MATHEMATICS OF OPERATIONS RESEARCH
Vol. 34, No. 2, May 2009, pp. 481–498
ISSN 0364-765X | EISSN 1526-5471 | 09 | 3402 | 0481

DOI 10.1287/moor.1090.0381

Online Scheduling with Bounded Migration

Peter Sanders
Fakultät für Informatik, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany,
sanders@ira.uka.de

Naveen Sivadasan
Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany,
ns@mpi-sb.mpg.de

Martin Skutella
TU Berlin, Institut für Mathematik, 10623 Berlin, Germany,
skutella@math.tu-berlin.de

Consider the classical online scheduling problem, in which jobs that arrive one by one are assigned to identical parallel machines with the objective of minimizing the makespan. We generalize this problem by allowing the current assignment to be changed whenever a new job arrives, subject to the constraint that the total size of moved jobs is bounded by some constant times the size of the arriving job. This constant is called the migration factor. For small migration factors, we obtain several simple online algorithms with constant competitive ratio. We also present a linear time "online approximation scheme," that is, a family of online algorithms with competitive ratio arbitrarily close to 1 and constant migration factor.

Key words: scheduling; approximation; online algorithm; sensitivity analysis
MSC2000 subject classification: Primary: 90C27, 90C31, 68Q25; secondary: 90C10, 68M20
OR/MS subject classification: Primary: production/scheduling, sequencing, multiple machine; secondary: integer programming
History: Received March 16, 2007; revised May 29, 2008 and November 28, 2008. Published online in *Articles in Advance* April 17, 2009.

1. Introduction. One of the most fundamental scheduling problems asks for an assignment of jobs to m identical parallel machines so as to minimize the makespan. (The makespan is the completion time of the last job that finishes in the schedule; it also equals the maximum machine load.) In the standard classification scheme of Graham et al. [16], this scheduling problem is denoted by $P \,\|\, C_{\max}$ and it is well known to be strongly NP-hard (Garey and Johnson [13]).

The *offline* variant of this problem assumes that all jobs are known in advance, whereas in the *online* variant the jobs are incrementally revealed by an adversary, and the online algorithm can only choose the machine for the new job without being allowed to move other jobs. Note that dropping this radical constraint on the online algorithm yields the offline situation.

We study a natural generalization of both offline and online problems. Jobs arrive incrementally, but upon arrival of a new job j, we are allowed to migrate *some* previous jobs to other machines. The total size of the migrated jobs, however, must be bounded by βp_j where p_j is the size of the new job. For *migration factor* $\beta = 0$ we get the online setting, and for $\beta = \infty$ we get the offline setting.

For an offline optimization problem, an *approximation algorithm* efficiently (in polynomial time) constructs schedules whose values are within a constant factor α of the optimum solution value. The number α is called *performance guarantee* or *performance ratio* of the approximation algorithm. A family of polynomial time approximation algorithms with performance guarantee $1 + \epsilon$ for all fixed $\epsilon > 0$ is called a *polynomial time approximation scheme* (PTAS).

In a similar way, *competitive analysis* evaluates solutions computed in the online setting. An online algorithm achieves *competitive ratio* α if it always maintains solutions whose objective values are within a factor α of the offline optimum. Here, in contrast to offline approximation results, the achievable values α are not determined by limited computing power, but by the apparent lack of information about parts of the input that will only be revealed in the future. As a consequence, for all interesting classical online problems, it is rather easy to come up with lower bounds that create a gap between the best possible competitive ratio α and 1. In particular, it is usually impossible to construct a family of $(1 + \epsilon)$-competitive online algorithms for such problems.

Related work. For the online machine-scheduling problem, Graham's *list-scheduling* algorithm keeps the makespan within a factor $2 - 1/m$ of the offline optimum (Graham [14]): Schedule a newly arriving job on a least loaded machine. It can also easily be seen that this bound is tight. For the offline setting, Graham [15] shows that sorting the jobs in the order of nonincreasing size before feeding them to the list-scheduling algorithm yields an approximation algorithm with performance ratio $4/3 - 1/(3m)$. Later, exploiting the relationship between

the machine-scheduling problem under consideration and the bin-packing problem, algorithms with improved approximation ratios have been obtained in a series of works (Coffman et al. [9], Friesen [12], Langston [20]).

Finally, PTASes for a constant number of machines and for an arbitrary number of machines are given in Graham [15], Sahni [25], and by Hochbaum and Shmoys [18], respectively. The latter PTAS partitions jobs into large and small jobs. The sizes of large jobs are rounded such that an optimum schedule for the rounded jobs can be obtained via dynamic programming. The small jobs are then added greedily, using Graham's list-scheduling algorithm. This approach can be refined to an algorithm with linear running time (see, e.g., Hochbaum [17]): Replace the dynamic program with an integer linear program on a fixed number of variables and constraints that can be solved in constant time (Lenstra [21]).

In a series of papers, increasingly complicated online algorithms with better and better competitive ratios beating the Graham bound 2 have been developed (Bartal et al. [6], Karger et al. [19], Albers [2]). The best result known to date is a 1.9201-competitive algorithm due to Fleischer and Wahl [11]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin and Chandrasekaran [24] (see also Rudin [23]). For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$ (Chen et al. [8], Sgall [29]). For more results on online algorithms for scheduling, we refer to the recent survey articles by Albers [3] and Sgall [30].

Strategies that reassign jobs were studied in the context of online load balancing where jobs arrive in and depart from a system of m machines online and the scheduler has to assign each incoming job to one of the machines. Deviating from the usual approach of comparing against the optimal *peak load* seen so far, Westbrook [32] introduced the notion of competitiveness against *current load*: An algorithm is α-competitive if after every round the makespan is within α factor of the optimal makespan for the current set of jobs. Each incoming job j has size p_j and reassignment cost c_j. For a job, the reassignment cost has to be paid for its initial assignment and then every time it is reassigned. Observe that the optimal strategy has to pay this cost once for each job for its initial assignment. Thus, the optimal (re)assignment cost S is simply the sum of reassignment costs of all jobs scheduled till now. Westbrook showed a 6-competitive strategy for identical machines with reassignment cost $3S$ for proportional reassignments, i.e., c_j is proportional to p_j, and $2S$ for unit reassignments, i.e., $c_j = 1$ for all jobs. Later Andrews et al. [4] improved it to competitive ratio 3.5981 with the same reassignment factors. They also showed $(3+\epsilon)$- and $(2+\epsilon)$-competitive strategies for the proportional and unit case, respectively, where the reassignment factor depends only on ϵ. For arbitrary reassignment costs they achieve 3.5981-competitiveness with 6.8285 reassignment factor. They also present a 32-competitive strategy with constant reassignment factor for related machines.

Our contribution. We consider the online setting where jobs arrive incrementally but there are no job departures. In §3 we describe a simple online algorithm that achieves approximation ratio $\frac{3}{2}$ using a moderate migration factor $\beta = 2$. Notice that this result already beats the lower bound 1.88 (1.58) on the competitive ratio of any classical (randomized) online algorithm without migration. Using a more sophisticated analysis, the migration factor can be decreased to $\frac{4}{3}$ while maintaining competitive ratio $\frac{3}{2}$. This result is tight because we can show that any robust[1] scheduling strategy with competitive ratio $\frac{3}{2}$ has migration factor of at least $\frac{4}{3}$. We also show another tightness result: The algorithms considered in §3 have the nice property that they work *locally*, i.e., they only migrate jobs from the machine to which the new job is being assigned. We prove that any robust local strategy has competitive ratio of at least $\frac{3}{2}$. In §4 we present a more sophisticated algorithm with migration factor $\frac{5}{2}$ and competitive ratio $\frac{4}{3}$. For the special case of two machines, it is shown in §5 that we can achieve competitive ratio $\frac{7}{6}$ with a migration factor of 1. Moreover, this ratio is tight for migration factor 1.

In §6 we present a family of online algorithms with competitive ratio $1+\epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. On the negative side, no constant migration factor suffices to maintain competitive ratio 1, i.e., optimality. We provide interpretations of these results in several different contexts:

(i) *Online algorithms*. Online scheduling with bounded job migration is a relaxation of the classical online paradigm. Obviously, there is a trade-off between the desire for high-quality solutions and the requirement to compute them online, that is, to deal with a lack of information. Our result can be interpreted in terms of the corresponding trade-off curve: Any desired quality can be guaranteed while relaxing the online paradigm only moderately by allowing for a constant migration factor.

(ii) *Sensitivity analysis*. Given an optimum solution to an instance of an optimization problem and a slightly modified instance, can the given solution be turned into an optimum solution for the modified instance without changing the solution too much? This is the compelling question in sensitivity analysis. As indicated above, for

[1] We give a precise definition of robustness in §3 after the proof of Theorem 3.2.

the scheduling problem under consideration one has to answer in the negative. Already, one additional job can change the entire structure of an optimum schedule. However, our result implies that the answer is positive if we only require near-optimum solutions.

(iii) *Approximation results*. Our result yields a new PTAS for the scheduling problem under consideration. Due to its online background, this PTAS constructs the solution incrementally. That is, it reads the input little by little, always maintaining a $(1+\epsilon)$-approximate solution. Indeed, it follows from the analysis of the algorithm that every update only takes constant time. In particular, the overall running time is linear and thus matches the previously best known approximation result.

We believe that each of these interpretations constitutes an interesting motivation for results like the one we present here in its own right and can therefore lead to interesting results for many other optimization problems. After reading a preliminary version of this paper (Sanders et al. [27]), Epstein and Levin [10] came up with similar results for the bin-packing problem.

