Preemptive online scheduling with reordering

György Dósa* Leah Epstein†

Abstract

We consider online preemptive scheduling of jobs, arriving one by one, on m identical parallel machines. A buffer of a fixed size $K > 0$, which assists in partial reordering of the input, is available to be used for the storage of at most K unscheduled jobs. We study the effect of using a fixed sized buffer (of an arbitrary size) on the supremum competitive ratio over all numbers of machines (the overall competitive ratio), as well as the effect on the competitive ratio as a function of m.

We find a tight bound on the competitive ratio for any m. This bound is $\frac{4}{3}$ for even values of m and slightly lower for odd values of m. We show that a buffer of size $\Theta(m)$ is sufficient to achieve this bound, but using $K = o(m)$ does not reduce the best overall competitive ratio which is known for the case without reordering, $\frac{e}{e-1}$. We further consider the semi-online variant where jobs arrive sorted by non-increasing processing time requirements. In this case it turns out to be possible to achieve a competitive ratio of 1. In addition, we find tight bounds as a function of the buffer size and the number of machines for this semi-online variant. Related results for non-preemptive scheduling were recently obtained by Englert, Özmen and Westermann.

1 Introduction

Scheduling of jobs arriving one by one (or *over list*) is a basic model in online scheduling [17]. The system consists of a set of m identical machines that can process a sequence of arriving jobs. Each job j, which has a processing time p_j associated with it (also called *size*), needs to be assigned upon arrival. The completion time, or load, of a machine is the total time needed to process the jobs assigned to it, including idle time in which the machine is waiting for a job to be executed (if idle time exists). The goal is to minimize the maximum completion time of any machine, also known as the *makespan*.

We consider online and semi-online preemptive scheduling of jobs. An arriving job can be split into parts, which need to be assigned to non-overlapping time slots, possibly on different machines. Idle time is allowed, and each machine can process at most one job at each time. In the online scenario, a job must be treated before the next job is revealed. For an algorithm $\mathcal{A}$, we denote its cost by $\mathcal{A}$ as well. An optimal offline algorithm that knows the complete sequence of jobs in advance, as well as its cost, are denoted by OPT. In this paper we measure the performance quality of algorithms using the (absolute) competitive ratio, which is the most common measure for the performance evaluation of online algorithms. The competitive ratio of $\mathcal{A}$ is the infimum $\mathcal{R}$ such that for any input, $\mathcal{A} \leq \mathcal{R} \cdot \text{OPT}$.

We consider a model where a reordering buffer, of a fixed size $K > 0$, is available. This buffer can store up to K unassigned jobs and thus assists in partial reordering of the input. Upon the arrival of a job, it is possible to either assign it completely to machines and time slots, or otherwise it is possible to store it in

*Department of Mathematics, University of Pannonia, Veszprém, Hungary, dosagy@almos.vein.hu.

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

the buffer rather than assigning it. If the buffer already contains K jobs, at least one of these jobs must be assigned to the machines in order to make room for the new job, or else the new job must be assigned.

Non-preemptive scheduling (i.e., the case where a job cannot be split into parts and it must be processed continuously on one machine), with a reordering buffer, was previously studied in several papers [3, 6, 12, 13, 19]. The main research question in these papers was to find the effect of using a reordering buffer on the competitive ratio, that is, finding the lowest competitive ratio which can be achieved if the online algorithm is supplied with a buffer, and whether this competitive ratio is achievable only in the limit, or whether there exists a size of a buffer which allows to achieve this bound. This competitive ratio can then be compared to the best possible competitive ratio which can be achieved without a buffer. Clearly, an offline algorithm can be seen as an algorithm which uses an unbounded buffer. Limiting the online algorithm to a fixed sized buffer still means in most cases that the algorithm cannot perform as well as an optimal offline algorithm. Consequently, the competitive ratio for every value of m is of interest, as well as the *overall* competitive ratio, which is the supremum competitive ratio over all values of m.

In all (non-preemptive) variants studied in the past, a finite length buffer already allows to achieve the best competitive ratio. In particular, for two identical machines, a buffer of size 1 is sufficient, as was proved by Kellerer et al. [12] and independently by Zhang [19]. For m identical machines, Englert, Özmen and Westermann [6] showed that a buffer of size $O(m)$ is sufficient. For the more general case of uniformly related machines, where machines may have different speeds, it was shown [3] that for two machines, a buffer of size 2 allows to achieve the best competitive ratio. In fact, for some speed ratios between the two machines, a buffer of size 1 is sufficient, while for some other speed ratios, a buffer of size 1 provably does not allow to achieve the best bound. Note that it was shown by [6] that a buffer of size $m-1$ reduces the competitive ratio for uniformly related machines below the lower bound of the case without reordering. In this paper we answer analogous questions for preemptive scheduling.

Preemptive online scheduling without reordering was studied by Chen, van Vliet, and Woeginger [2] (see also [1, 14, 16]). They designed an algorithm of the best possible competitive ratio for any number of machines m. This competitive ratio is a monotonically increasing function of m, $\frac{m^m}{m^m-(m-1)^m}$, which implies an overall competitive ratio of $\frac{e}{e-1} \approx 1.58$. A number of papers generalized this result for uniformly related machines [4, 7, 9, 10, 18].

We study an additional variant where it is known in advance that jobs arrive sorted by size, in a non-increasing order. This common semi-online variant of preemptive semi-online scheduling was analyzed by Seiden, Sgall and Woeginger for identical machines [15]. The overall tight bound on the competitive ratio shown by [15] is $\frac{1+\sqrt{3}}{2} \approx 1.366$. Semi-online preemptive scheduling on uniformly related machines was considered in [5, 8].

Our results. We give a fairly complete study of the case of m identical machines. We find a tight bound on the competitive ratio for general inputs for any number of machines, m. We show that a buffer of size $\lceil \frac{m-2}{2} \rceil$ is sufficient to achieve this bound. In fact, reordering via the usage of a buffer allows to reduce the tight competitive ratio to $\frac{4}{3}$ for even m, and to $\frac{4m^2}{3m^2+1} < \frac{4}{3}$ for odd m, whereas $K = o(m)$ does not reduce the overall competitive ratio, which remains $\frac{e}{e-1}$, as in [2]. Surprisingly, we find that the best competitive ratio, as a function of m, is not monotone, and the overall competitive ratio is $\frac{4}{3}$. Note that this value is the tight competitive ratio for $m = 2$, the only case where a buffer is not necessary. This is different from the non-preemptive problem, where for $m = 2$ the usage of a buffer reduces the best competitive ratio from $\frac{3}{2}$ to $\frac{4}{3}$ [6, 12, 19]. As a motivation for using this specific size of buffer, $\lceil \frac{m-2}{2} \rceil$, we show that for $m = 6$ machines, where our general result uses a buffer of size 2, a buffer of size 1 leads to a larger competitive ratio. We show tight bounds of $\frac{19}{14} \approx 1.35714$ on the competitive ratio for this case.

We further consider the semi-online variant where jobs arrive sorted by non-increasing processing time requirements. In this case we show that a buffer of size $m-1$ is sufficient to achieve a competitive ratio of 1, whereas a buffer of size $m-2$ is not sufficient. That is, the combination of a reordering buffer with jobs arriving in a sorted order is as good as receiving the entire set of jobs in advance. We show that the tight bound for this last case, where the buffer has a size of $m-2$, is $1+\frac{1}{m^2+m-1}$. Finally, we find tight bounds for all buffer sizes $1, 2, \ldots, m-3$, in these cases, the competitive ratio is $\max_{1\leq\mu\leq m-K-1} \frac{2m(m+\mu)}{2m^2+2K\mu+\mu^2+\mu}$ (where μ takes integer values, and K is the size of the buffer).

Our algorithms are based on a unified approach where largest jobs are kept in the buffer, and the created schedule is as imbalanced as possible to keep the less loaded machines free to receive new jobs. Using the master algorithm with different parameters results in distinct algorithms for the different cases. The lower bounds are based on a unified approach where one sequence is used, and all the considered inputs are subsequences of this sequence. This approach is general and allows the usage of several types of inputs for the different cases. These general approaches were used in the past for preemptive scheduling problems, but some important adaptations were required to be able to deal with the existence of a buffer.

2 Algorithms

2.1 The master algorithm

All our algorithms have a common structure which is explained in this section. These algorithms avoid the usage of idle time, and try to assign as much work as possible to the more loaded machines, while keeping K jobs is the buffer (or a smaller number of jobs, if the input sequence has terminated and the jobs left in the buffer are being assigned). Let n denote the number of jobs (which is not known to the online algorithm in advance).

The algorithm can be used for any $K \leq m$ (we will see later that larger values of K are not useful). In addition to the number of machines m and the size of the buffer K, the master algorithm uses a parameter $\mathcal{R}$, which is the required competitive ratio. In the case $K = 0$, the algorithm reduces to that used by [2].

We note that in the case of identical machines, idle time gives no advantage. The schedule obtained at some time can be fully expressed as m numbers, where the i-th number is the total length of intervals in which i machines are active. In order to assign a job, it is necessary to decide how to partition it into m parts, where the j-th part indicates the length of time in which it is assigned to run in parallel to j other jobs. We next explain using induction that how to keep a schedule without idle time. The initial empty schedule clearly satisfies this property. Assume that the schedule at a given time does not contain idle time. Assume that a part of size c_j is supposed to be assigned to run in parallel to j other jobs (for $0 \leq j \leq m-1$). In order to assign the next job without idle time as well, sort the machines by non-decreasing load. A part of the job which is supposed to run in parallel to j other jobs is assigned at the earliest possible time to the $m-j$-th machine in the sorted order. If it is possible to assign a part of size c_j to run in parallel to exactly j other jobs, then the time at which the $(m-j)$-th machine is not active but the $(m-j+1)$-th machine is active is at least of length c_j. Thus the new job is assigned in a valid way, without introducing idle time.

After initialization, and as long as at most K jobs have arrived, all jobs are stored in the buffer. If the input consists of at most K jobs, then each job is assigned to a separate machine, to run on this machine non-preemptively, starting from time 0, which results in an optimal solution. Otherwise, after $K+1$ jobs have arrived, and as long as jobs keep arriving, the algorithm keeps the K largest jobs seen so far in the buffer. After jobs stop arriving, the algorithm keeps removing a smallest job from the buffer and assigning

it using the same algorithm, until all jobs have been assigned. The loads after the assignment of i jobs are denoted by $L^i_1 \leq L^i_2 \leq \ldots \leq L^i_m$. Note that these are the loads after the arrival of $\min\{K+i, n\}$ jobs. Specifically, if $K+i \leq n$, then these are the loads after the arrival of $K+i$ jobs, out of which K are stored in the buffer. Otherwise, if $K+i > n$, these are the loads after all jobs have arrived and $n-i$ of them are in the buffer. We also use $Q_i = \sum_{k=1}^{m} L^i_k$, i.e., Q_i is the total size of the assigned jobs after i jobs have been assigned. This amount includes all scheduled jobs in both of the described cases. Let OPT_i denote the cost of OPT at this time, after i jobs have been assigned. This value takes into account the prefix of $\min\{K+i, n\}$ jobs, i.e., all jobs that arrived by this time.

We assume that each machine has an index in $\{1, 2, \ldots, m\}$. We first explain how to assign jobs to machines in a way that the sorted order of machines does not change, that is, the load L^i_g is always the load of machine g. Later we show how to modify the algorithm so that it uses at most one preemption per job, the sequence of loads remains as in the first variant of the algorithm, but the sorted order of machines changes frequently (i.e., L^i_g is the g-th load in the sorted order of loads, but it does not necessarily belong to machine g).

We say that the buffer is *full*, if it contains exactly K jobs. Since the algorithm for the case $n \leq K$ is completely defined above, we assume in what follows that $n > K$, i.e., there is at least one case where a job needs to be assigned while the buffer is full, and we describe the assignment of jobs for the cases where the buffer is full, or was full at some previous time. This last option means that jobs no longer arrive, and the jobs which remained in the buffer need to be assigned.

To assign a job, non-overlapping slots are reserved on the machines, and the job is assigned into these slots, one by one, by a decreasing order of indices of machines, until the job is completely assigned, or until all slots are full. In each case for which we use this master algorithm with specific parameters, we will show that the second option never occurs. If one of the used slots is not filled completely, then the earliest part of this slot is used, so that no idle time is created. The slots for the $(i+1)$-th job ever assigned are $[L^i_m, \mathcal{R} \cdot \text{OPT}_{i+1}]$ on machine m, and $[L^i_j, L^i_{j+1}]$ on each other machine, $1 \leq j < m$. The slots are clearly non-overlapping. Note that some of the slots may be empty, if there are at least two consecutive identical loads.

In order to prove upper bounds, we use two lower bounds on the cost of an optimal solution, which are the average load, implied by the sum of all jobs (including those in the buffer), and the maximum size of any job. The algorithm, however, needs to compute the exact value of OPT_i. We exploit the property that OPT_i is in fact equal to the maximum of these two bounds [11]. Therefore, calculating the slot on machine m for the assignment of a job can be done in constant time.

Clearly, as long as every job is assigned successfully, the competitive ratio of the master algorithm is at most $\mathcal{R}$. Therefore, in each case we consider, it is necessary to show that the algorithm never fails. In most cases we derive a set of invariants which are proved by induction and allow to prove this property. In one case we use a small number of invariants and a direct proof for the most important invariant rather than induction. In the latter case, the usage of a similar structure of proof to the former case, i.e., additional similar invariants together with induction, does not seem to be helpful. Since the algorithm is a generalization of the algorithm of [2], the main technical contribution here is the design of the correct set of invariants, or the design of a more direct proof. This is done for each case separately, since the exact value of $\mathcal{R}$ affects the execution of the algorithm and leads to very different schedules in the different cases. Note that for both variants and all values of m, the largest jobs are always the jobs which are kept in the buffer. In the case of general inputs, it is either the case that the new job is stored in the buffer, or that it is the job which is assigned. In the case of non-increasing sequences, the first K jobs are kept in the buffer until the sequence ends, and then they are

assigned in an order which is opposite to the order of their arrival.

