Max-min online allocations with a reordering buffer*

Leah Epstein† Asaf Levin‡ Rob van Stee§

Abstract

We consider online scheduling so as to maximize the minimum load, using a reordering buffer which can store some of the jobs before they are assigned irrevocably to machines. For m identical machines, we show an upper bound of $H_{m-1}+1$ for a buffer of size $m-1$. A competitive ratio below H_m is not possible with any fixed buffer size, and it requires a buffer of size $\Omega(m/\log m)$ to get a ratio of $O(\log m)$. For uniformly related machines, we show that a buffer of size $m+1$ is sufficient to get a competitive ratio of m, which is best possible for any fixed sized buffer. We show similar results (but with different constructions) for the restricted assignment model. We give tight bounds for two machines in all the three models.

These results sharply contrast to the (previously known) results which can be achieved without the usage of a reordering buffer, where it is not possible to get a ratio below a competitive ratio of m already for identical machines, and it is impossible to obtain an algorithm of finite competitive ratio in the other two models, even for $m=2$. Our results strengthen the previous conclusion that a reordering buffer is a powerful tool and it allows a significant decrease in the competitive ratio of online algorithms for scheduling problems. Another interesting aspect of our results is that our algorithm for identical machines imitates the behavior of a greedy algorithm on (a specific set of) related machines, whereas our algorithm for related machines completely ignores the speeds until all jobs have arrived, and then only uses the relative order of the speeds.

1 Introduction

Scheduling problems are most frequently described in a framework of assigning jobs to machines. There are various kinds of scheduling problems depending upon the properties of the machines, allowable assignments, and the cost criteria. Our goal is to study a natural model where the assignment of jobs is performed dynamically, in the sense that jobs are being considered one by one, and generally must be assigned in this order, while the information on future jobs is unknown. We consider parallel machines, and a problem variant which possesses features of both offline and online scenarios. These are not offline problems, since the input arrives gradually, but they are not purely online either, since we allow partial reordering of the input.

More specifically, in this paper we study a variant called *scheduling with a reordering buffer*, where a buffer, which can store a fixed number of unassigned jobs, is available. Thus each new job must be either assigned to a machine or stored in the buffer (possibly in the case where the buffer is already full, the

*An extended abstract of this work appears in the proceedings of ICALP 2010.

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

‡Faculty of Industrial Engineering and Management, The Technion, 32000 Haifa, Israel. levinas@ie.technion.ac.il.

§Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany. Research supported by the German Research Foundation (DFG). vanstee@mpi-inf.mpg.de.

algorithm is forced to evict another job from the buffer, which must be assigned to a machine immediately)[1]. Every assignment of a job to a machine is irrevocable.

In this paper, we are concerned with max-min allocations, that is, the goal is maximizing the minimum load. The concept of such allocations is related to the notion of fair division of resources [3]. Originally, this goal function was used for describing systems where the complete system relies on keeping all the machines productive for as long as possible, as the entire system fails even in a case that just one of the machines ceases to be active [20]. Additional motivations for the goal function come from issues of Quality of Service. From the networking aspect, this problem has applications to basic problems in network optimization such as fair bandwidth allocation. Consider pairs of terminal nodes that wish to communicate; we would like to allocate bandwidth to the connections in a way that no link unnecessarily suffers from starvation, and all links get a fair amount of resources. Another motivation is efficient routing of traffic. Consider parallel links between pairs of terminal nodes. Requests for shifting flow are assigned to the links. We are interested in having the loads of the links balanced, in the sense that each link should be assigned a reasonable amount of flow, compared to the other links. Yet another incentive to consider this goal function is congestion control by fair queuing. Consider a router that can serve m shifting requests at a time. The data pieces of various sizes, needing to be shifted, are arranged in m queues (each queue may have a different data rate), each pays a price which equals the delay that it causes in the waiting line. Our goal function ensures that no piece gets a "preferred treatment" and that they all get at least some amount of delay.

We are mainly interested in two models of parallel machines, namely identical machines [22] and uniformly related machines [4, 8]. The input is a stream of jobs, of indices $1, 2, \ldots$, where the single attribute of each job j is its non-negative processing time, π_j, also known as its size. The goal is always to partition the jobs into m subsets, where each subset is to be executed on one specific machine, that is, there are m machines, of indices $\{1, 2, \ldots, m\}$, and each job is to be assigned to one of them. Online algorithms see the input jobs one at a time, and need to assign each job before becoming familiar with the remaining input. In the identical machines model, the load of a machine is the total size of the jobs assigned to it, while for uniformly related machines (also known as related machines), each machine i has a speed s_i associated with it, and the load of a machine is the total size of jobs assigned to it, scaled by the speed of the machine. Without loss of generality, let the speeds be $s_1 \leq \cdots \leq s_m$.

An additional common variant considered here is *restricted assignment* [6]. The machines have identical speeds, though each job j is associated with a subset of the machines, $M_j \subseteq \{1, 2, \ldots, m\}$, and can be assigned only to a machine in M_j.

Notation Throughout the paper we use the following notation. The size of the buffer is denoted by K. We use OPT to denote an optimal solution as well as its profit (i.e., the load of its least loaded machine). For an (offline or online) algorithm ALG we denote its profit by ALG as well. For a maximization problem (such as the problem studied here), the competitive ratio of ALG is the infimum $\mathcal{R}$ such that for any input, $\mathcal{R} \cdot \text{ALG} \geq \text{OPT}$ (note that we use numbers greater than 1 for competitive ratios of a maximization problem). For a minimization problem, the value of a solution is called its *cost* and the competitive ratio of ALG is the infimum $\mathcal{R}$ such that for any input, $\text{ALG} \leq \mathcal{R} \cdot \text{OPT}$. If the competitive ratio of ALG is at most r, then we say that it is r-competitive. Let H_t denote the t-th harmonic number, that is, $H_t = \sum_{i=1}^{t} \frac{1}{i}$. The *total work*

[1]Note that our definition of a buffer size is slightly different from the definition of Englert et al. [17], where each job must be stored in the buffer before being assigned to a machine. That is, in our definition, the size of a buffer is lower by exactly 1 than the buffer size in their definition. For example, the model without a buffer is equivalent to our model with a buffer of size zero, and the model of [17] with a buffer of size 1.

or *work* assigned to a machine is defined as the total size of jobs assigned to it. We let L_i denote the load of machine i after the termination of an algorithm, and ℓ_i denotes the load of this machine at some time during the execution or after all input has arrived, but before the jobs remaining in the buffer at the end of the input sequence have been assigned. The corresponding time at which the values ℓ_i are examined will be defined unless it is clear from the context. Analogously, we use W_i and w_i to denote the total work assigned to machine i after the algorithm has completed its execution, and at some time during execution, respectively. The values L_i and W_i (and similarly, ℓ_i and w_i) may differ only in the case of uniformly related machines. We use T to denote the total size of all jobs in a given input.

Related work This problem (without a buffer) has been well studied (known by different names such as "machine covering" and "the Santa Claus problem") in the computer science literature (see e.g. [20, 13, 12, 27, 7, 23, 16, 18, 3, 19, 2]). For identical machines, it is known that any online algorithm has a competitive ratio of at least m (the proof is folklore, see [27, 5]), and this bound can be achieved using a greedy algorithm which assigns each job to the least loaded machine, as was shown by Woeginger [27]. Several changes to the basic model have been considered to avoid this very high competitive ratio. Randomized algorithms were studied in [5], where it was shown that the best competitive ratio which can be achieved using randomization is $\tilde{\Theta}(\sqrt{m})$. Several types of semi-online variants were considered, where one of the following was assumed: the largest processing time of any job is given in advance, the total size is given in advance, or both are given. It was shown that in these cases, the best competitive ratio remains $\Theta(m)$ [26, 9]. Here we show that our approach allows us to reduce the competitive ratio much more significantly. It should be noted, that an additional, much stronger, semi-online variant was studied as well, where it is assumed that jobs arrive sorted by non-increasing processing time. In this variant, which is much closer to an offline problem than our model, the competitive ratio is at most $\frac{4}{3}$ [12, 13].

For uniformly related machines, no algorithm with finite competitive ratio exists even for two machines [5]. The semi-online problem where jobs arrive sorted by non-increasing processing times, and the problem where the profit of an optimal algorithm is known in advance, admit m-competitive algorithms, which is best possible in both cases. If the two types of information are combined, a 2-competitive algorithm is known [5]. To the best of our knowledge, no positive results are known for the case of restricted assignment. It is known that no finite competitive ratio can be achieved for a purely online algorithm, even for two machines [10], and even in the model of hierarchical machines, where for every j, the set M_j is a prefix of the machine set.

In most variants of scheduling with a buffer studied in the past, a fixed length buffer (usually of size $O(m)$) already allowed to achieve the best possible competitive ratio (for any fixed sized buffer). Moreover, in almost all cases, the competitive ratio is significantly reduced, compared to the best possible purely online algorithm. We next survey the results for the min-max allocation problem (also known as the minimum makespan problem), whose goal is dual to our goal function, with a buffer.