Our results can also be interpreted within the framework of online load balancing (see the discussion above, Westbrook [32], and Andrews et al. [4]), with proportional reassignments and without job deletions. However, our requirements are stronger in the sense that a strategy with reassignment factor β ensures that when a new job j arrives, the total reassignment cost incurred (for scheduling it) is at most βc_j. This is different from the more relaxed constraint that after t rounds, the total reassignment cost incurred is at most $\beta \sum c_j$ (summing over all jobs seen until round t). Moreover, almost all of our scheduling strategies are *robust*, i.e., they convert *any* α-competitive schedule to an α-competitive schedule after assigning the newly arrived job, whereas in Westbrook [32] and Andrews et al. [4] it is required that the schedule so far is carefully constructed in order to ensure the competitiveness after assigning/deleting a job in the next round.

The underlying details of the presented online approximation scheme have the same roots as the original PTAS by Hochbaum and Shmoys [18] and its refinements (Hochbaum [17]). We distinguish between small and large jobs; a job is called large if its size is of the same order of magnitude as the optimum makespan. Because this optimum can change when a new job arrives, the classification of jobs must be updated dynamically. The size of every large job is rounded such that the problem of computing an optimum schedule for the subset of large jobs can be formulated as an integer linear program of constant size. A newly arriving job causes a small change in the right-hand side of this program. This enables us to use results from sensitivity analysis of integer programs in order to prove that the schedule of large jobs needs to be changed only slightly. Our PTAS is very simple, it uses only this structural result and does not use any algorithms from integer programming theory.

In §7 we discuss an application of bounded migration to configuring storage servers. This was the original motivation for our work. In this application, the objective is to maximize the minimum machine load. It is well known (Azar and Epstein [5]) that any online deterministic algorithm for this *machine-covering problem* has competitive ratio at least m (the number of machines). There is also a lower bound of $\Omega(\sqrt{m})$ for any randomized online algorithm. We develop a simple deterministic online strategy that is 2-competitive already for migration factor $\beta = 1$.

A preliminary version of this work appeared in proceedings of ICALP 2004 (Sanders et al. [27]).

2. Preliminaries. We consider the problem of scheduling a set of *jobs* $\{1,\dots,n\}$ on a set of m identical parallel *machines* $M = \{1,\dots,m\}$. Job $j \in \{1,\dots,n\}$ has a positive *processing time* or *size* p_j and arrives in round j. The number of jobs is unknown in advance and the processing time of job j is only revealed in round j. For a subset of jobs $N \subseteq \{1,\dots,n\}$, the *total processing time* of jobs in N is $p(N) := \sum_{j\in N} p_j$; moreover, let $p_{\max}(N) := \max_{j\in N} p_j$ denote the maximum processing time of a job in N.

A *schedule* S for a subset of jobs N is an assignment of the jobs in N to the set of machines M, i.e., $S\colon N \to M$. We denote the set of jobs scheduled on (i.e., assigned to) machine $i \in M$ by $S^{-1}(i) \subseteq N$. The *load of machine* i is $p(S^{-1}(i))$ and the *makespan* of schedule S is the maximum machine load $\max_{i\in M} p(S^{-1}(i))$. We say that machine $i \in M$ is a *least loaded* machine with respect to schedule S if $p(S^{-1}(i)) = \min_{i'\in M} p(S^{-1}(i'))$. For a subset of jobs N, let $\textsc{opt}(N)$ denote the *minimal makespan* of a schedule for N. The following well-known facts are due to Graham [14].

Observation 2.1. Let N be a set of jobs.

(i) The value $\textsc{lb}(N) := \max\{p(N)/m, p_{\max}(N)\}$ is a lower bound on $\textsc{opt}(N)$ satisfying

$$\textsc{lb}(N) \le \textsc{opt}(N) < 2\,\textsc{lb}(N). \tag{1}$$

(ii) Consider an arbitrary schedule for N with makespan κ. Assigning a new job $j \notin N$ to a least loaded machine yields a schedule for $N \cup \{j\}$ with makespan at most

$$\max\{\kappa, \textsc{opt}(N \cup \{j\}) + (1 - 1/m)p_j\}.$$

In the following sections we often consider the subset of jobs $N = \{1, \ldots, j-1\}$ that have arrived in the first $j-1$ rounds and a newly arrived job j. If N and j are clear from the context, we sometimes use the shorter notation $\text{OPT} := \text{OPT}(N)$ and $\text{OPT}' := \text{OPT}(N \cup \{j\})$.

We conclude this section by stating an easy but important corollary of observation 2.1(ii).

COROLLARY 2.1. *Given an α-approximate schedule for the jobs in N, assigning a new job of size at most $(1 + 1/(m-1))(\alpha - 1)\text{OPT}'$ to a least loaded machine yields an α-approximate schedule for $N \cup \{j\}$.*

3. Simple strategies with small migration factors. We start by presenting a very simple $(3/2 - 1/(2m))$-competitive algorithm with migration factor 2. The algorithm is based on the following procedure.

PROCEDURE 1.
Input: *A schedule for a set of jobs N and a new job $j \notin N$.*
Output: *A schedule for $N \cup \{j\}$.*
Choose one of the following two options that minimizes the resulting makespan:
Option 0: *Assign job j to a least loaded machine.*
Option 1: *Let i be the machine minimizing the maximum job size. Repeatedly remove jobs from this machine (in any order); stop before the total size of removed jobs exceeds $2p_j$. Assign job j to machine i. Assign the removed jobs successively to a least loaded machine.*

THEOREM 3.1. *Given a $(3/2 - 1/(2m))$-approximate schedule for N and a new job $j \notin N$, Procedure 1 outputs a $(3/2 - 1/(2m))$-approximate schedule for $N \cup \{j\}$. The migration factor is bounded by 2.*

PROOF. From the description of Procedure 1, it is clear that the migration factor is at most 2. We call a job *small* if its processing time is at most $\frac{1}{2}\text{OPT}'$; otherwise, it is called *large*. If the new job j is small, option 0 yields makespan of at most $(3/2 - 1/(2m))\text{OPT}'$ by Corollary 2.1. Thus, we can assume from now on that j is large.

Because there are at most m large jobs in $N \cup \{j\}$, all jobs on machine i chosen in option 1 are small. Thus, after removing jobs from machine i as described above, machine i is either empty or the total size of removed jobs exceeds the size of the large job j. In both cases, assigning job j to machine i cannot increase its load above $(3/2 - 1/(2m))\text{OPT}'$. Thus, using the same argument as above, assigning the removed small jobs successively to a least loaded machine yields a $(3/2 - 1/(2m))$-approximate schedule for the set of jobs $N \cup \{j\}$. □

COROLLARY 3.1. *Scheduling every newly arriving job according to Procedure 1 is a $(3/2 - 1/(2m))$-competitive algorithm with migration factor 2.*

Next we show that the migration factor can be decreased to $\frac{4}{3}$ without increasing the competitive ratio above $\frac{3}{2}$. This result is achieved by carefully modifying Procedure 1.

PROCEDURE 2.
Input: *A schedule for a set of jobs N and a new job $j \notin N$.*
Output: *A schedule for $N \cup \{j\}$.*
Choose one of the following $m+1$ options that minimizes the resulting makespan. (Break ties in favor of option 0.)
Option 0: *Assign job j to a least loaded machine.*
Option i [for $i \in \{1, \ldots, m\}$]: *Ignoring the largest job on machine i, consider the remaining jobs on machine i in order of nonincreasing size and remove a job unless the total size of removed jobs will exceed $\frac{4}{3}p_j$. Assign job j to machine i. Assign the removed jobs successively to a least loaded machine.*

In the analysis of Procedure 2, we make use of the following structural property of a schedule. A schedule for a set of jobs N has property $(*)$ if the load on any machine excluding its largest job is at most $\text{OPT}(N)$. For example, a schedule constructed by list-scheduling always has property $(*)$. We show that Procedure 2 maintains property $(*)$.

LEMMA 3.1. *Given a schedule for N satisfying property $(*)$ and a new job $j \notin N$, Procedure 2 outputs a schedule for $N \cup \{j\}$ satisfying property $(*)$.*

PROOF. Let $M' \subseteq M$ be the subset of machines touched by Procedure 2 for scheduling job j. It suffices to argue that the total load on machine $i \in M'$ excluding the job j' that entered the machine last is at most OPT'. Except for the very last case considered below, we will prove this stronger property for machine $i \in M'$.

If $j' \neq j$, then j' was assigned as part of the redistribution phase. Because redistribution is always performed on a currently least loaded machine, the load on this machine excluding this last job is at most OPT'.

It remains to consider the case $j' = j$. If option 0 is chosen and assigns j to machine i, then machine i is a least loaded machine of the initial schedule; in particular, its load excluding j is at most $\text{OPT} \leq \text{OPT}'$. Otherwise, j is assigned to machine i by option i, which we assume from now on. Because we always break ties in favor of option 0, the makespan of the schedule produced in option i is strictly smaller than the makespan of the schedule obtained in option 0. We distinguish two cases.

Case 1. The makespan of the schedule obtained by assigning j to a least loaded machine i' (option 0) is determined by machine i'. As the makespan obtained by option i is strictly smaller than the makespan obtained by option 0, the load of machine i minus the size of j is at most the makespan of a least loaded machine i' in the initial schedule, and thus at most $\text{OPT} \leq \text{OPT}'$.

