Optimal on-line algorithms to minimize makespan on two machines with resource augmentation [*]

Leah Epstein[†] Arik Ganot[‡]

Abstract

We study the problem of on-line scheduling on two uniformly related machines where the on-line algorithm has resources different from those of the off-line algorithm. We consider three versions of this problem, preemptive semi-online, non-preemptive on-line and preemptive on-line scheduling. For all these cases we design algorithms with best possible competitive ratios as functions of the machine speeds.

1 Introduction

The problem. We consider scheduling on two uniformly related machines with resource augmentation. Jobs arrive one by one, and each job has to be assigned before the next job arrives.
We consider three different versions, preemptive semi-online scheduling, non-preemptive on-line scheduling and preemptive on-line scheduling. *Semi-online* means that jobs must arrive in a decreasing order of size (processing time). *Preemptive* means that a job may be cut into a few pieces and scheduled on both machines, but two parts of the same job may not be scheduled concurrently on two different machines. This is allowed to both the on-line algorithm and the off-line algorithm.
The measure. A schedule is measured by *makespan*, i.e. the maximum completion time of any machine. An on-line algorithm is measured by its *competitive ratio*, which is defined $C = \sup_{\sigma} \left\{ \frac{ALG(\sigma)}{OPT(\sigma)} \right\}$, where σ is a list of jobs and $ALG(\sigma)$ and $OPT(\sigma)$ are the makespans of the schedules created by the on-line algorithm and an optimal off-line algorithm for that list of jobs. We use the notations OPT and ALG (omitting σ) when the sequence σ is clear from the context.
Resource augmentation. We consider cases where the resources may be *augmented*, i.e., an optimal off-line algorithm has two uniformly related machines with possibly different speeds, which may be faster or slower than the speeds of the machines of the on-line algorithm.

This is a generalization of competitive analysis. This generalization is useful since an on-line scenario is very different from an off-line one. Therefore machines in both cases do not necessarily should have the same speed. Comparison between an on-line algorithm with one set of speeds to an off-line algorithm with a different set of speeds can imply that in order for the on-line algorithm to be able to perform well, it needs to use faster machines.

Resource augmentation was introduced by Kalyanasundaram and Pruhs [25]. They studied scheduling problems which are known to have unbounded competitive ratio. They showed that it

[*]This work was submitted as a part of the M.Sc. thesis of the second author. A preliminary version of this paper appeared in the proceedings of The First Workshop on Approximation and Online Algorithms (WAOA'03), pages 109-122.

[†]Department of Mathematics, University of Haifa, 31905 Haifa, Israel. `lea@math.haifa.ac.il`.

[‡]School of Computer Science, Tel-Aviv University, Ramat-Aviv, Tel-Aviv, Israel.

is possible to attain a good competitive ratio if the machines of the on-line algorithm are slightly faster than the machines of the off-line algorithm.

Resource augmentation has been applied to a number of problems. It was used already in the paper where the competitive ratio was introduced [33]. In that paper, the performance of online paging algorithms was studied, where they have a larger cache memory than that of an optimal off-line algorithm. In several machine scheduling and load balancing problems [7, 24, 27, 3], the effect of adding more or faster machines has been studied.

Preliminaries. The load of a job (or a part of job) of size (processing time) p, scheduled on a machine of speed $1/z$, is pz. We define the *load* of a machine as the total *load* of the jobs (or parts of jobs) that are scheduled on that machine plus the sum of times when the machine is idle. The *weight* of a machine is the sum of the sizes of jobs (or part of jobs) that are scheduled on that machine. We slightly abuse the notation by identifying jobs with their sizes (processing times).

In the general case, resource augmentation on two uniformly related machines means that the on-line algorithm has a slow machine with speed a and a fast machine with speed b, and the off-line algorithm has a slow machine with speed A and a fast machine with speed B. However, with no loss of generality we may assume the on-line algorithm has a slow machine with speed $1/s$ ($s \geq 1$) and a fast machine with speed 1, and an optimal off-line algorithm has a slow machine with speed $1/q$ ($q \geq 1$) and a fast machine with speed 1. The speed $1/z$ implies that a machine of this speed becomes slower as z grows.

Note that this assumption can be made, since a schedule with makespan x on a pair of machines with speeds a and b ($a < b$), will have a makespan $x \cdot b$ on a pair of machines with speeds 1 and a/b. This is true for any schedule, thus it is possible to assume, without loss of generality, that the on-line algorithm has speeds 1 and $1/s = a/b$. Similarly, it is possible to assume, without loss of generality, that the off-line algorithm has speeds 1 and $1/q = A/B$. We use the parameters q and s throughout the paper.

Our results. In this paper we find the exact competitive ratio functions for all three versions we consider, those are functions of s and q. For the preemptive cases, we show they are valid for both a deterministic version of the problem and a randomized version. We give deterministic algorithms which achieve these competitive ratios, but the lower bounds hold for randomized algorithms.

We show that the competitive ratio of preemptive semi-online scheduling is 1 for $\{q \leq 3, s \leq \frac{2q}{3}\}$ or for $\{q \geq 3, s \leq \frac{q+1}{2}\}$, $\frac{3s(q+1)}{2q+3sq}$ for $\{q \leq 3, \frac{2q}{3} \leq s \leq 2\}$ and $\{q < 3, s > 2\}$, and $\frac{2s(q+1)}{q+1+2sq}$ for $\{q \geq 3, s > \frac{q+1}{2}\}$. For the last two cases we use idle time and prove that it is necessary. Moreover, we show that the competitive ratio of non-preemptive on-line scheduling is

$$\min\left\{\frac{(1+2q)s}{(s+1)q}, \frac{q+1}{q}\right\}$$

and the competitive ratio of preemptive on-line scheduling is $\frac{(q+1)^2 s}{q(1+s+sq)}$.

Previous work. *On-line scheduling.* Scheduling on identical and uniformly related was studied since 1966 when Graham introduced his greedy algorithm "List Scheduling" for identical machines [21]. The competitive ratio of that algorithm is $2 - 1/m$. Since then there was a sequence of improvements on the performance of algorithms [4, 26, 1] and a sequence of lower bounds [5, 1, 20], and the best current results are an algorithm of competitive ratio 1.9201, designed by Fleischer and Wahl [17] and a lower bound of 1.88 given by Rudin [29]. For uniformly related machines there exist constant competitive algorithms, see [2, 6]. The lowest upper bound known for the competitive ratio is 5.828 whereas the lower bound is 2.438. For two and three identical machines, the paper

[16] showed that the greedy algorithm has the best possible competitive ratio. The tight result for two uniformly related machines (without resource augmentation) is folklore and is given in [14]. The competitive ratio is $\min\{\frac{2q+1}{q+1}, 1+\frac{1}{q}\}$ for this problem, achieved by a greedy algorithm which assigns a job to the machine which would finish it first.

