s idle times.

For job fragments in $\mathcal{OPT}$ that are contained in times while $\mathcal{ONL}$ is busy, we simply associate them to the basic subschedule running in $\mathcal{ONL}$ at the same time.

In the following we consider job fragments in $\mathcal{OPT}$ that are run while $\mathcal{ONL}$ is idle. Consider one such job fragment and the expiry time of the job corresponding to this job fragment. It must belong to one of the following two cases:

(1) The expiry time is during some idle time in $\mathcal{ONL}$. This case is only possible if the job is not pending just before the expiry time, i.e., $\mathcal{ONL}$ started the job earlier and have either completed or preempted it already. (Recall that a preempted job will not be considered again according to our algorithm.) This means $\mathcal{ONL}$ started the job before its expiry time, meaning that it must be the first job in some basic subschedule B_0 (recall that the algorithm only preempts jobs if the preempting job is about to expire). In this case we associate this job fragment with B_0 .

(2) The expiry time is when $\mathcal{ONL}$ is running some job in some basic subschedule $B = (w_1, \dots, w_k)$. (Note that the job can have expired in $\mathcal{ONL}$ but $\mathcal{OPT}$ still runs it, because $\mathcal{OPT}$ may have completed part of the job earlier on.) We associate this job fragment with B .

From the above discussion, all job fragments started by $\mathcal{OPT}$ while $\mathcal{ONL}$ is idle is associated with some basic subschedules. We now consider the job fragments associated with each basic subschedule. Consider a basic subschedule $B = (w_1, w_2, \dots, w_k)$, and let f be the finishing time of the last job in B . Job fragments of total length at most $w_1 + \dots + w_k$ are started and ended within B and are therefore associated with B . In addition, there can be one job of length w_1 associated with B due to case (1) above.

Next, consider all job fragments $j_1, \dots, j_m$ associated with B due to case (2). Let j be the job corresponding to

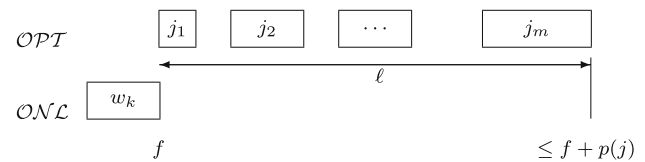


Fig. 2 Bounding case (2) job fragments

the last job fragment j_m (refer to Fig. 2). The expiry time of j is before f ; hence its deadline is before $f + p(j)$, i.e., this execution of j in $\mathcal{OPT}$ must finish before $f + p(j)$ and hence the total time ℓ that $\mathcal{OPT}$ spends processing these job fragments is at most $p(j)$, i.e., $\ell \leq p(j) = w(j)$. Note that in-between all the j_i 's there could be many job fragments not associated with B but this only makes the bound better.

Therefore, the total length of $\mathcal{OPT}$ jobs associated with B is at most $(w_1 + \dots + w_k) + w(j) + w_1$. Using Lemma 1, for $k \geq 2$,

$$\begin{aligned} |\mathcal{OPT}| &\leq \left(\frac{1}{\alpha\beta^{k-2}} + \frac{1}{\beta^{k-2}} + \dots + 1 \right) w_k + \beta w_k \\ &\quad + \frac{1}{\alpha\beta^{k-2}} w_k \\ &= \left(\frac{2}{2(\beta-1)\beta^{k-2}} + \frac{1}{\beta^{k-2}} + \dots + 1 + \beta \right) w_k \\ &= \left(\frac{1}{(\beta-1)\beta^{k-2}} + \frac{1 - 1/\beta^{k-1}}{1 - 1/\beta} + \beta \right) w_k \\ &= \left(\frac{1}{(\beta-1)\beta^{k-2}} - \frac{\beta}{(\beta-1)\beta^{k-1}} + \frac{\beta}{\beta-1} + \beta \right) w_k \\ &= \frac{\beta^2}{\beta-1} w_k. \end{aligned}$$

$\mathcal{ONL}$ gets a profit of w_k in B , but pays penalties of $\rho(w_1 + \dots + w_{k-1}) \leq \rho(1/\alpha\beta^{k-2} + 1/\beta^{k-2} + \dots + 1/\beta)w_k \leq \rho(1/\beta^{k-1} + 1/\beta^{k-2} + \dots + 1/\beta)w_k < \rho w_k/(\beta-1)$. Thus the competitive ratio r is at most

$$r \leq \frac{\beta^2/(\beta-1)}{1 - \rho/(\beta-1)} = \frac{\beta^2}{\beta-1-\rho}$$

This is minimum when $\beta = 2(1 + \rho)$, in which case $r = 4(1 + \rho)$.