Kellerer et al. [25] and Zhang [28] considered the case of two identical machines, and showed that a buffer of size 1 allows to achieve a competitive ratio of $\frac{4}{3}$, which is best possible. For m identical machines, Englert et al. [17] showed that a buffer of size $O(m)$ is sufficient. Their algorithm has the best possible competitive ratio for every value of m, while this ratio tends to 1.47 for large m. It is known that for (pure) online scheduling, no algorithm can have a competitive ratio smaller than 1.853 [1, 21]. For the more general case of uniformly related machines, it was shown in [14] that for two machines, a buffer of size 2 is sufficient to achieve the best competitive ratio. In fact, for some speed ratios between the two machines,

	Classical model	Lower bounds		Upper bounds		$m = 2$
	LB = UB	Ratio	Buffer size	Ratio	Buffer size	LB=UB
Identical machines	m [27, 5]	H_m	any	$H_{m-1}+1$	$m-1$	$\frac{3}{2}$
		$\omega(\log m)$	$o(\frac{m}{\log m})$			
Related machines	∞ [5]	∞	$m-2$	$2m-\Theta(1)$	$m-1$	$\frac{2s+1}{s+1}$
		m	any	m	$m+1$	
Restricted assignment	∞ [10]	∞	$m-2$	$2m$	$O(m^2)$	2
		m	any			

Table 1: Overview of previous and new results. The entry 'any' for a buffer size means that for any fixed buffer size there is an input sequence which proves this lower bound. All the algorithms for two machines use a buffer of size 1 and achieve the best possible competitive ratio for any fixed size buffer. The parameter $s \geq 1$, in the case of two related machines, is the ratio between the machine speeds.

a buffer of size 1 is sufficient, while for some other speed ratios, a buffer of size 1 provably is not enough to achieve the best bound. Note that it was shown by Englert et al. [17] that a buffer of size $m-1$ reduces the competitive ratio for uniformly related machines below the lower bound of the case without a reordering buffer. Specifically, they designed a 2-competitive algorithm for related machines, which uses a buffer of size $O(m)$, whereas without a buffer it is known that no algorithm can have a competitive ratio below 2.43 [8]. Finally, for the standard online scheduling problem in the restricted assignment model, there are tight bounds of $\Theta(\log m)$ on the competitive ratio [6]. The lower bound still holds if there is a buffer, since the construction of [6] holds for fractional assignment (where a job can be split arbitrarily among the machines that can run it); each job can be replaced by very small jobs, and so the usage of a buffer does not change the result.

The analogous questions for preemptive scheduling on identical machines were resolved in [15]. In this variant, jobs can be cut into parts, and these parts may be assigned to different machines, with the restriction that two parts of one job cannot be simultaneously processed on two machines. The (irrevocable) assignment of job includes its allocation to non-overlapping time slots, possibly on different machines. The best possible upper bound over all values of m is $\frac{4}{3}$, while for the case without a buffer, this value is approximately 1.58, as was shown by Chen et al. [11].

Our results See Table 1 for a summary. For identical machines, we design an algorithm which uses a buffer of size $m-1$, and has a competitive ratio of at most $H_{m-1}+1$. For $m=2$, we design a different $\frac{3}{2}$-competitive algorithm, which is optimal. We show a lower bound of H_m for any fixed sized buffer, and in addition, we show that for a buffer size of $o(\frac{m}{\log m})$, an upper bound of $O(\log m)$ cannot be achieved. Interestingly, our algorithm IMITATE, imitates the behavior of a greedy algorithm for uniformly related machines, with the speeds $\{1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{m}\}$.

For the cases of related machines and restricted assignment, we extend the known negative results of [5, 10], and show that a buffer of size at most $m-2$ does not admit an algorithm of finite competitive ratio. For any fixed buffer size, we design a lower bound of m on the competitive ratio of any algorithm. The

proofs of the lower bounds in the two models are very different, but the results are similar. We complement these result by $O(m)$-competitive algorithms, which use a fixed sized buffer. Specifically, for the related machines model we design an algorithm of competitive ratio m, which uses a buffer of size $m+1$, and an algorithm of competitive ratio below $2m-1$, which uses a buffer of size $m-1$. For two machines we design a different $\frac{2s+1}{s+1}$-competitive algorithm, where $s = \frac{s_2}{s_1}$, which uses a buffer of size 1. For the restricted assignment model we present an algorithm of competitive ratio at most $2m$ which uses a buffer of size $O(m^2)$. For two machines, we give a simpler algorithm of competitive ratio 2, which only requires a buffer of size 1. In contrast to the algorithm for identical machines, which attributes *phantom speeds* to the machines, the algorithm for related machines ignores the speeds while jobs are still arriving, and once all jobs have arrived, only uses the relative order of speeds, rather than their values.

Our results strengthen the previous conclusion that a reordering buffer is a powerful tool and it allows a significant decrease in the competitive ratio of online algorithms for scheduling problems.

Outline of the paper The main body of the paper deals with arbitrary values of m, while all results for $m = 2$ are given in Section 5. In Section 2, we present our results for m identical machines, in Section 3 we present our results for m uniformly related machines, and in Section 4 we discuss the model of restricted assignment for m machines.

2 Identical machines

We start with the most elementary case of identical machines. We first show limitations on the performance and the required buffer size of any algorithm.

2.1 Lower bounds

In some of the lower bound proofs, the input sequence starts with a large number of very small jobs. In such a case, having a buffer is not meaningful, since almost all jobs must be assigned. The difficulty is in spreading these small jobs among the machines without knowing the sizes of future jobs. The next proof has some similarities with a lower bound for the preemptive variant [24].

Theorem 1. *For any buffer size K, no online algorithm can have a competitive ratio below H_m.*

Proof. Let $M \in \mathbb{N}$ be an integer which is divisible by $m!$, and let $N = KmM$. Let $\varepsilon = \frac{1}{N}$. The first part of the input contains N jobs of size ε. Let $\ell_1 \leq \cdots \leq \ell_m$ be the resulting loads for a given algorithm, after the last of these jobs has arrived. Since at most K of them remain in the buffer, we have $1 - K\varepsilon \leq \sum_{i=1}^{m} \ell_i \leq 1$.

Next, j jobs of size $1/(m-j)$ arrive for some $0 \leq j \leq m-1$. These j jobs are called *large jobs*. For this input, the optimal profit is $\text{OPT}_j = 1/(m-j)$.

Since only j large jobs arrive, out of the $j+1$ machines with the lowest loads $\ell_1, \ldots, \ell_{j+1}$, at least one does not receive such a job, and all it could possibly receive would be the contents of the buffer, that is, at most K additional jobs of size ε. Therefore, in this scenario, the profit of the solution returned by the algorithm, denoted by ALG_j, satisfies $\text{ALG}_j \leq \ell_{j+1} + K\varepsilon$.

Let $\mathcal{R}$ be the competitive ratio of the algorithm. Then

$$\text{OPT}_j = \frac{1}{m-j} \leq \mathcal{R}(\ell_{j+1} + K\varepsilon) \quad \text{holds for all} \quad 0 \leq j \leq m-1.$$

Taking the sum over all j we get

$$\sum_{j=0}^{m-1} \frac{1}{m-j} \leq \mathcal{R} \left(\sum_{j=0}^{m-1} \ell_{j+1} \right) + \mathcal{R}mK\varepsilon$$

$$\Rightarrow H_m = \sum_{j=1}^{m} \frac{1}{j} = \sum_{j=0}^{m-1} \frac{1}{m-j} \leq \mathcal{R} \sum_{j=1}^{m} \ell_j + \mathcal{R}mK\varepsilon \leq \mathcal{R}(1 + mK\varepsilon) = \mathcal{R} \left(1 + \frac{1}{M} \right)$$

by the definition of N. We find that $\mathcal{R} \geq H_m \cdot \frac{M}{M+1} \to H_m$ for $M \to \infty$. □

The following proposition implies that in fact we need a buffer of size $\Omega(m/\log m)$ to get a competitive ratio which is logarithmic in m.

Proposition 2. *Let $f(m)$ be a function where $f(m) < m$ for all $m > 1$. For a buffer size $f(m)$, no algorithm can have a competitive ratio below $m/(f(m)+1)$.*

Proof. The proof is similar to the case $f(m) = 0$ (see [5]), i.e., the pure online case. The input starts with m jobs of size 1. At least $m - f(m)$ of them must be assigned to machines. If at least two of them were assigned to the same machine, no additional jobs arrive, the algorithm is left with at least one empty machine, so ALG $= 0$ while OPT $= 1$. Otherwise, $m-1$ jobs of size m arrive. Any machine which does not receive such a job has a load of at most $f(m)+1$, since it has at most one job of size 1 before the jobs remaining in the buffer are assigned, and the total size of the jobs in the buffer is at most $f(m)$. At this time OPT $= m$, which implies the claimed lower bound. □

2.2 Algorithm

While the lower bound constructions are relatively simple, it is harder to see what an algorithm of sufficiently small competitive ratio should do. Intuitively, given the lower bound construction in the proof of Theorem 1, the machines should be relatively imbalanced. We achieve this by acting as if the machines have speeds (the so-called phantom speeds). We next define the algorithm IMITATE, which has a buffer of size $m-1$. IMITATE always keeps the largest $m-1$ jobs in the buffer.

ALGORITHM IMITATE. We store the first $m-1$ jobs in the buffer. Upon an arrival of a job, in the case that the buffer already contains $m-1$ jobs, consider those jobs and the new job. Let the size of the smallest job among these m jobs be X. A job of size X will be assigned while the other jobs are stored in the buffer. The algorithm imitates the behavior of the so-called post-greedy algorithm for uniformly related machines. We define the phantom speed of the machine of index i to be $\frac{1}{i}$. Letting ℓ_i denote the current load of machine i, the next job is assigned to a machine which minimizes $i \cdot (\ell_i + X)$.

Upon termination of the input sequence, let $p_2 \leq \ldots \leq p_m$ be the sizes of jobs remaining in the buffer. (If there are only $j < m-1$ jobs in the buffer, then we let $p_i = 0$ for $2 \leq i \leq m-j$.) The job of size p_i is assigned to machine i (machine 1 does not get a job). We call this job *the final job* of machine i.

In what follows, we use ℓ_i to denote the load of machine i just before the assignment of the final job. By the definition of the algorithm, $L_i = \ell_i + p_i$ for $2 \leq i \leq m$ and $L_1 = \ell_1$. Note that if $p_2 = 0$, that is, the buffer contains less than $m-1$ jobs upon termination of the input sequence, then $\ell_i = 0$ for all $1 \leq i \leq m$. In this (not interesting) case, we have OPT $=$ ALG $= 0$. Thus, in what follows, we assume that the buffer contains $m-1$ jobs upon termination.

To analyze the algorithm, we start with proving the following lemmas.

We denote the total size of all jobs except for the final ones by Δ. That is, we let $\Delta = \sum_{i=1}^{m} \ell_i$.