Semi-online scheduling. Graham [22] also analyzed the greedy algorithm on identical machines for the case where sizes (processing times) of jobs are non-increasing getting the result $4/3-1/(3m)$. The results for two and three machines are 7/6 and 11/9. Seiden, Sgall and Woeginger [31] showed lower bounds of 7/6 and 1.18046 for two and three machines respectively. Note that the bound for two machines is tight, which means that the greedy algorithm has the best possible competitive ratio for the problem.

Most of the study for semi-online scheduling on non-identical machines involves a study of the greedy algorithm. For two machines, Mireault, Orlin and Vohra [28] gave a complete analysis of the greedy algorithm as a function of the speed ratio. They show that the interval $q \in [1, \infty)$ is partitioned into nine intervals, and introduced a function which gives the competitive ratio in each interval. The reference [19] shows that for any speed ratio, the performance ratio of the greedy algorithm is at most $\frac{1}{4}(1+\sqrt{17}) \approx 1.28$. This was generalized in [12]. In that paper, a complete analysis of the best competitive ratio as a function of the speed ratio was given. Here there are already fifteen intervals. In most intervals the greedy algorithm turns out to be the best algorithm, but there are several intervals where other algorithms are designed, and were shown to perform better and to have the best possible competitive ratios. For a general setting of m uniformly related machines, Friesen [18] showed that the overall approximation ratio of the greedy algorithm is between 1.52 and $\frac{5}{3}$. Dobson [9] claimed to improve the upper bound to $\frac{19}{12} \approx 1.58$. Unfortunately, his proof does not seem to be complete.

On-line preemptive scheduling. There are several tight results for preemptive *on-line* scheduling, in all cases the lower bounds hold also for randomized algorithms. Chen, Van Vliet and Woeginger gave a preemptive optimal algorithm and a matching lower bound for *identical machines* [8]. The competitive ratio of the algorithm is $m^m/(m^m-(m-1)^m)$. Seiden [30] also gave such an optimal algorithm. His algorithm, has a lower makespan than the one in [8] in many cases, however (naturally) it has the same competitive ratio. A lower bound of the same value was given independently by Sgall [32]. For *two uniformly related machines*, the tight competitive ratio is known to be $(q+1)^2/(q^2+q+1)$, given in [14] and independently in [34]. Those results are extended for a class of m uniformly related machines with *non-decreasing speed ratios* in [11]. The tight bound in that case is a function of all the speeds. A lower bound of 2 on the competitive ratio was given already in [15].

Semi-online preemptive scheduling. Seiden, Sgall and Woeginger [31] studied semi-online preemptive scheduling on *identical machines.* They showed that the most difficult case among non-increasing sequences is a sequence of unit size jobs. They gave a complete solution, where the competitive ratio is a function of the number m of machines.

The problem of preemptive semi-online scheduling on two uniformly related machines without resource augmentation was studied by Epstein and Favrholdt [13], who proved a competitive ratio of $\max\left\{\frac{(3(q+1)}{3q+2}, \frac{2q(q+1)}{2q^2+q+1}\right\}$ where $1/q$ is the speed of the slow machine. They also proved that idle time is necessary for $q > 2$.

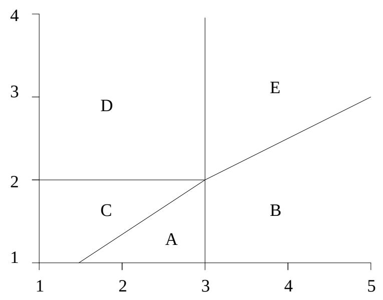

Figure 1: The partition to areas according to the five cases. The horizontal axis (respectively vertical axis) is q (respectively s).

2 Preemptive Semi-Online Scheduling

In this section we consider semi-online preemptive scheduling on two uniformly related machines. Recall that jobs arrive sorted in a non-increasing order of sizes. In the current section $c(q, s)$ denotes the optimal competitive ratio for this problem. We prove the following theorem.

Theorem 1 *The optimal competitive ratio of deterministic or randomized semi-online preemptive algorithm is,*

$$c(q,s) = \begin{cases} 1 & \textit{for } q \leq 3 \textit{ and } s \leq \frac{2q}{3} \ , \\ 1 & \textit{for } q \geq 3 \textit{ and } s \leq \frac{q+1}{2} \ , \\ \frac{3s(q+1)}{2q+3sq} & \textit{for } q \leq 3 \textit{ and } \frac{2q}{3} \leq s \leq 2 \ , \\ \frac{3s(q+1)}{2q+3sq} & \textit{for } q < 3 \textit{ and } s > 2 \ , \\ \frac{2s(q+1)}{q+1+2sq} & \textit{for } q \geq 3 \textit{ and } s > \frac{q+1}{2} \ . \end{cases}$$

In the last two cases, only an algorithm which uses idle time can achieve this competitive ratio. See Figure 1 for the five cases, where the regions are marked in alphabetical order according to the order of the cases.

We first prove the lower bound and then design the algorithms. The only algorithms which use idle time are for the cases where it is necessary. Note that due to resource augmentation, there are two areas where the slow machine of the on-line algorithm is relatively fast and therefore an "optimal" algorithm can be achieved. In those cases, adding speed to the on-line algorithm allows us to overcome the on-line restriction.

2.1 Lower Bound

We start with two theorems that appeared in the literature and are useful for proving our results.

The first theorem deals with an optimal preemptive schedule. Unlike the non-preemptive case, there exists a simple formula for calculating the optimal cost. The theorem is a special case of a theorem given in [23].

Theorem 2 *Given a set of jobs of total size P, the largest of which has size ℓ, the makespan of an optimal off-line algorithm (whose machine speeds are 1 and $1/q$) is*

$$\max\{\ell, \frac{Pq}{q+1}\} \ .$$

The second theorem gives a “recipe” of computing lower bounds for randomized on-line and semi-online algorithms. Similar theorems were given in [15, 11, 13] and are adaptations of a theorem from [32].

Theorem 3 *Consider a sequence of at least two jobs, where the total size of jobs is P, and J_{n-1}, J_n are the last two jobs (in this order). Let $OPT(J_i)$ be the preemptive optimal off-line cost after the arrival of J_i. The competitive ratio of any preemptive randomized on-line algorithm is at least*

$$\frac{Ps}{OPT(J_{n-1}) + sOPT(J_n)} \ .$$

Proof. Fix a sequence of random bits used by the algorithm. Let $ONL(J_i)$ be the makespan of the preemptive on-line algorithm after scheduling J_i. Let $T_1 = ONL(J_n)$, and let T_2 be the last time when both machines are busy simultaneously. Since a job cannot be processed concurrently on both machines, we get $ONL(J_{n-1}) \geq T_2$.
Suppose c is the competitive ratio of the on-line algorithm. Since only one machine can be non-idle during the time period between T_2 and T_1, we get

$$P \leq (1 + \frac{1}{s})T_2 + (T_1 - T_2) = T_1 + \frac{T_2}{s} \ .$$

Taking the expectation we get

$$P \leq E[T_1] + \frac{E[T_2]}{s} \leq E[ONL(J_n)] + \frac{E[ONL(J_{n-1})]}{s} \leq cOPT(J_n) + \frac{cOPT(J_{n-1})}{s} \ .$$

Hence

$$c \geq \frac{Ps}{OPT(J_{n-1}) + sOPT(J_n)} \ .$$

■

Note that we did not use the assumption that the jobs arrive at a decreasing order of size, thus this theorem applies for the on-line case as well.