If $k = 1$, then a similar analysis can show that the total weight of job fragments associated with $\mathcal{OPT}$ is at most $(1 + \alpha + 1)w_1 = 2\beta w_1$ while $\mathcal{ONL}$ gets w_1 , so $r = 2\beta = 4(1 + \rho)$ again. $\square$

3 Non-completion penalties

3.1 Proportional weight

In this subsection we give an algorithm for the proportional weight case. Before we state the algorithm, we first define

a few terms. At any point during $\mathcal{ONL}$'s execution, the job that is running is called the *current* job. On arrival, a job is either *accepted* or *rejected* immediately. An accepted job, if not eventually *completed*, is said to have been *evicted*. An accepted job can be evicted either because it is *preempted*, i.e., it was started but then preempted during execution; or *discarded*, i.e., it is removed before it can start. By *final schedule* we refer to the schedule produced by $\mathcal{ONL}$ containing the preempted and completed jobs (but not the discarded jobs) and when they are run. The *provisional schedule at time t* , denoted by S_t , refers to the schedule planned by $\mathcal{ONL}$ for all its accepted jobs at time t , including the current job. For $u \geq t$, $S_t(u)$ denotes the job planned to run (or running) at time instant u of S_t . If nothing is scheduled at u (it is idle) then $S_t(u) = \emptyset$.

Algorithm 2 Let $\beta > 1$ be a parameter that depends on ρ . Its value will be specified later. When a new job j arrives, identify a set of accepted jobs (including the current one) in the provisional schedule such that their total weight is less than $w(j)/\beta$ and their eviction would create a single time interval long enough to schedule j . (If there are more than one such job sets, choose any one.) If no such set of jobs exists then reject j . Otherwise, evict those jobs, and allocate j to start as early as possible in the newly-created empty time interval (leaving no idle time after the preceding job). The evicted jobs will not be considered again.

Note that the algorithm does not use any restart or resume ability. Clearly the algorithm satisfies the immediate decision property and hence also immediate notification. In the following proof however we analyze it in the immediate notification model with resume, i.e., we prove that it is competitive against an $\mathcal{OPT}$ that may preempt and resume jobs. This implies the same competitive ratio against optimal algorithms with restart or with immediate decision, because $\mathcal{OPT}$ can only become less powerful if not allowed to resume jobs, and $\mathcal{OPT}$ can always announce in advance the time interval during which it will schedule a job without hurting its performance.

We will see shortly how this algorithm can be implemented efficiently. Before we prove the competitive ratio, we first show several key properties of the resulting schedule.

Lemma 2 *At any time t , if $S_t(u) = \emptyset$ then $S_t(u') = \emptyset$ for all $u' > u$. That is, there is no idle time between jobs in the provisional schedule. Moreover, an eviction always evicts a suffix of the provisional schedule.*

Proof Clearly there is no idle time if the provisional schedule contains only one job (the current one). Newly arriving jobs are scheduled as early as possible in their time slot, that is, without leaving idle time before them. If such a new job evicts other jobs, then these jobs have total weight, and hence

total length, at most $1/\beta$ times that of the new job. That is, the total length of the evicted jobs is strictly smaller. By induction, the evicted jobs were scheduled consecutively. Hence, if there is any job which follows the newly scheduled job in the provisional schedule, this implies there was idle time between jobs in the previous provisional schedule, a contradiction. Hence there cannot be any jobs after the evicted jobs in the provisional schedule, which means the evicted part is a suffix. $\square$

The fact that the evicted part is a suffix allows for efficient implementation as there is only a linear number of candidate set of jobs to be considered.

Lemma 3 *Consider a time u . If at some time $t < u$, $S_t(u) \neq \emptyset$, then $S_{t'}(u) \neq \emptyset$ for all $t < t' \leq u$. That is, if some job is allocated to run at time u in the provisional schedule, then u will never become idle again.*

Proof Since jobs are not moved once they are assigned a time slot, the only reason the time instant u could become idle again is that the job that occupies time u is preempted. But since the preempting job must be longer than the total length of all preempted jobs, and the preempting job is scheduled so as to leave no idle time with the preceding job in the provisional schedule, so it must also occupy time u . $\square$

We now analyze the competitive ratio by mapping profits obtained in $\mathcal{OPT}$ to those obtained in $\mathcal{ONL}$. We partition the set of jobs completed by $\mathcal{OPT}$ into three sets: C (Completed), those completed by both $\mathcal{ONL}$ and $\mathcal{OPT}$; R (Rejected), those rejected immediately by $\mathcal{ONL}$ upon arrival; E (Evicted), those accepted by $\mathcal{ONL}$ on arrival but subsequently evicted (either discarded or preempted). Let $w(X)$ denote the total weight of jobs in a set X .

We consider the final schedule of $\mathcal{ONL}$, and partition it into *busy periods* and *idle periods* which are maximal time intervals during which $\mathcal{ONL}$ is executing jobs, or idle, respectively.