In previous work on scheduling with a buffer, in many cases, the largest jobs (or largest job, in the case $K = 1$) were those which are stored in the buffer. Intuitively, this seems to be the correct approach; the algorithm is aware of the exact sizes of the largest jobs and takes them into account in the other scheduling decisions, but it postpones their assignment until the later. Nevertheless, in [3] one of the algorithms of optimal competitive ratio, which uses $K = 1$, has two cases, where in one of the cases the larger available job is assigned while the smaller job is stored in the buffer. We note an interesting difference with the algorithms of [6]. Our algorithm uses the same method of assignment for all jobs, even after no additional jobs arrive. It is possible in fact to use the same algorithm also in the case $n \leq K$ and avoid cases in the definition of the algorithm. However, since this case is very simple, so we prefer the current presentation, to avoid cases in the proof.

Finally, we show how to modify the algorithm to use at most one preemption per job instead of at most $m - 1$ preemptions. Assume that machine j has the j-th load before the assignment of a job. If the job is assigned to slots on machines $j, j + 1, \ldots, m$, where $j < m - 1$, i.e., it was assigned using at least two preemptions, then instead of using the slots on machines $j + 2, \ldots, m$, it is possible to assign all these parts continuously on machine $j + 1$. Due to the definition of the algorithm, the only idle time in the processing of the job (but not in the schedule) may occur between the end of the time slot in which it is assigned to run on machine j and the beginning of the time slot on machine $j + 1$, since the slot on machine j is not necessarily completely occupied. In this case the assignment of the part to machine j remains unchanged. The load of machine $j + 1$ becomes the largest load (i.e., the m-th load), while the load of a machine y (for $j + 2 \leq y \leq m$) becomes the $(y - 1)$-th load. If the slot on machine j is occupied completely, and $j < m$, then it is possible to schedule the job on machine j non-preemptively and without idle time (if $j = m$, then the job is already scheduled non-preemptively). In this case, the resulting load of machine j becomes the m-th load, while the load of machine y (for $j + 1 \leq y \leq m$) becomes the $(y - 1)$-th load. In both cases, the resulting sorted list of loads is the same as before, but the machines having these loads switch places in this list.

2.2 General inputs

The algorithm uses a buffer of size $K = \lfloor \frac{m-1}{2} \rfloor = \lceil \frac{m-2}{2} \rceil$, that is, $K = \frac{m}{2} - 1$ for even values of m, and $K = \frac{m-1}{2}$ for odd values of m. Let $\mathcal{R} = \frac{4}{3}$ if m is even, and $\mathcal{R} = \frac{4m^2}{3m^2+1}$ if m is odd. Thus the bound for odd m is strictly below $\frac{4}{3}$, and achieves it lowest value for $m = 3$, for which the bound equals to $\frac{9}{7} \approx 1.28571$. For $m = 2$ the competitive ratio is $\frac{4}{3}$, but the size of the buffer which we use is zero, thus the result of [2] for $m = 2$ already covers this case. By the results of Section 3, this result is best possible, i.e., the usage of a buffer does not improve the performance in this case.

We define a set of invariants which must hold after every assignment. Invariant j (for $1 \leq j \leq \lfloor \frac{m}{2} \rfloor$) at time i is defined by

$$\sum_{k=1}^{j} L_k^i \leq (\mathcal{R} - 1) \cdot \frac{j \cdot Q_i}{m - j} .$$

The motivation of these invariants comes from a situation where very small jobs of a total size of x arrive first, after which some number $j \leq \lfloor \frac{m}{2} \rfloor$ of identical jobs of size $\frac{x}{m-j}$ arrive. An optimal solution spreads the small jobs over $m - j$ machines, while the best response of the algorithm would be to use the j least loaded machines. The invariants make sure that in such a case the competitive ratio is not exceeded. Note that even though such a construction could be a good candidate for a lower bound proof, the lower bound

proof of Section 3 considers an input with a much smaller number of cases.

We use some further definitions and notations. After $i \geq 0$ jobs were assigned, no matter whether a new job arrives, or if no additional jobs arrive, but the buffer is non-empty, let the sorted list of sizes of jobs, which includes the jobs in the buffer and the new job (if exists) be $Y_0^{i+1} \leq Y_1^{i+1} \leq \ldots \leq Y_{\ell-1}^{i+1}$, where $\ell \leq K+1$. If a new job has just arrived (and thus the buffer is full), and we are about to assign a job, then $\ell = K+1$. We are going to assign the smallest job, thus $Q_{i+1} = Q_i + Y_0^{i+1}$. In addition, as explained in Section 2.1, $\text{OPT}_{i+1} = \max\{Y_{\ell-1}^{i+1}, \frac{1}{m} \cdot (Q_i + \sum_{k=0}^{\ell-1} Y_k^{i+1})\}$, i.e., OPT_{i+1} is the optimal cost for a schedule for all received jobs so far, just before Y_0^{i+1} is scheduled. Using $Y_0^{i+1} \leq Y_k^{i+1}$ for any $1 \leq k \leq \ell-1$, we have $\text{OPT}_{i+1} \geq Y_0^{i+1}$ and $\text{OPT}_{i+1} \geq \frac{1}{m} \cdot (Q_i + jY_0^{i+1})$, which holds for any $1 \leq j \leq \ell$.

Lemma 1 *For a given value of i, if after i jobs were assigned the buffer is full, then the j-th invariant holds for time i and every $1 \leq j \leq \lfloor \frac{m}{2} \rfloor$. If the buffer contains $\ell < K$ jobs at this time, then the j-th invariant holds for time i and every $1 \leq j \leq \ell$.*

As mentioned above, the main technical difficulty lies in finding the correct invariants. The proof of Lemma 1 is technical, and is given in Section 4.1. The proof idea is that if machines $1, \ldots, j$ did not receive any parts of the $i+1$-th assigned job, then there is no change in their loads, i.e., $L_k^i = L_k^{i+1}$, while $Q_{i+1} \geq Q_i$. Otherwise, except for the most loaded machine, the load of each machine that received a part of the job, except for possibly the machine of lowest index that received a part, is exactly the previous load of the next machine, that is, a machine $k < m$ which received a part of the job, but not the last part, satisfies $L_k^{i+1} = L_{k+1}^i$. There are several cases, according to the value of j, and in particular, the cases $j = \lfloor \frac{m}{2} \rfloor$ and $j < \lfloor \frac{m}{2} \rfloor$ are considered separately. In order to prove the invariant for j, for $j < \lfloor \frac{m}{2} \rfloor$, the inductive hypothesis for $j+1$ is used. However, if $j = \lfloor \frac{m}{2} \rfloor$, there is no $j+1$-th invariant, and only the j-th invariant can be used, so the proof is slightly different from the previous case, and in addition, the two cases of even m and odd m are not identical, since the value of $\mathcal{R}$ is used in an early stage of the proof.

Lemma 2 *For every $i+1$ ($i \geq 0$), if the invariants hold for time i ($i < n$), then the algorithm assigns the $(i+1)$-th job successfully.*

Proof. We show that there is enough space for the job, that is, the total size of the slots is no smaller than the size of the job to be assigned. This gives the following condition: $\mathcal{R} \cdot \text{OPT}_{i+1} - L_1^i \geq Y_0^{i+1}$.

Using invariant 1 at time i,

$$L_1^i \leq (\mathcal{R}-1) \cdot \frac{Q_i}{m-1} \, ,$$

and the following bounds on the optimal cost: $\text{OPT}_{i+1} \geq Y_0^{i+1}$ and $m \cdot \text{OPT}_{i+1} \geq Q_i + Y_0^{i+1}$, we get

$$\begin{aligned} L_1^i + Y_0^{i+1} &\leq (\mathcal{R}-1) \cdot \frac{Q_i}{m-1} + Y_0^{i+1} = \frac{\mathcal{R}-1}{m-1} \cdot (Q_i + Y_0^{i+1}) + Y_0^{i+1} \cdot (\frac{m-\mathcal{R}}{m-1}) \\ &\leq \frac{\mathcal{R}-1}{m-1} \cdot m \cdot \text{OPT}_{i+1} + \text{OPT}_{i+1} \cdot (\frac{m-\mathcal{R}}{m-1}) = \mathcal{R} \cdot \text{OPT}_{i+1} \, , \end{aligned}$$

for any $\mathcal{R} \leq 2$. ■

Theorem 12 will show that the obtained bounds are tight within algorithms for a fixed size buffer. We summarize these bounds in the following.

Theorem 3 *An application of the master algorithm, using $K = \frac{m}{2} - 1$ and $\mathcal{R} = \frac{4}{3}$ for even values of m, and $K = \frac{m-1}{2}$ and $\mathcal{R} = \frac{4m^2}{3m^2+1}$, for odd values of m, is successful, i.e., results in an algorithm of a competitive ratio of at most $\mathcal{R}$.*

2.2.1 One special case

We investigate the special case $m = 6$ separately. In this case, the application of the master algorithm with $K = \frac{m-2}{2} = 2$ leads to a competitive ratio of $\frac{4}{3}$. We show that a buffer of size 1 does not allow to achieve the best possible competitive ratio, $\frac{4}{3}$. In this last case, $K = 1$, we show that the best competitive ratio which can be achieved is $\mathcal{R} = \frac{19}{14} \approx 1.3571 > \frac{4}{3}$. To prove the upper bound, the algorithm of Section 2.2 is applied with $\mathcal{R} = \frac{19}{14}$. Using a different method of analysis we prove the following theorem. The proved bound is tight, by Theorem 19.

Theorem 4 *An application of the master algorithm for $m = 6$ using $K = 1$ and $\mathcal{R} = \frac{19}{14}$ is successful.*

In this proof we use two invariants. The first one is

$$L_5^i + L_6^i \geq \mathcal{R} \cdot \frac{Q_i}{3} = \frac{19}{42} \cdot Q_i$$

and the second one is

$$L_1^i \leq \frac{\mathcal{R} - 1}{5} \cdot Q_i = \frac{Q_i}{14} \, .$$

The motivation of both invariants is to show that the loads of machines satisfy the condition that the least loaded machine is loaded so lightly that it can receive the last job which needs to be assigned without violating the required competitive ratio. That is, if it turns out that the sequence terminated, and only the job remaining in the buffer still needs to be assigned, there is a way to assign this job. The second condition requires the two most loaded machines to be loaded quite heavily, which serves as a tool in proving that the least loaded machine is loaded lightly. Note that the average load of a machine is $\frac{Q_i}{6}$, so if the machines were balanced, the load of two machines would have been $\frac{14}{42} \cdot Q_i$ rather than at least $\frac{19}{42} \cdot Q_i$. The first invariant is proved by induction while the second one is proved directly.

Lemma 5 *The first invariant holds for any $i \geq 0$, for which after i jobs have been assigned, the buffer contains a job.*

Proof. We prove the invariant by induction. Before any jobs are assigned we have $L_5^0 + L_6^0 = 0 = Q_0$. Assume that the invariant holds for a given value of i. Let X_{i+1} be the size of the $(i+1)$-th job which is ever scheduled. If all parts of the job are assigned to machines 5 and 6, then

$$L_5^{i+1} + L_6^{i+1} = L_5^i + L_6^i + X_{i+1} \geq \frac{19}{42} \cdot Q_i + X_{i+1} > \frac{19}{42} \cdot (Q_i + X_{i+1}) = \frac{19}{42} \cdot Q_{i+1} \, ,$$

since $Q_{i+1} = Q_i + X_{i+1}$.

Otherwise, the slots allocated for this job on the sixth and fifth machines are full, so $L_6^{i+1} = \mathcal{R} \cdot \text{OPT}_{i+1}$ and $L_5^{i+1} = L_6^i$. Since $L_6^i \geq L_5^i$ we have $L_5^{i+1} \geq \frac{1}{2} \cdot (L_5^i + L_6^i) \geq \frac{19}{84} \cdot Q_i = \frac{19}{84} \cdot (Q_{i+1} - X_{i+1})$. Recall that we assume that after the assignment a job remains in the buffer. This job has a size of at least X_{i+1}, so we have $\text{OPT}_{i+1} \geq \frac{1}{6} \cdot (Q_{i+1} + X_{i+1})$, and consequently $L_6^{i+1} \geq \frac{19}{84} \cdot (Q_{i+1} + X_{i+1})$.

Thus

$$L_5^{i+1} + L_6^{i+1} \geq (\frac{19}{84} \cdot Q_{i+1} + \frac{19}{84} \cdot X_{i+1}) + (\frac{19}{84} \cdot Q_{i+1} - \frac{19}{84} \cdot X_{i+1}) = \frac{19}{42} \cdot Q_{i+1} \, .$$

■

The proof of the next technical lemma can be found in Section 4.2.

Lemma 6 *The second invariant holds for any $i \geq 0$, for which after i jobs have been assigned, the buffer contains a job, as long as the first i jobs ever assigned are assigned successfully.*

Lemma 7 *For every time $i+1$ ($i \geq 0$), if the invariants hold at time i, then the algorithm assigns the next job successfully.*

Proof. Let X_{i+1} be the size of the $(i+1)$-th job which is ever scheduled. We need to show that the total size of slots is sufficient, i.e., $\mathcal{R} \cdot \text{OPT}_{i+1} - L_1^i \geq X_{i+1}$. We have $L_1^i \leq \frac{Q_i}{14} = \frac{Q_{i+1}-X_{i+1}}{14}$. Thus

$$L_1^i + X_{i+1} \leq \frac{Q_{i+1} - X_{i+1}}{14} + X_{i+1} = \frac{Q_{i+1}}{14} + \frac{13}{14} \cdot X_{i+1} \leq \text{OPT}_{i+1} \cdot (\frac{6}{14} + \frac{13}{14}) = \mathcal{R} \cdot \text{OPT}_{i+1} \ ,$$

using $\text{OPT}_{i+1} \geq X_{i+1}$ and $\text{OPT}_{i+1} \geq \frac{Q_{i+1}}{6}$. ■

Note that it is not assumed in Lemma 7 that there is an additional job in the buffer, thus it holds for the very last job which is assigned as well. The next corollary completes the proof.