Lemma 4 *The competitive ratio of any on-line preemptive randomized algorithm is at least*

$$\max\left\{ r_1 = \frac{2s(q+1)}{q+1+2sq} \ , r_2 = \frac{3s(q+1)}{2q+3sq}, 1 \right\} \ .$$

Proof. Consider three sequences consisting of one, two and three unit size jobs. For one job, no algorithm can do better than scheduling all of it on the fast machine at time 0, achieving a competitive ratio of 1. For sequences of 2 and 3 jobs, we get the lower bound by applying Theorem 3. ■

Lemma 5 *For the cases $\{q < 3 \ , \ s > 2\}$ and $\{q \geq 3, s > \frac{q+1}{2}\}$ an on-line algorithm which does not use idle time cannot achieve a competitive ratio of r_2 and r_1, respectively.*

Proof. Consider a sequence of two unit jobs. Set $i = 1$ for $q \geq 3$, $i = 2$ for $q < 3$. Recall that OPT denotes the makespan of an optimal off-line algorithm. Under our assumptions, the first job cannot be assigned to the slow machine, since in this case $OPT = 1$. This would cause the competitive ratio to become s, which breaches our competitive ratio, due to the following.

$$r_1 - s = \frac{2s(q+1)}{q+1+2sq} - s = s\frac{q+1-2sq}{q+1+2sq} < 0$$

since $s > \frac{q+1}{2}$ in this case, and

$$r_2 - s = \frac{3s(q+1)}{2q+3sq} - s = s\frac{q+3-3sq}{q(2+3s)} < 0$$

since $3sq > 6$ and $q+3 \le 6$. Thus the first job must be scheduled on the fast machine at time 0. For two jobs, $OPT = \frac{2q}{q+1}$, thus the second job must be scheduled on the fast machine only after time 1, up to a time which is no later than time $\frac{2q}{q+1}r_i$. This means at least a part of size $1 - (\frac{2q}{q+1}r_i - 1)$ of the second job must be scheduled on the slow machine. For *both* cases we consider, this schedule is illegal since $s\left(2 - \frac{2q}{q+1}r_i\right) > 1$ due to the following.

For r_1, $2s - 1 - s\frac{2q}{q+1}r_1 = \frac{2s-q-1}{q+1+2sq} > 0$ for $s > \frac{q+1}{2}$. For r_2, $2s - 1 - s\frac{2q}{q+1}r_2 = \frac{s-2}{2+3s} > 0$ for $s > 2$. This implies that the second job is scheduled concurrently on both machines. ■

2.2 Algorithms Without Idle Time

In this section we present algorithms for the three cases in which creation of idle time in the schedules can be avoided. By the previous section, three such cases may exist, $\{s \le \frac{2q}{3}, q \le 3\}$, $\{\frac{2q}{3} < s < 2, q \le 3\}$ and $\{s \le \frac{q+1}{2}, q > 3\}$.

Lemma 6 *For* $\max\{1, \frac{2q}{3}\} \le s \le 2, q \le 3$*, there exists an on-line algorithm of competitive ratio* $r_2 = \frac{3s(q+1)}{2q+3sq}$ *which does not use idle time.*

Proof. The first job J_1 is scheduled on the fast machine. Without loss of generality, we may assume it has size 1.
As long as the total size of jobs does not exceed $1 + \frac{1}{q}$, $OPT = 1$. Thus, we can use the interval 1 to r_2 on the fast machine without violating the competitive ratio. For the next jobs we use that interval first, and when it is full we go on to the time interval $[0, s\left(1 + \frac{1}{q} - r_2\right)]$ on the slow machine. When these two intervals are filled, the total size of the jobs scheduled is $1 + \frac{1}{q}$. At this point, some job may be only partially assigned. Denote this job J_p. Since

$$(1 + \frac{1}{q} - r_2)s = \left(1 + \frac{1}{q} - 3s\frac{q+1}{2q+3sq}\right)s = 2s\frac{q+1}{q(2+3s)} = \frac{2s}{2+3s}\frac{q+1}{q} \le \frac{4}{8}\cdot\frac{2}{1} \le 1 ,$$

the load on the slow machine does not exceed 1, hence even if until this point some other job was split between the two machines, its two parts do not overlap in time.
In this situation, the ratio of the loads of the two machines is 3:2 (and the ratio of weights of the machines is $3s : 2$). From now on, we keep this ratio, so that the fast machine is always more loaded. Any arriving job of size x is split into two pieces of size $\frac{3sx}{3s+2}$ (fast machine) and $\frac{2x}{3s+2}$ (slow machine). For the case that the total sum of sizes P is large enough ($P \ge 1 + \frac{1}{q}$), $OPT = \frac{qP}{q+1}$. The on-line load on the fast machine (which is larger) is $\frac{3sP}{3s+2}$, and the competitive ratio is r_2. This completes the description of the algorithm. To complete the proof we need to show the following.
a. The second part of J_p is scheduled properly.
b. Any future job J is scheduled properly.
Proof of a - case 1, J_p was the second job to arrive.
Since it is scheduled on the fast machine only after time 1, we need only to show that on the slow machine, it will be completed no later than time 1. Let x be the size of the second part of J_p. We need to schedule $\frac{2x}{3s+2}$ of the job (which consumes $\frac{2xs}{3s+2}$ units of time on the slow machine). The

size of the second job is at most 1, and $\frac{1}{q}$ of it was scheduled, so $x \leq 1 - \frac{1}{q}$. Thus, after scheduling J_p, we get that the load on the slow machine is

$$\frac{2s}{3s+2} \cdot \frac{q+1}{q} + \frac{2xs}{3s+2} \leq \frac{2s}{3s+2}\left(1 + \frac{1}{q} + x\right) \leq \frac{4s}{3s+2} \leq 1 \ .$$

Proof of a - case 2, J_p was the third job to arrive or a later job.
If J_p does not have its first part scheduled on the fast machine, then this case is similar to a case here the first part and the second part are two different jobs, whose correctness is proven in the next section. Otherwise, $J_p < r_2 - 1$, since it must be smaller than the second job, and the second job must be scheduled entirely on the first machine by the definition of the algorithm. In this case, scheduling it on the slow machine will cause a load of no more than $(r_2-1)s = \frac{3s^2-2sq}{2q+3sq} \leq \frac{3s^2-2s}{2+3s} \leq 1$ (for $s \leq 2$). Thus, the part scheduled on the slow machine does not intersect J_2, and thus it does not intersect J_3.
Proof of b.
We need to show that the interval on the slow machine is enough to schedule $\frac{2x}{3s+2}$. Let P be the total size of previous jobs. Then $\frac{3s}{3s+2}P$ is the load of the fast machine and $\frac{2s}{3s+2}P$ is the load of the slow machine just before scheduling J. The extra load that J causes is $\frac{2xs}{3s+2}$. Hence we need $\frac{Ps}{3s+2} \geq \frac{2sx}{3s+2}$. This clearly holds since $x \leq \frac{P}{2}$ because J is at least the third job. ■