Lemma 4 *No job in R has its expiry time fall inside an idle period in the final schedule of $\mathcal{ONL}$.*

Proof Suppose a job j in R has its expiry time inside an idle period $[t, u)$. As the final schedule of $\mathcal{ONL}$ is idle at time t , by Lemma 3, the provisional schedule when j arrives, $S_{r(j)}$, must also be idle at time t , i.e., $S_{r(j)}(t) = \emptyset$. Hence by Lemma 2, $S_{r(j)}(t') = \emptyset$ for all $t' \in [t, \infty)$, and thus $\mathcal{ONL}$ could have run the whole of j starting at t (instead of rejecting it). $\square$

Consider a busy period which consists of k basic sub-schedules, i.e., with k completed jobs $j_1, \dots, j_k$. Each completed job may be preceded by some preempted jobs, which are (transitively) preempted by the associated completed job.

The following lemma argues that although a job in R may be scheduled in $\mathcal{OPT}$ in such a way that it overlaps several busy periods, it is sufficient to associate it with just one busy period: the one during which it expires.

Lemma 5 *If the expiry time of a job j in R is within a busy period with completed jobs $j_1, \dots, j_k$, then (i) j is released on or after the beginning of this busy period, and (ii) $w(j) \leq \beta \sum_{i=1}^k w(j_i)$.*

Proof (i) Let $[t, u)$ be the busy period. If $r(j) < t$, then as $\mathcal{ONL}$ is idle just before t , by Lemma 3, the provisional schedule when j arrives is also idle at t , i.e., $S_{r(j)}(t) = \emptyset$. Hence by Lemma 2, $S_{r(j)}(t') = \emptyset$ for all times $t' \in [t, \infty)$, and so j could not be rejected.

(ii) Let $q_1, \dots, q_m$ be the jobs that are in the provisional schedule at time $r(j)$. Similar to (i), since the final schedule is idle at u , Lemmas 3 and 2 together gives $S_{r(j)}(t') = \emptyset$ for all times $t' \in [u, \infty)$. Thus all these jobs are scheduled to start on or after t and complete before u in the provisional schedule, i.e., in the busy period $[t, u)$. Moreover $w(j) \leq \beta \sum_{i=1}^m w(q_i)$ or else j would evict them. Each of these q_i 's may be evicted later by other jobs, but the evicting jobs must appear in the same busy period since there is no gap in the provisional schedule. Thus this must eventually lead to one or more of the j_i 's. Each of these evicting jobs must have larger weight than the total weight of its evicted jobs, thus if $w(j) \leq \beta \sum_{i=1}^m w(q_i)$ then $w(j) \leq \beta \sum_{i=1}^k w(j_i)$ as well. $\square$

Theorem 2 *Algorithm 2 has a competitive ratio of $2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 2}$ with $\beta = 1 + \rho + \sqrt{\rho^2 + 3\rho + 2}$ for the proportional weight case, in the immediate notification with resume model.*

Proof From the above two lemmas, it is sufficient to analyze each busy period separately: each job in R is associated with the busy period during which it expires, each job in E is associated with the busy period where it was planned, and each job in C is associated with the busy period during which it is completed. From now on, we consider one busy period with k completed jobs $j_1, \dots, j_k$, and C, R, E refer to jobs associated with this busy period only. Let $W = \sum_{i=1}^k w(j_i)$. Clearly $w(C) \leq W$. In the following we give bounds on $w(R)$ and $w(E)$.

A completed job j_i evicts jobs of total weight at most $w(j_i)/\beta$ by the definition of the algorithm. Each of these evicted jobs may in turn have evicted other jobs of total weight at most $1/\beta$ of their own weight, and so on. Thus the sum of weights of evicted jobs associated with j_i is at most $w(j_i)/\beta + w(j_i)/\beta^2 + \dots < w(j_i)/(\beta - 1)$. Therefore, $w(E) < \sum_{i=1}^k \frac{1}{\beta-1} w(j_i) = \frac{1}{\beta-1} W$.

Let L be the length of this busy period. Since weights are equal to lengths, and only jobs in C and E appear in the final

schedule, $L \leq w(C) + w(E)$. Note that preempted jobs do not have their full lengths appearing in the busy period, and discarded jobs do not appear in the final schedule at all. Still, we have

$$L \leq \sum_{i=1}^k \left(\frac{w(j_i)}{\beta-1} + w(j_i) \right) = \sum_{i=1}^k \frac{\beta}{\beta-1} w(j_i) = \frac{\beta}{\beta-1} W.$$

For each job j in R associated with this busy period, its release time must be during this busy period and its deadline must be at most $p(j)$ after the end of this busy period. Hence the total length, or weight, of jobs in R associated with this busy period is at most $L + \beta W$ (using Lemma 5). Therefore

$$\begin{aligned} |\mathcal{OPT}| &= w(C) + w(E) + w(R) \\ &\leq W + \frac{1}{\beta-1} W + (L + \beta W) \\ &\leq \left(\frac{2\beta}{\beta-1} + \beta \right) W. \end{aligned}$$