Case 2. The makespan of the schedule obtained by assigning j to a least loaded machine i' (Option 0) is not determined by machine i'. In this case, it must be determined by machine $i \neq i'$ because this is the only machine whose makespan can possibly be decreased by choosing option i. Remember that we break ties in favor of option 0. Thus, the makespan of machine i must have been decreased in option i. Because we did not touch the largest job on machine i, it still has the desired property. □

THEOREM 3.2. *Given a $\frac{3}{2}$-approximate schedule for N satisfying property $(*)$ and a new job $j \notin N$, Procedure 2 outputs a $\frac{3}{2}$-approximate schedule for $N \cup \{j\}$ satisfying property $(*)$. The migration factor is bounded by $\frac{4}{3}$.*

PROOF. The bound on the migration factor is clear from the description of Procedure 2. Moreover, it follows from Lemma 3.1 that the computed schedule for $N \cup \{j\}$ has property $(*)$. To show the approximation result, we distinguish three cases depending on the size of job j.

Case 1. If $p_j \leq \frac{1}{2}\text{OPT}'$, option 0 yields a schedule of makespan at most $\frac{3}{2}\text{OPT}'$ by Corollary 2.1.

Case 2. If $p_j \geq \frac{3}{4}\text{OPT}'$, then $\frac{4}{3}p_j \geq \text{OPT}$ so that for any machine i it is feasible to migrate all but the largest job due to property $(*)$. Because there are at most $m-1$ jobs in N of size larger than $\frac{1}{2}\text{OPT}'$, there is a machine i that only contains jobs of size at most $\frac{1}{2}\text{OPT}'$. Then option i yields a schedule with makespan at most $\frac{3}{2}\text{OPT}'$. (Apply the same argument as in Case 1 for the reassignment of removed jobs.)

Case 3. It remains to consider the situation when $p_j = \frac{1}{2}\text{OPT}' + \delta$ for some $0 < \delta < \frac{1}{4}\text{OPT}'$. In the following a job in N is called *huge* if its size is strictly larger than $\text{OPT}' - \delta$ and *tiny* if its size is strictly smaller than δ. The subsets of huge and tiny jobs are denoted by N_{H} and N_{T}, respectively. Notice that in an optimal schedule for N a huge job can only share a machine with tiny jobs.

CLAIM. *In the given schedule S for N there exists a machine i that does not contain huge jobs and $p(S^{-1}(i) \setminus N_{\text{T}}) \leq \text{OPT}$.*

In an optimal schedule for N the jobs in $N \setminus (N_{\text{H}} \cup N_{\text{T}})$ are scheduled on the $m - |N_{\text{H}}|$ machines that do not contain a huge job. This yields $p(N \setminus (N_{\text{H}} \cup N_{\text{T}})) \leq (m - |N_{\text{H}}|)\text{OPT}$. By a simple averaging argument, among the $m - |N_{\text{H}}|$ machines that do not contain a huge job in the given schedule S there is one as claimed above.

It remains to show that option i yields a $\frac{3}{2}$-approximate schedule for $N \cup \{j\}$. Because $p(S^{-1}(i) \setminus N_{\text{T}}) \leq \text{OPT}$, there is at most one job of size strictly larger than $\frac{1}{2}\text{OPT}$ on machine i. Consequently, every job removed during option i has size at most $\frac{1}{2}\text{OPT}$, and the reassignment of these jobs does not cause a problem with respect to the bound $\frac{3}{2}\text{OPT}'$. We still need to show that the load of machine i after assigning job j does not exceed $\frac{3}{2}\text{OPT}'$.

If there is a tiny job that is not removed from machine i, then the total size of removed jobs is larger than $\frac{4}{3}p_j - \delta \geq p_j$, and we are done because the load of machine i in the given schedule S is at most $\frac{3}{2}\text{OPT}$. We can assume in the following that all tiny jobs are removed from machine i.

If $S^{-1}(i) \setminus N_{\text{T}}$ contains only one job, we are done because this job is not huge such that together with job j it does not exceed the bound $\frac{3}{2}\text{OPT}'$ on the load of machine i. Finally, if $S^{-1}(i) \setminus N_{\text{T}}$ contains more than one job, then, as already argued above, the second-largest job has size at most $\frac{1}{2}\text{OPT} \leq \frac{4}{3}p_j$ and is therefore removed (remember that the jobs are considered in order of nonincreasing size during the removal phase). Because this job is not tiny, it has size at least δ and the total size of removed jobs is at least $\delta + p(S^{-1}(i) \cap N_{\text{T}})$. Therefore the load of machine i after the removal phase is at most $p(S^{-1}(i) \setminus N_{\text{T}}) - \delta \leq \text{OPT} - \delta$ and we can afford to assign job j to machine i. □

In §4 we present a more sophisticated algorithm with migration factor 4 and competitive ratio $\frac{4}{3}$. Moreover, in §5 we present a specialized algorithm for the case of two machines with migration factor 1 and competitive ratio $\frac{7}{6}$; and we also show that this result is tight, i.e., migration factor 1 does not allow for a smaller competitive ratio.

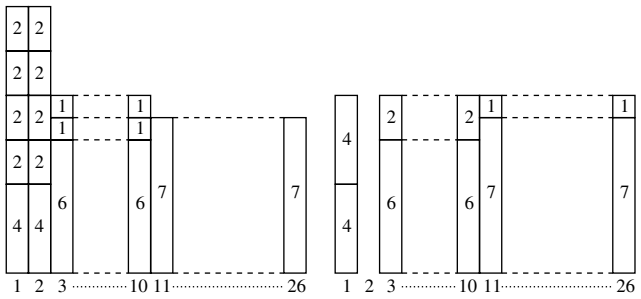

FIGURE 1. An instance with 26 machines and 50 jobs.

Notes. The load of the machines is depicted on the vertical axis. On the right-hand side an optimal schedule with makespan 8 is given. On the left-hand side there is a $\frac{3}{2}$-approximate schedule-fulfilling property $(*)$. If a new job of size $6+\epsilon$ arrives, jobs of total size of at least 8 have to be moved in order to obtain a schedule that is still $\frac{3}{2}$-approximate.

Robustness. The scheduling strategies for minimizing the makespan discussed in this section (and in §4) are robust in the following sense. The only invariant that we require in their analyses is that before the arrival of a new job the current schedule is α-approximate. Job j can then be incorporated, yielding again an α-approximate schedule. In other words, we do not require that the current schedule is carefully constructed so far, to maintain the competitiveness in the next round. Only for Procedure 2 must the schedule additionally satisfy that, on any machine, the load excluding the largest job is at most the optimum makespan. This is a rather mild assumption because even the simple list-scheduling algorithm and almost all natural scheduling strategies ensure this.

Negative results. Theorems 3.1 and 3.2 raise the question of which migration factor is really necessary to achieve competitive ratio $\frac{3}{2}$. We can prove that Procedure 2 is optimal in the sense that any robust scheduling strategy needs a migration factor of at least $\frac{4}{3}$ in order to maintain competitive ratio $\frac{3}{2}$.

LEMMA 3.2. *There exists a $\frac{3}{2}$-approximate schedule-fulfilling property $(*)$ such that, for any $0<\epsilon<4/21$, upon arrival of a new job, migration factor $\frac{4}{3}-\epsilon$ is necessary to achieve $\frac{3}{2}$-competitiveness.*

PROOF. The situation is depicted in Figure 1. There are 26 machines and 50 jobs. A $\frac{3}{2}$-approximate schedule-fulfilling property $(*)$ with makespan 12 is given on the left-hand side of the figure whereas an optimal schedule with makespan 8 is given on the right-hand side. In both schedules, machines 3 to 10 are identically packed and the same holds for machines 11 to 26. Notice that machine 2 is empty in the optimal schedule. Therefore, upon arrival of a new job of size $24/(4-3\epsilon)$, the optimal makespan remains 8. Notice that $6<24/(4-3\epsilon)<7$ due to our choice of ϵ.

To incorporate the new job into the schedule on the left-hand side of Figure 1 without increasing the makespan, jobs of total size of at least 8 have to be migrated: First of all, notice that it does not make sense to assign the new job to one of the machines currently containing a job of size 7. If the new job is assigned to one of the machines currently containing a job of size 6, this job has to be removed from the machine. In order to reassign it without increasing the makespan, two more jobs of size 1 have to be moved. Finally, in order to schedule the new job on one of the first two machines, jobs of total size of at least 8 have to be moved.

Thus, the migration factor is at least $8\cdot(4-3\epsilon)/24=4/3-\epsilon$. □

An additional feature of Procedures 1 and 2 is that they are *local* in the sense that they migrate jobs only from the machine to which the newly arrived job is assigned.

LEMMA 3.3. *There is a class of optimal schedules for which, upon arrival of a new job, it is not possible to achieve a competitive ratio smaller than $\frac{3}{2}$ using only local migration (even if an arbitrary migration factor is allowed).*

PROOF. The following optimal schedule on m machines, upon the arrival of a new job, enforces a competitive ratio of at least $3/(2+2/m)$ for any amount of local migration. This bound converges to $\frac{3}{2}$ for large m. Machines 1 and 2 each contain one job of size $\frac{1}{2}$ and $m/2$ jobs of size $1/m$ (we assume that m is even). All other machines contain a single job of size 1; see Figure 2. The newly arriving job has size 1. The optimum makespan is $1+1/m$, and the makespan achievable by any local strategy is $\frac{3}{2}$ (by scheduling the new job, e.g., on machine 1 and migrating all small jobs to other machines). □

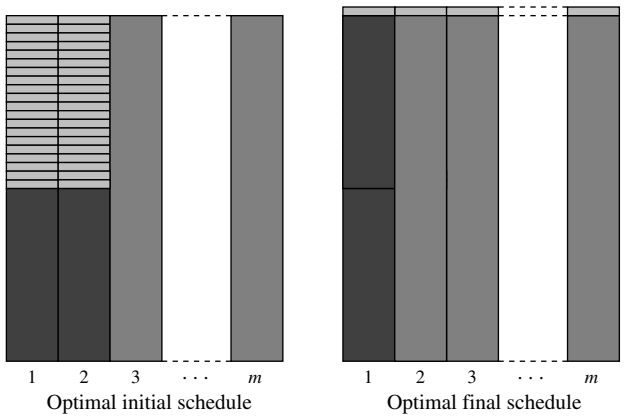

FIGURE 2. An instance where local migration is not better than $\frac{3}{2}$-competitive.