Corollary 8 *All jobs are scheduled successfully.*

2.3 Non-increasing job sizes

We assume that jobs arrive sorted by non-increasing sizes. We use the master algorithm with the parameters $\mathcal{R} = \max_{\mu \in \{0,1,2,\ldots,m-K-1\}} \frac{2m(m+\mu)}{2m^2+2K\mu+\mu^2+\mu}$, for $1 \leq K \leq m-1$. In the case $K = m-1$, $\mathcal{R} = 1$, so the algorithm finds an optimal schedule.

Recall that in this semi-online variant the algorithm keeps the first K jobs (which are the largest jobs) in the buffer, and that we assume $n > K$. Let $Z_1 \geq Z_2 \geq \ldots \geq Z_K$ be the sizes of jobs which are stored in the buffer. We do not use indices here since there is no change in the contents of the buffer until the last job arrives. After $n-s$ jobs are assigned, for $0 \leq s < K$, the buffer contains only the jobs of sizes $Z_1, Z_2, \ldots, Z_s$. Let X_i denote the size of the i-th job which is assigned. Since Z_1 is the size of the largest job in the sequence, if the buffer contains ℓ jobs after the i-th job is assigned, then we have $\text{OPT}_i = \max\{Z_1, \frac{1}{m} \cdot (Q_i + \sum_{k=1}^{\ell} Z_k)\}$. Thus there exists an integer $1 \leq f \leq n+1$ so that $\text{OPT}_i = Z_1$ for all $i < f$ and $\text{OPT}_i > Z_1$ for all $i \geq f$.

We next define a set of invariants which must hold after every assignment, and prove Lemma 9 below in Section 4.3. Invariant j at time i is defined by

$$\sum_{k=1}^{j} L_k^i + \sum_{k=1}^{j} Z_k \leq j \cdot \mathcal{R} \cdot \text{OPT}_i \ .$$

The motivation of these invariants comes from the possibilities of assigning the jobs which are stored in the buffer, if no additional jobs arrive. Every set of largest j jobs cannot run in parallel on more than j machines. The invariants consider the load resulting from assigning the largest j jobs to the least loaded j machines. Therefore, the indices for which the invariant needs to hold depend on the number of jobs in the buffer.

To prove the K-th invariant, if the invariant does not hold immediately using induction, we use a direct proof, similar to the direct proof of Lemma 6.

Lemma 9 *For a given i, let ℓ denote the number of jobs in the buffer after i jobs were assigned successfully. The j-th invariant holds for i and every $1 \leq j \leq \ell$.*

In order to prove that the invariants hold at some given time, it is necessary to assume that the last assigned job was assigned successfully. We prove that this is the case if the invariants were true just before the assignment of this job.

Lemma 10 *For every time $i+1$ ($i \geq 0$ and $i+1 \leq n$), if there is at least one job in the buffer, then the algorithm assigns the next job successfully.*

Proof. To prove that the assignment of the $(i+1)$-th job ever assigned is successful, we need to show that the total size of the slots is sufficient. We use the first invariant. There is at least one job in the buffer, so by Lemma 9, so $L_1^i + Z_1 \leq \mathcal{R} \cdot \text{OPT}_i \leq \mathcal{R} \cdot \text{OPT}_{i+1}$. Since the size of no job in the sequence exceeds the size Z_1, i.e., $X_{i+1} \leq Z_1$, we are done. ■

Since we have proved that if the invariants hold at a given time, then the next job is assigned successfully, and if a job was assigned successfully, then the invariants hold after its assignment, we get that all jobs are assigned successfully.

We have proved the following theorem.

Theorem 11 *The application of the master algorithm for $1 \leq K \leq m-1$ with*

$$\mathcal{R} = \max_{\mu \in \{0,1,2,\ldots,m-K-1\}} \frac{2m(m+\mu)}{2m^2 + 2K\mu + \mu^2 + \mu}$$

is successful. In particular, the application of the master algorithm using the parameters $K = m-1$ and $\mathcal{R} = 1$ is successful.

By Corollary 20, these bounds are best possible.

3 Lower bounds

In this section we prove the following theorem, which implies the optimality of the results of Section 2.2 (excluding Section 2.2.1). The optimality of the result of Section 2.2.1 is shown later, in Theorem 19, and the optimality of the result of Section 2.3 will follow from Corollary 20.

Theorem 12 *No algorithm for general inputs which uses a fixed size buffer can have a smaller competitive ratio than $\frac{4}{3}$ for even m, and no algorithm which uses a fixed size buffer can have a smaller competitive ratio than $\frac{4m^2}{3m^2+1}$ for odd m. An algorithm which uses a buffer of size $o(m)$ has an overall competitive ratio of $\frac{e}{e-1}$, that is, the usage of a buffer of this size is not helpful.*

In order to prove lower bounds, we provide a "recipe" for designing lower bounds for the problem. This method is an adaptation of the method used in [10, 16], which takes into account the existence of a buffer.

We restrict ourselves to inputs which have a specific form. All the considered inputs are prefixes of one sequence. The sequence consists of a non-negative number of blocks $b \geq 0$, where each block contains identically sized jobs, and finally, the sequence of blocks may be followed by t additional jobs. Let n_i be the number of jobs in the i-th block, and s_i be the size of each such job. We require $n_i \geq K+1$. The t additional jobs arriving after all blocks are called *further* jobs. Let q_j denote the size of the j-th further job, for $1 \leq j \leq t$. We also require that $t + \sum_{i=1}^{b} n_i \geq m$. We denote such a sequence, defined for a specific value of K, by σ.

We use OPT_j to denote the optimal makespan for the sequence of jobs up to (and including) the j-th **further** job. In addition, we let $\text{OPT}_{i,j}$ denote the optimal makespan for the sequence of jobs up to (and including) the j-th job of the i-th block.

For each block i, we define the last $n_i - K$ optimal costs ($\text{OPT}_{i,j}$ for $K+1 \leq j \leq n_i$) as *crucial*.

The typical analysis of the behavior of online algorithms on such sequences is based on the property that an online algorithm must assign the jobs block by block. The difficulty in the analysis, in the case that a buffer can be used, lies in the option of the algorithm to keep at most K jobs from previous blocks in the buffer at the time when the jobs of a new block (or the further jobs) start arriving. In order to be able to assume that the algorithm still processes the jobs sorted by blocks, we neglect the assignment of a number of jobs of each block which are assigned later. Specifically, we remove K jobs of each block from the final output. These are the last K jobs of each block which are assigned, so as a result, no jobs are present in the buffer after a block was presented, and the jobs remaining in the schedule are in fact assigned block by block. We still do not know if they are assigned in the order of arrival, but since all jobs in a block have the same size, we can assume that this is the case. For further jobs, this is not necessarily true, so we need to consider an arbitrary order of assignment for these jobs.

More precisely, for the analysis we define a modified sequence, which is based on a specific output of the online algorithm, as follows. For each block i, remove from the schedule the last K jobs of this block (i.e., of size s_i) which were ever assigned. If the execution of the algorithm on the original sequence is restricted to the modified sequence (and the treatment of the removed jobs is simply neglected), then at the time of arrival of the first job of a block, and also at the time of arrival of the first further job, the buffer is empty. This holds since the buffer can contain at most K such jobs, which are the last assigned jobs of this block, and thus all these jobs were removed from the schedule. Therefore, the order of assignment is according to blocks, and the further jobs are assigned last, in some order.

We define a sequence of costs $\mathcal{C}_j$ for $1 \leq j \leq m$. These costs are upper bounds on the optimal cost at the time of assignment of the last m jobs in the modified sequence.

Consider first the sequence of all crucial optimal costs of all blocks (i.e., all optimal costs, neglecting the first K optimal costs of each block). If $t > K$, this sequence is followed by the optimal costs OPT_j for $K+1 \leq j \leq t$, and finally, K times the cost OPT_t. It $t \leq K$, then the last t values are t times OPT_t. The last m costs in this sequence are denoted by $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$. Note that the last $\min\{m, t, K+1\}$ values in this sequence are OPT_t, which is the optimal cost of the entire input sequence.

We consider the relation between the $\mathcal{C}_j$ values and the optimal costs at the time of assignment of jobs. Let $a_1, a_2, \ldots, a_m$ be the last m jobs of the modified sequence ever assigned. If $t > m$ then all those jobs are further jobs. Otherwise, the last t of those jobs are further jobs. Let O_j denote the optimal cost (of the input that already arrived, for the original sequence) at the time of assignment of a_j. We next prove the relation $O_j \leq \mathcal{C}_j$ for all m values of j.

Consider the time at which a job x, which is the k-th assigned job of a given block i, is assigned. If $n_i \geq k + K$, then in the original sequence, no jobs of future blocks, or further jobs have arrived until this time, and at most K additional jobs of the same block have arrived. Therefore, an upper bound the optimal cost can be computed based on the jobs all previous blocks, and $k + K$ jobs of the current block, i.e., this upper bound is $\text{OPT}_{i,k+K}$.

Consider next the situation where x is the k-th assigned further job. If the sequence contains more than K further jobs arriving after x, at most K jobs can be stored in the buffer, so at the time of assignment of the k-th assigned further job (which is not necessarily the k-th arriving further job), at most $k + K$ further jobs have arrived. Therefore, an upper bound the optimal cost can be computed based on the jobs all previous

blocks, and the first $k+K$ further jobs, i.e., this upper bound is OPT_{k+K}. Clearly, if $k+K > t$ then an upper bound can be computed based on the entire input.

Lemma 13 *For every $1 \leq j \leq m$, $O_j \leq \mathcal{C}_j$.*

Proof. Assume first that $t \leq m$. Consider first the last t jobs ever assigned. Since after each block the buffer is empty, these jobs are exactly the t further jobs. We prove $O_j \leq \mathcal{C}_j$ for $m-t+1 \leq j \leq m$.

If $t \leq K+1$, then clearly $O_j \leq \mathcal{C}_j$ for $m-t+1 \leq j \leq m$, since $\mathcal{C}_j = \text{OPT}_t$ by definition of the $\mathcal{C}_j$ values, and OPT_t is the optimal cost of the entire input.

If $t \geq K+2$, then similarly, $\mathcal{C}_j = \text{OPT}_t$ for $m-K \leq j \leq m$, and we need to consider the case $m-t+1 \leq j \leq m-K-1$. Consider the time when $k-1$ further jobs were already assigned (for $1 \leq k \leq t-K-1$), and the k-th job needs to be assigned. Since the buffer contains at most K jobs, and $k-1$ further jobs were assigned, then at most $k+K$ further jobs have arrived, and the optimal cost is at most $\text{OPT}_{k+K} = \mathcal{C}_{m-t+k}$. Thus $O_j \leq \mathcal{C}_j$ for the required values of j.

Next, if $t \neq m$, consider a job a_j of block i, i.e., this job is the j-th jobs out of the last m assigned jobs in the modified sequence, and the k-th assigned job of block i so that $k+K \leq n_i$. The cost $\mathcal{C}_j$ was defined to be the optimal cost of the input up to the i-th block, including $k+K$ jobs of block i, i.e. it is $\text{OPT}_{i,k+K}$. Since at most $k+K$ jobs of this block could arrive by the time that a_j is assigned, we get $O_j \leq \mathcal{C}_j$.

If $t > m$, the the last assigned m jobs are all further jobs. If $m \leq K+1$, then $\mathcal{C}_j = \text{OPT}_t$ for $1 \leq j \leq m$, and we are done. Otherwise, the proof is similar to the case $K+2 \leq t \leq m$, but we are only interested in the times of assignment of the jobs $a_1, \ldots, a_{m-K-1}$, so the first $t-m$ further jobs (according to the time of assignment) are not considered. ■

Theorem 14 *Given a sequence σ as defined above. The competitive ratio of any algorithm which uses a buffer of size K is at least*

$$\frac{\sum_{j=1}^{t} q_j + \sum_{i=1}^{b} (n_i - K) \cdot s_i}{\sum_{k=1}^{m} \mathcal{C}_k}.$$

Proof. Consider the modified schedule. Let T_i (for $i = 1, \ldots, m$) denote the makespan of the algorithm when $i-1$ jobs out of the jobs of the modified sequence still need to be assigned. Consider the time axis of the schedule and let τ_i denote the maximum time in the final schedule of the modified sequence at which at least i machines are still active, for $1 \leq i \leq m$. If there is no such time for some i then we let $\tau_i = 0$. Clearly, $\tau_1 \geq \tau_2 \geq \ldots \geq \tau_m$ must hold. We show that $T_i \geq \tau_i$. For $i = 1$ this is clear. Next, remove the jobs $a_m, a_{m-1}, \ldots, a_1$, one by one, in this order (which is opposite to the order which they were assigned).

After the removal of a set of the $\ell - 1 > 0$ jobs which were assigned last (jobs $a_m, \ldots, a_{m-\ell+2}$), the maximum completion time is at least τ_ℓ, since the activity on the machines at the point in time τ_ℓ was affected (by the removal) on at most $\ell - 1$ machines, and at least one machine remains active at time τ_ℓ. Therefore $T_\ell \geq \tau_\ell$.