$\mathcal{ONL}$ gets a profit of W for completing the jobs, but pays a penalty for all evicted jobs, thus $|\mathcal{ONL}| \geq W - \rho(\frac{1}{\beta-1} W)$. Therefore the competitive ratio r is at most

$$r \leq \frac{\frac{2\beta}{\beta-1} + \beta}{1 - \frac{\rho}{\beta-1}} = \frac{2\beta + \beta(\beta-1)}{\beta-1-\rho} = \frac{\beta^2 + \beta}{\beta-1-\rho}.$$

It can be verified by differentiation that the minimum is attained when we set $\beta = 1 + \rho + \sqrt{\rho^2 + 3\rho + 2}$, and then $r = 2\rho + 3 + 2\sqrt{\rho^2 + 3\rho + 2}$. $\square$

3.2 Unit length, restart model

Next we consider the unit length case. We study the same algorithm, Algorithm 2. Again, the algorithm obeys the immediate decision and hence notification property. The analysis in this subsection, however, only works for immediate notification with restart (and hence immediate decision): it relies on the fact that $\mathcal{OPT}$ completes each job in a unit length time window. In the next subsection we will consider the resume model.

Note that in this case, since all jobs are of the same length, evicting a single accepted job (except the current job, which only has a portion of its length left) will create enough space to fit in the new job. Thus an eviction will either discard one job that is not the current job, or evict two jobs, preempting the current job and discarding the following one. This allows for efficient implementation as the checking for eviction only need to be made against each individual job in the provisional schedule before the new job's deadline, or the current job plus the job following it.

Theorem 3 *Algorithm 2 has a competitive ratio of $4\rho + 5 + 4\sqrt{\rho^2 + 5\rho/2 + 3/2}$ for the unit length case, in the immediate notification with restart model.*

Proof We can follow a similar method to the proof of Theorem 2. We define C , E and R as before, but for the whole schedule. Similar to the proof of Theorem 2 we have $w(C) \leq W$ and $w(E) \leq \frac{1}{\beta-1}W$, where W is the sum of weight of all jobs completed by $\mathcal{ONL}$. However we bound R differently.

Consider the time t when a job j in R is scheduled by $\mathcal{OPT}$. Since it is rejected, at the time of its rejection the time interval $[t, t+1]$ is occupied by some job(s) in the provisional schedule such that their total weight is at least $1/\beta$ that of j . In fact, since jobs are of the same length, there are at most two such jobs. We charge $w(j)$ to these jobs proportional to their weights, i.e., if there are two such ‘blocking’ jobs x and y then they receive a charge of $(w(x)/(w(x)+w(y)))w(j)$ and $(w(y)/(w(x)+w(y)))w(j)$ respectively, and if there is only one job then it receives a charge of $w(j)$. Each such charge to a job x in the provisional schedule is therefore of value at most $\beta w(x)$. Each job in the provisional schedule is eventually either completed or evicted; thus they belong to either C or E . Because of the unit-length property, each job in C or E blocks at most two jobs in R . Moreover, once inserted into the provisional schedule, they do not move to a different time interval, and hence each such charge will only be associated with one job in C or E . Hence, each job in C or E receives at most two charges. Thus we have $w(R) < 2\beta(w(C)+w(E))$. Furthermore $w(C)$ and $w(E)$ are bounded as in Theorem 2. So

$$\begin{aligned} |\mathcal{OPT}| &= w(C) + w(E) + w(R) \\ &\leq (2\beta + 1)(w(C) + w(E)) \\ &\leq (2\beta + 1) \left(W + \frac{1}{\beta - 1} W \right) \\ &= (2\beta + 1) \left(\frac{\beta}{\beta - 1} \right) W. \end{aligned}$$

Again $|\mathcal{ONL}| \geq W - \rho(\frac{1}{\beta-1}W)$. Thus the competitive ratio is at most

$$r \leq \frac{(2\beta + 1)\beta/(\beta - 1)}{1 - \rho/(\beta - 1)} = \frac{(2\beta + 1)\beta}{\beta - 1 - \rho}.$$

This is minimized by choosing $\beta = 1 + \rho + \sqrt{\rho^2 + 5\rho/2 + 3/2}$, in which case $r = 4\rho + 5 + 4\sqrt{\rho^2 + 5\rho/2 + 3/2}$. $\square$

3.3 Unit length, resume model

It turns out that Algorithm 2, which does not use resume, also gives an upper bound, albeit somewhat weaker, for the resume model. We follow the same line of analysis, in particular we bound the C and E jobs in the same way, but we bound the R jobs differently.

We first note a structural property of the $\mathcal{OPT}$ schedule.