Lemma 7 *For $s \leq \frac{q+1}{2}, q \geq 3$ and for $s \leq \frac{2q}{3}, q \leq 3$, there exist on-line algorithms of competitive ratio 1 which do not use idle time.*

Proof. The first job J_1 is scheduled on the fast machine. Without loss of generality, we may assume it has size 1. As long as the total size of jobs does not exceed $1 + \frac{1}{q}$, $OPT = 1$. Hence we can proceed by scheduling the next $\frac{1}{q}$ total size of jobs on the slow machine without violating the competitive ratio (since $\frac{s}{q} < 1$). When these jobs have been scheduled, the total size of the jobs scheduled is $1 + \frac{1}{q}$. At this point, some job may be only partially assigned. Denote this job J_p. We will show that even if this job was split between the two machines, its two parts do not overlap in time.

In this situation, the ratio of the loads is q:s. From now on, we keep this ratio, so that the fast machine is always more loaded. Any arriving job of size x is split into two pieces of size $\frac{qx}{q+1}$ (fast machine) and $\frac{x}{q+1}$ (slow machine). For the case that the total sum of sizes P is large enough ($P \geq 1 + \frac{1}{q}$), $OPT = \frac{qP}{q+1}$. The load of the on-line algorithm on the fast machine (which is larger) is $\frac{qP}{q+1}$, and the competitive ratio is 1. This completes the description of the algorithm. To complete the proof we need to show the following.
a. The second part of J_p is scheduled properly.
b. Any future job J is scheduled properly.
Proof of a - case 1, J_p was the second job to arrive.

A part of the second job is scheduled on the fast machine only after time 1, and another part of it is scheduled on the slow machine. Thus we need only to show that the part assigned to the slow machine will be completed no later than time 1.

Let x be the size of the second part of J_p. We need to schedule $\frac{x}{1+q}$ of the job (which consumes $\frac{xs}{1+q}$ units of time on the slow machine). The size of the second job is at most 1, and $\frac{1}{q}$ of it was scheduled, so $x \leq 1 - \frac{1}{q}$. Thus, after scheduling J_p, we get in the first case, that the load on the slow machine is,

$$\frac{s}{q} + \frac{xs}{1+q} \leq \frac{s(1+q)+(q-1)s}{q(1+q)} = \frac{2s}{1+q} \leq 1 \ .$$

In the second case, by using $s \le \frac{2q}{3}$, and then by using $q \le 3$, that the load on the slow machine is,

$$\frac{s}{q} + \frac{xs}{1+q} \le \frac{2}{3} + \frac{2xq}{3(1+q)} \le \frac{4q}{3(1+q)} \le 1 \ .$$

Proof of a - case 2, J_p was the third job to arrive or a later job.
In this case, the size of J_2 is smaller than or equal to $\frac{1}{q}$, or else J_p would have been a part of J_2, thus $J_p \le \frac{1}{q}$. Let x be the size of the second part of J_p. We need to schedule $\frac{x}{1+q}$ of the job (which consumes $\frac{xs}{1+q}$ units of time on the slow machine). In the first case, we get the load $\frac{s}{q} + \frac{xs}{1+q} \le \frac{2s+sq}{q(1+q)} \le \frac{q+2}{2q} < 1$ (since $q > 2, x \le \frac{1}{q}, s \le \frac{q+1}{2}$). In the second case we get the load $\frac{s}{q} + \frac{xs}{1+q} \le \frac{2}{3} + \frac{2xq}{3(1+q)} \le \frac{2}{3} + \frac{2}{3(1+q)} \le 1$, since $q \ge 1$.
Proof of b.
Let x be the size of the arriving job J, and let P be the total size of previous jobs. Before scheduling J, the load of the slow machine is $\frac{sP}{1+q}$, and the load of the fast machine is $\frac{qP}{1+q}$. We need to show that the interval on the slow machine is enough to contain $\frac{sx}{1+q}$. In the first case, since J is at least the third job, $P \ge 1 + x$. Thus (using $x \le 1$) we get

$$\begin{aligned}\frac{sx}{1+q} + \frac{sP}{1+q} - \frac{qP}{1+q} &\le \frac{\frac{q+1}{2}x}{q+1} + \frac{\frac{q+1}{2}P}{q+1} - \frac{qP}{q+1} = \frac{\frac{q+1}{2}x - \frac{q-1}{2}P}{q+1} \\ \le \frac{\frac{q+1}{2}x - \frac{q-1}{2}(1+x)}{q+1} &\le \frac{2x+1-q}{2(q+1)} \le \frac{3-q}{2(q+1)} \le 0 \ .\end{aligned}$$

In the second case, we get $\frac{sx}{1+q} + \frac{sP}{1+q} - \frac{qP}{1+q} \le \frac{2qx-qP}{3(q+1)} \le 0$ since $x \le \frac{P}{2}$ (as J is at least the third job). ■

2.3 Algorithms With Idle Time

In this section we present algorithms for cases which were previously proven to require idle time to reach our desired competitive ratio. Two such cases exist, $\{q < 3, s > 2\}$, for which we will prove a competitive ratio of $r_2 = \frac{3s(q+1)}{2q+3sq}$, and $\{q \ge 3, s > \frac{q+1}{2}\}$ for which we will prove a competitive ratio of $r_1 = \frac{2s(q+1)}{q+1+2sq}$.

Lemma 8 *For any pair of (q, s) such that $q < 3$ and $s > 2$, there exists an on-line preemptive algorithm with competitive ratio of $\frac{3s(q+1)}{2q+3sq}$. For any pair of (q, s) such that $q \ge 3$ and $s > \frac{q+1}{2}$, there exists an on-line preemptive algorithm with competitive ratio of $\frac{2s(q+1)}{q+1+2sq}$.*

Proof. We define the algorithm. Without loss of generality, we assume the first job has size 1. Let $r = \frac{3s(q+1)}{2q+3sq}$ if $q < 3$, and $\frac{2s(q+1)}{q+1+2sq}$ otherwise. The first job is cut into two pieces, one of size $\frac{r-1}{s-1}$ which is scheduled on the slow machine until time r, and the other of size $\frac{s-r}{s-1}$ which is scheduled on the fast machine starting from time 0. Note that $s > 1$ and $\frac{r-1}{s-1} + \frac{s-r}{s-1} = 1$, thus we need to show that the sizes of both parts are non-negative. We need to show $1 \le r \le s$. For the first case, we have $\frac{3s(q+1)}{2q+3sq} \ge 1$ if and only if $3s \ge 2q$, which clearly holds since $3s > 6 > 2q$. Moreover, $\frac{3s(q+1)}{2q+3sq} \le s$ if and only if $q + 3 \le 3sq$, which holds since $3sq \ge 6q \ge q + 5$, for $q \ge 1$. In the second case, $\frac{2s(q+1)}{q+1+2sq} \ge 1$ if and only if $2s \ge q + 1$, which holds according to the definition of this interval, moreover, $\frac{2s(q+1)}{q+1+2sq} \le s$ if and only if $q + 1 \le 2sq$, which holds since $sq \ge q \ge 1$ for $q \ge 1$ and $s \ge 1$.