Let $\mathcal{R}$ be the competitive ratio of the algorithm. Thus $T_j \leq \mathcal{R} \cdot O_{m-j+1}$ for every $1 \leq j \leq m$. By Lemma 13, $O_{m-j+1} \leq \mathcal{C}_{m-j+1}$, so $T_j \leq \mathcal{R}\mathcal{C}_{m-j+1}$. Let W be the sum of all jobs sizes in the modified input, i.e., $W = \sum_{j=1}^{t} q_j + \sum_{i=1}^{b} (n_i - K) \cdot s_i$ and let $\tau_{m+1} = 0$. We next claim that $W \leq \sum_{k=1}^{m} T_k$. Consider the time interval between times τ_{k+1} and τ_k for some $1 \leq k \leq m$. During this time, at each point in time,

there are at most k active machines. Thus the total size of parts of job processed during this time interval is at most $k \cdot (\tau_k - \tau_{k+1})$. Therefore (using $\tau_{m+1} = 0$),

$$W \leq \sum_{k=1}^{m} k \cdot (\tau_k - \tau_{k+1}) = \sum_{k=1}^{m} k \cdot \tau_k - \sum_{k=1}^{m} k \cdot \tau_{k+1} = \sum_{k=1}^{m} k \cdot \tau_k - \sum_{k=1}^{m} (k-1) \cdot \tau_k = \sum_{k=1}^{m} \tau_k \leq \sum_{k=1}^{m} T_k \ ,$$

since $\tau_k \leq T_k$ for any $1 \leq k \leq m$. Therefore we have $W \leq \sum_{k=1}^{m} T_k \leq \sum_{k=1}^{m} \mathcal{R} \cdot \mathcal{C}_{m-k+1} = \sum_{j=1}^{m} \mathcal{R} \cdot \mathcal{C}_j$. The lower bound follows. ■

We use Theorem 14 to prove all the lower bounds in the section. We start with a lower bound for general inputs. The lower bound is simple and it only uses a single block before the further jobs. This lower bound can be used for different values of K.

We define a class of lower bound sequences σ_1 which is used below. This type of sequence contains a block of very small jobs, and $t < m$ further jobs. Let N be a large integer, and let $\delta = \frac{1}{N}$. The first block of σ_1 contains KN jobs of size $\frac{1}{KN} = \frac{\delta}{K}$. The number of further jobs, and their sizes are defined separately for each case.

Lemma 15 *For any fixed value of K, the competitive ratio of any algorithm which uses a buffer of size K is at least $\frac{4}{3}$ for even m and at least $\frac{4m^2}{3m^2+1}$ for odd m.*

Proof. The main idea of this lower bound is that a large number of tiny jobs of a total size of 1 arrive. Since the jobs are small, the buffer has no effect, and the algorithm has to distribute almost all jobs among the machines. Since jobs are tiny, preemption is not helpful. After the assignment of almost all small jobs takes place, roughly $\frac{m}{2}$ large jobs arrive. Each large job has a size of roughly $\frac{2}{m}$, so if these jobs arrive, it is revealed that the small jobs were supposed to occupy only half of the machines until time $\frac{2}{m}$. It is either the case that sufficiently many machines are not busy at sufficiently early times, in which case the assignment of the small jobs has a relatively completion time, or otherwise, after the assignment of larger jobs, the maximum completion time is sufficiently large. These two possible scenarios result in a lower bound of $\frac{4}{3}$ on the competitive ratio. If m is odd, the lower bound is slightly lower.

We use σ_1 and $t = \lceil \frac{m}{2} \rceil$ further jobs of size $\frac{1}{\lfloor \frac{m}{2} \rfloor}$. The size of further jobs was chosen so that in an optimal assignment of the complete sequence, each further job is assigned to a dedicated machine, while the other jobs are spread evenly on the other machines. In this schedule, all machines have a load of $\frac{1}{\lfloor \frac{m}{2} \rfloor}$.

For even m, the total size of jobs in the modified sequence is $2 - \delta$, $\mathcal{C}_j \leq \frac{1}{m}$ for $1 \leq j \leq \frac{m}{2}$, and $\mathcal{C}_j \leq \frac{2}{m}$ for $\frac{m}{2} + 1 \leq j \leq m$. This gives a lower bound of $\frac{2-\delta}{\frac{m}{2} \cdot \frac{1}{m} + \frac{m}{2} \cdot \frac{2}{m}}$. For small enough δ, this value tends to $\frac{4}{3}$.

For odd values of m, the total size of jobs in the modified sequence is $1 - \delta + \frac{m+1}{2} \cdot \frac{2}{m-1}$, $\mathcal{C}_j \leq \frac{1}{m}$ for $1 \leq j \leq \frac{m-1}{2}$, and $\mathcal{C}_j \leq \frac{2}{m-1}$ for $\frac{m+1}{2} \leq j \leq m$. This gives a lower bound of $\frac{1-\delta+\frac{m+1}{m-1}}{\frac{m-1}{2} \cdot \frac{1}{m} + \frac{m+1}{2} \cdot \frac{2}{m-1}}$. For small enough δ, the lower bound tends to $\frac{4m^2}{(m-1)^2+2m(m+1)} = \frac{4m^2}{3m^2+1}$. ■

We next analyze the lower bound resulting from sequences of the type σ_1.

Lemma 16 *Let $t < m$. The competitive ratio of any algorithm which uses a buffer of size $K < t$ is at least*

$$\frac{1 + \sum_{i=1}^{t} q_i}{\frac{m-t}{m} + \sum_{i=1}^{t-K} \mathrm{OPT}_{i+K} + K \cdot \mathrm{OPT}_t} \ ,$$

where q_j is the size of the j-th further job in the sequence σ_1, and the value OPT_j is the optimal cost of the sequence up to the j-th further job.

Proof. We use Theorem 14. The total size of the jobs of the modified sequence is $1-\delta+\sum_{i=1}^{t} q_i$. We have $\mathcal{C}_j = \text{OPT}_t$ for $m-K+1 \le j \le m$, $\mathcal{C}_j = \text{OPT}_{j+K-m+t}$ for $m-t+1 \le j \le m-K$, and $\mathcal{C}_j \le \frac{1}{m}$ for $1 \le j \le m-t$. ■

We next consider a special case of σ_1, where the list of further jobs consists of $K+1$ identical jobs, followed by a sequence of jobs with increasing sizes. The sequence is constructed so that an optimal schedule for the sequence up to the $(K+1)$-th further job is flat, that is, each further job is assigned to a dedicated machine, and the other jobs are spread evenly on the other machines, giving the same load to each machine. The additional further jobs form an increasing sequence of sizes, which allows exactly a flat optimal schedule.

Corollary 17 *The competitive ratio of any algorithm which uses a buffer of size $K \le \lceil \frac{m-2}{2} \rceil$ is at least*

$$\frac{m^t}{(t+tK-tm-Km)\cdot m^{K-1}\cdot(m-1)^{t-K-1}+(K+m)\cdot m^{t-1}} .$$

for any $K+1 \le t \le m-1$.

Proof. We apply Lemma 16 with $K+1 \le t \le m-1$. Let $Q_j = 1+\sum_{i=1}^{j} q_i$.

We use $q_1 = q_2 = \cdots = q_{K+1} = \frac{1}{m-K-1}$. Thus $Q_{K+1} = 1+\frac{K+1}{m-K-1} = \frac{m}{m-K-1}$, so $\text{OPT}_{K+1} = \frac{1}{m-K-1}$.

We let $q_j = (\frac{m}{m-1})^{j-K-1}\cdot q_1$ for $j = K+2, K+3, \ldots, t$. Thus $q_t \ge q_{t-1} \ge \cdots \ge q_1$ and $q_j = (\frac{m}{m-1})q_{j-1}$, or equivalently, $(m-1)q_j = m\cdot q_{j-1}$, for $K+2 \le j \le t$.

We next prove by induction that for $K+1 \le j \le t$, $Q_j = m\cdot q_j$, or equivalently, $Q_{j-1} = (m-1)\cdot q_j$ (the equivalence follows from since $Q_j = Q_{j-1}+q_j$). For $j = K+1$ we have $Q_{K+1} = \frac{m}{m-K-1} = m\cdot q_{K+1}$. For $j \ge K+2$ we have $Q_{j-1} = m\cdot q_{j-1}$, therefore $Q_j = m\cdot q_{j-1}+q_j = (m-1)\cdot q_j + q_j = m\cdot q_j$, since $(m-1)q_j = m\cdot q_{j-1}$.

Since $q_j = \frac{Q_j}{m}$ for $j \ge K+1$, we get that in this case $\text{OPT}_j = q_j$. For $j < K+1$, $Q_j < m\cdot q_j$ and therefore $\text{OPT}_j = q_j$. We have

$$1+\sum_{i=1}^{t} q_i = Q_t = m\cdot q_t = m\cdot\left(\frac{m}{m-1}\right)^{t-K-1}\cdot\frac{1}{m-K-1} .$$

On the other hand,

$$\begin{aligned}
&\frac{m-t}{m}+\sum_{i=1}^{t-K}\text{OPT}_{i+K}+K\cdot\text{OPT}_t = \frac{m-t}{m}+\sum_{i=K+1}^{t}\text{OPT}_i+K\cdot\left(\frac{m}{m-1}\right)^{t-K-1}\cdot\frac{1}{m-K-1}\\
=\ &\frac{m-t}{m}+\sum_{i=K+1}^{t}(\frac{m}{m-1})^{i-K-1}\cdot\frac{1}{m-K-1}+K\cdot\left(\frac{m}{m-1}\right)^{t-K-1}\cdot\frac{1}{m-K-1}\\
=\ &\frac{m-t}{m}+\sum_{j=1}^{t-K}(\frac{m}{m-1})^{j-1}\cdot\frac{1}{m-K-1}+K\cdot\left(\frac{m}{m-1}\right)^{t-K-1}\cdot\frac{1}{m-K-1}\\
=\ &\frac{m-t}{m}+\frac{1}{m-K-1}\cdot\left(\frac{(\frac{m}{m-1})^{t-K}-1}{\frac{m}{m-1}-1}+K\cdot\left(\frac{m}{m-1}\right)^{t-K-1}\right) .
\end{aligned}$$

This results in a lower bound of

$$\begin{aligned} & \frac{m \cdot (\frac{m}{m-1})^{t-K-1}}{\frac{(m-t)\cdot(m-K-1)}{m} + K \cdot (\frac{m}{m-1})^{t-K-1} + (m-1) \cdot (\frac{m}{m-1})^{t-K} - (m-1)} \\ = & \frac{m^t}{(m-t) \cdot (m-K-1) \cdot m^{K-1} \cdot (m-1)^{t-K-1} + K \cdot m^{t-1} + m^t - m^K \cdot (m-1)^{t-K}} \\ = & \frac{m^t}{(t+tK-tm-Km) \cdot m^{K-1} \cdot (m-1)^{t-K-1} + (K+m) \cdot m^{t-1}} \end{aligned}$$

on the competitive ratio.

For the case $K = 1$, $t = m - 2$ this results in a lower bound of

$$\frac{m^m}{(3m - m^2 - 4) \cdot m^2 \cdot (m-1)^{m-4} + (m+1) \cdot m^{m-1}} .$$

■

We next show as a corollary of the lower bound above that using a buffer of size $o(m)$ gives a competitive ratio which tends to $\frac{e}{e-1}$ for $m \to \infty$. Thus, the size of the buffer must be a linear function of m in order to improve over the upper bound of the case where no buffer is used.

Corollary 18 *Any algorithm using a buffer of size $o(m)$ has an overall competitive ratio of at least $\frac{e}{e-1}$.*

Proof. Using $t = m - 1$, we have a lower bound of

$$\begin{aligned} & \frac{m^{m-1}}{(2m - K - 1 - m^2) \cdot m^{K-1} \cdot (m-1)^{m-2-K} + (K+m) \cdot m^{m-2}} \\ = & \frac{(\frac{m}{m-1})^{m-1}}{\frac{2m-K-1-m^2}{(m-1)^2} \cdot (\frac{m}{m-1})^{K-1} + \frac{K+m}{m} \cdot (\frac{m}{m-1})^{m-1}} , \end{aligned}$$

which tends to $\frac{e}{e-1}$ for large m and $K = o(m)$. ■

This completes the proof of Theorem 12. We now turn to proving the additional lower bounds of this section. In the construction of the sequence of the following proof, two blocks are used before the further jobs.

Theorem 19 *Any algorithm with $K = 1$ and $m \geq 5$ has a competitive ratio of at least $\frac{4m^3-12m^2+4m}{3m^3-11m^2+18m-24}$, which gives a lower bound of $\frac{19}{14}$ for $m = 6$.*

Proof. Consider an input with two blocks, for $m \geq 5$ and $K = 1$. The first block consists again of very small jobs, of total size 1 (similarly to the first block of σ_1). The next block contains $m - 3 \geq 2$ jobs, each of size $\frac{1}{m-2}$. There are two further jobs which have a size of $\frac{2m-5}{(m-2)^2}$ (which is larger than the size of each job of the block). The total size of jobs in the modified sequence is $\frac{2m^2-6m+2}{(m-2)^2} - \delta$.

We have $\mathcal{C}_m = \mathcal{C}_{m-1} = \frac{2m-5}{(m-2)^2}$. For $3 \leq j \leq m-2$, $\mathcal{C}_j = \frac{1}{m} \cdot (\frac{j-1}{m-2} + 1)$, and finally $\mathcal{C}_1 = \mathcal{C}_2 = \frac{1}{m}$. Thus

$$\begin{aligned} \sum_{j=1}^{m} \mathcal{C}_j &= \frac{2}{m} + 2 \cdot \frac{2m-5}{(m-2)^2} + \sum_{g=1}^{m-4} \frac{m+g-1}{m(m-2)} \\ &= \frac{2}{m} + \frac{(m-1)(m-4)}{m(m-2)} + \frac{(m-4)(m-3)}{2m(m-2)} + \frac{2(2m-5)}{(m-2)^2} = \frac{3m^3 - 11m^2 + 18m - 24}{2m(m-2)^2} . \end{aligned}$$

Using Theorem 14, we get a lower bound of $\frac{4m^3-12m^2+4m}{3m^3-11m^2+18m-24}$. For $m = 6$, the resulting lower bound is $\frac{19}{14} \approx 1.3571$. ■

Note that the lower bound for the case $m = 6$, $K = 1$ resulting from Corollary 17 (with $t = 4$) is only $\frac{648}{481} \approx 1.3472$. The lower bound for $m = 7$ given by Theorem 19 is $\frac{203}{148} \approx 1.3716216$, which is an improvement over the lower bound which is implied of Corollary 17 as well.

Next, we consider the case of non-increasing job sizes and prove the following.