Lemma 6 Suppose $\mathcal{OPT}$ schedules a job j_1 in more than one fragment, say in time intervals $[a, b)$ and $[c, d)$, and

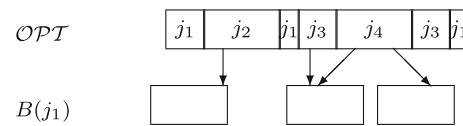


Fig. 3 Blocking sets of a job j_1 with a possible assignment of charges from jobs nested inside. Note that although in this picture charges of all nested jobs are made to $B(j_1)$, in general they may be made to jobs that are not in $B(j_1)$ as their own blocking sets may be different

another job j_2 is ever started at some point t between b and c . Then no part of j_2 can be scheduled outside of $[b, c)$ by $\mathcal{OPT}$. (That is, j_2 must be completely scheduled inside $[b, c)$, though not necessarily in one piece.)

Proof We can without loss of generality assume that every job that $\mathcal{OPT}$ starts will eventually complete, and that $\mathcal{OPT}$ schedules in an earliest deadline first manner (breaking ties in some consistent way), i.e., at any time instant, if there are two pending jobs then the one with the earlier deadline will be scheduled. This follows from a standard exchange argument. Since j_2 was started at t while j_1 is not yet complete, j_2 must have an earlier deadline than j_1 . Hence the second fragment of j_1 cannot start until j_2 is completed. Similarly, if j_2 was started before a then its earlier deadline implies that j_1 cannot start until j_2 is complete, contradicting that j_2 is executed at t . $\square$

It follows from the above lemma that $\mathcal{OPT}$ has a nested structure, i.e., if a job is scheduled in multiple fragments, then any jobs scheduled between two fragments must be wholly contained between the two fragments (see Fig. 3).

For each job j in R , let $s(j)$ denote the time when j is first started in $\mathcal{OPT}$, and $f(j)$ denote the time when j is completed in $\mathcal{OPT}$. The span of j is defined as the time interval $[s(j), f(j))$.

Lemma 7 Consider a job j in $\mathcal{OPT}$ with $l - 1$ other jobs nested within. Then $f(j) - s(j) \geq l$. Moreover if $f(j) - s(j) < 2$ then it must be that $f(j) - s(j) = 1$, i.e., there cannot be nested jobs inside j , and that j must be scheduled in one piece in $\mathcal{OPT}$.

Proof It follows from the fact that all these $l - 1$ other jobs are fully scheduled inside j and all jobs are of length 1. The second statement follows similarly. When there are no other jobs in between we can without loss of generality assume $\mathcal{OPT}$ schedules a job in one piece. $\square$

For each job j in R , we consider the set of jobs in the provisional schedule at the instant when j arrives (i.e., $S_r(j)$) that overlaps, fully or partially, the span of j . We call it the *blocking set* of j , denoted by $B(j)$. The job in $B(j)$ that intersects $s(j)$, if there is one, is called the *head*; similarly the one that intersects $f(j)$, if it exists, is called the *tail*. All other jobs in $B(j)$ forms the *body*.

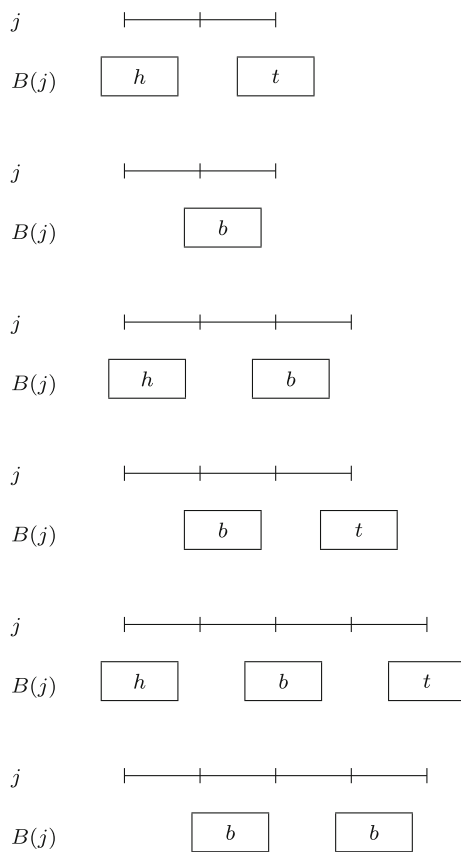


Fig. 4 Minimum number of blocking jobs, with spans of 2–4 (h = head, b = body, t = tail)

Lemma 8 Consider a job j in R with span at least l , where $l \geq 2$ is a positive integer. There must be at least one non-head job in $B(j)$. If the first non-head job in $B(j)$ starts on or after $s(j) + 1$ in the provisional schedule, then there must be a head job and there are at least $\lceil l/2 \rceil$ non-tail jobs in $B(j)$. Otherwise, there are at least $\lfloor l/2 \rfloor$ body jobs in $B(j)$.