Lemma 3. $(H_{m-1}+1)\ell_1 \geq \Delta$.

Proof. We first show that $\ell_1 \geq (i-1)\ell_i$ for $i \geq 2$. If $\ell_i = 0$, then we are done. Otherwise let μ be the size of the last job ever assigned to machine i prior to the allocation of the final jobs. Let λ_1 be the load of machine 1 at the time when the job of size μ was assigned. Then $\lambda_1 + \mu \geq i\ell_i$ by the post-greedy assignment. Since $\ell_i \geq \mu$ and $\ell_1 \geq \lambda_1$, we get $\ell_1 \geq \lambda_1 \geq i\ell_i - \mu \geq (i-1)\ell_i$.

We conclude that for all $i \geq 2$, $\ell_1 \geq (i-1)\ell_i$, and hence $\frac{\ell_1}{i-1} \geq \ell_i$. We sum up the last inequality for all $2 \leq i \leq m$, and get $\sum_{i=2}^{m} \frac{\ell_1}{i-1} \geq \sum_{i=2}^{m} \ell_i = \Delta - \ell_1$. On the other hand, $\sum_{i=2}^{m} \frac{\ell_1}{i-1} = H_{m-1}\ell_1$, so $(H_{m-1}+1)\ell_1 \geq \Delta$. □

Lemma 4. *For any $k > 1$, $H_m k L_k \geq \Delta$.*

Proof. We first show that $k \cdot L_k \geq i \cdot \ell_i$ for all $1 \leq i \leq m$ and $2 \leq k \leq m$. If $\ell_i = 0$ or $i = k$, this trivially holds. Otherwise, let μ be the size of the last job ever assigned to machine i (neglecting the final job). Let λ_k be the load of machine k at the time in which the job of size μ was assigned. Then $k(\lambda_k + \mu) \geq i\ell_i$ by the post-greedy assignment rule. We have $L_k = \ell_k + p_k \geq \lambda_k + \mu$, using the properties $p_k \geq \mu$, since just before the assignment of the final jobs, the buffer contains the largest $m-1$ jobs in the input, and $\ell_k \geq \lambda_k$, since ℓ_k is the load of machine k just before the final jobs are assigned, while λ_k is the load of this machine at an earlier time. Therefore $L_k \geq \lambda_k + \mu \geq \frac{i \cdot \ell_i}{k}$.

Summing up over $1 \leq i \leq m$, we get

$$\Delta = \sum_{i=1}^{m} \ell_i \leq \sum_{i=1}^{m} \frac{k \cdot L_k}{i} = kL_k \sum_{i=1}^{m} \frac{1}{i} = kL_k H_m.$$

□

Lemma 5. *For any $1 \leq k \leq m$, $\text{OPT} \leq \frac{1}{k}(\Delta + \sum_{j=2}^{k} p_j)$.*

Proof. Consider an optimal solution. The total size of jobs that are not assigned to machines containing the $m-k$ largest final jobs (of sizes $p_{k+1}, \ldots, p_m$) is at least $k \cdot \text{OPT}$. The total size of these jobs is in fact at most $\Delta + \sum_{j=2}^{k} p_j$. □

The following theorem follows from the lemmas above.

Theorem 6. *The competitive ratio of* IMITATE *is at most $H_{m-1}+1$.*

Proof. Consider machine k, for $k > 1$. For $i < k$, by definition, we have $p_i \leq p_k$. Since $L_k = \ell_k + p_k \geq p_k$, we get $p_i \leq L_k$. Thus we get

$$\begin{aligned}
k \cdot \text{OPT} &\leq \Delta + \sum_{j=2}^{k} p_j && \text{by Lemma 5} \\
&\leq H_m k L_k + \sum_{j=2}^{k} p_j && \text{by Lemma 4} \\
&\leq L_k \left(kH_m + k - 1\right) && \text{since } p_i \leq L_k \text{ for } i \leq k \\
&\leq kL_k \left(H_m - \frac{1}{m} + 1\right) && \text{using } k \leq m.
\end{aligned}$$

Thus $L_k(H_{m-1}+1) \geq \text{OPT}$. As for machine 1, the inequality $\text{OPT} \leq \frac{1}{k}(\Delta + \sum_{j=2}^{k} p_k)$ from Lemma 5 reduces to $\text{OPT} \leq \Delta$, so Lemma 3 implies $L_1(H_{m-1}+1) \geq \text{OPT}$. □

3 Uniformly related machines

3.1 Lower bounds

Theorem 7. *For a buffer of size at most $m-2$, no algorithm can have a finite competitive ratio on related machines.*

Proof. Let the speed of machine i be S^{i-1} for some large S. Without loss of generality assume that the buffer has size $m-2$. The input sequence starts with $m-1$ jobs of sizes

$$S, S^2, \ldots, S^{m-1}.$$

At least one of these jobs cannot remain in the buffer. Let S^k be the size of an assigned job. Let j be the machine it is assigned to. If $j \leq k$, a final job of size 1 arrives. Otherwise, a job of size S^m arrives.

In the first case, there is a machine in $\{k+1, \ldots, m\}$ which does not receive a job in $\{S^k, \ldots, S^{m-1}\}$. This machine has a load of at most $S^{k-1}/S^k = 1/S$ (there are m jobs, so if a machine gets two jobs then $\text{ALG} = 0$). In this case, $\text{OPT} = 1$.

In the second case, if the machines $\{k+1, \ldots, m\}$ have all jobs $\{S^k, \ldots, S^m\}$, then machines $\{1, \ldots, k\}$ only have $k-1$ jobs and $\text{ALG} = 0$. Therefore, we may assume as before that each machine must have one job exactly. Thus the machine $j > k$ having the job of size S^k has a load of at most $S^k/S^k = 1$, while $\text{OPT} = S$. Letting $S \to \infty$, we get an unbounded competitive ratio in both cases. □

Theorem 8. *For any buffer size K, no algorithm can have a competitive ratio below m for related machines.*

Proof. Let the speed of machine i be S^{i-1} for some large S. The first phase is the same as for identical machines (see Theorem 1). That is, N small jobs of size ε arrive, where $N >> K$, $N \in \mathbb{N}$, $m!|N$ and $\varepsilon = 1/N$. Let $w_1, \ldots, w_m$ be the total work assigned to each machine after the last job of the first phase arrives. We choose a machine $1 \leq j \leq m$ such that $w_j \leq \frac{1}{m}$. Such a machine must exist since $\sum_{i=1}^{m} w_i \leq 1$.

Next $m-1$ jobs arrive. Specifically, the next jobs are of sizes

$$1/S^{j-1}, 1/S^{j-2}, \ldots, 1/S, S, S^2, \ldots, S^{m-j}\ ,$$

that is, for each size in $\{S^i | 1-j \leq i \leq m-j, i \neq 0\}$, one job of this size arrives. Thus if $j = 1$ all arriving jobs have sizes above 1 and if $j = m$ then all arriving jobs have sizes below 1. We have that the optimal profit in this case is $\text{OPT}_j = 1/S^{j-1}$ (assign the job of size $1/S^i$ to machine $j-i$ for $i \neq 0$ and assign all small jobs to machine j; note that i can be negative).

If at least one of the jobs $S, S^2, \ldots, S^{m-j}$ is *not* assigned to a machine in $\{j+1, \ldots, m\}$, then the profit of the algorithm is at most $(1 + \frac{1}{S-1})/S^j$, which is an upper bound on the total size of the other jobs divided by the speed of the slowest machine in this set. In this case, the competitive ratio is at least $S-1$, which is at least m for large enough S. Otherwise, the load of machine j is at most

$$\frac{w_j + K\varepsilon + 1/S^{j-1} + \cdots + 1/S}{S^{j-1}} \leq \frac{w_j + K\varepsilon + 1/(S-1)}{S^{j-1}}\ .$$

We get a competitive ratio of

$$\frac{1}{w_j + K\varepsilon + 1/(S-1)} \geq \frac{1}{1/m + K\varepsilon + 1/(S-1)} .$$

Letting $S \to \infty$ and $\varepsilon \to 0$, we conclude that the competitive ratio is at least m. □

3.2 An algorithm for related machines and a buffer of size $m+1$

We next present an algorithm of competitive ratio m which uses a buffer of size $m+1$. In the next section, we design another algorithm of a competitive ratio slightly less than $2m-1$, which uses a buffer of size $m-1$. Our algorithms ignore the exact speeds, and only use their relative order.

Here we show that by using just two extra buffer positions compared to the minimum number of positions required to get an algorithm of finite competitive ratio, it is possible to get an optimal competitive ratio of m and hence save a factor of almost 2 compared to the algorithm of the next section.

The algorithm works as follows. The first $m+1$ jobs are stored in the buffer. Upon arrival of a job, it is possibly swapped with a job of the buffer to maintain the property that the buffer contains the largest jobs seen so far.

The algorithm runs List Scheduling (LS) [22] on the machines, while ignoring the speeds. That is, a job is assigned to a minimally loaded machine, that is, a machine for which the total work assigned to it so far is minimum. Let the remaining jobs in the buffer when the input ends be $p_0 \leq \cdots \leq p_m$. We slightly abuse notation and let p_i denote also the size of job p_i. If the buffer contains at most $m-1$ jobs of strictly positive sizes, then any assignment is optimal, since in this case, OPT $= 0$. If there are m such jobs in the buffer (that is, p_0 does not exist or $p_0 = 0$), then assigning job p_i to machine i results in an optimal solution. The remaining case that the buffer contains $m+1$ jobs of positive sizes (and thus OPT > 0) is analyzed below (even if no other jobs were assigned).

The jobs $\{p_0, p_1, \ldots, p_m\}$ are assigned in one of the following $2m-2$ ways, depending on which option gives the largest profit.

1. Assign p_0 to a minimally loaded machine (in terms of the total work assigned to it), and p_i to machine i for all $i = 1, 2, \ldots, m$.
2. Assign the jobs as in the previous case, but move p_j to machine m for some $1 \leq j \leq m-1$.
3. Assign p_i to machine i for all $i = 1, 2, \ldots, m$. Assign p_0 to a machine k for some $2 \leq k \leq m-1$.