From now on, future jobs are scheduled on the first (possibly idle) time on the machine they are scheduled on. If a job is scheduled on the slow machine and there is no idle time left, it will be scheduled after time r.
As in previous algorithms, as long as the total size of the jobs does not exceed $1+\frac{1}{q}$, these jobs are scheduled on the fast machine until time r, and then on the slow machine. When the total size of the jobs is $1+\frac{1}{q}$, there is still idle time on the slow machine (by the definition of the algorithm), the idle time ends when the total size of the jobs is $r(1+\frac{1}{s})$, which is greater than $1+\frac{1}{q}$, since $r(1+\frac{1}{s})-1-\frac{1}{q} \geq \min\{\frac{q^2-1}{(q+1+2sq)q}, \frac{q+1}{q(2+3s)}\} \geq 0$ for $q \geq 1$. The loads on the machines, not including idle time, is r on the fast machine and $s\left(1+\frac{1}{q}-r\right)$ on the slow machine. The ratio of the loads is $\frac{3}{2}$ if $q \leq 3$, and $\frac{2}{1+\frac{1}{q}}$ if $q > 3$. Since even after the first job $OPT \geq 1$, the competitive ratio is maintained up to this point.
We continue by keeping the ratio between the loads. For $q \leq 3$ this means cutting any job of size x into two pieces, one of size $\frac{3sx}{3s+2}$, which is assigned to the first available time on the fast machine, and the other of size $\frac{2x}{3s+2}$, which is assigned to the first available time on the slow machine. For $q > 3$ this means cutting any job of size x into two pieces, one of size $\frac{2xsq}{q+1+2sq}$ which is assigned to the first available time on the fast machine, and the other of size $\frac{x(q+1)}{q+1+2sq}$ which is assigned to the first available time on the slow machine. Note that in both cases it is possible for the assignment of some job to take up the idle time on the slow machine, in which case the piece assigned to the slow machine is cut into two pieces, and is continued after time r.
Again we need to prove the leftover on the job for which the total size reached $1+\frac{1}{q}$ is scheduled properly, and that any future job J are assigned properly. Since the free time before time r on both machines is disjoint after scheduling the first job, there is no difference between a job and the leftover of a job.
Case A, $q \leq 3$.
Let P be the total size of previous jobs (such that $P \geq 1+\frac{1}{q}$). Let x be the size of job J. Then $\frac{3s}{3s+2}P$ is the load of the fast machine and $\frac{2s}{3s+2}P$ is the load of the slow machine before scheduling J. Scheduling J will cause an extra load of $\frac{2xs}{3s+2}$ on the slow machine. Hence, we need to show that $\frac{Ps}{3s+2} \geq \frac{2xs}{3s+2}$. This clearly holds if is at least the third job, which implies $x \leq \frac{P}{2}$. Otherwise, J is a leftover and then $x \leq 1-\frac{1}{q}$. For $q \leq 3$, this implies that $x \leq 1-\frac{1}{q} \leq \frac{1}{2}(1+\frac{1}{q}) \leq \frac{P}{2}$.
Case B, $q > 3$.
Let P be the total size of previous jobs (such that $P \geq 1+\frac{1}{q}$). Let x be the size of job J. Then $\frac{2qs}{2qs+q+1}P$ is the load of the fast machine and $\frac{(q+1)s}{2qs+q+1}P$ is the load of the slow machine before scheduling J. Scheduling J will cause an extra load of $\frac{s(q+1)x}{q+1+2qs}$ on the slow machine. Hence, we need to show that $\frac{(q+1)xs}{q+1+2qs} \leq \frac{(q-1)sP}{q+1+2qs}$, which is implied by $x \leq \frac{q-1}{q+1}P$. If J is the third job or a later one then $x \leq \frac{P}{2} \leq \frac{q-1}{q+1}P$. Otherwise, J is a leftover and $x \leq 1-\frac{1}{q}$. For $q > 3$, this implies

$$x \leq 1-\frac{1}{q} = \frac{(q-1)(q+1)}{q^2+q} \leq \frac{q-1}{q+1} \cdot \frac{q+1}{q} \leq \frac{q-1}{q+1}P \ .$$

■

3 Non-Preemptive On-Line Scheduling

In this section we study the problem of deterministic non-preemptive scheduling with respect to makespan on two machines. There is no restriction on the arriving jobs. Since preemption is not allowed, we can assume that the algorithms cannot use idle time, as idle time does not improve the situation. In the current section $c(q,s)$ denotes the optimal competitive ratio for this problem. We prove the following theorem.

Theorem 9 *The optimal competitive ratio of deterministic on-line non-preemptive algorithm is* $c(q,s) = \frac{(1+2q)s}{(s+1)q}$ *for* $s < \frac{q+1}{q}$ *and* $c(q,s) = \frac{q+1}{q}$ *for* $s \geq \frac{q+1}{q}$.

Note that $1 \leq c(q,s) \leq 2$.

3.1 Upper Bound

We define the algorithm LIST. Upon arrival of a job, for each machine, LIST has to compute the time that the job would be completed if it is assigned to it. Then it schedules the job on a machine where this time is minimized.

Lemma 10 *LIST has a competitive ratio of at most* $\min\left\{\frac{(1+2q)s}{(s+1)q}, \frac{q+1}{q}\right\}$.

Proof. Consider the time just before the arrival of the job which determines the makespan. Let x be the weight of the fast machine of the on-line algorithm, and let y be the weight on the slow machine of the on-line algorithm at this point. Let ℓ be the size of the next job. Denote by ALG the makespan of the schedule created by LIST, and the competitive ratio by $c = \frac{ALG}{OPT}$. We get $OPT \geq \frac{x+y+\ell}{1+\frac{1}{q}} = \frac{q(x+y+\ell)}{1+q}$ and $OPT \geq \ell$. The on-line algorithm will assign the next job to the machine where it will finish first. Thus $ALG \leq x + \ell$ and $ALG \leq (y+\ell)s$. We have $(s+1)ALG \leq s(x+y+2\ell) \leq s(x+y+\ell)+s\ell \leq \frac{s(1+q)}{q}OPT + sOPT$. Thus $ALG \leq OPT \cdot \frac{s(1+q)+qs}{(s+1)q} \rightarrow c \leq \frac{s(1+2q)}{q(s+1)}$. Moreover, LIST is definitely no worse than scheduling every job on the fast machine - hence, $c \leq \frac{q+1}{q}$. Note that $c = \frac{q+1}{q}$ for every $s \geq \frac{q+1}{q}$. This value varies between 1 and 2, and is equal to $\phi \approx 1.618$ for s=q. ■

3.2 Lower Bound

We prove that no deterministic on-line algorithm can perform better than LIST.