Corollary 20 *For the case of non-increasing job sizes, let $0 \leq K \leq m-2$. The competitive ratio of any algorithm is at least $\max_{1 \leq \mu \leq m-K-1} \frac{2m(m+\mu)}{2m^2+2K\mu+\mu^2+\mu}$ (where μ takes integer values). Specifically, if $K \leq m-2$, then no algorithm can compute an optimal solution. For $K \geq m-1$, no algorithm can have a competitive ratio below 1. Thus, the algorithms of Section 2.3 are best possible both in terms of competitive ratio and the size of the used buffer.*

Proof. If $K \geq m-1$, a sequence which consists of a single job shows that the competitive ratio cannot be below 1. Let $1 \leq K \leq m-2$ and fix an integer value of μ, where $1 \leq \mu \leq m-K-1$. We use $b = 0$ and $n = m + \mu$ and use jobs of size 1. We have $\text{OPT}_i = 1$ for $i \leq m$, and $\text{OPT}_{m+i} = \frac{m+i}{m}$, for $m+1 \leq i \leq m+\mu$. In the sequence $\mathcal{C}_1, \ldots, \mathcal{C}_m$, the last K values are $\text{OPT}_{m+\mu}$, and the previous $m-K$ values are $\text{OPT}_{\mu+K+1}, \ldots, \text{OPT}_{m+\mu}$. We get the values 1 ($m-K-\mu$ times), $\frac{m+1}{m}, \frac{m+2}{m}, \ldots, \frac{m+\mu-1}{m}$, and $\frac{m+\mu}{m}$ ($K+1$ times). Using Theorem 14 we find that the resulting value of the lower bound is

$$\begin{aligned}
& \frac{m+\mu}{(m-K-\mu) + \sum_{i=1}^{\mu-1} \frac{m+i}{m} + (K+1) \cdot \frac{m+\mu}{m}} \\
= & \frac{2m \cdot (m+\mu)}{2m \cdot (m-K-\mu) + 2 \cdot \sum_{i=1}^{\mu-1} (m+i) + 2 \cdot (K+1) \cdot (m+\mu)} \\
= & \frac{2m \cdot (m+\mu)}{2m^2 - 2mK - 2m\mu + 2 \cdot (\mu-1) \cdot m + \mu \cdot (\mu-1) + 2Km + 2m + 2K\mu + 2\mu} \\
= & \frac{2m \cdot (m+\mu)}{2m^2 + 2K\mu + \mu^2 + \mu} \, .
\end{aligned}$$

Note that using $\mu = 0$ yields a lower bound of 1, and using $\mu \geq m-K$ yields a lower bound which is no larger than the case $\mu = m-K-1$, since for each increase by 1 in the value of μ, i.e., using $\mu' = \mu + 1$ the numerator grows by $2m$ while the denominator grows by $2K + 2\mu + 2 \geq 2(m-1) + 2 \geq 2m$, which does not increase the value of the lower bound. ■

4 Proofs of technical lemmas

4.1 Proof of Lemma 1

We prove the claim by induction on i. For $i = 0$, $L_k^i = 0$ for all k, and $Q_i = 0$ imply that all invariants hold. Assume now that all invariants hold for a given time i, and consider the invariants for time $i+1$. If none of machines $1, 2, \ldots, \lfloor \frac{m}{2} \rfloor$ received any parts of the $(i+1)$-th assigned job, then $L_k^{i+1} = L_k^i$ for $1 \leq k \leq \lfloor \frac{m}{2} \rfloor$, and thus all the invariants hold for time $i+1$, since $Q_{i+1} \geq Q_i$. Otherwise, let $1 \leq z \leq \lfloor \frac{m}{2} \rfloor$ denote the minimum index of a machine which received a non-zero part of the job. Similarly to the above, for any $k < z$, $L_k^{i+1} = L_k^i$, and thus for any $j < z$, the j-th invariant holds at time i. We need to prove the j-th

invariant for $j = z, z+1, \ldots, \lfloor \frac{m}{2} \rfloor$, if K jobs remain in the buffer (i.e., $i+1+K \leq n$), and otherwise for $j = z, z+1, \ldots, \ell$, if the number of jobs in the buffer after the assignment is $\ell < K$.

Note that for even m, if $i+K = n$, then at time i all $\frac{m}{2}$ invariants hold, while after the $(i+1)$-th job is assigned, it is required that only the invariants for $j \leq \frac{m}{2} - 2$ hold. Our proof would result in the invariants for $j \leq \frac{m}{2} - 1$ holding, that is, one invariant which can be proved would not be necessary. However, if m is odd, after the $(i+1)$-th job is assigned, it is required that only the invariants for $j \leq \frac{m-1}{2} - 1$ hold, which is exactly what is proved here.

After the assignment, since the slots of machines $z+1, z+2, \ldots, m$ were fully used, the loads of these machines satisfy $L_j^{i+1} = L_{j+1}^i$, for $z+1 \leq j \leq m-1$, and $L_m^{i+1} = \mathcal{R} \cdot \text{OPT}_{i+1}$. Therefore, for any $z \leq j \leq \lfloor \frac{m}{2} \rfloor$, the following inequality holds:

$$\begin{aligned}
\sum_{k=1}^{j} L_k^{i+1} &= Q_{i+1} - \sum_{k=j+1}^{m} L_k^{i+1} = Q_{i+1} - \sum_{k=j+1}^{m-1} L_{k+1}^i - \mathcal{R} \cdot \text{OPT}_{i+1} \\
&= Q_{i+1} - \sum_{k=j+2}^{m} L_k^i - \mathcal{R} \cdot \text{OPT}_{i+1} = Q_{i+1} - Q_i + \sum_{k=1}^{j+1} L_k^i - \mathcal{R} \cdot \text{OPT}_{i+1} \\
&= Y_0^{i+1} + \sum_{k=1}^{j+1} L_k^i - \mathcal{R} \cdot \text{OPT}_{i+1} \, .
\end{aligned}$$

Case 1. $j < \lfloor \frac{m}{2} \rfloor$. In this case $m - j - 1 > 0$ holds, and we can use the invariant of $j+1$ at time i. This invariant holds since either the buffer is full after the assignment of the $(i+1)$-th job, and thus it was also full before it, or otherwise the buffer contained at least $j+1$ jobs before the assignment. We have $\sum_{k=1}^{j+1} L_k^i \leq (\mathcal{R}-1) \cdot \frac{(j+1) \cdot Q_i}{m-j-1}$. Thus it is sufficient to prove

$$Y_0^{i+1} + (\mathcal{R}-1) \cdot \frac{(j+1) \cdot Q_i}{m-j-1} - \mathcal{R} \cdot \text{OPT}_{i+1} - (\mathcal{R}-1) \cdot \frac{j \cdot Q_{i+1}}{m-j} \leq 0.$$

Sub-case 1.1. If $Q_i < (m-j-1) \cdot Y_0^{i+1}$, then we use $\text{OPT}_{i+1} \geq Y_0^{i+1}$ and $Q_{i+1} = Q_i + Y_0^{i+1}$ to get

$$\begin{aligned}
& Y_0^{i+1} + \frac{(\mathcal{R}-1) \cdot (j+1) \cdot Q_i}{m-j-1} - \mathcal{R} \cdot \text{OPT}_{i+1} - \frac{(\mathcal{R}-1) \cdot j \cdot Q_{i+1}}{m-j} \\
\leq \; & Y_0^{i+1} \cdot \left(1 - \mathcal{R} - \frac{j \cdot (\mathcal{R}-1)}{m-j}\right) + Q_i \cdot \frac{m \cdot (\mathcal{R}-1)}{(m-j)(m-j-1)} \\
\leq \; & Y_0^{i+1} \cdot \left(1 - \mathcal{R} - \frac{j \cdot (\mathcal{R}-1)}{m-j} + \frac{m \cdot (\mathcal{R}-1)}{m-j}\right) = \frac{(\mathcal{R}-1) \cdot Y_0^{i+1}}{m-j}(-(m-j) - j + m) = 0 \, .
\end{aligned}$$

Since $m - j - 1 > 0$, the coefficient of Q_i in the second expression is positive, so the substitution is valid.

Sub-case 1.2. If $Q_i \geq (m-j-1) \cdot Y_0^{i+1}$, then we use $\text{OPT}_{i+1} \geq \frac{1}{m} \cdot (Q_i + (j+1) \cdot Y_0^{i+1})$ (since there are $\ell \geq j+1$ jobs stored in the buffer), and $Q_{i+1} = Q_i + Y_0^{i+1}$. It suffices to prove that

$$Y_0^{i+1} + (\mathcal{R}-1) \cdot \frac{(j+1) \cdot Q_i}{m-j-1} \leq \mathcal{R} \cdot \frac{1}{m} \cdot (Q_i + (j+1) \cdot Y_0^{i+1}) + (\mathcal{R}-1) \cdot \frac{j \cdot \left(Q_i + Y_0^{i+1}\right)}{m-j},$$

or equivalently,

$$Y_0^{i+1} + (\mathcal{R}-1)\cdot\frac{j+1}{m-j-1}\cdot Q_i \le \left(\frac{\mathcal{R}}{m} + \frac{(\mathcal{R}-1)\cdot j}{m-j}\right)\cdot Q_i + \left(\frac{\mathcal{R}\cdot(j+1)}{m} + \frac{(\mathcal{R}-1)\cdot j}{m-j}\right)\cdot Y_0^{i+1}.$$

Then it is easy to see that the coefficient of Y_0^{i+1} on the right hand side is less than 1, since if m is even, then $\mathcal{R} = 4/3$, $j \le \frac{m}{2} - 1$ and

$$\mathcal{R}\cdot\frac{j+1}{m} + (\mathcal{R}-1)\cdot\frac{j}{m-j} < \mathcal{R}\cdot\frac{m/2}{m} + (\mathcal{R}-1)\cdot\frac{m/2}{m/2} = \mathcal{R}/2 + \mathcal{R} - 1 = \frac{3}{2}\mathcal{R} - 1 \le 1\,.$$

Otherwise, if m is odd, then $\mathcal{R} = \frac{4m^2}{3m^2+1}$, $j \le \frac{m-1}{2} - 1$ and

$$\begin{aligned}
\mathcal{R}\cdot\frac{j+1}{m} + (\mathcal{R}-1)\cdot\frac{j}{m-j} &< \mathcal{R}\cdot\frac{m+1}{2m} + (\mathcal{R}-1)\cdot\frac{m-1}{m+1} \\
&= \mathcal{R}\frac{(m+1)^2 + 2m\cdot(m-1)}{2m\,(m+1)} - \frac{m-1}{m+1} \\
= \frac{4m^2}{3m^2+1}\cdot\frac{3m^2+1}{2m\cdot(m+1)} - \frac{m-1}{m+1} &= \frac{2m}{m+1} - \frac{m-1}{m+1} = 1\,.
\end{aligned}$$

It follows that in the next expression, which we would like to prove, the coefficient of Y_0^{i+1} is nonnegative:

$$\left(1 - \frac{\mathcal{R}}{m}\cdot(j+1) - \frac{(\mathcal{R}-1)\cdot j}{m-j}\right)\cdot Y_0^{i+1} \le \left(\frac{\mathcal{R}}{m} + \frac{(\mathcal{R}-1)\cdot j}{m-j} - (\mathcal{R}-1)\cdot\frac{j+1}{m-j-1}\right)\cdot Q_i.$$

Then no matter whether the coefficient in the right hand side is positive or not, it suffices to prove (by substituting $Q_i \ge (m-j-1)\cdot Y_0^{i+1}$) that

$$\left(1 - \frac{\mathcal{R}}{m}\cdot(j+1) - \frac{(\mathcal{R}-1)\cdot j}{m-j}\right) \le \left(\frac{\mathcal{R}}{m} + \frac{(\mathcal{R}-1)\cdot j}{m-j} - (\mathcal{R}-1)\cdot\frac{(j+1)}{m-j-1}\right)\cdot(m-j-1)\,,$$

or equivalently,

$$1 - \frac{\mathcal{R}}{m}\cdot(j+1) - \frac{(\mathcal{R}-1)\cdot j}{m-j} \le \frac{\mathcal{R}}{m}\cdot(m-j-1) + \frac{(\mathcal{R}-1)\cdot j}{m-j}\cdot(m-j-1) - (\mathcal{R}-1)\cdot(j+1)\,,$$

which is equivalent to

$$1 + (\mathcal{R}-1)\cdot(j+1) \le \mathcal{R}\cdot\frac{m-j-1+j+1}{m} + (\mathcal{R}-1)\cdot\frac{j\cdot(m-j-1)+j}{m-j}\,,$$

which is finally equivalent to the following condition which clearly holds.

$$(\mathcal{R}-1)\cdot j + \mathcal{R} \le \mathcal{R} + (\mathcal{R}-1)\cdot j\,.$$

Case 2. $j = \frac{m}{2}$, this case needs to be considered only if m is even, and $\frac{m}{2} - 1$ jobs remain in the buffer after the assignment. We prove this case separately as follows.

The invariant for j for time i is $\sum_{k=1}^{\frac{m}{2}} L_k^i \le (\mathcal{R}-1)\cdot Q_i$, and therefore $\sum_{k=\frac{m}{2}+1}^{m} L_k^i \ge (2-\mathcal{R})\cdot Q_i$. Since $L^i_{\frac{m}{2}+1} \le L^i_{\frac{m}{2}+2} \le \ldots \le L^i_m$, we have $\sum_{k=\frac{m}{2}+2}^{m} L_k^i \ge \frac{\frac{m}{2}-1}{\frac{m}{2}}\cdot(2-\mathcal{R})\cdot Q_i$. We get $\sum_{k=1}^{\frac{m}{2}+1} L_k^i \le (\frac{2}{m}\cdot(2-\mathcal{R}) + \mathcal{R} - 1)\cdot Q_i$.