In the case $m = 2$, the algorithm can be easily modified to keep only two jobs in the buffer, rather than three, since the third option does not exist, and p_0 is always assigned by the same rule as the previous jobs.

In our analysis below, we show that there is always at least one assignment which shows that the competitive ratio is at most m.

Theorem 9. *The competitive ratio of this algorithm is at most m.*

Proof. We scale the input such that OPT $= 1$. Recall that T denotes the total size of all jobs in the input. Then $T \geq \sum_{i=1}^{m} s_i$. If $p_i \geq s_i/m$ for $i = 1, \ldots, m$, then we are done using the first assignment, since $L_i = \frac{W_i}{s_i} \geq \frac{p_i}{s_i} \geq \frac{1}{m}$. Else, let k be the maximum index for which $p_k < s_k/m$.

If we analyze the first assignment or an assignment of the second type, then we let w_i denote the total work assigned to machine i, neglecting $p_1, \ldots, p_m$ but not p_0. That is, since p_0 is assigned greedily in all

these assignments, it is counted as part of some w_i. If we analyze an assignment of the third type, then p_0 is neglected as well, and it is not counted as a part of any w_i.

Once the type of assignment which is being considered is stated, we let j denote an index of a machine for which w_j is maximum, that is,

$$j = \arg\max_i w_i.$$

The next two claims are used several times in the proof.

Claim 10. *For any* $1 \leq t \leq m$,

$$T - \sum_{i=t+1}^{m} p_i \geq s_1 + \cdots + s_t \ .$$

Proof. Consider an optimal assignment. For a given value of t, in this assignment there are at least t machines that do not have jobs in the set $\{p_{t+1}, \ldots, p_m\}$ assigned to them. The total size of all jobs assigned to those machines cannot exceed $T - \sum_{i=t+1}^{m} p_i$. Since $\text{OPT} = 1$, the total size of all jobs assigned to those machines is at least their total speed, which is at least $\sum_{i=1}^{t} s_i$ (since this is the minimum total speed of any subset of t machines). □

Claim 11. *For any pair of machines* i_1, i_2 *and index* $0 \leq t \leq m$, $w_{i_1} \leq w_{i_2} + p_t$ *holds.*

Proof. The claim clearly holds if $w_{i_1} = 0$ or if $i_1 = i_2$. Otherwise, let x be the size of the last job assigned to machine i_1 which is included in w_{i_1}. Due to the assignment rule, $w_{i_2} + x \geq w_{i_1}$ (this holds even if the job of size x is p_0, since if it is included in w_{i_1} then this job is assigned by the LS rule as well). Since $x \leq p_t$ for any $0 \leq t \leq m$, the claim follows. □

We consider three cases.

Case 1: $k = m$ (i.e., $p_m < s_m/m$). We analyze only options where p_0 is assigned greedily to the least loaded machine. Recall that in such cases p_0 is counted as a part of some w_i.

In this case we prove the following claim.

Claim 12. *For any* $1 \leq i < m$, $\sum_{t=1}^{m} w_t \geq s_i$ *holds.*

Proof. We have

$$\sum_{t=1}^{m} w_t = T - \sum_{t=1}^{m} p_t \geq T - m \cdot \frac{s_m}{m} \geq s_i$$

due to the definition of T, the property $p_i \leq p_m < \frac{s_m}{m}$ and $T \geq \sum_{t=1}^{m} s_t \geq s_i + s_m$ (applied in this order). □

If $j = m$, then we analyze the first assignment option. Recall that T is the total size of all the jobs, that is, $T = \sum_{i=1}^{m}(w_i + p_i)$. The final load of machine m is

$$L_m = \frac{W_m}{s_m} = \frac{w_m + p_m}{s_m} = \frac{\max_i w_i + \max_i p_i}{s_m} \geq \frac{T/m}{s_m} \geq \frac{1}{m} \ ,$$

where the last inequality holds since $T \geq \sum_{t=1}^{m} s_t \geq s_m$. As for $i < m$,

$$L_i = \frac{W_i}{s_i} = \frac{w_i + p_i}{s_i} \geq \frac{w_m}{s_i} \geq \frac{1}{m s_i} \sum_{t=1}^{m} w_t \geq \frac{1}{m}$$

holds since by Claim 11, we have $w_i + p_i \geq w_m$, $w_m = \max_t w_t \geq \frac{\sum_{t=1}^{m} w_t}{m}$, and $\sum_{t=1}^{m} w_t \geq s_i$ by Claim 12 (applied in this order).

If $j < m$, consider the second type of assignment, for this specific value of j. By Claim 12, $\sum_{t=1}^{m} w_t \geq s_i$, $w_j = \max_t w_t \geq \frac{\sum_{t=1}^{m} w_t}{m}$ and so

$$w_j \geq \frac{s_i}{m} \text{ for all } i < m.$$

By Claim 11, $w_i + p_i \geq w_j$, so

$$L_i = \frac{W_i}{s_i} = \frac{w_i + p_i}{s_i} \geq \frac{w_j}{s_i} \geq \frac{1}{m} \text{ for } 1 \leq i < m, i \neq j.$$

For machine j, already $\frac{w_j}{s_j} \geq \frac{1}{m}$ so we get $L_j = \frac{W_j}{s_j} = \frac{w_j}{s_j} \geq \frac{1}{m}$. Finally, for machine m,

$$L_m = \frac{W_m}{s_m} = \frac{w_m + p_m + p_j}{s_m}.$$

Using Claim 11, we have $w_m + p_j \geq w_j$, and since $w_j = \max_i w_i$, we get

$$W_m = (w_m + p_j) + p_m \geq \max_i w_i + \max_{\{1 \leq i \leq m\}} p_i \geq \frac{T}{m} \geq \frac{s_m}{m}.$$

Case 2: $1 < k < m$. Consider the third assignment option, where p_0 is assigned to machine k. By the definition of the w_i values in this case, $T = \sum_{i=1}^{m} w_i + \sum_{i=0}^{m} p_i$.

For machines $k + 1 \leq i \leq m$, we have $p_i \geq \frac{s_i}{m}$, so $L_i = \frac{W_i}{s_i} = \frac{w_i + p_i}{s_i} \geq \frac{p_i}{s_i} \geq \frac{1}{m}$ and we are done. Since $p_k < \frac{s_k}{m}$, $p_i \leq p_k < \frac{s_k}{m}$ for $0 \leq i \leq k$, so we have

$$\sum_{i=0}^{k} p_i < (k+1) \cdot \frac{s_k}{m} \leq s_k$$

where the last inequality holds since $k < m$.

Using Claim 10,

$$\sum_{i=1}^{m} w_i + \sum_{i=0}^{k} p_i \geq s_1 + \cdots + s_k,$$

which implies $\sum_{i=1}^{m} w_i \geq s_{k-1}$ (using the property $k > 1$) and therefore $w_j \geq s_{k-1}/m$. By Claim 11, $w_i + p_i \geq w_j$ for any machine $1 \leq i \leq m$, so for $1 \leq i \leq k-1$, we find

$$L_i = \frac{W_i}{s_i} = \frac{w_i + p_i}{s_i} \geq \frac{w_j}{s_{k-1}} \geq \frac{1}{m}.$$

Using Claim 11, $w_k + p_0 \geq w_j$, so

$$W_k = w_k + p_0 + p_k \geq w_j + p_k \geq \frac{\sum_{i=1}^{m} w_i}{m} + \frac{\sum_{i=0}^{k} p_i}{k+1} \geq \frac{\sum_{i=1}^{m} w_i}{m} + \frac{\sum_{i=0}^{k} p_i}{m} \geq \sum_{i=1}^{k} \frac{s_i}{m} \geq \frac{s_k}{m},$$

since $w_j \geq w_i$ for all $1 \leq i \leq m$, $p_k \geq p_i$ for all $0 \leq i \leq k$, and $k + 1 \leq m$ (applied in this order).

Case 3: $k = 1$. We have $p_i \geq s_i/m$ for $i = 2, \ldots, m$, so we only need to consider machine 1. Consider the first assignment. By Claim 10, $\sum_{i=1}^{m} w_i + p_1 \geq s_1$. By Claim 11 we have $w_1 + p_1 \geq w_i$ for $2 \leq i \leq m$, so

$$s_1 \leq \sum_{i=1}^{m} w_i + p_1 = (w_1 + p_1) + \sum_{i=2}^{m} w_i \leq m(w_1 + p_1) \ ,$$

or equivalently, $w_1 + p_1 \geq s_1/m$, and therefore $L_1 = \frac{W_1}{s_1} = (w_1 + p_1)/s_1 \geq 1/m$. □

3.3 An algorithm for related machines and a buffer of size $m - 1$

We define an algorithm which uses only a buffer of size $m - 1$. Its competitive ratio will be smaller than $2m - 1$ (compared to m which we obtained in the previous section). The motivation for the behavior of this algorithm is that the slowest machine, that is, machine 1, can get a sufficient share of the jobs even without receiving a large job out of the buffer. This option is particularly important in the case that the $m - 1$ jobs remaining in the buffer are relatively large, and an optimal solution assigns all other jobs to machine 1.

The algorithm works as follows. The first $m - 1$ jobs are stored in the buffer. Upon the arrival of a job, let z be the minimum sized job out of the m available jobs, as well as its size. We show how z is assigned, while the other $m - 1$ jobs are stored in the buffer.

Let $\rho(m)$ be a positive function of the number of machines, where $\rho(m) > 1$ for all m. Let w_i denote the current total work assigned to machine i. If $w_1 < \frac{\sum_{i=1}^{m} w_i + z}{\rho(m)}$, then assign z to machine 1. Otherwise assign it to a machine i ($2 \leq i \leq m$), minimizing $w_i + z$.

When no more jobs arrive, let $p_2 \leq p_3 \leq \ldots \leq p_m$ be the jobs remaining in the buffer (and their sizes). Assign job p_i to machine i. If the buffer contains less than $m - 1$ jobs, assign them arbitrarily. Below we only analyze the case where $\text{OPT} > 0$.