4. A $\frac{4}{3}$-competitive strategy with migration factor $\frac{5}{2}$. Taking up the line of research pursued in §3, we show that competitive ratio $\frac{4}{3}$ can be achieved by a more sophisticated algorithm with migration factor $\frac{5}{2}$. This result, as well as the algorithm and its proof, have been pointed out by Sgall [31]. The result improves upon an earlier $\frac{4}{3}$-competitive strategy with migration factor 4 mentioned in Sanders et al. [27].

PROCEDURE 3.
Input: *A schedule S for a set of jobs N and a new job $j \notin N$.*
Output: *A schedule for $N \cup \{j\}$.*
Among the schedules generated by the following $m+1$ options, consider only those with migration factor less than $\frac{5}{2}$ and output one with minimum makespan.
Option i [for $i \in \{0, \ldots, m\}$]: *Classify the jobs in $N \cup \{j\}$ as follows. The i largest jobs are called* large; *the next $2(m-i)$ largest jobs are* medium; *the remaining jobs are* small.
If j is small, then assign it to a least loaded machine.
Otherwise, if j is medium, then find a machine i' with no large and at most one medium job in schedule S.[2] *Repeatedly remove small jobs from machine i' (in arbitrary order) until no small job remains or the total size of removed jobs exceeds p_j. Assign j to i' and the removed jobs successively to a least loaded machine.*
Otherwise, if j is large and if there is a machine i' with no large or medium job in schedule S, then proceed with machine i' as in the last case.
Otherwise, j is large and there are two machines i' and i'' containing no large and exactly one medium job each.[3] *Let the medium jobs on machines i' and i'' be j' and j'', respectively, and assume that $p_{j'} \le p_{j''}$. Remove job j' and the biggest small job (if any) from machine i'. Repeatedly remove further jobs from machine i' (in arbitrary order) until no job remains or the total size of removed jobs exceeds p_j. Assign job j to machine i'. Repeatedly remove small jobs from machine i'' (in arbitrary order) until no small job remains or the total size of removed jobs exceeds $p_{j'}$. Assign job j' to machine i''. Assign the small jobs removed from i' and i'' successively to a least loaded machine.*

THEOREM 4.1. *Given a $\frac{4}{3}$-approximate schedule for N and a new job $j \notin N$, Procedure 3 outputs a $\frac{4}{3}$-approximate schedule for $N \cup \{j\}$. The migration factor is less than $\frac{5}{2}$.*

PROOF. Let i be the number of jobs in $N \cup \{j\}$ with processing time strictly greater than $\frac{2}{3}\text{OPT}'$. We prove that the schedule computed in option i has makespan of at most $\frac{4}{3}\text{OPT}'$, and the total size of migrated jobs is less than $\frac{5}{2}p_j$.
By the classification applied in option i, large jobs are exactly those with a processing time greater than $\frac{2}{3}\text{OPT}'$. Thus, medium jobs have a size of at most $\frac{2}{3}\text{OPT}'$. Moreover, we claim that small jobs have size at most $\frac{1}{3}\text{OPT}'$: In an optimum schedule for $N \cup \{j\}$ with makespan OPT′ and i large jobs of size greater than $\frac{2}{3}\text{OPT}'$, there can be at most $2(m-i)$ further jobs of size greater than $\frac{1}{3}\text{OPT}'$.

[2] Because there are exactly i large jobs and $2(m-i)-1$ medium jobs in S, such a machine exists.

[3] Each machine contains at least one large or medium job. The total number of large and medium jobs in S is exactly $2m-i-1$. Thus, there are at least $i+1$ machines that contain exactly one medium or large job. Because there are only $i-1$ large jobs in S, the existence of the two machines i' and i'' follows.

If j is small, no migration occurs and we are done by Corollary 2.1. If j is medium, the migration factor is at most 2: The total size of jobs removed from machine i' is less than p_j plus the size of one small job. After removing small jobs from machine i' and assigning job j to it, the load of machine i' has either decreased or there are only two jobs on the machine that are both medium. In any case, the new load of machine i' is at most $\frac{4}{3}\text{OPT}'$. Reassigning the removed small jobs does not increase the makespan beyond $\frac{4}{3}\text{OPT}'$ due to Corollary 2.1.

If j is large and there is a machine i' with no large or medium job, the migration factor is again at most 2 and the upper bound on the makespan easily follows as well. It remains to consider the case where j is large and there are two machines i' and i'' containing no large and exactly one medium job each. We first argue that the makespan remains at most $\frac{4}{3}\text{OPT}'$. After removing jobs and assigning j, the load of machine i' has either decreased or job j is the only job on machine i'. In both cases the new load of i' is at most $\frac{4}{3}\text{OPT}'$. The same bound holds for the load of machine i'' after removing small jobs and assigning j' (same argument as in the case "j is medium" above). Moreover, reassigning the removed small jobs does not increase the makespan beyond $\frac{4}{3}\text{OPT}'$ due to Corollary 2.1. To prove the bound $\frac{5}{2}$ on the migration factor, we distinguish two cases.

Case 1. At most one small job is removed from machine i'. Then the amount removed from i' is at most $p_{j'} + \frac{1}{3}\text{OPT}'$. Because $p_{j''} \geq p_{j'}$ and the original load of machine i'' is at most $\frac{4}{3}\text{OPT}'$, the amount removed from i'' is at most $\frac{4}{3}\text{OPT}' - p_{j''} \leq \frac{4}{3}\text{OPT}' - p_{j'}$. Thus, the total migration is at most $\frac{5}{3}\text{OPT}' = \frac{5}{2} \cdot \frac{2}{3}\text{OPT}' < \frac{5}{2}p_j$.

Case 2. At least two small jobs are removed from machine i'. Then the first removed job has size at most $p_j - p_{j'}$ and the same holds for all further removed jobs. Therefore, the total size of jobs removed from i' is at most $p_j + (p_j - p_{j'})$. Moreover, the total size of jobs removed from machine i'' is at most $p_{j'}$ plus the size of one small job. Thus, the total migration is at most $2p_j + \frac{1}{3}\text{OPT}' < \frac{5}{2}p_j$. This concludes the proof. □

5. The two-machine case. In this section we present an online algorithm for the two-machine case that achieves competitive ratio $\frac{7}{6}$ with migration factor 1. Before we discuss the algorithm, we first prove that this result is best possible.

THEOREM 5.1. *Any online algorithm with migration factor at most* 1 *has a competitive ratio of at least* $\frac{7}{6}$.

PROOF. Consider an instance consisting of four jobs with sizes $\frac{1}{3}$, $\frac{1}{3}$, $\frac{1}{2}$, and $\frac{1}{2}$. The optimal makespan is obviously $\text{OPT} = \frac{5}{6}$. It is easy to verify that the only $\frac{7}{6}$-approximate schedule is the optimal schedule that assigns one job of size $\frac{1}{3}$ and one job of size $\frac{1}{2}$ to each machine. After the arrival of a new job j of size $p_j = \frac{1}{3}$, the optimal makespan is $\text{OPT}' = 1$. With migration factor 1, however, one can only achieve makespan $\frac{7}{6}$. □

Before we present our particular online algorithm, we first prove the *existence* of an online algorithm with competitive ratio $\frac{7}{6}$ and migration factor 1. We even show that there is such an online algorithm that is robust and uses only local migration; see §3 for the definitions of robustness and local migration.

LEMMA 5.1. *Consider an arbitrary* $\frac{7}{6}$*-approximate schedule for a given set of jobs* N *on two machines and a newly arriving job* j*. There is a subset of jobs* X *of total size* $p(X) \leq p_j$ *residing on one of the two machines so that scheduling job* j *on this machine and migrating the jobs in* X *to the other machine yields a* $\frac{7}{6}$*-approximate schedule for* $N \cup \{j\}$.

PROOF. If $p_j \leq \frac{1}{3}\text{OPT}'$, it follows from Corollary 2.1 that assigning job j to a least loaded machine yields the desired schedule, i.e., X can be chosen to be the empty set in this case. In the following we assume that $p_j > \frac{1}{3}\text{OPT}'$.