Proof It follows from the fact that $\mathcal{ONL}$ schedules each job in one piece (no resume used), each job is of length 1, and any gap in the provisional schedule between two jobs in $B(j)$ must have length < 1 (otherwise j would not be rejected). See Fig. 4 for the worst-case arrangements. $\square$

Essentially, the blocking set is the set of jobs that prevents j from being scheduled (in the same time interval where $\mathcal{OPT}$ schedules it) because of their weights: each job in the body must have weight at least $w(j)/\beta$ or otherwise j can evict that job and be put into the provisional schedule, instead of being rejected. For head or tail jobs, it may require evicting the job next to them as well in order for j to be scheduled. In those cases, we can only argue that the sum of weights of the two jobs is at least $w(j)/\beta$.

Our approach is to charge each job j in R to one or two jobs of $B(j)$, in such a way that each job x that ever appeared in some $B(j)$ will receive at most three charges, and that each

charge is of weight at most $\beta w(x)$. Since each job that has ever appeared in some $B(j)$ will be either in C or E , this indirectly bounds R .

Note that a job, once added to the provisional schedule, will stay at the same location until it is either completed or evicted. When we consider different jobs in $\mathcal{OPT}$, even if they are ‘close’ to each other in $\mathcal{OPT}$, the provisional schedules at their release times (and hence their blocking sets) may be very different, but this does not affect our analysis. What is important is that each job, during its lifetime in the provisional schedule, only receives charges from jobs in $\mathcal{OPT}$ that are ‘close’ to them.

First we consider the innermost nested jobs, i.e. those with span 1 and are scheduled in one piece in $\mathcal{OPT}$ (Lemma 7). We create charges as in the restart case in the previous subsection. If there is a job in $B(j)$ the eviction of which alone can schedule j (which must be the case for a body job), a charge of value $w(j)$ is created and charged to it; we call this a *whole charge*. Otherwise j is rejected because of the combined blocking of two jobs, then we create two charges to these two jobs, distributing the value of $w(j)$ to each charge proportional to the two job weights. We call them *split charges*. Each whole or split charge to a job x has weight at most $\beta w(x)$.

For any two jobs such that one is not nested within the other, their spans must be disjoint and therefore their blocking sets must also be disjoint except that the tail of one may be the head of the other. We use this fact to assign our charges so that each group of nested jobs can be considered separately from the others.

Consider a job j with $l - 1$ jobs nested within, $l \geq 2$. Only whole charges will be made, and we also make sure that the whole charges of j and all nested jobs, except the innermost ones, are only made to the head or body of the blocking set, never the tail. As long as this property is maintained over the entire schedule, the body of $B(j)$ will not receive charges from outside of j and its nested jobs, and the same holds for the head except that it may receive one charge from the innermost nested job in a preceding group of jobs.

To assign the charge for j , we first recursively assign charges for jobs nested within j . Suppose among these $l - 1$ nested jobs, there are y split charges made to the non-tail jobs in $B(j)$.

Lemma 9 If there are y split charges in this group of jobs, then the split charges must come from at least $\lceil y/2 \rceil$ jobs in $\mathcal{OPT}$, and are assigned to at least $\lceil y/2 \rceil$ different jobs in $B(j)$. Moreover, if y is even, one of these two terms is at least one greater (i.e., either there are at least $\lceil y/2 \rceil + 1$ jobs in $\mathcal{OPT}$ making split charges to at least $\lceil y/2 \rceil$ jobs in $B(j)$, or there are at least $\lceil y/2 \rceil$ jobs in $\mathcal{OPT}$ making split charges to at least $\lceil y/2 \rceil + 1$ jobs in $B(j)$).

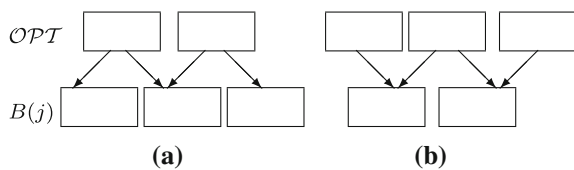


Fig. 5 An example with $y = 4$. **(a)** If it comes from two $\mathcal{OPT}$ jobs, it must be charged to at least three different jobs in $B(j)$. **(b)** If it is charged to only two jobs, it must come from at least three jobs

Proof The first part is easy to see because each job in $\mathcal{OPT}$ makes at most two split charges and each job in $B(j)$ can take at most two split charges. In addition, if y is even, it is impossible for the y charges to be packed into exactly $y/2$ $\mathcal{OPT}$ jobs and assigned to exactly $y/2$ jobs in $B(j)$, because split charges originating from the same job in $\mathcal{OPT}$ must be assigned to different jobs in $B(j)$; see Fig. 5. $\square$

We consider different cases.