Let $\mathcal{R}(m) = \frac{2m-1+\sqrt{4m^2-8m+5}}{2}$. Note that $\mathcal{R}(2) = \frac{3+\sqrt{5}}{2} \approx 2.618$. We first state some properties of $\mathcal{R}(m)$, which follow from elementary algebra. We abbreviate $\mathcal{R}(m)$ by $\mathcal{R}$.

Proposition 13. *$\mathcal{R}$ satisfies: $2m - \frac{3}{2} < \mathcal{R} < 2m - 1$, and $(2m - 1 - \mathcal{R})\mathcal{R} = m - 1$.*

We next analyze the algorithm.

Theorem 14. *For every value of m, using $\rho(m) = \mathcal{R}(m)$, the competitive ratio of the algorithm is exactly $\mathcal{R}(m)$. Moreover, this is the best competitive ratio which can be achieved by any positive function $\rho(m)$.*

Proof. We first prove an upper bound of $\mathcal{R}(m)$ on the competitive ratio. We again scale the input such that $\text{OPT} = 1$ and let T be the total size of all the jobs. Let w_i denote the total work assigned to machine i before the assignment of the jobs of the buffer. Let $T' = T - \sum_{i=2}^{m} p_i$, so that $\sum_{i=1}^{m} w_i = T'$.

There is at least one job assigned to machine 1, since it receives the very first job ever assigned.

Claim 15. *The final load of machine 1 satisfies $L_1 \geq \frac{\text{OPT}}{\mathcal{R}}$.*

Proof. We first prove $w_1 \geq \frac{T'}{\mathcal{R}}$. Let w_i^t denote the total work assigned to machine i just before the t-th job ever assigned is assigned. If n' jobs are assigned by the main loop, then according to this notation, $w_i = w_i^{n'+1}$. Let x_t denote the size of the t-th assigned job. Let $T_t = \sum_{i=1}^{m} w_i^{t+1} = \sum_{i=1}^{m} w_i^t + x_t$, be the total size of jobs (excluding jobs in the buffer) at the time of assignment of the t-th job. Thus $T' = T_{n'}$ and $T_0 = 0$.

We prove by induction that $w_1^{t+1} \geq \frac{T_t}{\mathcal{R}}$. This holds for $t = 0$. Assume that it holds for t and we prove it for $t+1$. If the algorithm assigns the job of size x_{t+1} to machine 1, then $w_1^{t+2} = w_1^{t+1} + x_{t+1} \geq \frac{T_t}{\mathcal{R}} + x_{t+1} = \frac{T_{t+1}}{\mathcal{R}} + \frac{\mathcal{R}-1}{\mathcal{R}} x_{t+1} > \frac{T_{t+1}}{\mathcal{R}}$. Otherwise, by the rules of the algorithm, $w_1^{t+2} = w_1^{t+1} \geq \frac{T_{t+1}}{\mathcal{R}}$.

We have $\text{OPT} \leq \frac{T'}{s_1}$, since there is at least one machine i of OPT that has no job out of $\{p_2, p_3, \ldots, p_m\}$, and its load (in OPT) is at most $\frac{T'}{s_i} \leq \frac{T'}{s_1}$. By applying $w_1 \geq \frac{T'}{\mathcal{R}}$ we get $L_1 = \frac{W_1}{s_1} = \frac{w_1}{s_1} \geq \frac{T'}{\mathcal{R} \cdot s_1} \geq \frac{\text{OPT}}{\mathcal{R}}$. □

Assume by contradiction that the competitive ratio is not maintained. We already proved $L_1 \geq \frac{\text{OPT}}{\mathcal{R}}$, so we let $i \geq 2$ be a maximum index of a machine with $L_i = \frac{w_i + p_i}{s_i} < \frac{\text{OPT}}{\mathcal{R}} = \frac{1}{\mathcal{R}}$.

Claim 16. *For any machine $k > 1$, such that $k \neq i$, $w_k < \frac{s_i}{\mathcal{R}}$ holds.*

Proof. If machine k did not receive any jobs before job p_k, then the claim trivially holds. Otherwise, let z_k denote the last job assigned to machine k before p_k and its size. Since z_k was not assigned to machine i, $w_i \geq w_k - z_k$. Using $p_i \geq z_k$, since the jobs of the buffer are largest, we get $\frac{s_i}{\mathcal{R}} > w_i + p_i \geq w_k$. □

Claim 17. $w_1 \leq \frac{T'}{\mathcal{R}} + p_i$.

Proof. Consider the last job assigned to machine 1, and denote it and its size by z_1. Just before the assignment of this last job, the work assigned to machine 1 was smaller than the total size of all jobs including z_1 divided by $\mathcal{R}$. This total size is at most T'. Using $z_1 \leq p_i$ we get the claim. □

Summarizing, we have $w_k < \frac{s_i}{\mathcal{R}}$ for any $k \notin \{1, i\}$ by Claim 16, $w_1 \leq \frac{T'}{\mathcal{R}} + p_i$ by Claim 17 and $w_i + p_i < \frac{s_i}{\mathcal{R}}$ by our choice of i and since $\text{OPT} = 1$. Summing up these bounds for all $1 \leq k \leq m$ we get

$$\begin{aligned} p_i + \sum_{k=1}^{m} w_k = p_i + T' &< \frac{T'}{\mathcal{R}} + p_i + (m-1)\frac{s_i}{\mathcal{R}} \\ \Rightarrow T' \cdot (\mathcal{R} - 1) &< (m-1) s_i \,, \end{aligned} \tag{1}$$

which holds by reorganization of the equation.

On the other hand, consider some subset of i machines which have no jobs of the set $\{p_{i+1}, \ldots, p_m\}$ in OPT. At least one of those machines has a speed of at least s_i. The total size of jobs assigned to these machines in OPT is at most $T' + \sum_{j=2}^{i} p_j$, so $T' + \sum_{j=2}^{i} p_j \geq s_i$. Since $p_i \leq w_i + p_i < \frac{s_i}{\mathcal{R}}$, and $p_j \leq p_i$ for $j \leq i$, we have $T' + (i-1)\frac{s_i}{\mathcal{R}} \geq s_i$. Since $i \leq m$, this implies $T' + (m-1)\frac{s_i}{\mathcal{R}} \geq s_i$.

Thus, multiplying by $\mathcal{R} - 1$ and reorganizing we have $(\mathcal{R}-1)T' \geq (\mathcal{R}-1)s_i(1 - \frac{m-1}{\mathcal{R}})$, and using (1) we get $(\mathcal{R}-1)s_i(1 - \frac{m-1}{\mathcal{R}}) < (m-1)s_i$, or equivalently $\mathcal{R}^2 - 2m\mathcal{R} + \mathcal{R} < -m + 1$.

Since $\mathcal{R}^2 - 2m\mathcal{R} + \mathcal{R} = 1 - m$, we get a contradiction.

We next show that the analysis is tight, and moreover, that using a different function $\rho(m)$ would not result in a smaller competitive ratio.

Consider a function $\rho(m)$ and let N be a large integer. Consider an input consisting of N jobs of size $\frac{s_1}{N}$, followed by $m-1$ jobs of size s_m. For a sufficiently large value of N, the total work assigned to the first machine is arbitrarily close to $\frac{s_1}{\rho(m)}$, so its load is approximately $\frac{1}{\rho(m)}$. The first machine does not receive any of the last $m-1$ jobs since these are the jobs stored in the buffer. The optimal profit for this input is at least 1. Thus, if $\rho(m) = \mathcal{R}(m)$ then the competitive ratio is exactly $\mathcal{R}(m)$. If $\rho(m) > \mathcal{R}(m)$ then the competitive ratio is higher.

We next consider cases where $\rho(m) \leq \mathcal{R}(m)$. Let $\Delta = \mathcal{R}(m) - 2m + 2$, so $\frac{1}{2} < \Delta < 1$. Consider the case of a set of m machines, out of which, $m-1$ are of speed $\varepsilon = \frac{1}{N}$ for a large integer $N > 2m$, while machine m has a speed of $\mathcal{R}(m)$. The input consists of $m-1$ jobs of size 1, a job of size Δ, a job of size 2ε, $m-2$ jobs of size ε, and additional $m-1$ jobs of size 1. Clearly, OPT ≥ 1. For simplicity we assume that in the action of the algorithm, ties are broken in favor of faster machines (i.e., machines of higher indices). If this is not the case, it is possible to add small jobs to the instance so that after the assignment of these jobs, machine m is more loaded than machines $2, 3, \ldots, m-1$.

The algorithm stores the first $m-1$ jobs in the buffer until termination. The job of size Δ is assigned to the first machine, machine m gets a job of size 2ε, and each other machine receives a job of size ε. The first machine does not receive any of these jobs since $\Delta > \frac{1}{2} > m\varepsilon > \frac{m\varepsilon}{\rho(m)}$.

We prove that $\rho(m) < \frac{m-1+m\varepsilon+\Delta}{\Delta}$ holds. ¿From this it will follow that $\Delta < \frac{m-1+m\varepsilon+\Delta}{\rho(m)}$, so at least one of the next $m-1$ jobs is assigned to machine 1. Let $\Gamma = \frac{m-1+\Delta+m\varepsilon}{\Delta}$. We have $\Gamma > \frac{m-1+\mathcal{R}-2m+2}{\mathcal{R}-2m+2} = \frac{\mathcal{R}-m+1}{\mathcal{R}-2m+2} = \mathcal{R}$, by Proposition 13. Thus $\rho(m) \leq \mathcal{R}(m) < \Gamma$.

Thus, there are at most $m-2$ jobs of size 1 assigned to machines $2, 3, \ldots, m$, and since machine m is more loaded than the other machines then it does not receive any of these jobs. Finally, in the step where the jobs of the buffer are distributed, machine m receives one job of size 1. Thus its final load is $\frac{1+2\varepsilon}{\mathcal{R}(m)}$ while OPT ≥ 1.