Consider an optimal schedule for $N \cup \{j\}$ where job j is scheduled on the first machine. We describe the difference between this optimal schedule and the given schedule for N; see Figure 3. Let δ_1 be the subset of jobs assigned to the first machine in both schedules. Similarly, Δ_2 is the set of jobs assigned to the second machine in both schedules. The remaining two sets δ_2 and Δ_1 capture the differences between these two schedules except for job j, which is only present in the optimal schedule (see Figure 3). In the following we assume without loss of generality that $p(\delta_1) \leq p(\delta_2)$ (otherwise, exchange the two machines in the given schedule for N). Because

$$p(\delta_1) + p(\delta_2) + p_j \leq \text{OPT}', \tag{2}$$

we get $p(\delta_1) < \frac{1}{3}\text{OPT}'$ as $p_j > \frac{1}{3}\text{OPT}'$. Also notice that

$$p(\Delta_1) + p(\Delta_2) \leq \text{OPT}'. \tag{3}$$

Without exceeding migration factor 1, we can remove the subset of jobs δ_1 from the first machine, assign j to this machine, and then assign all jobs in δ_1 to the least loaded machine. If $p_j + p(\Delta_1) \leq \frac{7}{6}\text{OPT}'$, we are

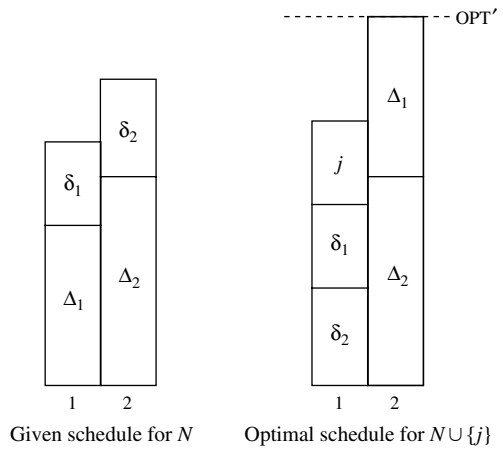

FIGURE 3. Comparison of given schedule for N and optimal schedule for $N \cup \{j\}$ on two machines.

done by Corollary 2.1; set $X := \delta_1$ or $X := \varnothing$, depending on which machine has a smaller load after j has been scheduled. We thus assume from now on that

$$p_j + p(\Delta_1) > \tfrac{7}{6}\text{OPT}' \tag{4}$$

and thus, by (3),

$$p_j > p(\Delta_2) + \tfrac{1}{6}\text{OPT}'. \tag{5}$$

We distinguish two cases.

Case 1. $p(\Delta_2) \le \frac{1}{3}\text{OPT}'$. Because $p_j + \delta_2 \le \text{OPT}'$ by (2), removing the jobs in Δ_2 from the second machine, assigning job j to this machine, and then assigning all jobs in Δ_2 to the least loaded machine yields the desired result by Corollary 2.1; set $X := \Delta_2$ or $X := \varnothing$, depending on which machine has a smaller load after j has been scheduled.

Case 2. $p(\Delta_2) > \frac{1}{3}\text{OPT}'$. Then $p_j > \frac{1}{2}\text{OPT}'$ by (5) and thus $p(\delta_1) + p(\delta_2) < \frac{1}{2}\text{OPT}'$ by (2). Because $p(\Delta_1) < \frac{2}{3}\text{OPT}'$ by (3), we get

$$p(\Delta_1) + p(\delta_1) + p(\delta_2) < \tfrac{7}{6}\text{OPT}'. \tag{6}$$

If $p_j + p(\Delta_2) \le \frac{7}{6}\text{OPT}'$, then $X := \delta_2$ is a feasible choice and we are done. Otherwise, $p_j + p(\Delta_2) > \frac{7}{6}\text{OPT}'$ together with (4) and (3) implies that $p_j > \frac{2}{3}\text{OPT}'$. As a consequence of (2) we get $p(\delta_1) + p(\delta_2) < \frac{1}{3}\text{OPT}'$. Moreover, because $p(\delta_1) \le p(\delta_2)$ we obtain $p(\delta_1) < \frac{1}{6}\text{OPT}'$. By (3) we get $p(\Delta_1) + p(\Delta_2) + p(\delta_1) < \frac{7}{6}\text{OPT}'$. Therefore $X := \Delta_2$ is a feasible choice and the proof is complete. □

Of course, the result in Lemma 5.1 can be easily turned into an online algorithm with competitive ratio $\frac{7}{6}$ and migration factor 1 using brute force: Simply determine the subset of jobs X by complete enumeration. The following more-efficient algorithm is based on the intuition that it is sufficient to know those jobs in X whose size is at least $\frac{1}{3}\text{OPT}'$. The remaining jobs can be chosen greedily. We call a job in the initial schedule S *large* if it is among the three largest jobs on its machine. Otherwise, it is *small*. Because in a $\frac{7}{6}$-approximate schedule for N there are at most three jobs of size at least $\frac{1}{3}\text{OPT}'$ on each machine, we know that small jobs have size at most $\frac{1}{3}\text{OPT}'$. For $i \in \{1, 2\}$, we define a family of subsets of N by

$$\mathcal{L}_i := \{Y \subseteq S^{-1}(i) \mid p(Y) \le p_j \text{ and } Y \text{ only contains large jobs}\}.$$

Because there are at most three large jobs on machine i in schedule S, the family $\mathcal{L}_i$ contains at most 2^3 subsets of jobs.

PROCEDURE 4.

Input: *A schedule S for a set of jobs N and a new job $j \notin N$.*

Output: *A schedule for $N \cup \{j\}$.*

We define an option for each $i = 1, 2$ and $Y \in \mathcal{L}_i$. Choose one of these (at most 16) options that minimizes the resulting makespan.

Option i, Y: *Migrate all jobs in Y to the other machine and assign job j to machine i. Consider the small jobs in $S^{-1}(i)$ in arbitrary order: Migrate a small job unless the total size of migrated jobs will exceed p_j or migrating the job will not lead to an improved makespan.*

THEOREM 5.2. *Given a $\frac{7}{6}$-approximate schedule for N and a new job $j \notin N$, Procedure 4 outputs a $\frac{7}{6}$-approximate schedule for $N \cup \{j\}$. The migration factor is at most* 1.

PROOF. The bound on the migration factor is clear from the description of Procedure 4. Let $i \in \{1, 2\}$ and $X \subseteq S^{-1}(i)$ be a subset as in Lemma 5.1. Moreover, let $Y \subseteq X$ be the subset of large jobs in X. We prove that option i, Y yields a $\frac{7}{6}$-approximate schedule for $N \cup \{j\}$. We distinguish two cases.

Case 1. $X = \varnothing$. In this case, assigning job j to machine i yields a $\frac{7}{6}$-approximate schedule for $N \cup \{j\}$. In option $i, \varnothing$, our algorithm assigns j to machine i and afterwards only migrates small jobs if this decreases the makespan. Therefore, it also outputs a $\frac{7}{6}$-approximate schedule for $N \cup \{j\}$.

Case 2. $X \neq \varnothing$. In this case we may assume that assigning job j to any of the two machines yields a schedule with makespan strictly larger than $\frac{7}{6}\text{OPT}'$ (otherwise we can choose $X := \varnothing$ and are back in Case 1). In other words,

$$p(S^{-1}(1)) > \tfrac{7}{6}\text{OPT}' - p_j \qquad \text{and} \qquad p(S^{-1}(2)) > \tfrac{7}{6}\text{OPT}' - p_j. \tag{7}$$

We refer to machine i in the following as the *first machine* and to the other machine as the *second machine*. We first argue that the load of the second machine in the schedule computed by option i, Y is at most $\frac{7}{6}\text{OPT}'$. Because $Y \subseteq X$, migrating the large jobs in Y cannot increase the load of the second machine beyond $\frac{7}{6}\text{OPT}'$. Thus, before we start to migrate small jobs, the load of the second machine is at most $\frac{7}{6}\text{OPT}'$. It follows from Corollary 2.1 that the load of the second machine remains below $\frac{7}{6}\text{OPT}'$ if we improve the makespan by migrating small jobs of size at most $\frac{1}{3}\text{OPT}'$.

It remains to show that the load of the first machine i in the schedule computed by option i, Y is at most $\frac{7}{6}\text{OPT}'$. We distinguish two subcases.

Case 2a. All small jobs in $S^{-1}(i)$ are migrated to the second machine in option i, Y. In this case we have migrated all jobs in X, and possibly some more. In particular, the load of machine i in our schedule is at most the load of machine i in the $\frac{7}{6}$-approximate schedule described in Lemma 5.1.

Case 2b. There is a small job $k \in S^{-1}(i)$ that is not migrated in option i, Y. There can be two reasons for not migrating job k. If k is not migrated because this does not decrease the makespan, then the load of machine i is at most $\frac{7}{6}\text{OPT}'$ by Corollary 2.1, and we are done. Otherwise, the total size of jobs that have already been migrated is larger than $p_j - p_k \geq p_j - \frac{1}{3}\text{OPT}'$. Because of (7), the load of the second machine is at least $\frac{7}{6}\text{OPT}' - p_j + p_j - \frac{1}{3}\text{OPT}' = \frac{5}{6}\text{OPT}'$. Because the total load of the two machines is at most $2\text{OPT}'$, the load of machine i is at most $\frac{7}{6}\text{OPT}'$. □

6. An online approximation scheme with constant migration. The results presented in §§3, 4, and 5 raise the question of how far the competitive ratio for online algorithms with a constant migration factor can be decreased. We first prove that optimality (i.e., competitive ratio 1) cannot be achieved. However, for any fixed $\epsilon > 0$ we can get down to competitive ratio $1 + \epsilon$.

The following lemma states that online algorithms with a constant migration factor and competitive ratio 1 do not exist.