Lemma 11 *No deterministic on-line algorithm can have a better competitive ratio than* $\frac{(1+2q)s}{(s+1)q}$ *if* $s < \frac{q+1}{q}$, *and no deterministic on-line algorithm can have a better competitive ratio than* $\frac{q+1}{q}$, *if* $s \geq \frac{q+1}{q}$.

Proof. First, we prove the lemma for $s \geq \frac{q+1}{q}$. In this case, $c = \frac{q+1}{q}$. We use a sequence of two jobs which is 1, q. Denote the on-line algorithm by ALG. If ALG schedules the first job on the slow machine, it gets a competitive ratio of $s \geq \frac{q+1}{q}$. Thus in order to maintain a competitive ratio of less than $\frac{q+1}{q}$ it must schedule the first job on the fast machine. In this case, its makespan is no smaller than $q+1$ after the second job, since scheduling it on the fast machine gives a makespan of $q+1$, and scheduling it on the slow machine gives a makespan of $qs \geq q+1$, while an optimal

offline algorithm easily achieves a makespan of q.
For $s < \frac{q+1}{q}$, Set $\alpha = \frac{q+1-qs}{s-q+qs}$. Note that since $s < \frac{q+1}{q}$, $\alpha > 0$. We start with presenting the algorithm a job of size 1. We have the following cases.
Case 1. The first job is assigned to the slow machine.
Set $T_i = (1 + \frac{1}{\alpha})^i$. Let j be an integer such that $T_j \leq 1 + q, T_{j+1} > 1 + q$. Such a number j must exist since $T_0 = 1$ and $\lim\limits_{n\to\infty} T_n = \infty$. In this case, we proceed with the following sequence, $b_0 = 1$ (The job which was presented at first), $b_1 = \frac{1}{\alpha}, b_i = \frac{1}{\alpha}(1+\frac{1}{\alpha})^{i-1}$ for all $i \geq 2$. Note that $T_i = \sum\limits_{n=0}^{i} b_n$, for all $i \geq 0$. We give jobs from this sequence until some job b_t is scheduled on the fast machine, or until job $b_t = b_{j+1}$ is scheduled (on one of the machines), i.e. in total $j + 2$ jobs are scheduled. The following cases may occur.
Case 1a, $t < j + 1$ and there is at least one job on each machine.
After the first time the algorithm scheduled a job on the fast machine, we present a job of size qT_t. The off-line algorithm achieves a makespan of qT_t (scheduling all but the last job on the slow machine, and the last job on the fast machine). The on-line algorithm can either schedule that job on the slow machine or to the fast machine. If it is scheduled on the slow machine, we get a competitive ratio of

$$\begin{aligned}\frac{sT_{t-1} + sqT_t}{qT_t} &= \frac{s(1 + \frac{1}{\alpha})^{t-1} + sq(1 + \frac{1}{\alpha})^t}{q(1 + \frac{1}{\alpha})^t} = \frac{s}{(1 + \frac{1}{\alpha})q} + s \\ &= \frac{s(q + 1 - qs + qs + q)}{q(s + 1)} = \frac{(1 + 2q)s}{(s + 1)q} .\end{aligned}$$

Otherwise, it is scheduled on the fast machine, and the competitive ratio is

$$\begin{aligned}\frac{b_t + qT_t}{qT_t} &= \frac{\frac{1}{\alpha}(1 + \frac{1}{\alpha})^{t-1} + q(1 + \frac{1}{\alpha})^t}{q(1 + \frac{1}{\alpha})^t} = 1 + \frac{1}{q(1 + \alpha)} = 1 + \frac{1}{q(1 + \frac{1+q-qs}{qs+s-q})} \\ &= \frac{qs + s - q}{q(s + 1)} + 1 = \frac{2qs + s}{q(s + 1)} = \frac{s(1 + 2q)}{(s + 1)q} .\end{aligned}$$

Case 1b, $t = j + 1$ and all jobs are on one machine.
We will show that either after $j + 1$ jobs or after $j + 2$ jobs, the competitive ratio is at least c. We denote by OPT_k, ALG_k the makespans after scheduling job b_k of an optimal off-line algorithm and the on-line algorithm, respectively. In these two cases ($k = j, j + 1$), an optimal off-line algorithm can put the first job on the slow machine and all the next jobs on the fast machine, achieving makespans of $OPT_j \leq q$, $OPT_{j+1} \leq T_j - 1$. The on-line algorithm places all jobs on the slow machine, achieving makespans $ALG_j = sT_j, ALG_{j+1} = sT_{j+1}$. If $\frac{sT_j}{q} \geq \frac{s(1+2q)}{q(s+1)}$ the proof is complete since $\frac{ALG_j}{OPT_j} \geq c$. Otherwise, $T_j < \frac{1+2q}{s+1}$ and $T_{j+1} = T_j(1 + \frac{1}{\alpha}) < \frac{1+2q}{s+1}(1 + \frac{1}{\alpha}) = \frac{1+2q}{q+1-qs}$. Thus, the competitive ratio is at least $\frac{sT_{j+1}}{T_{j+1}-1} = s\left(1 + \frac{1}{T_{j+1}-1}\right) \geq s\left(1 + \frac{1}{\frac{1+2q}{q+1-qs}-1}\right) = s\frac{2q+1}{(s+1)q}$.
Case 2, The first job is scheduled on the fast machine.
Set $S_i = (1 + \alpha)^i$. Let j be an integer such that $S_j \leq 1 + q, S_{j+1} > 1 + q$, which exists since $\alpha > 0$. In this case, we proceed with the following sequence, $a_0 = 1$ (The job which was presented at first), $a_1 = \alpha, a_i = \alpha(1+\alpha)^{i-1}$ for all $i \geq 2$. Note that $S_i = \sum\limits_{n=0}^{i} a_n$. We give jobs from this sequence until some job is scheduled on the slow machine, or until $j + 2$ jobs are scheduled. Let k be the index of the last job, that is $k = j + 1$ if $j + 2$ jobs were scheduled, and otherwise k is the index of