The first example shows that if $\rho(m) \geq \mathcal{R}(m)$ then the algorithm cannot have a competitive ratio below $\mathcal{R}(m)$, while the second example shows that if $\rho(m) \leq \mathcal{R}(m)$ then the algorithm cannot have a competitive ratio below $\mathcal{R}(m)$ by taking $\varepsilon \to 0$ in this example. Therefore, the analysis is tight, and the competitive ratio cannot be reduced using a different function $\rho(m)$. □

4 Restricted assignment

4.1 Lower bounds

In this section we prove lower bounds for the case of restricted assignment.

Theorem 18. *For a buffer of size at most $m-2$, no algorithm can have a finite competitive ratio in the restricted assignment setting.*

Proof. The first part of the input consists of $m-1$ unit jobs with the machine sets $\{1,2\}, \{2,3\}, \ldots, \{m-1, m\}$. If the buffer size is $K < m-1$, then at least one job with (say) the processing set $\{i, i+1\}$ is assigned before the next item arrives. Denote the machine that this job is assigned to by $j \in \{i, i+1\}$. The next and final unit size job that arrives can only be assigned to machine j.

Now, it can be seen that some machine will remain empty, since only $m-2$ jobs can be assigned to the $m-1$ machines that are not j, whereas it is always possible to distribute the jobs equally over the machines and get a load of 1 everywhere. □

Theorem 19. *For any fixed size buffer (of size K), any online algorithm has a competitive ratio of at least m.*

Proof. Let N be a large integer. The first phase of the input consists of jobs of a total size of 1, composed of NK jobs of size $\varepsilon = \frac{1}{NK}$, with the machine set $\{1, 2, \ldots, m\}$. Let i be a machine which receives a total load of at most $\frac{1}{m}$ after the last job of the first phase has arrived. Such a machine must exists since the total

load of the m machines is at most 1. The input continues with $m-1$ jobs. For each $j \neq i$, $1 \leq j \leq m$, there is a job of size 1, with the processing set $\{j\}$. At this time, $\text{OPT} = 1$, since machine i can receive all small jobs. On the other hand, considering machine i, we get that $\text{ALG} \leq \frac{1}{m} + K\varepsilon = \frac{1}{m} + \frac{1}{N}$. Letting N tend to infinity, we get a lower bound of m. □

4.2 An algorithm for restricted assignment with a fixed size buffer

We now present an algorithm for restricted assignment. At each time, for every machine i, we keep the m largest jobs which can be processed on i in the buffer. Every job which is not in this set (or a job which stops being in this set) is assigned in a greedy fashion to a least loaded machine which can process it. The buffer may therefore contain at most m^2 jobs and thus, the required size of the buffer is m^2. Assume $\text{OPT} > 0$, so every machine has at least one job that can be assigned to it.

At the end of the input we assign the jobs from the buffer in an optimal way (i.e., we test all possible assignments and pick the best one). This enumeration requires exponential time. We will analyze a fixed assignment which depends on a (fixed) optimal solution OPT, which we define next. Our goal is to show that our algorithm can find this assignment or a more profitable one (since it tests all possible assignments of the jobs stored in the buffer), and that its profit is at least $\text{OPT}/(2m)$.

Let S be the set of machines i such that OPT has at least one job of size $\text{OPT}/(2m)$ assigned to i. Then, we would like to pick for every machine in S one job from the buffer of size at least $\text{OPT}/(2m)$, and assign it to that machine. Note that a machine for which there are less than m jobs which it can process must belong to S. This holds since in the optimal solution it has at most $m-1$ jobs assigned to it, and the largest of these therefore has a size of at least $\text{OPT}/(m-1)$.

Lemma 20. *Such an assignment of jobs of size at least* $\text{OPT}/(2m)$ *is possible.*

Proof. Let j_i be the largest job (of size at least $\text{OPT}/(2m)$) assigned to machine i by OPT. If j_i is in the buffer, then we assign it to machine i. We do this for every $i \in S$. So far jobs were only assigned to their machines in OPT, so each job was assigned to at most one machine.

At the end of this process there might be additional machines in S for which we did not assign any job, that is, the job in OPT which we planned to assign to each such machine is not in the buffer. Therefore, the reason that for such a machine i we did not assign jobs is that there are more than m jobs which can be processed by i and which are larger than j_i. This means that i is a machine which initially (after the termination of the input) has m jobs in the buffer which can be assigned to it. At least one such job was not assigned before (since at most $m-1$ jobs from the buffer have been assigned), so we can assign it to i. Applying this procedure for one machine at a time, we will get an allocation of one job for each machine in S and such a job has size at least $\text{OPT}/(2m)$ as we required. □

After dealing with the set S, we continue to allocate one job to each machine not in S. We apply the following rule. For each machine $i \notin S$ we allocate to i the largest job in the buffer, which can be run on i, and which was not allocated before. We again assign a job to one machine at a time. We can always allocate a job from the buffer to i because initially there are at least m jobs which can be processed by i, and every machine is allocated exactly one job from the buffer.

Lemma 21. *If machine $i \notin S$ is allocated a job j, then this job is not smaller than any job which can be processed on machine i and which is either*

- *still in the buffer after each machine received a job out of the buffer, or*
- *allocated by the list scheduling algorithm before the input terminated.*

Proof. Job j is larger than any job which is left in the buffer after every machine was allocated one job out of those left in the buffer (taking into account only the jobs which can be processed by machine i). The jobs assigned greedily before the final step are no larger than j, since the buffer keeps the m largest jobs which i can receive. □

The jobs remaining in buffer, after each machine received a job out of the buffer, are assigned one by one, so that each job is assigned greedily to the least loaded machine that can process it. We say that a job j is small if it was allocated greedily (either at the earlier stage or after the input ends). Other jobs are called large. Therefore, the algorithm has allocated exactly one large job to each machine, and if OPT has assigned a job of size at least $\text{OPT}/(2m)$ to machine i, then the large job of i is of size at least $\text{OPT}/(2m)$.

Theorem 22. *Every machine is allocated a load of at least* $\text{OPT}/(2m)$.

Proof. A machine $i \in S$ is allocated a large job of size at least $\text{OPT}/(2m)$ as we discussed above. Hence, it suffices to consider a machine $i \notin S$ whose large job is of size less than $\text{OPT}/(2m)$.

Fix such a machine i. For every machine $j \neq i$ we denote by C_j the set of jobs which OPT assigns to i and the algorithm assigns to j. Note that C_j may contain a large job, but in this case the large job is of size at most $\text{OPT}/(2m)$ (as otherwise $i \in S$ which is a contradiction).

We consider the total size of small jobs in C_j. Denote by x the last small job from the set C_j assigned to j. Note that by the greedy fashion in which jobs are assigned we conclude that if we discard x from C_j, the total size of remaining small jobs in C_j is at most the total size of small jobs assigned to i (as otherwise the algorithm would not assign x to j). Recall that the large job of i is at least as large as x, and hence we conclude that the total size of small jobs of C_j is at most the total size of jobs assigned to machine i. Summing up over all j we conclude that the total size of small jobs which OPT assigns to machine i and the algorithm assigns to other machines is at most $m-1$ times the load of machine i in the solution returned by the algorithm. The total size of large jobs which OPT assigns to machine i is at most $\text{OPT}/2$ (as each such job has size less than $\text{OPT}/(2m)$). Hence, the total size of small jobs which OPT assigns to machine i is at least $\text{OPT}/2$. Therefore, the algorithm assigns at least $\text{OPT}/(2m)$ to machine i, and the claim follows. □

Note that if there is a restriction that the algorithm must run in polynomial time, then it is possible to apply an approximation algorithm as follows. Let OPT' be the profit of an optimal solution to the problem of adding the jobs stored in the buffer to the assignment. We have showed $\text{OPT}' \geq \frac{\text{OPT}}{2m}$. Consider the instance in which every job j, which is already assigned at the termination of the input to some machine i, is restricted not to its original set of machines M_j but to $\{i\}$. Every job j remaining in the buffer is restricted to its set M_j. This is an offline instance of the problem, and therefore, a C-approximation algorithm would find an assignment of jobs to machines with a profit at least $\frac{\text{OPT}'}{C}$. This results in an assignment augmented by the jobs remaining in the buffer, having a profit of at least $\frac{\text{OPT}}{2mC}$. Using the approximation algorithm of Bansal and Sviridenko [7] for the case of restricted assignment, for which $C = \Theta(\frac{\log\log m}{\log\log\log m})$, we can get a slightly worse competitive ratio of $O(\frac{m\log\log m}{\log\log\log m})$ using a polynomial time algorithm.

5 Tight results for two machines

In this section we consider the case of two machines. We design algorithms which use a reordering buffer of size 1, which is the minimum possible size of a buffer which allows to reduce the general results for online algorithms. Our algorithms are optimal in two ways. First, they have the best possible competitive ratio. Second, they use the minimum size buffer which allows to achieve this bound, i.e., a buffer of size 1. We start this study by retrospective point of view on Algorithm IMITATE for the case of two machines. This discussion will lead us to the properties of the algorithm for two related machines which is best possible for any value of the speed ratio (also giving a best possible result for identical machines). We will conclude this section by presenting a best possible result for the restricted assignment model.

5.1 Algorithm IMITATE - Retrospective

One possible improvement of Algorithm IMITATE would be to change the phantom speeds of the machines. This could only (possibly) slightly reduce the competitive ratio for small m.

Consider the following two examples for $m = 2$, and a phantom speed of $s \leq 1$ for the second machine (and a buffer of size 1). Let $q = \frac{1}{s}$. The first input contains three jobs of sizes $q(q+1) = q^2 + q$, $(q-1)(q+1) = q^2 - 1$ and $q + 1$. Note that the jobs arrive sorted by decreasing size. The first job is stored in the buffer and it is the final job of the second machine. The second job is assigned first (to the first machine). For the third job, we have $(q^2-1)+(q+1) = q(q+1)$, so this job can be assigned to the second machine. This gives ALG $= q^2 - 1$, while OPT $= q^2 + q$, which gives a ratio of $\frac{q}{q-1}$.