LEMMA 6.1. *Any online algorithm that always maintains an optimal solution has migration factor $\Omega(m)$.*

PROOF. Consider a scheduling instance with m machines and $2m - 2$ jobs, two of size i/m for all $i = 1, \ldots, m - 1$. Up to permutations of machines, any optimum schedule has the structure depicted in the left part of Figure 4. The optimum makespan is $1 - 1/m$. When a new job of size 1 arrives, the optimum makespan increases to 1. Again, the structure of an optimum schedule for the enlarged instance is unique; see the right-hand side of Figure 4. From each machine in $\{2, \ldots, m - 1\}$, at least one of the two jobs has to move. Hence the total size of jobs that have to be migrated is at least

$$\frac{1}{m} \sum_{i=1}^{m-2} \min\{i, m - 1 - i\} \geq \frac{1}{m} \sum_{i=1}^{\lfloor (m-2)/2 \rfloor} i,$$

which is in $\Omega(m)$. □

In the remainder of this section, $\epsilon > 0$ is a fixed constant. (In the following we assume without loss of generality that $\epsilon < 1$.) We develop an algorithm with competitive ratio $1 + O(\epsilon)$ and constant migration factor $\beta(\epsilon)$. This algorithm is based on a combination of several techniques known from the area of polynomial time approximation schemes for machine-scheduling problems. The root of the presented method is the polynomial time approximation scheme by Hochbaum and Shmoys [18] and its refinements (Hochbaum [17]). The

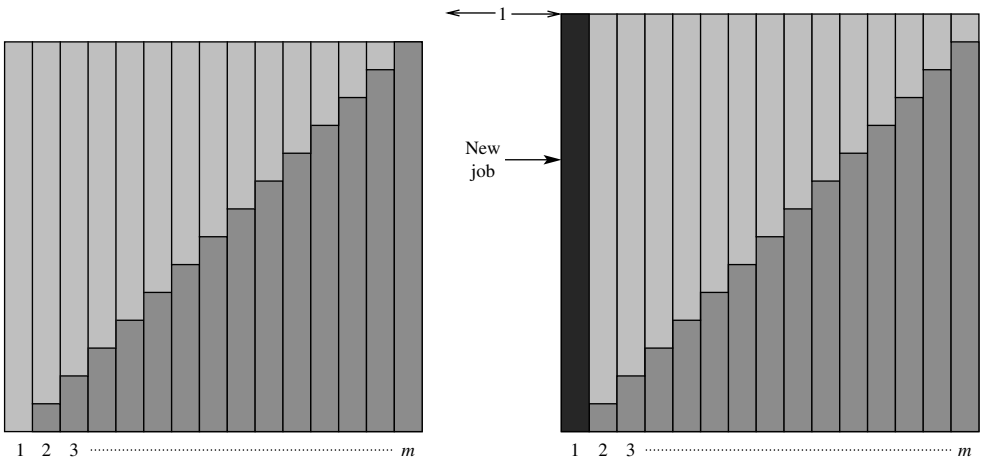

FIGURE 4. An instance where all machine configurations have to change to maintain optimality.

pivotal point of our online approach is to establish a connection to classical sensitivity analysis for integer linear programming.

The following observation belongs to the folklore in the field of scheduling; see, e.g., Afrati et al. [1].

OBSERVATION 6.1. Rounding up each job's processing time to the nearest integer power of $1+\epsilon$ increases the makespan of an arbitrary schedule by at most a factor of $1+\epsilon$. In particular, in specifying a $(1+O(\epsilon))$-competitive algorithm we can assume that all processing times are integer powers of $1+\epsilon$.

Notice that rounding job sizes is not critical with respect to constant migration factors either. In the remainder of this section we thus consider the situation when all processing times are integer powers of $1+\epsilon$. The current set of jobs is denoted by N and the newly arriving job by j. Moreover, we set $N' := N \cup \{j\}$. A job in N' is called *large* if its processing time is at least $\epsilon\,\mathrm{LB}(N')$ where, according to Observation 2.1,

$$\mathrm{LB}(N') := \max\left\{\frac{p(N')}{m},\ p_{\max}(N')\right\};$$

otherwise, it is called *small*. The subsets of large and small jobs in N' are denoted by N'_{L} and N'_{S}, respectively. Notice that N' is the disjoint union of N'_{L} and N'_{S}. We also define a much finer partition of the jobs in N' into classes of equal-size jobs N'_i, $i \in \mathbb{Z}$, with

$$N'_i := \{j' \in N' \mid p_{j'} = (1+\epsilon)^i\}.$$

Moreover, let $N_{\mathrm{L}} := N'_{\mathrm{L}} \cap N$, $N_{\mathrm{S}} := N'_{\mathrm{S}} \cap N$, and $N_i := N'_i \cap N$ for $i \in \mathbb{Z}$. Then N is the disjoint union of N_{L} and N_{S}; also the subsets N_i, $i \in \mathbb{Z}$, form a partition of N.

It is a well-known fact from the area of approximation algorithms for machine-scheduling problems that the crucial skeleton of a schedule is given by the assignment of large jobs. Due to Corollary 2.1, the small jobs can later be filled in greedily. We start by describing the set of all schedules of large jobs as the set of integral lattice points of a polyhedron.

Let $I := \{i \in \mathbb{Z} \mid \epsilon\,\mathrm{LB}(N') \le (1+\epsilon)^i \le p_{\max}(N')\}$ denote the index set of those classes that might contain large jobs such that $N'_{\mathrm{L}} = \bigcup_{i\in I} N'_i$ and $N_{\mathrm{L}} = \bigcup_{i\in I} N_i$. We can bound the number of such job classes as follows.

OBSERVATION 6.2. The number $|I|$ of classes of large jobs is at most $O((1/\epsilon)\log(1/\epsilon))$.

PROOF. By definition of I it holds that

$$|I| \le 1 + \log_{1+\epsilon}\frac{p_{\max}(N')}{\epsilon\,\mathrm{LB}(N')} \le 1 + \log_{1+\epsilon}\frac{1}{\epsilon} \le 1 + \frac{2}{\epsilon}\log\frac{1}{\epsilon} \in O\left(\frac{1}{\epsilon}\log\frac{1}{\epsilon}\right).$$

This concludes the proof. □

Given an assignment of large jobs to machines, we say that a particular machine obeys *configuration* $k\colon I \to \mathbb{N}_0$ if, for all $i \in I$, exactly $k(i)$ jobs of size $(1+\epsilon)^i$ are assigned to this machine. The set of configurations that can occur in any schedule for N'_{L} is denoted by

$$\mathbf{K} := \{k\colon I \to \mathbb{N}_0 \mid k(i) \le |N'_i| \text{ for all } i \in I\}.$$

Up to permutations of machines and up to permutations of equal-size jobs, an arbitrary schedule for N'_{L} can be described by specifying, for each $k \in \mathbf{K}$, the number y_k of machines that obey configuration k. Conversely, a vector $y \in \mathbb{N}_0^{\mathbf{K}}$ specifies a feasible m-machine-schedule for N'_{L} if and only if

$$\sum_{k \in \mathbf{K}} y_k = m, \tag{8}$$

i.e., the schedule uses exactly m machines, and

$$\sum_{k \in \mathbf{K}} k(i)\, y_k = |N'_i| \quad \text{for all } i \in I, \tag{9}$$

i.e., the set of jobs assigned to the machines is N'. We denote the set of vectors $y \in \mathbb{N}_0^{\mathbf{K}}$ satisfying (8) and (9) by $\mathbf{S}'$. Thus, $\mathbf{S}'$ represents the set of all schedules for N'_{L} (up to permutations of machines and up to permutations of equal-size jobs). If we replace $|N'_i|$ with $|N_i|$ on the right-hand side of (9), then constraints (8) and (9) characterize the set $\mathbf{S}$ of all schedules for N_{L}.

For a configuration $k \in \mathbf{K}$, let

$$\mathrm{load}(k) := \sum_{i \in I} (1+\epsilon)^i\, k(i)$$

denote the load of a machine obeying configuration k. The makespan of a schedule $y \in \mathbf{S}$ or $y \in \mathbf{S}'$ is thus equal to $\max\{\mathrm{load}(k) \mid y_k > 0\}$. For $\mu \geq 0$ we denote the set of configurations of load of at most μ by

$$\mathbf{K}(\mu) := \{k \in \mathbf{K} \mid \mathrm{load}(k) \leq \mu\}.$$

The subsets of all schedules for N_{L} and N'_{L} with makespan at most μ are denoted by

$$\mathbf{S}(\mu) := \{y \in \mathbf{S} \mid y_k = 0 \text{ if } \mathrm{load}(k) > \mu\}$$

and

$$\mathbf{S}'(\mu) := \{y \in \mathbf{S}' \mid y_k = 0 \text{ if } \mathrm{load}(k) > \mu\},$$

respectively.

In the following, we usually interpret a schedule $y \in \mathbf{S}(\mu)$ or $y \in \mathbf{S}'(\mu)$ as a vector in $\mathbb{N}_0^{\mathbf{K}(\mu)}$ by ignoring all zero entries corresponding to configurations in $\mathbf{K} \backslash \mathbf{K}(\mu)$. We can thus write

$$\mathbf{S}(\mu) = \{y \in \mathbb{N}_0^{\mathbf{K}(\mu)} \mid A(\mu) y = b\}$$

and

$$\mathbf{S}'(\mu) = \{y \in \mathbb{N}_0^{\mathbf{K}(\mu)} \mid A(\mu) y = b'\},$$

where $A(\mu)$ is a matrix in $\mathbb{N}_0^{(1+|I|)\times|\mathbf{K}(\mu)|}$ and b, b' are vectors in $\mathbb{N}_0^{1+|I|}$. The first row of the linear system $A(\mu)y = b'$ corresponds to constraint (8); the remaining $|I|$ rows correspond to constraints (9).

The next observation is crucial for applying sensitivity analysis of integer linear programming later.

Observation 6.3. The vectors b and b' are equal for all but at most one entry, where they differ by one.