Case 1: The first non-head job in $B(j)$ starts on or after $s(j) + 1$ in the provisional schedule. This implies that there is a head job, and that the head job has weight at least $w(j)/\beta$, meaning that it can accept whole charges. The head may receive a charge from a previous job; let $x \in \{0, 1\}$ be an indicator variable with $x = 1$ if such a charge exists and 0 otherwise. By Lemma 8 and 9, the number of non-tail jobs in $B(j)$, all of which can take charges, is at least $\max(\lceil l/2 \rceil, \lceil (x + y)/2 \rceil)$. (The x is either a split charge, or if it is a whole charge then there will be no split charge because it must come from the same innermost nested job, so it can be treated as a split charge here.) Recall that each job can take up to three charges. So the remaining number of charges that can be made to these non-tail jobs in $B(j)$, after taken up by the $x + y$ charges, is therefore at least $3 \max(\lceil l/2 \rceil, \lceil (x + y)/2 \rceil) - (x + y)$. The number of jobs nested within j that make whole charges, including j itself, is at most $l - \lceil y/2 \rceil$ after deducting those that make split charges (Lemma 9). If

$$3 \max(\lceil l/2 \rceil, \lceil (x + y)/2 \rceil) - (x + y) \geq l - \lceil y/2 \rceil \quad (1)$$

then we have enough room to make all the charges. It can be verified that (1) holds for all x and y .

Case 2: The first non-head job starts before $s(j) + 1$. It means even if there is a head job, its weight may be less than $w(j)/\beta$, meaning it may not be able to take whole charges. (It can always take split charges.) In this case we show how to assign all whole charges to body jobs only. We also need not consider the charge from earlier jobs that may exist (the x variable in (1)) as it can only be assigned to the head.

If there is a head job and at least one of the y split charges goes to the head, then by only considering the body jobs and using Lemma 8, (1) becomes

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil - 1) - (y - 1) \geq l - \lceil y/2 \rceil \quad (2)$$

Otherwise, all the jobs that receive the y split charges must be body jobs, so the formula becomes

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil) - y \geq l - \lceil y/2 \rceil \quad (3)$$

Both formulas can be shown to hold when l is even, or when l is odd and y is odd.

For the case where l is odd and y is even, we apply Lemma 9: if at least one of the y split charges goes to the head, then (2) should be modified to either

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil) - (y - 1) \geq l - \lceil y/2 \rceil \quad (4)$$

or

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil - 1) - (y - 1) \geq l - \lceil y/2 \rceil - 1 \quad (5)$$

Otherwise, (3) should be modified to either

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil + 1) - y \geq l - \lceil y/2 \rceil \quad (6)$$

or

$$3 \max(\lceil l/2 \rceil, \lceil y/2 \rceil) - y \geq l - \lceil y/2 \rceil - 1 \quad (7)$$

All four formulas hold for odd l and even y .

Note that since all the $l - 1$ other jobs are nested inside j , as long as they (except the innermost ones) only charge to the head and body of their own blocking sets, these charges will never be made to the tail of $B(j)$, maintaining the property we set before.

From the above it follows that $w(R) \leq 3\beta(w(C) + w(E))$. Using this, a similar analysis to that of Theorem 3 follows:

$$\begin{aligned} |\mathcal{OPT}| &= w(C) + w(E) + w(R) \\ &\leq (3\beta + 1)(w(C) + w(E)) \\ &= (3\beta + 1) \left(\frac{\beta}{\beta - 1} \right) W \\ |\mathcal{NCL}| &\geq W - \rho \left(\frac{1}{\beta - 1} W \right) \end{aligned}$$

Competitive ratio $r \leq \frac{(3\beta+1)\beta/(\beta-1)}{1-\rho/(\beta-1)} = \frac{(3\beta+1)\beta}{\beta-1-\rho}$, which is minimized by choosing $\beta = 1 + \rho + \sqrt{\rho^2 + 7\rho/3 + 4/3}$. We therefore have

Theorem 4 Algorithm 2 has a competitive ratio of $6\rho + 7 + 6\sqrt{\rho^2 + 7\rho/3 + 4/3}$ for the unit length case, in the immediate notification with resume model.

3.4 Lower bounds

Note that the lower bound of $4(1 + \rho)$ in Fung (2008) (either proportional weight or unit length) for preemption penalty can be applied to the non-completion penalty model as well with either immediate decision or notification. This is because the proof used only tight jobs; in such cases acceptance or rejection must be decided on arrival anyway, there

is no choice of when to schedule the job, and an evicted job cannot be started again later and will therefore not complete. Thus a preemption penalty is the same as a non-completion penalty.

It may be reasonable to assume in the immediate notification or decision models that all jobs come with certain laxity > 1 ; for example, in a home delivery situation, it is common that there are several days of lead time between order and delivery. Often, larger laxity allows for smaller competitive ratios (see e.g., Goldwasser 2003). However here we show:

Theorem 5 *For immediate decision with proportional weight, no algorithm can be better than 2-competitive even if all jobs have arbitrarily large laxity.*