The second input consists of very small jobs of a total size of $q + 1$. The buffer only contains one job, and the rest of the input is assigned so that approximately a total size of q is assigned to the first machine and a total size of 1 is assigned to the second machine. Since OPT $= \frac{q+1}{2}$, we get a ratio of $\frac{q+1}{2}$.

The value of q which minimizes $\max\{\frac{q+1}{2}, \frac{q}{q-1}\}$ is $q = \sqrt{2} + 1 \approx 2.4142$, which gives a competitive ratio of at least 1.7071. Applying the analysis of Section 2.2 for two machines and a general value of s results in exactly this competitive ratio, while the lower bound implied by Theorem 1 is $H_2 = \frac{3}{2}$.

We will present a different algorithm, which achieves this optimal competitive ratio of H_2 for two identical machines. This algorithm is a special case of the algorithm of Section 5.2. The following theorem follows from Theorem 27.

Theorem 23. *There is an algorithm with a competitive ratio of $\frac{3}{2}$ for two identical machines which uses a buffer of size* 1.

5.2 An optimal algorithm for two related machines

The main drawback of the algorithm of Section 3.2 is that it ignores the contents of the buffer. In some cases the jobs in the buffer are relatively small so that this does not matter, since larger jobs can possibly arrive later. However, if the buffer already contains large jobs compared to the current load of the machines, the algorithm can use this information. In this case it can allow the machines to become more imbalanced if this can be evened out using the job in the buffer.

Let the machines be denoted by M_1 (slow machine, speed 1) and M_2 (fast machine, speed $s \geq 1$). Note that we use this notation even if $s = 1$.

Time t is the time when the t-th job assignment is about to take place. The total work assigned to M_i at time t (before assigning a job, i.e., after $t-1$ jobs were assigned) is denoted by w_i^t; we have $w_1^1 = w_2^1 = 0$.

Let the size of the job to be assigned at time t be x^t, and the other size (of the job in the buffer) be $y^t \geq x^t$. We slightly abuse notation and let x^t and y^t denote both the jobs and their sizes. At all times when we have two jobs, do the following.

$$\begin{cases} \text{if } \dfrac{w_2^t + y^t}{s} \geq \dfrac{w_1^t + x^t}{s+1} & \text{assign } x^t \text{ to } M_1 \\ \text{else} & \text{assign } x^t \text{ to } M_2 \end{cases}$$

The algorithm tries to assign x^t to M_1 as often as possible. In particular, the first assigned job is assigned to M_1. Finally, assign the last job remaining in the buffer to M_2. The reason for this last job assignment is that the load on M_2 will always be smaller than the load of M_1. However, we do not need this property explicitly in our proof and we will not show it.

Lemma 24. *If job x^t is assigned to M_2 at time $t \geq 2$, then*

$$y^t < \frac{s}{s+1}(w_1^t + x^t) - w_2^t \qquad \textit{and} \qquad x^t \leq s \cdot w_1^t.$$

Proof. If a job is assigned to M_2 at time t, then we have $s(w_1^t + x^t) > (s+1)(w_2^t + y^t)$. We find $y^t < \frac{s}{s+1}(w_1^t + x^t) - w_2^t$. The bound on x^t now follows directly from $x^t \leq y^t$ and $w_2^t \geq 0$. □

Lemma 25. *For all $t \geq 1$, we have $w_2^t + y^t \geq \frac{s}{s+1} w_1^t$.*

Proof. We use induction. The claim holds before any jobs are assigned. If a job is assigned to M_1, then $w_1^{t+1} = w_1^t + x^t$, whereas $w_2^{t+1} = w_2^t$. The job was assigned to M_1 and therefore $w_2^t + y^t \geq \frac{s}{s+1} w_1^{t+1}$. Note that $y^{t+1} \geq y^t$ for all $t \geq 1$, since the job in the buffer is replaced only if a larger job arrives. Hence, $w_2^{t+1} + y^{t+1} \geq w_2^t + y^t \geq \frac{s}{s+1} w_1^{t+1}$.

If a job is assigned to M_2, then using the inductive hypothesis, the claim holds since $w_1^{t+1} = w_1^t$, $w_2^{t+1} > w_2^t$ and $y^{t+1} \geq y^t$. □

Lemma 26. *If $s \cdot w_1^t < (s+1) w_2^t$, and the last job assigned to M_2 was assigned at time $t_0 < t$, then the job y^{t_0} is still in the buffer up to time $t-1$, i.e., it is unassigned before the assignment of time t, and $s(w_1^t + x^{t_0}) > (s+1) w_2^t$.*

Proof. We have $w_2^t = w_2^{t_0+1} = w_2^{t_0} + x^{t_0} \leq w_2^{t_0} + y^{t_0} < \frac{s}{s+1}(w_1^{t_0} + x^{t_0}) < \frac{s}{s+1}(w_1^t + x^{t_0})$. By the assumption $s \cdot w_1^t < (s+1) w_2^t$, we get $w_1^{t_0} + x^{t_0} > \frac{s+1}{s} w_2^t > w_1^t$. So between time t_0 and time $t-1$, a total size of jobs strictly lower than x^{t_0} (and therefore less than y^{t_0}) was assigned to M_1, and nothing was assigned to M_2 after time t_0. This shows that job y^{t_0} is still in the buffer after the assignment at time $t-1$. □

The following theorem guarantees competitive ratios of $3/2$ and 2 for the identical and related machines model respectively.

Theorem 27. *The competitive ratio of this algorithm is at most $(2s+1)/(s+1)$.*

Proof. Let t be the time at which the penultimate job assignment takes place, that is, the assignment of last job which is not the job remaining in the buffer. In this proof, we abbreviate w_1^t, w_2^t, x^t, y^t by w_1, w_2, x, y.

Case 1: the penultimate job (x) is assigned to M_2. Job y is assigned to M_2 as well. If after the assignment of y, M_1 has the lowest load, then the profit of the returned solution is w_1 and the optimal profit is at most

$$\frac{w_1+w_2+x+y}{s+1} \leq \frac{w_1+w_2+x+\frac{s}{s+1}(w_1+x)-w_2}{s+1} \leq \frac{(w_1+x)\frac{2s+1}{s+1}}{s+1} \leq \frac{w_1(2s+1)}{s+1},$$

using that $y < \frac{s}{s+1}(w_1+x)-w_2$ (first inequality) and $x \leq sw_1$ (last inequality) by Lemma 24.

If M_1 has the highest load, the profit of the algorithm is $L_2 = (w_2+x+y)/s$, so by Lemma 25, the competitive ratio is at most

$$\frac{w_1+w_2+x+y}{\frac{s+1}{s}(w_2+x+y)} = \frac{s}{s+1} + \frac{sw_1}{(s+1)(w_2+x+y)} \leq \frac{2s+1}{s+1}\,.$$

Case 2: the penultimate job is assigned to M_1. The last job is assigned to M_2, which then has a final load of $L_2 = (w_2+y)/s \geq (w_1+x)/(s+1)$ by the assignment condition at time t. The optimal profit is at most $(w_1+x+w_2+y)/(s+1)$.

Case 2a: $w_2+y \leq s(w_1+x)$. Then the profit of our algorithm is $(w_2+y)/s$, and the competitive ratio is at most

$$\frac{w_1+x+w_2+y}{\frac{s+1}{s}(w_2+y)} = \frac{s}{s+1} + \frac{s(w_1+x)}{(s+1)(w_2+y)} \leq \frac{2s+1}{s+1}\,.$$

Case 2b: $s(w_1+x) < w_2+y \leq 2s(w_1+x)$. The profit of our algorithm is w_1+x, and the competitive ratio is at most $(w_1+x+w_2+y)/((s+1)(w_1+x)) \leq \frac{2s+1}{s+1}$, using $w_2+y \leq 2s(w_1+x)$.

Case 2c: $w_2+y > 2s(w_1+x)$ and $w_1+x \geq \frac{s+1}{s}w_2$. Using the last two conditions, we have $y \geq s(w_1+w_2+x)$. In this case, an optimal assignment consists of assigning y to the fast machine, and assigning all other jobs to the slow machine. The optimal profit is at most w_1+w_2+x, giving a competitive ratio of at most

$$\frac{w_1+w_2+x}{w_1+x} \leq 1 + \frac{w_2}{w_1+x} \leq \frac{2s+1}{s+1}\,. \tag{2}$$

Case 2d: $w_2+y > 2s(w_1+x)$ and $w_1+x < \frac{s+1}{s}w_2$. At time $t+1$, we have $w_1^{t+1} = w_1+x$ and $w_2^{t+1} = w_2$. Therefore, we can apply Lemma 26 for time $t+1$. Let t_0 be the last time a job was assigned to M_2. If $x = y^{t_0}$, we find $w_1+x = w_1+y^{t_0} \geq w_1+x^{t_0} > \frac{s+1}{s}w_2$ by Lemma 26, which leads to a contradiction. Therefore $y = y^{t_0}$. Then by our algorithm, at time t_0, $w_2^{t_0}+y < \frac{s}{s+1}(w_1^{t_0}+x^{t_0})$ and on the other hand $w_2^{t_0}+x^{t_0}+y = w_2+y > 2s(w_1+x)$. This implies

$$\begin{aligned} x^{t_0} &> 2s(w_1+x) - \frac{s}{s+1}(w_1^{t_0}+x^{t_0}) \\ \Rightarrow \frac{2s+1}{s+1}x^{t_0} &> 2s(w_1+x) - \frac{s}{s+1}w_1^{t_0} \\ \Rightarrow x^{t_0} &> \frac{2s(s+1)}{2s+1}(w_1+x) - \frac{s}{2s+1}w_1^{t_0} > s(w_1^{t_0}+x) > sw_1^{t_0} \end{aligned}$$

which contradicts the bound $x^{t_0} \leq sw_1^{t_0}$ from Lemma 24. □

5.3 A matching lower bound for two related machines

In this section we show that for every speed ratio s, the algorithm of the previous section is optimal, even if we are given a larger buffer. That is, we will prove the following theorem.