Proof. The first entry of both b and b' is equal to m (see the right-hand side of (8)). The remaining entries are equal to $|N_i|$ and $|N'_i|$, respectively. Notice that $|N_i| = |N'_i|$ unless $j \in N'_i$. In the latter case, $|N'_i| = |N_i| + 1$. □

The next lemma contains the key insight for the result presented in this section.

Lemma 6.2. *Consider a schedule for N_{L} and denote its makespan by μ. This schedule can be turned into a schedule for N'_{L} with makespan at most* $\max\{\mu, \mathrm{OPT}(N'_{\mathrm{L}})\}$ *while touching at most* $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ *machines.*

Proof. Let $\mu' := \max\{\mu, \mathrm{OPT}(N'_{\mathrm{L}})\}$. If $N_{\mathrm{L}} = N'_{\mathrm{L}}$ (i.e., if j is small), there is nothing to be shown. Moreover, if $\mu' \geq 2\,\mathrm{OPT}(N'_{\mathrm{L}})$, the required schedule can be obtained by simply assigning j to the least loaded machine. Notice that the load of this machine as well as the size of job j are at most $\mathrm{OPT}(N'_{\mathrm{L}})$. We can therefore assume in the following that $j \in N'_{\mathrm{L}}$ and $\mu < 2\mathrm{OPT}(N'_{\mathrm{L}})$.

Let $y \in \mathbf{S}(\mu')$ denote the given schedule for N_{L}. Then, y satisfies

$$A(\mu')y = b, \quad y \in \mathbb{N}_0^{\mathbf{K}(\mu')}. \tag{10}$$

We are looking for a schedule $y' \in \mathbf{S}'(\mu')$, that is, y' must satisfy

$$A(\mu')y' = b', \quad y' \in \mathbb{N}_0^{\mathbf{K}(\mu')}. \tag{11}$$

By Observation 6.3, the right-hand sides of the systems of linear equations in (10) and (11) are equal for all but one entry, where they differ by one. Using a sensitivity result from integer linear programming[4] there exists a solution y' to (11) satisfying

$$\|y - y'\|_\infty \le 3|\mathbf{K}(\mu')|\Delta,$$

where Δ is an upper bound on the absolute value of any subdeterminant of the matrix $A(\mu')$. The number of machines whose configurations are different in schedules y and y' is bounded from above by

$$\|y - y'\|_1 \le |\mathbf{K}(\mu')| \cdot \|y - y'\|_\infty \le 3|\mathbf{K}(\mu')|^2 \Delta. \tag{12}$$

To complete the proof, we have to show that the right-hand side of (12) is constant. First we give an upper bound on the number of configurations $|\mathbf{K}(\mu')|$, i.e., on the number of machine configurations with load of at most μ'. Because each job in N'_{L} has a size of at least

$$\epsilon \,\mathrm{LB}(N') \overset{(1)}{\ge} \frac{\epsilon}{2}\,\mathrm{OPT}(N') \ge \frac{\epsilon}{2}\,\mathrm{OPT}(N'_{\mathrm{L}}) > \frac{\epsilon}{4}\,\mu',$$

there are fewer than $4/\epsilon$ jobs in any configuration $k \in \mathbf{K}(\mu')$ with load at most μ'; in particular, it holds that $k(i) < 4/\epsilon$ for all $i \in I$. This yields

$$|\mathbf{K}(\mu')| \le \left(\frac{4}{\epsilon}\right)^{|I|} = 2^{|I|\log(4/\epsilon)} \in 2^{O((1/\epsilon)\log^2(1/\epsilon))} \tag{13}$$

by Observation 6.2.

We now turn to Δ, which must be an upper bound on the absolute value of any subdeterminant of the matrix $A(\mu')$. The entries in the first row of $A(\mu')$ are all 1 (see (8)) and the remaining entries are of the form $k(i) < 4/\epsilon$ (see (9)). The maximum dimension of a square submatrix of $A(\mu')$ is at most the number of rows which is $1 + |I|$. Hence, the absolute value of any subdeterminant is upper-bounded by

$$\Delta := (1+|I|)!\left(\frac{4}{\epsilon}\right)^{|I|} \le \left((1+|I|)\frac{4}{\epsilon}\right)^{|I|} = 2^{|I|(\log(1+|I|)+\log(4/\epsilon))} \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}. \tag{14}$$

The claimed result follows by plugging the upper bound (13) on $|\mathbf{K}(\mu')|$ and the upper bound (14) on Δ into (12). □

We now return to the complete set of jobs N' also including small jobs. Building upon Lemma 6.2, we show how to incorporate small jobs in the following theorem.

THEOREM 6.1. *Any $(1+\epsilon)$-approximate schedule for N can be turned into a $(1+\epsilon)$-approximate schedule for $N' = N \cup \{j\}$ such that the total size of jobs that have to be moved is bounded by a constant $\beta(\epsilon)$ times p_j with $\beta(\epsilon) \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}$.*

PROOF. We distinguish two cases. If the newly arrived job is small, i.e., $p_j < \epsilon\,\mathrm{LB}(N')$, then j can simply be assigned to a least loaded machine by Corollary 2.1 and no job in N has to be moved.

It remains to consider the situation when job j is large. The given schedule for N induces a schedule for N_{L} with makespan $\mu \le (1+\epsilon)\,\mathrm{OPT}(N) \le (1+\epsilon)\,\mathrm{OPT}(N')$. By Lemma 6.2, the latter schedule can be turned into a schedule for N'_{L} with makespan of at most

$$\max\{\mu, \mathrm{OPT}(N'_{\mathrm{L}})\} \le (1+\epsilon)\,\mathrm{OPT}(N'),$$

by touching only $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ machines. In the following, this subset of machines of constant size is denoted by M'. We construct a schedule for N' as follows:

(i) Start with the schedule for N'_{L} discussed above.

(ii) The small jobs in N_{S} that the given schedule for N assigned to one of the machines in $M \setminus M'$ are assigned to the same machine again.

(iii) The remaining jobs in $N_{\mathrm{S}} = N'_{\mathrm{S}}$ are assigned one after another to a least loaded machine.

[4] For the convenience of the reader, we quote the used result: COROLLARY 17.2a (from Schrijver [28]). Let A be an integral $m \times n$-matrix, so that each subdeterminant of A is at most Δ in absolute value, let b' and b'' be column m-vectors, and let c be a row n-vector. Suppose $\max\{cx \mid Ax \le b';\, x \text{ integral}\}$ and $\max\{cx \mid Ax \le b'';\, x \text{ integral}\}$ are finite. Then for each optimum solution z' of the first maximum there exists an optimum solution z'' of the second maximum such that $\|z' - z''\|_\infty \le n\Delta(\|b' - b''\|_\infty + 2)$.

The makespan of the partial schedule constructed in steps (i) and (ii) is bounded by the maximum of the makespan of the given schedule for N and the optimal makespan of the schedule for N'_L. It is thus bounded by $(1+\epsilon)\text{OPT}(N')$. Assigning small jobs greedily to a least loaded machine in step (iii) therefore results in a $(1+\epsilon)$-approximate schedule for N' by Corollary 2.1.

Finally, notice that in the whole process, only jobs that have initially been scheduled on machines M' are moved. The total size of these jobs is bounded from above by

$$2\,\text{OPT}(N')\,|M'| \overset{(1)}{\leq} 4\text{LB}(N')|M'| \leq p_j \frac{4}{\epsilon}|M'|.$$

We can therefore set $\beta(\epsilon) := (4/\epsilon)|M'| \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}$. □

COROLLARY 6.1. *Theorem 6.1 can be strengthened by adding the requirement that the schedules for N and N' only differ on constantly many machines.*

PROOF. In the proof of Theorem 6.1, the only critical step in the construction of the schedule for N' is the reassignment of small jobs. In general, the number of small jobs that need to be reassigned is not constant. Moreover, in the worst case, every such small job is assigned to a different machine.

This problem can be solved by grouping small jobs into batches of size roughly $\epsilon\,\text{LB}(N')$. Each batch can be treated as one single small job. In this way we can reduce the number of small jobs that need to be reassigned to a constant. □

We can finally state the main result of this section.

THEOREM 6.2. *There exists a $(1+\epsilon)$-competitive online algorithm with constant migration factor $\beta(\epsilon) \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}$ such that the running time for incorporating a newly arrived job is constant. More precisely, the algorithm maintains and updates a data structure in time $2^{2^{O((1/\epsilon)\log^2(1/\epsilon))}}$ in each iteration.*[5] *From this data structure a $(1+\epsilon)$-approximate schedule can be obtained in time linear in the number of jobs (and not depending on ϵ).*

PROOF. Before we discuss the required data structure in detail, we first give a rough outline and intuition.

We again consider the scheduling strategy described in the proof of Theorem 6.1. In Step (i) we ignore all small jobs and construct the schedule for the large jobs N'_L. This schedule is stored as a vector $y \in \mathbb{N}_0^{\mathbf{K}}$ (array) in our data structure. It follows from the proof of Lemma 6.2 that only a constant number of different possibilities have to be taken into consideration in order to find this schedule. The schedule for N'_L in Step (i) can thus be found in constant time. Step (ii) is trivial; nothing has to be changed on the machines in $M \setminus M'$.