Proof Consider any laxity $\alpha > 1$. Fix an arbitrarily small $0 < \epsilon < 1$. At time 0 release a job with length 1 and a very long deadline. Any online algorithm $\mathcal{ONL}$ needs to accept and start this job at some time $[t, t + 1)$ or else no more jobs are released and the competitive ratio is infinite. Choose a large positive integer N and define $x = (1 - \epsilon)/N$. N should be large enough so that $x(\alpha - 1) < \epsilon$. At each time $t, t + x, t + 2x, \dots, t + (N - 1)x$, a short job of length x and laxity α arrives, but if $\mathcal{ONL}$ ever accepts any one of these short jobs, the subsequent short jobs are not released. The last of these jobs has deadline $t + (N - 1)x + x\alpha = t + Nx + x(\alpha - 1) < t + (1 - \epsilon) + \epsilon = t + 1$, so all these jobs have deadlines before $t + 1$. No online algorithm would accept any of these short jobs as their weight is only x but the job with weight 1 will be missed (not to say a penalty of ρ has to be paid). Thus $\mathcal{ONL}$ obtains a profit of 1. $\mathcal{OPT}$ can schedule all jobs, getting a profit of $1 + Nx = 1 + (1 - \epsilon) = 2 - \epsilon$. Thus the competitive ratio cannot be better than $2 - \epsilon$. $\square$

4 Concluding Remarks

In this paper we give an optimal bound for scheduling jobs with proportional weight and preemption penalty. Note that optimal bounds are still not known for the unit length case, where the $4(1 + \rho)$ bound is optimal only for tight jobs. Perhaps more importantly though, we studied non-completion penalties in preemptive scheduling and gave competitive algorithms.

A natural question would be to establish upper and lower bounds that separate the competitiveness of the cases of immediate decision, immediate notification and the no-commitment case.

Acknowledgments Part of this work was done while the author was on study leave granted by the University of Leicester.

References

- Baruah, S. K., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L. E., et al. (1992). On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4, 125–144.
- Ding, J., Ebenlendr, T., Sgall, J., & Zhang, G. (2007). Online scheduling of equal-length jobs on parallel machines. In *Proceedings of 15th European Symposium on Algorithms*, Vol. 4698 of Lecture Notes in Computer Science (pp. 427–438). New York: Springer.
- Ding, J., & Zhang, G. (2006). Online scheduling with hard deadlines on parallel machines. In *Proceedings of 2nd International Conference on Algorithmic Aspects in Information and Management*, Vol. 4041 of Lecture Notes in Computer Science (pp. 32–42). New York: Springer.
- Dürr, C., Jez, L., & Nguyen, K. T. (2009). Online scheduling of bounded length jobs to maximize throughput. In *Proceedings of 7th International Workshop in Approximation and Online Algorithms*, Vol. 5893 of Lecture Notes in Computer Science (pp. 116–127). New York: Springer.
- Ebenlendr, T., & Sgall, J. (2008). A lower bound for scheduling of unit jobs with immediate decision on parallel machines. In *Proceedings of 6th Workshop on Approximation and Online Algorithms*, Vol. 5426 of Lecture Notes in Computer Science (pp. 43–52). New York: Springer.
- Fung, S. P. Y. (2008). Lower bounds on online deadline scheduling with preemption penalties. *Information Processing Letters*, 108(4), 214–218.
- Fung, S. P. Y., Zheng, F., Chan, W.-T., Chin, F. Y. L., Poon, C. K., & Wong, P. W. H. (2008). Improved on-line broadcast scheduling with deadlines. *Journal of Scheduling*, 11(4), 299–308.
- Goldwasser, M. H., & Kerbikov, B. (2003). Admission control with immediate notification. *Journal of Scheduling*, 6, 269–285.
- Hoogeveen, H., Potts, C. N., & Woeginger, G. J. (2000). On-line scheduling on a single machine: Maximizing the number of early jobs. *Operations Research Letters*, 27(5), 193–197.
- Koren, G., & Shasha, D. (1995). *D^{over}*: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24, 318–339.
- Nguyen, K. T. (2011). Improved online scheduling in maximizing throughput of equal length jobs. In *Proceedings of 6th International Computer Science Symposium in Russia*, Vol. 6651 of Lecture Notes in Computer Science (pp. 429–442). New York: Springer.
- Thibault, N., & Laforest, C. (2009). Online time constrained scheduling with penalties. In *Proceedings of 23rd IEEE International Symposium on Parallel and Distributed Processing*.
- Ting, H.-F. (2006). A near optimal scheduler for on-demand data broadcasts. In *Proceedings of 6th Italian Conference on Algorithms and Complexity*, Vol. 3998 of Lecture Notes in Computer Science (pp. 163–174). New York: Springer.
- Woeginger, G. J. (1994). On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1), 5–16.
- Zheng, F., Dai, W., Xiao, P., & Zhao, Y. (2005). Competitive strategies for on-line production order disposal problem. In *Proceedings of 1st International Conference on Algorithmic Applications in Management* (pp. 46–54).
- Zheng, F., Xu, Y., & Poon, C. K. (2009). On job scheduling with preemption penalties. In *Proceedings of 5th International Conference on Algorithmic Aspects in Information and Management*, Vol. 5564 of Lecture Notes in Computer Science (pp. 315–325). New York: Springer.