Therefore, it is enough to prove

$$Y_0^{i+1} + (\frac{2}{m} \cdot (2-\mathcal{R}) + \mathcal{R} - 1) \cdot Q_i - \mathcal{R} \cdot \text{OPT}_{i+1} - (\mathcal{R}-1) \cdot Q_{i+1} \leq 0 \,.$$

We use $\text{OPT}_{i+1} \geq \frac{1}{m} \cdot (Q_i + \frac{m}{2} \cdot Y_0^{i+1})$ (since in addition to the job of size Y_0^{i+1} which is being assigned, the buffer contains $K = \frac{m}{2} - 1$ jobs of at least its size), $Q_{i+1} = Q_i + Y_0^{i+1}$ and $\mathcal{R} = \frac{4}{3}$ to get that it is enough to consider

$$\begin{aligned} Y_0^{i+1} + \left(\frac{2}{m} \cdot (2-\mathcal{R}) + \mathcal{R} - 1\right) \cdot Q_i &- \mathcal{R} \cdot \left(\frac{Q_i}{m} + \frac{Y_0^{i+1}}{2}\right) - (\mathcal{R}-1) \cdot (Q_i + Y_0^{i+1}) \\ &= Y_0^{i+1} \cdot \left(2 - \frac{3\mathcal{R}}{2}\right) + \frac{1}{m} \cdot Q_i \cdot (4 - 3\mathcal{R}) = 0 \,. \end{aligned}$$

Case 3. $j = \frac{m-1}{2}$, this case needs to be considered only if m is odd, and $\frac{m-1}{2}$ jobs remain in the buffer after the assignment.

The invariant for j for time i is $\sum_{k=1}^{\frac{m-1}{2}} L_k^i \leq (\mathcal{R}-1) \cdot \frac{\frac{m-1}{2}}{m-\frac{m-1}{2}} \cdot Q_i = \frac{m^2-1}{3m^2+1} \cdot \frac{m-1}{m+1} \cdot Q_i = \frac{(m-1)^2}{3m^2+1} \cdot Q_i$, and therefore $\sum_{k=\frac{m+1}{2}}^{m} L_k^i \geq (1 - \frac{m^2-2m+1}{3m^2+1}) \cdot Q_i = \frac{2\left(m^2+m\right)}{3m^2+1} \cdot Q_i$. Since $L^i_{\frac{m+1}{2}} \leq L^i_{\frac{m+1}{2}+1} \leq \ldots \leq L^i_m$, we have $\sum_{k=\frac{m+1}{2}+1}^{m} L_k^i \geq \frac{\frac{m+1}{2}-1}{\frac{m+1}{2}} \cdot \frac{2\left(m^2+m\right)}{3m^2+1} \cdot Q_i = \frac{m-1}{m+1} \cdot \frac{2m(m+1)}{3m^2+1} \cdot Q_i = \frac{2m(m-1)}{3m^2+1} \cdot Q_i$. We get $\sum_{k=1}^{\frac{m+1}{2}} L_k^i \leq (1 - \frac{2m(m-1)}{3m^2+1}) \cdot Q_i = \frac{m^2+2m+1}{3m^2+1} \cdot Q_i$.

Therefore, it is sufficient to prove

$$\begin{aligned} Y_0^{i+1} + \frac{m^2+2m+1}{3m^2+1} \cdot Q_i - \mathcal{R} \cdot \text{OPT}_{i+1} - (\mathcal{R}-1) \cdot \frac{m-1}{m+1} \cdot Q_{i+1} \leq 0, \text{ i.e.,} \\ Y_0^{i+1} + \frac{m^2+2m+1}{3m^2+1} \cdot Q_i - \frac{4m^2}{3m^2+1} \cdot \text{OPT}_{i+1} - \frac{m^2-1}{3m^2+1} \cdot \frac{m-1}{m+1} \cdot Q_{i+1} \leq 0 \end{aligned}$$

We use $Q_{i+1} = Q_i + Y_0^{i+1}$ and $\text{OPT}_{i+1} \geq \frac{1}{m} \cdot (Q_i + \frac{m+1}{2} \cdot Y_0^{i+1})$ to get that

$$\begin{aligned} &Y_0^{i+1} + \frac{m^2+2m+1}{3m^2+1} \cdot Q_i - \frac{4m^2}{3m^2+1} \cdot \text{OPT}_{i+1} - \frac{m^2-2m+1}{3m^2+1} \cdot Q_{i+1} \\ &\leq Y_0^{i+1} + \frac{m^2+2m+1}{3m^2+1} \cdot Q_i - \frac{4m}{3m^2+1} \cdot (Q_i + \frac{m+1}{2} \cdot Y_0^{i+1}) - \frac{m^2-2m+1}{3m^2+1} \cdot \left(Q_i + Y_0^{i+1}\right) \\ &= \left(1 - \frac{2m \cdot (m+1)}{3m^2+1} - \frac{m^2-2m+1}{3m^2+1}\right) \cdot Y_0^{i+1} + \frac{m^2+2m+1-4m-\left(m^2-2m+1\right)}{3m^2+1} \cdot Q_i = 0. \end{aligned}$$

4.2 Proof of Lemma 6

We start with a short outline of the proof. We need to prove an invariant which states that the first machine is loaded relatively lightly. Thus, we only need to consider situations where this machine actually received some parts of jobs. We consider the following three times. The last time at which this machine received a part of a job, which means that the slot on the second machine was occupied completely at this time. The time strictly earlier, when the second machine received a part of a job (and the slot of the third machine is occupied completely), and a time strictly before this time when the third machine received a part of a job.

Since we already have an invariant regarding the total load of machines 5 and 6, we follow the steps of the algorithm starting from the time mentioned above, at which in fact the slot on the fourth machine was used completely. These three times in the execution are seen as key times, for which bounds on the optimal cost are calculated with respect to loads, and with respect to the largest job size, which is the size of the job in the buffer at this time. There are several cases, based on the relation between the sizes of at most six jobs, which are the three jobs mentioned above, and the jobs stored in the buffer at the times of their assignment.

To prove the invariant for a given value of i, where a job is stored in the buffer after i jobs have been assigned, we note that if $L_1^i = 0$ we are done, thus we assume $L_1^i > 0$.

Let A be a job which is just have been assigned by the algorithm. Consider the smallest index of a machine k ($0 \leq k \leq 6$) that received a non-empty part of A ($k = 0$ if the job is larger than the slots allocated for it). We say that job A **overflows** machine j for all machines j such that $k < j \leq 6$. That is, A overflows machine j if the slot allocated for A on machine j was fully used. Since all i jobs, which were assigned so far, were assigned successfully, no job could overflow machine 1. We say that the load of the m-th machine (i.e. the load of machine 6) is **full** after t jobs were assigned, if it equals to the maximum allowed load, i.e., it is equal to $\mathcal{R} \cdot \text{OPT}_t$. This condition holds if the t-th job ever assigned overflows the machine $m = 6$.

We present additional definitions which have a major role in the proof. Let v_3 be the last job among the first i assigned jobs, in the order of assignment of jobs (which is not necessarily the order of arrival of jobs) which overflows machine 2. Such a job must exist, otherwise, machine 1 would have remained empty. Let v_2 be the last job **strictly before** v_3 (in the order of assignment) that overflows machine 3 (and possibly also machine 2). This job must exist because otherwise machine 2 would have been empty at the time of assignment of v_3, and v_3 could not overflow it. Finally, let v_1 be the last job **strictly before** v_2 that overflows machine 4. Similarly to the above, this job also must exist.

Furthermore, we denote by u_k the job stored in the buffer at the time when v_k is being assigned, (for each $k = 1, 2, 3$). The job u_3 exists since the buffer contains a job after i jobs were assigned. We abuse notation to let u_k and v_k denote also the sizes of these jobs. For a job p, let $Q(p)$ denote the sum of sizes of jobs assigned strictly before p. Let $\text{OPT}(p)$ denote the optimal cost at the time of assignment of p (including the job stored in the buffer, if exists). For simplicity of notation, we let $L_j = L_j^i$ for $1 \leq j \leq 6$.

Consider the moment when v_1 is assigned and recall that it overflows machine 4. From this fact it follows that just after the assignment of v_1, the load of machine 4 will be equal to the previous load of machine 5, the load of machine 5 will be equal to the previous load of machine 6, and the load of machine 6 will be full. Using Lemma 5 for the time just before v_1 is assigned, we have the following lower bound on the sum of the loads of the last three machines just after v_1 is assigned: $\frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1)$.

Since v_1 is the last job before v_2 that overflows machine 4, each job assigned strictly after v_1 and strictly before v_2 does not overflow machine 4. This means that all parts of these jobs are assigned only to the last three machines. Denote the sum of the sizes of these job by S_{12}. Then, just before v_2 is assigned, the sum of the loads of the last three machines is at least

$$\frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1) + S_{12} \, .$$

Let us consider the moment when v_2 is assigned, and recall that it overflows machine 3. Similarly to the previous case, just after the assignment of v_2, the loads of each machine j ($3 \leq j \leq 5$) will be equal to the previous load of machine $j + 1$, and the load of machine 6 will be again full, that means that the total load of the last four machines is at least

$$\frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1) + S_{12} + \frac{19}{14} \cdot \text{OPT}(v_2) \, .$$

Then the jobs assigned between v_2 and v_3, not including these jobs (denote the total size of those jobs by S_{23}) are assigned completely to the last four machines, and we get that just before v_3 is assigned, the sum of the loads of the last four machines is at least

$$\frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1) + S_{12} + \frac{19}{14} \cdot \text{OPT}(v_2) + S_{23}.$$

Consider the moment when v_3 is assigned, and recall that it overflows machine 2. We get that just after v_3 is assigned, the total load of the last five machines is at least:

$$\frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1) + S_{12} + \frac{19}{14} \cdot \text{OPT}(v_2) + S_{23} + \frac{19}{14} \cdot \text{OPT}(v_3) \ .$$

Finally, let S_{34} denote the total size of all jobs assigned strictly after v_3 up to the i-th assigned job. After assigning the those remaining jobs up to the i-th job, we get that since no further job overflows machine 2, the total load of the last five machines satisfies

$$\begin{aligned} &L_2 + L_3 + L_4 + L_5 + L_6 \\ &\geq \frac{19}{42} \cdot Q(v_1) + \frac{19}{14} \cdot \text{OPT}(v_1) + S_{12} + \frac{19}{14} \cdot \text{OPT}(v_2) + S_{23} + \frac{19}{14} \cdot \text{OPT}(v_3) + S_{34}. \end{aligned}$$

For simplicity of notation, let $Q = Q(v_1)$. We have the next inequalities:

$$\begin{aligned} \text{OPT}(v_1) &= \max\left\{\frac{Q + v_1 + u_1}{6}, u_1\right\} \ , \\ \text{OPT}(v_2) &= \max\left\{\frac{Q + v_1 + S_{12} + v_2 + u_2}{6}, u_2\right\} \ , \\ \text{OPT}(v_3) &= \max\left\{\frac{Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + u_3}{6}, u_3\right\} \ . \end{aligned}$$

In the technical part of the proof, instead of the maximum values on the right hand side of these bounds on $\text{OPT}(v_k)$, we will use convex combinations of the two terms in each maximum. We also use $v_i \leq u_i$. We need to show that

$$L_1 \leq \frac{1}{14} \cdot (Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + S_{34}) \ .$$

Now we separately treat several cases as follows.

Case 1. $v_2 \leq u_1$ and $v_3 \leq u_2$. Then, (also using $v_1 \leq u_1$ and $v_3 \leq u_3$), we get

$$\begin{aligned} &L_2 + L_3 + L_4 + L_5 + L_6 \\ &\geq \frac{19}{42} \cdot Q + \frac{57}{42} \cdot \left(\frac{19}{57} \cdot \frac{Q + v_1 + v_2}{6} + \frac{19}{57} \cdot v_1 + \frac{19}{57} \cdot v_2\right) + S_{12} \\ &+ \frac{57}{42} \cdot \left(\frac{44}{57} \cdot \frac{Q + v_1 + v_2 + v_3}{6} + \frac{13}{57} \cdot v_3\right) + S_{23} \\ &+ \frac{57}{42} \cdot \left(\frac{57}{57} \cdot \frac{Q + v_1 + v_2 + 2v_3}{6} + \frac{0}{57} \cdot v_3\right) + S_{34} \\ &\geq \frac{39}{42} \cdot Q + \frac{39}{42} \cdot v_1 + \frac{39}{42} \cdot v_2 + \frac{39}{42} \cdot v_3 + S_{12} + S_{23} + S_{34} \ . \end{aligned}$$

It follows that

$$\begin{aligned} L_1 &= Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + S_{34} - (L_2 + L_3 + L_4 + L_5 + L_6) \\ &\leq \frac{1}{14} \cdot (Q + v_1 + v_2 + v_3) \ . \end{aligned}$$

Case 2. $v_2 \leq u_1$ and $v_3 > u_2$. Then in the second maximum we cannot substitute u_2 by v_3. On the other hand, from the condition follows, that u_2 is already have been assigned when we are just assigning v_3, thus S_{23} is not empty, and $S_{23} \geq u_2 \geq v_2$. We also apply $u_2 \geq v_2$ and $u_1 \geq v_1$ to get

$$\begin{aligned} &L_2 + L_3 + L_4 + L_5 + L_6 \geq \frac{19}{42} \cdot Q \\ &+ \frac{57}{42} \cdot \left(\frac{36}{57} \cdot \frac{Q + v_1 + v_2}{6} + \frac{19}{57} \cdot v_1 + \frac{2}{57} \cdot v_2 \right) + S_{12} \\ &+ \frac{57}{42} \cdot \left(\frac{57}{57} \cdot \frac{Q + v_1 + 2 \cdot v_2}{6} + \frac{0}{57} \cdot v_2 \right) + S_{23} \\ &+ \frac{57}{42} \cdot \left(\frac{27}{57} \cdot \frac{Q + v_1 + 2 \cdot v_2 + 2 \cdot v_3}{6} + \frac{30}{57} \cdot v_3 \right) + S_{34} \\ &= \frac{39}{42} \cdot Q + \frac{39}{42} \cdot v_1 + \frac{36}{42} \cdot v_2 + \frac{39}{42} \cdot v_3 + S_{12} + S_{23} + S_{34} \ . \end{aligned}$$