the job after which the sequence is stopped. The following cases may occur,
Case 2a, $k < j+1$ and there is at least one job on each machine.
After the first time the algorithm scheduled a job on the fast machine, we give the on-line algorithm a job of size qS_k. The off-line algorithm achieves a makespan of qS_k (scheduling all but the last job on the slow machine, and the last job to the fast machine). An on-line algorithm can either schedule the last job on the fast machine or on the slow machine.
If it is scheduled on the fast machine, the competitive ratio is at least
$\frac{S_{k-1}+qS_k}{qS_k} = \frac{(1+\alpha)^{k-1}+q(1+\alpha)^k}{q(1+\alpha)^k} = 1 + \frac{1}{q(1+\alpha)} = 1 + \frac{1}{q(1+\frac{1+q-qs}{qs+s-q})} = \frac{qs+s-q}{q(s+1)} + 1 = \frac{2qs+s}{q(s+1)} = \frac{s(1+2q)}{(s+1)q}$
If J is scheduled on the slow machine, the competitive ratio is at least
$\frac{sa_k+sqS_k}{qS_k} = \frac{s\alpha(1+\alpha)^{k-1}+sq(1+\alpha)^k}{q(1+\alpha)^k} = \frac{s\alpha}{(1+\alpha)q} + s = \frac{s+qs-qs^2}{(s+1)q} + s = s\frac{2q+1}{(s+1)q}$
Case 2b, $k = j+1$ and all jobs are on one machine.
We show that after either $j+1$ jobs or $j+2$ jobs, the competitive ratio is at least c. We denote by OPT_t, ALG_t the makespans after scheduling job a_t $(t = k, k+1)$ of an optimal algorithm and the on-line algorithm, respectively. In this case, an optimal off-line algorithm can put the first job on its slow machine and all the next jobs on its fast machine, achieving makespans of $OPT_j \le q$ and $OPT_{j+1} \le S_{j+1} - 1$. The on-line algorithm has scheduled the first $j+2$ jobs on the fast machine, achieving makespans $ALG_j = S_j$, $ALG_{j+1} = S_{j+1}$. If $\frac{Sj}{q} \ge \frac{s(1+2q)}{q(s+1)}$ the proof is complete since $\frac{ALG_j}{OPT_j} \ge c$. Otherwise, $S_j < \frac{s(1+2q)}{s+1}$ and $S_{j+1} = S_j(1+\alpha) < \frac{s(1+2q)}{s+1}(1+\alpha) = (1+2q)\,\frac{s}{s-q+qs}$. Thus, the competitive ratio is at least

$$\frac{S_{j+1}}{S_{j+1}-1} = 1 + \frac{1}{S_{j+1}-1} \ge 1 + \frac{1}{\frac{(1+2q)s}{s-q+qs} - 1} = s\frac{2q+1}{(s+1)\,q} \ .$$

■

4 Preemptive On-Line Scheduling

We study the problem of deterministic and randomized scheduling with respect to makespan on two machines. Idle time is allowed but our algorithms do not use it.

In this section we let $r(s,q)$ denote the competitive ratio function. We prove the following theorem.

Theorem 12 *The optimal competitive ratio of any deterministic or randomized on-line preemptive algorithm is* $r(s,q) = \frac{(1+q)^2 s}{(1+s+sq)q}$.

Note that for all s and q, $1 \le r(s,q) \le 2$ holds.

4.1 Lower Bound

Lemma 13 *No deterministic or randomized on-line algorithm can achieve a competitive ratio better than* $\frac{(1+q)^2 s}{(1+s+sq)q}$.

Proof. We get this from theorem 1 stated in section 2, by using a sequence of very small jobs (also called "sand") with total size 1, followed by a job with size q. An optimal off-line algorithm achieves a makespan of $\frac{1}{1+\frac{1}{q}}$ on the sand. This is done by assigning an amount of $\frac{1}{1+\frac{1}{q}}$ of the sand to the fast machine and the rest which is an amount of $\frac{1}{q\left(1+\frac{1}{q}\right)}$ of it to the slow machine. A makespan of q on the entire sequence is achieved by assigning all the sand to the slow machine and the other job to the fast machine. We get a competitive ratio of at least

$$\frac{Ps}{OPT(J_{n-1}) + sOPT(J_n)} = \frac{(q+1)s}{\frac{1}{1+\frac{1}{q}} + qs} = \frac{(q+1)^2 s}{q + sq(q+1)} = \frac{(q+1)^2 s}{q(1+s+qs)} = r(s,q) \ .$$

■

4.2 Upper Bound

We use the following notation. A new (arriving) job has size ℓ. The makespan of an optimal off-line algorithm before scheduling ℓ is OPT. The load of the fast machine of the on-line algorithm before scheduling ℓ is L_1, and the load of the slow machine of the on-line algorithm before scheduling ℓ is L_2. The total size of all the jobs before ℓ is $w = L_1 + \frac{L_2}{s}$. Analogously, L_1', L_2', w' and OPT' are the load on the fast machine of the on-line algorithm, the load on the slow machine of the on-line algorithm and the total size of the presented jobs and the makespan of an optimal off-line algorithm after scheduling ℓ. In this section, we use the variable r to denote $r(s,q)$ for a fixed pair of s and q that we are dealing with.

Lemma 14 *For a given sequence of jobs, $a_1, a_2, \ldots, a_n$, Let $w = \sum_{i=1}^{n} a_n$. If there exists some i for which $a_i > q(w - a_i)$, then i is unique and the makespan of the optimal off-line algorithm is a_i. Otherwise, the makespan of the optimal off-line algorithm is $w/(1+1/q)$.*

Proof. If a_i exists then it must be unique since $a_i > qw/(q+1) \geq w/2$. The bound on the makespan of an optimal off-line algorithm follows from Theorem 2. ■

We define algorithm F. Whenever a new job ℓ arrives, schedule a part as large as possible of it within the time interval $[L_1, rOPT']$ on the fast machine starting from L_1, and process the rest (if necessary) on the slow machine starting from L_2.

Lemma 15 *At any state of algorithm F, $L_2 \leq \frac{L_1}{q+1}$.*

Proof. We prove the claim by induction on the number of jobs assigned. First, when no jobs are assigned, the claim holds trivially. Suppose it holds prior to the arrival of a job ℓ. If ℓ is not scheduled on the slow machine it continues to hold, so we may assume some part of ℓ is scheduled on the slow machine. In this case, it follows from the definition of our algorithm that $L_1' = rOPT' \geq r\frac{w+\ell}{1+\frac{1}{q}} = \frac{rqw'}{q+1} = r\frac{q}{q+1}(\frac{L_2'}{s} + L_1') = \frac{rq}{s(q+1)}L_2' + \frac{rq}{q+1}L_1'$. From this we get $L_2' \leq \frac{s(L_1' - \frac{qr}{q+1}L_1')}{\frac{qr}{q+1}} = L_1'(\frac{s(q+1)}{qr} - s)$. Using the definition of r, $r = \frac{(1+q)^2 s}{(1+s+sq)q}$, we get $\frac{s(q+1)}{qr} = \frac{s(q+1)}{q} \cdot \frac{(1+s+sq)q}{(1+q)^2 s} = \frac{1+s+sq}{1+q}$ and moreover $\frac{s(q+1)}{qr} - s = \frac{1+s+sq}{q+1} - \frac{sq+s}{q+1} = \frac{1}{q+1}$. We conclude that $L_2' \leq \frac{L_1'}{q+1}$ ■

Thus, we never use more than the interval on the slow machine $[L_2, \frac{r}{q+1}OPT' = \frac{L_1'}{q+1}]$ to schedule ℓ. It remains to show that this is a legal assignment - i.e., ℓ is not scheduled concurrently on both machines. It is enough to show that at every state of the algorithm $L_2' \leq L_1$.