Theorem 28. *For any buffer size K, no algorithm can have a competitive ratio smaller than $(2s+1)/(s+1)$ for two related machines with speed ratio $s \geq 1$.*

Proof. Assume that machine 1 has speed 1 and the speed of machine 2 is s for $s \geq 1$. We fix an $\mathcal{R}$-competitive algorithm which uses a buffer of size K. Let $N \in \mathbb{N}$ be a large number such that $N >> K$, and let $\varepsilon = \frac{1}{N}$. The input sequence starts with N small jobs of size ε. Denote by w_1 the total work of the jobs assigned to machine 1 by the algorithm, and by w_2 the total work of jobs assigned to machine 2 by the algorithm, after the last job of size ε has arrived. At the end of this part of the input $\text{OPT} \geq \frac{1}{s+1} - \varepsilon$ (by assigning the jobs to the two machines in an almost balanced way) and since the algorithm is $\mathcal{R}$-competitive the following inequalities hold:

$$1 - K\varepsilon \leq w_1 + w_2 \leq 1, \tag{3}$$

$$w_1 + K\varepsilon \geq \frac{1}{\mathcal{R}(s+1)} - \frac{1}{\mathcal{R}}\varepsilon, \tag{4}$$

$$w_2 + K\varepsilon \geq \frac{s}{\mathcal{R}(s+1)} - \frac{s}{\mathcal{R}}\varepsilon, \tag{5}$$

where (3) holds due to the total size of small jobs, and (4) and (5) hold in order to maintain a competitive ratio of at most $\mathcal{R}$, and since stopping the input at this time would result in $L_1 = W_1 \leq w_1 + K\varepsilon$ and $L_2 = \frac{W_2}{s} \leq \frac{w_2 + K\varepsilon}{s}$.

At this point the input either stops or continues with one additional job of size s. If this additional job arrives, then $\text{OPT} = 1$ whereas the profit of the algorithm is at most $\max\{w_1 + K\varepsilon, (w_2 + K\varepsilon)/s\}$, since the new job can be assigned to the least loaded machine, while the other machine receives no additional jobs.

We give an upper bound for $\max\{w_1, w_2/s\}$. From (3) and (5), we find $w_1 \leq 1 - w_2 \leq 1 - \frac{s}{\mathcal{R}(s+1)} + (K + \frac{s}{\mathcal{R}})\varepsilon$. Similarly, from (3) and (4), we conclude that $w_2 \leq 1 - w_1 \leq 1 - \frac{1}{\mathcal{R}(s+1)} + (K + \frac{1}{\mathcal{R}})\varepsilon$. Hence $\max\{w_1, w_2/s\} \leq \max\{1 - \frac{s}{\mathcal{R}(s+1)} + (K + \frac{s}{\mathcal{R}})\varepsilon, (1 - \frac{1}{\mathcal{R}(s+1)} + (K + \frac{1}{\mathcal{R}})\varepsilon)/s\}$.

We next show that the first term of this maximum is the largest:

$$\begin{aligned} s - \frac{s^2}{\mathcal{R}(s+1)} \geq 1 - \frac{1}{\mathcal{R}(s+1)} &\Leftrightarrow \mathcal{R}s(s+1) - s^2 \geq \mathcal{R}(s+1) - 1 \\ &\Leftrightarrow (\mathcal{R}-1)s^2 + \mathcal{R}s \geq \mathcal{R}s + \mathcal{R} - 1 \end{aligned}$$

which holds because $\mathcal{R} \geq 1$ and $s \geq 1$. Thus, the profit of the solution returned by the algorithm is at most $1 - \frac{s}{\mathcal{R}(s+1)} + (K + \frac{s}{\mathcal{R}})\varepsilon$, whereas $\text{OPT} = 1$. Therefore, since ε can be arbitrary small compared to $K + s/\mathcal{R} \leq K + s$, the following inequality holds:

$$1 - \frac{s}{\mathcal{R}(s+1)} \geq \frac{1}{\mathcal{R}}\,.$$

This is equivalent to $\mathcal{R} \geq (2s+1)/(s+1)$. □

5.4 An optimal algorithm for two machines in the restricted assignment model

The algorithm keeps the largest job which has the processing set $\{1,2\}$ in the buffer. If any job with the set $\{1\}$ or the set $\{2\}$ arrives, it is assigned right away to its unique possible machine. If a job of the processing set $\{1,2\}$ arrives and the buffer is empty, then it is stored in the buffer. Otherwise, let $y \geq x$ be the sizes of the two jobs with the processing set $\{1,2\}$, which were not assigned yet. The job of size y is stored in the buffer, while the job of size x is assigned to a machine of minimum current load.

When the input stops, if there is a job in the buffer, it is assigned to a machine of minimum load as well.

Theorem 29. *The algorithm has a competitive ratio of* 2.

Proof. If the input contains at most one job with the processing set $\{1,2\}$, then the algorithm is optimal. Assume that there are at least two jobs with the processing set $\{1,2\}$, thus the algorithm was at least once confronted with the situation that it has two jobs with this processing set unassigned.

Let X and Y denote the sizes of these two jobs at the last time that this happens. Note that the last job remaining in the buffer is the largest job with this processing set. This is the job of size Y.

Let ℓ_i denote the load of machine i before the last job is assigned. Assume without loss of generality that $\ell_1 \leq \ell_2$, and hence the job of size Y is assigned to machine 1. Then since there is a machine in an optimal solution that does not contain the job of size Y then $\text{OPT} \leq T - Y$. Thus $\ell_2 \geq \frac{T-Y}{2} \geq \frac{\text{OPT}}{2}$. If $\ell_1 + Y \geq \ell_2$, then $\ell_1 + Y \geq \frac{\text{OPT}}{2}$.

Assume therefore $\ell_1 + Y < \ell_2$. If machine 2 never received a job of the processing set $\{1,2\}$, then the algorithm is optimal. Otherwise, consider the last job of this processing set that it ever received, and denote its size by Z. Let ℓ'_i denote the load of machine i just after this assignment of Z. By the rules of the algorithm, $\ell'_1 + Z \geq \ell'_2$. We have $\text{OPT} \leq \ell_1 + \ell'_2$, since machine 1 cannot receive any of the jobs assigned to machine 2 after the job of size Z was assigned by the algorithm.

We get $\text{OPT} \leq \ell_1 + \ell'_2 \leq \ell_1 + \ell'_1 + Z \leq 2\ell_1 + Y$ (using that the size of the job in the buffer is monotonically non-decreasing over time). Thus $\ell_1 + Y \geq \frac{2\ell_1+Y}{2} \geq \frac{\text{OPT}}{2}$. □

References

[1] S. Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.

[2] A. Asadpour, U. Feige, and A. Saberi. Santa Claus meets hypergraph matchings. In *Proc. 11th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 10–20, 2008.

[3] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM J. Comput.*, 39(7):2970–2989, 2010.

[4] 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(3):486–504, 1997.

[5] Y. Azar and L. Epstein. On-line machine covering. In *Proc. 5th European Symp. on Algorithms (ESA)*, volume 1284 of *LNCS*, pages 23–36. Springer, 1997.

[6] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995.

[7] N. Bansal and M. Sviridenko. The Santa Claus problem. In *Proc. 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40. ACM, 2006.

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

[9] S.-Y. Cai. Semi-online machine covering. *Asia-Pacific J. of Oper. Res.*, 24(3):373–382, 2007.

[10] O. Chassid and L. Epstein. The hierarchical model for load balancing on two machines. *Journal of Combinatorial Optimization*, 15(4):305–314, 2008.

[11] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18:127–131, 1995.

[12] J. Csirik, H. Kellerer, and G. Woeginger. The exact LPT-bound for maximizing the minimum completion time. *Operations Research Letters*, 11:281–287, 1992.

[13] B. L. Deuermeyer, D. K. Friesen, and M. A. Langston. Scheduling to maximize the minimum processor finish time in a multiprocessor system. *SIAM Journal on Discrete Mathematics*, 3(2):190–196, 1982.

[14] Gy. Dósa and L. Epstein. Online scheduling with a buffer on related machines. *Journal of Combinatorial Optimization*, 20(2):161–179, 2010.

[15] Gy. Dósa and L. Epstein. Preemptive online scheduling with reordering. *SIAM Journal on Discrete Mathematics*, 25(1):21–49, 2011.

[16] T. Ebenlendr, J. Noga, J. Sgall, and G. J. Woeginger. A note on semi-online machine covering. In *Proc. 3rd International Workshop in Approximation and Online Algorithms (WAOA)*, pages 110–118, 2005.

[17] M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *Proc. 48th Symp. Foundations of Computer Science (FOCS)*, pages 603–612, 2008.

[18] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.

[19] U. Feige. On allocations that maximize fairness. In *Proc. 19th Symp. on Discrete Algorithms (SODA)*, pages 287–293, 2008.

[20] D. K. Friesen and B. L. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Research*, 6(1):74–87, 1981.

[21] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proc 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–565, 2000.

[22] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell Sys. Tech. J.*, 45:1563–1581, 1966.

[23] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovasz local lemma. In *Proc. 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 397–406, 2010.

[24] Y. Jiang, Z. Tan, and Y. He. Preemptive machine covering on parallel machines. *Journal of Combinatorial Optimization*, 10(4):345–363, 2005.

[25] H. Kellerer, V. Kotov, M. G. Speranza, and Zs. Tuza. Semi online algorithms for the partition problem. *Operations Research Letters*, 21:235–242, 1997.

[26] Z. Tan and Y. Wu. Optimal semi-online algorithms for machine covering. *Theoretical Computer Science*, 372(1):69–80, 2007.

[27] G. J. Woeginger. A polynomial time approximation scheme for maximizing the minimum machine completion time. *Operations Research Letters*, 20(4):149–154, 1997.

[28] G. Zhang. A simple semi on-line algorithm for $P2//C_{\max}$ with a buffer. *Information Processing Letters*, 61:145–148, 1997.