Then it follows that

$$\begin{aligned} L_1 &= Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + S_{34} - (L_2 + L_3 + L_4 + L_5 + L_6) \\ &\leq \frac{3}{42} \cdot Q + \frac{3}{42} \cdot v_1 + \frac{6}{42} \cdot v_2 + \frac{3}{42} \cdot v_3 \\ &\leq \frac{1}{14} \cdot (Q + v_1 + v_2 + S_{23} + v_3) \, . \end{aligned}$$

Case 3. $v_2 > u_1$ and $v_3 > u_2$. Similarly to the previous case, we have $S_{12} \geq u_1 \geq v_1$ and $S_{23} \geq u_2 \geq v_2$. In addition, we apply $u_2 \geq v_2$ to get

$$\begin{aligned} &L_2 + L_3 + L_4 + L_5 + L_6 \geq \frac{19}{42} \cdot Q \\ &+ \frac{57}{42} \cdot \left(\frac{48}{57} \cdot \frac{Q + 2v_1}{6} + \frac{9}{57} \cdot v_1 \right) + S_{12} \\ &+ \frac{57}{42} \cdot \left(\frac{45}{57} \cdot \frac{Q + 2v_1 + 2v_2}{6} + \frac{12}{57} \cdot v_2 \right) + S_{23} \\ &+ \frac{57}{42} \cdot \left(\frac{27}{57} \cdot \frac{Q + 2v_1 + 2v_2 + 2v_3}{6} + \frac{30}{57} \cdot v_3 \right) + S_{34} \\ &\geq \frac{39}{42} \cdot Q + \frac{36}{42} \cdot v_1 + \frac{36}{42} \cdot v_2 + \frac{39}{42} \cdot v_3 + S_{12} + S_{23} + S_{34} \ . \end{aligned}$$

Then it follows that

$$\begin{aligned} L_1 &= Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + S_{34} - (L_2 + L_3 + L_4 + L_5 + L_6) \\ &\leq \frac{3}{42} \cdot Q + \frac{6}{42} \cdot v_1 + \frac{6}{42} \cdot v_2 + \frac{3}{42} \cdot v_3 \\ &\leq \frac{1}{14} \cdot (Q + v_1 + S_{12} + v_2 + S_{23} + v_3) \ . \end{aligned}$$

Case 4. $v_2 > u_1$ and $v_3 \le u_2$. Since u_1 is already have been assigned when we are just assigning v_2, thus S_{12} is not empty, and $S_{12} \ge u_1 \ge v_1$. We also apply $u_1 \ge v_1$, and $u_2 \ge v_2$ to get

$$\begin{aligned}
L_2 + L_3 + L_4 + L_5 + L_6 &\ge \frac{19}{42} \cdot Q \\
&+ \frac{57}{42} \cdot \left(\frac{57}{57} \cdot \frac{Q + 2v_1}{6} + \frac{0}{57} \cdot v_1 \right) + S_{12} \\
&+ \frac{57}{42} \cdot \left(\frac{6}{57} \cdot \frac{Q + 2v_1 + v_2 + v_3}{6} + \frac{32}{57} \cdot v_2 + \frac{19}{57} \cdot v_3 \right) + S_{23} \\
&+ \frac{57}{42} \cdot \left(\frac{57}{57} \cdot \frac{Q + 2v_1 + v_2 + 2 \cdot v_3}{6} + \frac{0}{57} \cdot v_3 \right) + S_{34} \\
&\ge \frac{39}{42} \cdot Q + \frac{36}{42} \cdot v_1 + \frac{39}{42} \cdot v_2 + \frac{39}{42} \cdot v_3 + S_{12} + S_{23} + S_{34} \, .
\end{aligned}$$

Then it follows that

$$\begin{aligned}
L_1 &= Q + v_1 + S_{12} + v_2 + S_{23} + v_3 + S_{34} - (L_2 + L_3 + L_4 + L_5 + L_6) \\
&\le \frac{3}{42} \cdot Q + \frac{6}{42} v_1 + \frac{3}{42} \cdot v_2 + \frac{3}{42} \cdot v_3 \le \frac{1}{14} \cdot (Q + v_1 + S_{12} + v_2 + v_3) \, .
\end{aligned}$$

4.3 Proof of Lemma 9

To prove the claim, use induction in all cases, except for one case where we need to prove the invariant for $j = K$. In this case we perform an analysis similar to that of the proof of Lemma 6 in the sense that we consider certain times at which machines received parts of jobs. In this case, since we are dealing with an arbitrary number of machines, the number of key times is a function of m, namely it is $m - K$. On the other hand, if the buffer contains K jobs after the assignment of a job, then these are exactly the largest K jobs in the input, that is, the first K jobs of the input. This reduces the number of cases in the bounds on the optimal cost, since the size of the largest job is simply Z_1. The cases here depend on the first time at which the lower bound on the value of the optimal cost exceeds Z_1.

We prove the claim by induction on i. For $i = 0$, $L_k^i = 0$ holds for all k, and we get $\sum_{k=1}^{j} Z_i \le j \cdot Z_1 \le j \cdot \mathcal{R} \cdot \text{OPT}_0$, since $\text{OPT}_0 \ge Z_1$.

Assume now that the required invariants hold for some $i - 1 \ge 0$. Let ℓ be the number of jobs in the buffer after the i-th job has been assigned. By Lemma 10 this job was assigned successfully.

If none of machines $1, 2, \ldots, \ell$ received any parts of the assigned job, then $L_k^i = L_k^{i-1}$ for $1 \le k \le \ell$, and thus all the invariants hold for i, since $\text{OPT}_i \ge \text{OPT}_{i-1}$. Otherwise, let $1 \le z \le \ell$ denote the minimum index of a machine which received a non-zero part of the job. Similarly to the above, for any $k < z$, $L_k^i = L_k^{i-1}$, and thus for any $j < z$, the j-th invariant holds at time $i - 1$.

We therefore need to prove the invariants at time i for $z \le j \le \ell$. We have $z \le \ell \le K \le m - 1$. If after the assignment of the i-th job, the buffer contains $\ell = K$ jobs, we will consider the case $j = K$ separately. In this latter case the buffer contained K jobs before this assignment as well. The assigned job is not one of the initial K jobs, which are still stored in the buffer.

We first consider the case $z \le j \le \min\{\ell, K-1\}$. We have $\sum_{k=1}^{j} L_k^i + \sum_{k=1}^{j} Z_k = Q_i - \sum_{k=j+1}^{m} L_k^i + \sum_{k=1}^{j} Z_k$. After the assignment of the i-th job, the new load of machine $z < p \le m - 1$ is L_{p+1}^{i-1}, and the new load of machine m is $\mathcal{R} \cdot \text{OPT}_i$. Since $j \ge z$, we can use this equality for $j + 1 \le k \le m$.

Note that the number of jobs in the buffer before the assignment is at least $j+1$. If the number of jobs in the buffer after the i-th assignment is K, then the buffer was full also before the assignment and using $j < K$ the last property holds. Otherwise, the assigned job was from the buffer and the number of jobs before the assignment was $\ell + 1$. Therefore, in both these cases we can use the $(j+1)$-th invariant for time $i-1$.

We get at most

$$Q_i - \sum_{k=j+2}^{m} L_k^{i-1} - \mathcal{R} \cdot \text{OPT}_i + \sum_{k=1}^{j} Z_k = Q_i - Q_{i-1} + \sum_{k=1}^{j+1} L_k^{i-1} - \mathcal{R} \cdot \text{OPT}_i + \sum_{k=1}^{j} Z_k$$
$$\leq X_i + (j+1) \cdot \mathcal{R} \cdot \text{OPT}_{i-1} - \sum_{k=1}^{j+1} Z_k - \mathcal{R} \cdot \text{OPT}_i + \sum_{k=1}^{j} Z_k \leq j \cdot \mathcal{R} \cdot \text{OPT}_i + X_i - Z_{j+1}$$

(since $\text{OPT}_{i-1} \leq \text{OPT}_i$). If it left to show $X_i \leq Z_{j+1}$. If X_i is not one of the K first jobs in the sequence, then this is clear. Otherwise, since we are dealing with machine j, the buffer contains at least j jobs to be assigned later. Therefore, the current job has a size in $\{Z_{j+1}, \ldots, Z_{m-1}\}$, and thus $X_i \leq Z_{j+1}$.

We next consider the case $j = K$, for which we need to prove

$$\sum_{k=1}^{K} L_k^i + \sum_{k=1}^{K} Z_k \leq K \cdot \mathcal{R} \cdot \text{OPT}_i \ . \tag{1}$$

For a job p, we use the notations $\text{OPT}(p)$ and $Q(p)$ as in the proof of Lemma 6. In addition, we let $L_j = L_j^i$. Since $z \leq K$, the i-th assigned job overflows machine $K+1$. We denote this job by v_{m-K}, and define a sequence of $m-K$ jobs as follows. The job v_j is the last job assigned strictly before the job v_{j+1}, which overflows machine $m-j+1$. For a job v_k we again let v_k denote both the job and its size.

For every $1 \leq j \leq m-K-1$, We let $S_{j,j+1}$ denote the total size of jobs assigned strictly after v_j and strictly before v_{j+1}. We prove by induction that at the time just after the job v_j is assigned, the sum of loads of machines $m-j+1, \ldots, m$ is at least

$$\mathcal{R} \cdot OPT(v_1) + S_{1,2} + \mathcal{R} \cdot OPT(v_2) + S_{2,3} + \ldots + S_{j-1,j} + \mathcal{R} \cdot OPT(v_j).$$

To prove this, we start with the base case $j = 1$. Since v_1 overflows machine m, its load just after the assignment of v_1 must be $\mathcal{R} \cdot OPT(v_1)$.

Assume next that the claim holds for some j, such that $1 \leq j \leq m-K+1$. To prove the property for v_{j+1}, consider the jobs assigned strictly after v_j and strictly before v_{j+1}.

By definition, since v_j was the last job assigned strictly before v_{j+1} that overflows machine $m-j+1$, all parts each job which is assigned strictly after v_j and strictly before v_{j+1} are assigned to machines $m-j+1, \ldots, m$. Therefore, at the time just before v_{j+1} is assigned, the total load of machines $m-j+1, \ldots, m$ is their total load just after the assignment of v_j plus $S_{j,j+1}$, that is, it is at least

$$\mathcal{R} \cdot OPT(v_1) + S_{1,2} + \mathcal{R} \cdot OPT(v_2) + S_{2,3} + \ldots + S_{j-1,j} + \mathcal{R} \cdot OPT(v_j) + S_{j,j+1}.$$

Consider now the moment a which v_{j+1} is assigned. Since it overflows machine $m-j$, the load of each machine $m-j \leq t \leq m-1$ becomes the previous load of machine $t+1$, and the load of machine m becomes $\mathcal{R}\text{OPT}(v_{j+1})$. Therefore, just after the assignment of v_{j+1}, the total load of machines $m-j, \ldots, m$ is at least

$$\mathcal{R} \cdot OPT(v_1) + S_{1,2} + \mathcal{R} \cdot OPT(v_2) + S_{2,3} + \ldots + S_{j-1,j} + \mathcal{R} \cdot OPT(v_j) + S_{j,j+1} + \mathcal{R} \cdot \text{OPT}(v_{j+1}) \ ,$$

as required.

We get that when the i-th jobs, v_{m-K}, has been just assigned, it holds that

$$\sum_{k=K+1}^{m} L_k \geq \mathcal{R} \cdot OPT(v_1) + S_{1,2} + \mathcal{R} \cdot OPT(v_2) + S_{2,3} + \ldots + S_{m-K-1,m-K} + \mathcal{R} \cdot OPT(v_{m-K}).$$

We use the notations $Z = Z_1 + \ldots + Z_K$ and $Q = Q(v_1)$. Then we have the next inequalities for all j:

$$OPT(v_j) = \max \left\{ \frac{Q + Z + v_1 + \sum_{k=2}^{j} (S_{k-1,k} + v_k)}{m}, Z_1 \right\}. \tag{2}$$

Let $S = S_{1,2} + S_{2,3} + \ldots + S_{m-K-1,m-K}$, and $V = \sum_{j=1}^{m-K} v_j$. Since $\sum_{j=1}^{K} L_j = \sum_{j=1}^{m} L_j - \sum_{j=K+1}^{m} L_j = Q + V + S - \sum_{j=K+1}^{m} L_j$, and $\text{OPT}_i = \text{OPT}(v_{m-K})$, to prove (1), it suffices to prove that the following holds

$$Q + Z + V + S \leq \mathcal{R} \cdot (K \cdot \text{OPT}(v_{m-K}) + \text{OPT}(v_1) + \text{OPT}(v_2) + \ldots + \text{OPT}(v_{m-K})) + S\ ,$$

or equivalently,

$$Q + Z + V \leq \mathcal{R} \cdot (K \cdot \text{OPT}(v_{m-K}) + \text{OPT}(v_1) + \text{OPT}(v_2) + \ldots + \text{OPT}(v_{m-K}))\ . \tag{3}$$

We next replace the values $\text{OPT}(v_i)$ on the right hand side by the bounds of (2). Note that omitting the $S_{k-1,k}$ terms from the formulas, we get a stronger statement, and we will prove this stronger statement. Let $\text{MAX}(v_i)$ denote the modified values of $\text{OPT}(v_i)$ after deleting the terms $S_{k-1,k}$. Thus we will show that

$$Q + Z + V \leq \mathcal{R} \cdot (K \cdot \text{MAX}(v_{m-K}) + \text{MAX}(v_1) + \text{MAX}(v_2) + \ldots + \text{MAX}(v_{m-K})) \tag{4}$$

holds.

If $\text{MAX}(v_{m-K}) = Z_1$ then we show that statement (4) holds trivially. We get $Q + Z + \sum_{k=1}^{m-K} v_k \leq m \cdot Z_1$, which implies $Q + Z + \sum_{k=1}^{j} v_k \leq m \cdot Z_1$ for all $1 \leq j \leq m - K$, and in particular $Z_1 \geq \frac{Q+Z+V}{m}$. Thus, $\text{MAX}(v_1) = \text{MAX}(v_2) = \ldots = \text{MAX}(v_{m-K}) = Z_1$, and we need to prove $Q + V + Z \leq \mathcal{R} \cdot m \cdot Z_1$, which holds for any $\mathcal{R} \geq 1$.