Lemma 16 *For a given w, ℓ the smallest available space for ℓ is achieved when $L_2 = \frac{L_1}{q+1}$.*

Proof. Given the above restriction, the space available for ℓ is $rOPT' - L_1$ on the fast machine and $L_1 - L_2$ on the slow machine. This implies that the total space available is $rOPT' - L_1 + \frac{L_1 - L_2}{s}$, where $L_2 = sw - sL_1$. Take the derivative of this function w.r.t L_1 we get $\frac{\partial}{\partial L_1}\left(rOPT' - L_1 + \frac{L_1 - s(w-L_1)}{s}\right) = \frac{1}{s}$, which means that the space for ℓ increases as L_1 increases. Thus the minimal space is achieved where L_1 is minimal, that is, where L_2 is maximal. Then from the previous lemma we achieve that the minimum space is achieved at $L_2 = \frac{L_1}{q+1}$. ■

Lemma 17 *At every state of algorithm F, the assignment is legal.*

Proof. By the previous lemma, we only need to show this for $L_2 = \frac{L_1}{q+1}$. In this case, the total size of the jobs on the slow machine is $\frac{L_1}{s(q+1)}$. Thus, we get

$$w = L_1 + \frac{L_1}{s(q+1)} \to w = \frac{s(q+1)+1}{s(q+1)} L_1 \to L_1 = \frac{s(q+1)}{sq+s+1} w \to L_2 = \frac{s}{sq+s+1} w \ .$$

Showing the assignment is legal is achieved by showing the interval $[L_1, rOPT']$ on the fast machine and $[L_2, L_1]$ on the slow machine are enough to contain ℓ. Thus, we need to show

$$\ell \le rOPT' - \frac{s(q+1)}{sq+s+1} w + \frac{s(q+1)-s}{s(sq+s+1)} w \ ,$$

or

$$0 \le rOPT' - \frac{s(q+1)}{sq+s+1} w + \frac{s(q+1)-s}{s(sq+s+1)} w - \ell \ .$$

We have the following cases,
Case 1, $\ell \le qw$.
In this case, we use the lower bound of $\frac{w+\ell}{1+\frac{1}{q}}$ for OPT'. We get

$$\begin{aligned} rOPT' - \frac{s(q+1)}{sq+s+1} w + \frac{s(q+1)-s}{s(sq+s+1)} w - \ell &\ge \frac{qr(w+\ell)}{q+1} - \frac{s(q+1)-q}{sq+s+1} w - \ell \\ = \frac{(1+q)s\ell + qw - \ell(sq+s+1)}{sq+s+1} &= \frac{qw-\ell}{sq+s+1} \ge 0 \end{aligned}$$

since $\ell \le qw$.
Case 2, $\ell > qw$.
In this case, we use the lower bound of ℓ for OPT'. We get

$$\begin{aligned} rOPT' - \frac{s(q+1)}{sq+s+1} w + \frac{s(q+1)-s}{s(sq+s+1)} w - \ell &\ge rl - \frac{s(q+1)-q}{sq+s+1} w - \ell \\ = \frac{\ell(s+\frac{s}{q}-1) - w(sq+s-q)}{sq+s+1} &\ge \frac{w(qs+s-q-sq-s+q)}{sq+s+1} = 0 \ . \end{aligned}$$

The last inequality is true since $\ell > qw$ and $s + \frac{s}{q} > 1$. ■

5 Conclusion

We gave tight bounds for three models. We did not study the on-line non-preemptive model. In [12] where this model is studied, the interval $[1, \infty)$ of the speed ratio between the two machines is split into 15 intervals, each of which stands for a totally different case. The proofs are very technical and therefore it seems that an extension to the resource augmented model would be too complicated.

Another interesting generalization of all variants studied in this paper is the case of m uniformly related machines. Recently Ebenlendr, Jawor and Sgall [10] extended the result of [11] and designed preemptive online algorithms of optimal competitive ratios for all speed combinations. A study with resource augmentation seems to be difficult due to the large number of possible cases.

There are certainly many other two machine models to which resource augmentation can be applied.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459-473, 1999.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, 44:486–504, 1997.

[3] N. Bansal, K. Dhamdhere, J. Könemann, and A. Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, pages 260–270, 2003.

[4] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51(3):359–366, 1995.

[5] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.

[6] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1), 108–121, 2000.

[7] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. *Nordic Journal of Computing*, 6(2):181–193, 1999.

[8] B. Chen, A. van Vliet, and G. J. Woeginger. An Optimal Algorithm for Preemptive On-line Scheduling. *Operations Research Letters*, 18:127–131, 1995.

[9] G. Dobson. Scheduling Independent Tasks on Uniform Processors. *SIAM Journal on Computing*, 13(4):705–716, 1984.

[10] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. In *Proc. of the 14th Annual European Symposium on Algorithms (ESA2006)*, pages 327–339, 2006.

[11] L. Epstein. Optimal Preemptive On-Line Scheduling on Uniform Processors with Non-Decreasing Speed Ratios. *Operations Research Letters*, 29(2):93–98, 2001.

[12] L. Epstein and L. M. Favrholdt. Optimal non-preemptive semi-online scheduling on two related machines. *Journal of Algorithms*, 57(1): 49–73, 2005.

[13] L. Epstein and L. M. Favrholdt. Optimal preemptive semi-online scheduling to minimize makespan on two related machines. *Operations Research Letters*, 30(4):269–275, 2002.

[14] L. Epstein, J. Noga, S. S. Seiden, J. Sgall, and G. J. Woeginger. Randomized Online Scheduling on Two Uniform Machines. *Journal of Scheduling*, 4(2):71–92, 2001.

[15] L. Epstein and J. Sgall. A Lower Bound for On-Line Scheduling on Uniformly Related Machines. *Operations Research Letters*, 26(1):17–22, 2000.

[16] U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[17] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3(5):343–353, 2000.

[18] D. K. Friesen. Tighter Bounds for LPT Scheduling on Uniform Processors. *SIAM Journal on Computing*, 16(3):554–560, 1987.

[19] T. Gonzalez, O. H. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[20] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2000)*, pages 564–565, 2000.

[21] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[22] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.

[23] E. Horwath, E. C. Lam, and R. Sethi. A Level Algorithm for Preemptive Scheduling. *J. Assoc. Comput. Mach.*, 24:32–43, 1977.

[24] B. Kalyanasundaram and K. Pruhs. Maximizing job completions online. *Journal of Algorithms*, 49(1):63–85, 2003.

[25] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):214–221, 2000.

[26] D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.

[27] T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 623–632, 1999.

[28] P. Mireault, J. B. Orlin, and R. V. Vohra. A Parametric Worst Case Analysis of the LPT Heuristic for Two Uniform Machines. *Operations Research*, 45:116–125, 1997.

[29] J. F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, May 2001.

[30] S. Seiden. Preemptive Multiprocessor Scheduling with Rejection. *Theoretical Computer Science*, 262(1-2):437–458, 2001.

[31] S. Seiden, J. Sgall, and G. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5):215–221, 2000.

[32] J. Sgall. A Lower Bound for Randomized On-Line Multiprocessor Scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.

[33] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[34] J. Wen and D. Du. Preemptive On-Line Scheduling for Two Uniform Processors. *Operations Research Letters*, 23:113–116, 1998.