Therefore, let $1 \leq \alpha \leq m - K$ be the minimal index for which holds that $\text{MAX}(v_\alpha) > Z_1$. The next inequalities follow from the definition of α:

$$\frac{Q + Z + v_1 + v_2 + \ldots + v_{\alpha-1}}{m} \leq Z_1, \text{ and} \tag{5}$$

$$\frac{Q + Z + v_1 + v_2 + \ldots + v_{\alpha-1} + v_\alpha}{m} > Z_1, \tag{6}$$

where the first inequality holds only if $\alpha > 1$. Now let us substitute the values $\text{MAX}(v_j)$ in the right hand side of (4). We would like to show that

$$Q + Z + V \leq \mathcal{R} \cdot \sum_{j=1}^{\alpha-1} Z_1 + \mathcal{R} \cdot \left(\sum_{j=\alpha}^{m-K} \frac{Q + Z + v_1 + \ldots + v_j}{m} + K \cdot \frac{Q + Z + v_1 + \ldots + v_{m-K}}{m} \right)\ . \tag{7}$$

Let us denote $L = m - \alpha + 1$. From the condition $1 \leq \alpha \leq m - K$ it follows that $K + 1 \leq L \leq m$. Then inequality (7) can be rewritten as

$$Q + Z + V \leq \mathcal{R} \cdot (\alpha - 1) \cdot Z_1 + \frac{\mathcal{R} \cdot L}{m} \cdot (Q + Z + v_1 + \ldots + v_\alpha) \tag{8}$$
$$+ \frac{\mathcal{R} \cdot (L-1)}{m} \cdot v_{\alpha+1} + \frac{\mathcal{R} \cdot (L-2)}{m} \cdot v_{\alpha+2} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \cdot v_{m-K} \, .$$

We prove (8) by distinguishing two cases as follows:

Case 1. $\mathcal{R} \cdot L \geq m$. The condition means that the coefficient of $(Q + Z + v_1 + \ldots + v_\alpha)$ is at least 1. Then we decrease both sizes by $(Q + Z + v_1 + \ldots + v_\alpha)$. Then the remainder, $(\frac{\mathcal{R} \cdot L}{m} - 1) \cdot (Q + Z + v_1 + \ldots + v_\alpha)$ on the right hand side is non-negative, using the definition of α and inequality (6), we estimate this remainder from below by $\left(\frac{\mathcal{R} \cdot L}{m} - 1\right) \cdot m \cdot Z_1$, so the coefficient of Z_1 becomes $\mathcal{R} \cdot (\alpha - 1) + \mathcal{R} \cdot L - m = (\mathcal{R} - 1) \cdot m \geq 0$.

Thus it suffices to prove that

$$v_{\alpha+1} + \ldots + v_{m-K} \leq (\mathcal{R} - 1) \cdot m \cdot Z_1$$
$$+ \frac{\mathcal{R} \cdot (L-1)}{m} \cdot v_{\alpha+1} + \frac{\mathcal{R} \cdot (L-2)}{m} \cdot v_{\alpha+2} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \cdot v_{m-K}.$$

If $\frac{\mathcal{R} \cdot (K+1)}{m} \geq 1$, then all coefficients of v_j on the right hand side are no smaller than those of the left hand side, and in addition, the coefficient of Z_1 is non-negative, so we are done. Otherwise, since $\mathcal{R} \cdot L \geq m$, there exists a value $\alpha' \geq \alpha$ and $L' = m - \alpha' + 1$ for which $\mathcal{R} \cdot L' \geq m$ and $\mathcal{R} \cdot (L' - 1) < m$. It is sufficient to consider the sum of all coefficients on the right hand side (neglecting those of $v_{\alpha+1}, \ldots, v_{\alpha'}$) and show that it is no smaller than the sum of coefficients on the left hand side (neglecting those of $v_{\alpha+1}, \ldots, v_{\alpha'}$ here as well). The claim will follow by using the fact that Z_1 is the maximum job size, and the fact that all coefficients are non-negative.

Thus we need to show that

$$\begin{aligned} m - K - \alpha' &\leq (\mathcal{R} - 1) \cdot m + + \frac{\mathcal{R} \cdot (L' - 1)}{m} + \frac{\mathcal{R} \cdot (L-2)}{m} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \\ &= \mathcal{R} \cdot m - m + \frac{\mathcal{R}}{m} \cdot \left[\left(L' - 1\right) + \ldots + (K+1) \right] \\ &= \mathcal{R} \cdot m - m + \frac{\mathcal{R}}{m} \cdot \frac{(L' + K) \cdot (m - K - \alpha')}{2} \, . \end{aligned}$$

By reordering we get

$$2m - K - \alpha' \leq \mathcal{R} \cdot m + \frac{\mathcal{R}}{m} \cdot \frac{(L' + K) \cdot (m - K - \alpha')}{2}, \text{ i.e.,}$$
$$2m \cdot \left(2m - K - \alpha'\right) \leq \mathcal{R} \cdot \left[2m^2 + (L' + K) \cdot \left(m - K - \alpha'\right) \right]$$

substituting $L = m - \alpha + 1$ and $L' = m - \alpha' + 1$, we get

$$2m \cdot \left(2m - K - \alpha'\right) \leq \mathcal{R} \cdot \left[2m^2 + (m - \alpha' + 1 + K) \cdot \left(m - K - \alpha'\right) \right] \, .$$

We next substitute $\mu = m - K - \alpha'$, i.e. $\alpha' = m - K - \mu$, (then from the condition $1 \le \alpha' \le m - K$ it follows that $0 \le \mu \le m - K - 1$), and we get

$$2m \cdot (m + \mu) \le \mathcal{R} \cdot \left[2m^2 + (m - (m - K - \mu) + 1 + K) \cdot \mu\right]$$

that is, $$2m \cdot (m + \mu) \le \mathcal{R} \cdot \left[2m^2 + (2K + \mu + 1) \cdot \mu\right]$$

and finally $$\frac{2m \cdot (m + \mu)}{2m^2 + (2K + \mu + 1) \cdot \mu} \le \mathcal{R}$$

which is one of the terms in the definition of $\mathcal{R}$ if $\mu > 1$. If $\mu = 0$ then this condition clearly holds as well. Note that the proof is valid for $\alpha = 1$, since (5) was not used explicitely.

Case 2. $\mathcal{R} \cdot L < m$. Note that if $\alpha = 1$ we get $L = m - \alpha + 1 = m$, then $\mathcal{R} \cdot L = \mathcal{R} \cdot m \ge m$, thus $\alpha > 1$, and (5) can be used. The condition means that the coefficient of $(Q + Z + v_1 + \ldots + v_\alpha)$ is strictly smaller than 1. Then inequality (8) can be rewritten as

$$\begin{aligned} Q + Z + V \le{}& \mathcal{R} \cdot (\alpha - 1) \cdot Z_1 + \frac{\mathcal{R} \cdot L}{m} \cdot (Q + Z + v_1 + \ldots + v_{\alpha-1}) \\ &+ \frac{\mathcal{R} \cdot L}{m} \cdot v_\alpha + \frac{\mathcal{R} \cdot (L-1)}{m} \cdot v_{\alpha+1} + \frac{\mathcal{R} \cdot (L-2)}{m} \cdot v_{\alpha+2} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \cdot v_{m-K} \,. \end{aligned}$$

We again decrease both sizes by $(Q + Z + v_1 + \ldots + v_{\alpha-1})$. Since the coefficient of this expression on the right hand size is strictly smaller than 1, the new coefficient $\frac{\mathcal{R} \cdot L}{m} - 1$ of $(Q + Z + v_1 + \ldots + v_{\alpha-1})$, is negative. We use inequality (5), that is, $m \cdot Z_1 \ge Q + Z + v_1 + \ldots + v_{\alpha-1}$. As a result, the coefficient of Z_1 decreases by $m \cdot \left(1 - \frac{\mathcal{R} \cdot L}{m}\right) = m - \mathcal{R} \cdot L$, and becomes $(\mathcal{R} - 1) \cdot m \ge 0$.

Thus it suffices to prove that

$$\begin{aligned} v_\alpha + \ldots + v_{m-K} \le{}& (\mathcal{R} - 1) \cdot m \cdot Z_1 \\ &+ \frac{\mathcal{R} \cdot L}{m} \cdot v_\alpha + \frac{\mathcal{R} \cdot (L-1)}{m} \cdot v_{\alpha+1} + \frac{\mathcal{R} \cdot (L-2)}{m} \cdot v_{\alpha+2} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \cdot v_{m-K} \,. \end{aligned}$$

Here the coefficient of each v_j on the right hand side is smaller than 1, so it is only needed to show that the sum of the coefficients in the left hand side (it is $m - K - \alpha + 1$) is not greater than the sum of the coefficients in the right hand side. Thus we would like to show that

$$\begin{aligned} m - K - \alpha + 1 &\le (\mathcal{R} - 1) \cdot m + \frac{\mathcal{R} \cdot L}{m} + \frac{\mathcal{R} \cdot (L-1)}{m} + \frac{\mathcal{R} \cdot (L-2)}{m} + \ldots + \frac{\mathcal{R} \cdot (K+1)}{m} \\ &= \mathcal{R} \cdot m - m + \frac{\mathcal{R}}{m} \cdot [L + (L-1) + \ldots + (K+1)] \\ &= \mathcal{R} \cdot m - m + \frac{\mathcal{R}}{m} \cdot \frac{(L + K + 1) \cdot (m - K - \alpha + 1)}{2} \,. \end{aligned}$$

By reordering we get

$$2m - K - \alpha + 1 \le \mathcal{R} \cdot m + \frac{\mathcal{R}}{m} \cdot \frac{(L + K + 1) \cdot (L - K)}{2} \,,$$

or equivalently $$2m \cdot (2m - K - \alpha + 1) \le \mathcal{R} \cdot \left[2m^2 + (L + K + 1) \cdot (L - K)\right] \,.$$

Next, substituting $L = m - \alpha + 1$, we get

$$2m \cdot (2m - K - \alpha + 1) \le \mathcal{R} \cdot \left[2m^2 + (m - \alpha + K + 2) \cdot (m - K - \alpha + 1)\right]$$

and finally substituting $\mu = m-K-\alpha+1$, i.e. $\alpha = m-K-\mu+1$, then from the condition $2 \leq \alpha \leq m-K$ it follows that $1 \leq \mu \leq m-K-1$, and we get

$$2m \cdot (m+\mu) \leq \mathcal{R} \cdot \left[2m^2 + (m-(m-K-\mu+1)+K+2) \cdot \mu\right]$$

that is,

$$2m \cdot (m+\mu) \leq \mathcal{R} \left[2m^2 + (2K+\mu+1) \cdot \mu\right]$$

and finally

$$\frac{2m \cdot (m+\mu)}{2m^2 + (2K+\mu+1) \cdot \mu} \leq \mathcal{R}$$

which holds due to the definition of $\mathcal{R}$.

5 Conclusion

We studied preemptive scheduling with reordering and showed that a buffer of size $\Theta(m)$ is necessary and sufficient to achieve the best competitive ratios for both general sequences and for non-increasing sequences. All the algorithms do not use idle time, which is not helpful in the case of identical machines.

One direction for future research is to find the tight competitive ratio for every pair K, m of a buffer size and number of machines. This goal is already achieved here for the case of non-increasing job sequences.

An additional direction is to generalize the results for uniformly related machines. Note that the methods of [5], which allow to design algorithms of optimal competitive ratio for many semi-online variants, do not supply a solution for the model studied here. It is not difficult to see that for two uniformly related machines, similarly to two identical machines, a buffer would not reduce the competitive ratio. We conjecture that for any $m > 2$ and any speed combination, the usage of a reordering buffer can decrease the competitive ratio.

The algorithms considered here are deterministic. Allowing randomization would not be helpful since even though the lower bounds are stated deterministically, all the lower bounds of Section 3 can be extended for randomized algorithms by considering expected loads of machines rather than the loads.

References

[1] B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51(5):219–222, 1994.

[2] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18(3):127–131, 1995. Also in ESA 1994.

[3] Gy. Dósa and L. Epstein. Online scheduling with a buffer on related machines. *Journal of Combinatorial Optimization*. To appear, DOI: 10.1007/s10878-008-9200-y.

[4] T. Ebenlendr, W. Jawor, and J. Sgall. Preemptive online scheduling: Optimal algorithms for all speeds. *Algorithmica*, 53(4):504-522, 2009.

[5] T. Ebenlendr and J. Sgall. Semi-online preemptive scheduling: One algorithm for all variants. In *Proc. of the 26th Annual Symposium on Theoretical Aspects of Computer Science (STACS2009)*, pages 349–360, 2009.

[6] 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.

[7] L. Epstein. Optimal preemptive on-line scheduling on uniform processors with non-decreasing speed ratios. *Operations Research Letters*, 29(2):93–98, 2001. Also in STACS 2001.

[8] 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.

[9] 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.

[10] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, 26(1):17–22, 2000.

[11] E. Horwath, E. C. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32–43, 1977.

[12] H. Kellerer, V. Kotov, M. G. Speranza, and Zs. Tuza. Semi online algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

[13] S. Li, Y. Zhou, G. Sun, and G. Chen. Study on parallel machine scheduling problem with buffer. In *Proc. of the 2nd International Multisymposium on Computer and Computational Sciences (IMSCCS2007)*, pages 278–281, 2007.

[14] S. Seiden. Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, 262(1-2):437–458, 2001.

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

[16] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997.

[17] J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, editors, *Online Algorithms - The State of the Art*, chapter 9, pages 196–231. Springer, 1998.

[18] J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Operations Research Letters*, 23(3-5):113–116, 1998.

[19] G. Zhang. A simple semi on-line algorithm for $P2//C_{\max}$ with a buffer. *Information Processing Letters*, 61(3):145–148, 1997.