We finally turn to Step (iii), where the small jobs that were previously assigned to M' are successively reassigned to a least loaded machine. To perform this step in constant time, we apply the same trick as in the proof of Corollary 6.1 and assume that small jobs are grouped into batches of size roughly $\epsilon\,\text{LB}(N')$. Each batch can be treated as one single small job such that the number of small jobs that we have to deal with in Step (iii) is constant. We finally have to argue that we can always find a least loaded machine in constant time. In fact, because we are only looking for a $(1+O(\epsilon))$-approximate schedule, it is not really necessary to find a least loaded machine. It suffices to find a machine whose load differs from the minimum load by at most $\epsilon\,\text{LB}(N')$. We can therefore assume that the machines are assigned to $O(1/\epsilon)$ buckets according to their current load where every bucket contains machines whose load is within a range of at most $\epsilon\,\text{LB}(N')$. In Step (iii) it then suffices to always choose a machine from the nonempty bucket that contains the machines of smallest loads. This can obviously be done in $O(1/\epsilon)$ time.

We now give a detailed description of the data structure. We assume in the following without loss of generality that $\epsilon < 1$. A $(1+\epsilon)$-approximate schedule for N can be represented using the following simple data structure. We assume that initially the schedule is given to us in this form. Later we show how to update it in constant time while scheduling job j. The machine configurations are represented using structures as shown in Figure 5.

There is an array of *Config Head*s of dimension $|\mathbf{K}(4\,\text{LB}(N'))| \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}$ (notice that $4\,\text{LB}(N') \geq 2\text{OPT}(N') > (1+\epsilon)\text{OPT}(N')$), one for each possible configuration $k \in \mathbf{K}(4\,\text{LB}(N'))$. Each *Config Head* corresponding to a configuration k stores the number of machines y_k scheduled according to this configuration. Moreover, each *Config Head* points to the list of machines (list of *Machine Nodes*) obeying that configuration.

Each *Machine Node* points to the list of jobs scheduled on that machine, grouped into batches. Each large job is put into a batch of its own. The small jobs on a machine are grouped into at most $O(1/\epsilon)$ batches of size at most $\epsilon\,\text{LB}(N)$. Thus there is only a constant number $O(1/\epsilon)$ of batches on any machine. Each batch has

[5] Here we make the common assumption that operations with numbers of polynomial size take constant time.

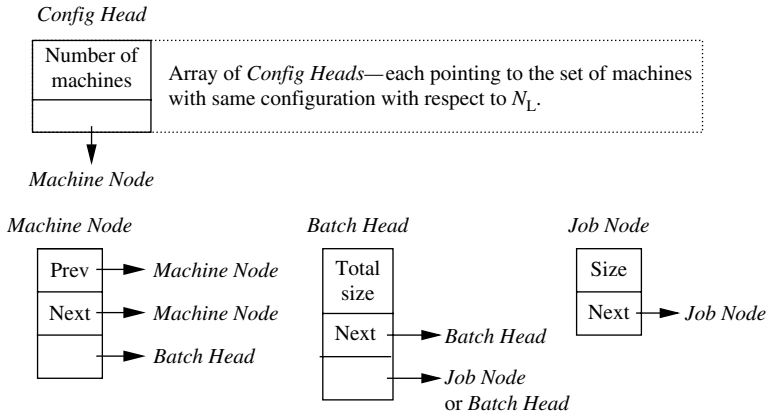

FIGURE 5. The date structure used to represent a schedule.

a *Batch Head* that points to the list of jobs (*Job Nodes*) in the batch. Because these jobs can recursively be grouped into smaller batches again, the list might also contain *Batch Heads*. The jobs that are scheduled on a machine are thus represented by a tree rooted at the *Machine Node*, whose inner nodes are batches and whose leaves are the jobs. Because every inner node has at least two child nodes, the size of the tree is linear in the number of jobs on the machine (i.e., leaf nodes).

Finally, the machines are partitioned into $O(1/\epsilon)$ buckets according to their current load. Each bucket contains machines whose load lies within an interval of size of at most $\epsilon \operatorname{LB}(N')$.

We next discuss the details of updating the data structure when scheduling job j. If j is small (i.e., $p_j < \epsilon \operatorname{LB}(N')$), then:

(i) Pick any machine i from the nonempty bucket containing the least loaded machines. Because there are $O(1/\epsilon)$ buckets, this takes $O(1/\epsilon)$ time.

(ii) Find a batch of small jobs on machine i of total size at most $\epsilon \operatorname{LB}(N') - p_j$ and add job j to this batch. If no such batch exists, create a new batch for job j. Consider all batches and merge small batches into larger batches of size at most $\epsilon \operatorname{LB}(N')$. All this takes $O(1/\epsilon)$ time because there are at most $O(1/\epsilon)$ batches on machine i.

(iii) Reassign machine i to a bucket according to its new load. Also, this takes $O(1/\epsilon)$ time.

The overall time needed to assign a small job to an approximately least loaded machine is thus $O(1/\epsilon)$. If the new job j is large (i.e., $p_j \geq \epsilon \operatorname{LB}(N')$), we proceed as follows:

(iv) Enumerate all $2^{2^{O((1/\epsilon)\log^2(1/\epsilon))}}$ schedules (vectors) $y' \in \mathbf{S}'$ for N'_L with $\|y - y'\|_\infty \in 2^{O((1/\epsilon)\log^2(1/\epsilon))}$. Choose one with minimum makespan. This takes $2^{2^{O((1/\epsilon)\log^2(1/\epsilon))}}$ time.

(v) The componentwise difference between y and y' specifies a subset of configurations and a nonzero number of machines that should be modified for each such configuration. For each configuration we remove the required number of machines from the front of the machine list pointed to by the respective *Config Head*. Because the total number of required machines is $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ (see Lemma 6.2), this takes $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ time.

(vi) Remove all batches of small jobs from these machines and reschedule the remaining jobs among these machines to achieve the configurations indicated by y'. Because the number of jobs and batches on each machine is $O(1/\epsilon)$, this takes $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ time.

(vii) Assign the machines to their new *Config Head*s and to buckets according to their load. This takes time $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ again.

(viii) Each of the $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ batches of small jobs is being reassigned as in the small job case discussed above. This takes $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ time.

The running time of the entire procedure is $2^{2^{O((1/\epsilon)\log^2(1/\epsilon))}}$.

There is one more issue that needs to be discussed. Whenever a new job arrives, the lower bound LB on the makespan of an optimal schedule might increase so that the classification of large and small jobs has to be updated. This means that the list of configurations that are currently being used changes. Certain classes of formerly large jobs are now small, and therefore neglected in the updated configurations. Notice, however, that machines belonging to the same *Config Head* before the update still belong to a common *Config Head* after the update. It is thus sufficient to update the array of *Config Heads*, potentially merging several old *Config Heads* into a new one without touching all *Machine Nodes*. This can be done in $2^{O((1/\epsilon)\log^2(1/\epsilon))}$ time.

Notice that there is no need to assign the jobs that became small immediately to batches of small jobs. On each machine this is done as soon as the machine is being touched in the future. We always preserve the invariant that the number of batches of small jobs is $O(1/\epsilon)$ on any machine.

Due to the monotonically increasing lower bound LB, we also need to update the bucket structure from time to time by merging two buckets into one. This can obviously also be done in $O(1/\epsilon)$ time.

Notice finally that from the described data structure a corresponding schedule can be obtained in linear time. For each *Machine Node*, parse the tree containing the jobs scheduled on the machine. As mentioned above, the size of the tree is linear in the number of jobs contained in it. This concludes the proof. □

The migration factor $\beta(\epsilon)$ we obtain is exponential in $1/\epsilon$. This raises the question whether a migration factor that is polynomially bounded in $1/\epsilon$ is possible. We do not know of an instance proving the converse and leave the question as an open problem.

As a corollary to Theorem 6.2, we obtain the following known result; see, e.g., Hochbaum [17].

COROLLARY 6.2. *There is a polynomial time approximation scheme with linear running time for the scheduling problem* $P \| C_{\max}$.

7. Maximizing the minimum machine load. An alternative, yet less frequently used, objective for machine scheduling is to maximize the minimum machine load. We have, however, a concrete application using this objective function that was the original motivation for our interest in bounded migration: Storage area networks (SAN) usually connect many disks of different capacity and grow over time. A convenient way to hide the complexity of a SAN is to treat it as a single big, fault-tolerant disk of huge capacity and throughput (Brinkmann et al. [7], Sanders [26]). A simple scheme with many nice properties implements this idea if we manage to partition the SAN into several subservers (Sanders [26]) of about equal size. Mapping to the scheduling framework, the disks correspond to jobs and the subservers correspond to machines. Each subserver stores the same amount of data. For example, if we have two subservers, each of them stores all the data to achieve a fault tolerance comparable to mirroring in ordinary redundant arrays of inexpensive disks (RAID) (Patterson et al. [22]). More subservers allow for a more flexible trade-off between fault tolerance, redundancy, and access granularity. In any case, the capacity of the server is determined by the *minimum* capacity of a subserver. Moreover, it is not acceptable to completely reconfigure the system when a new disk is added to the system or when a disk fails. Rather, the user expects a "proportionate response," i.e., if she adds a disk of x GByte she will not be astonished if the system moves data of this order of magnitude, but she would complain if much more is moved. Our theoretical investigation confirms that this "common sense" expectation is indeed reasonable.

We concentrate on the case without job departures (disk failures) and present a $\frac{1}{2}$-competitive online algorithm with migration factor 1. In order to motivate the scheduling procedure, we first discuss a negative result. The following lemma shows that it is not possible to start with an arbitrary $\frac{1}{2}$-approximate schedule for N and obtain a $\frac{1}{2}$-approximate schedule for $N \cup \{j\}$ with constant migration factor.

LEMMA 7.1. *There is a* $\frac{1}{2}$*-approximate schedule on* m *machines such that upon the arrival of a new job it is not possible to obtain a* $\frac{1}{2}$*-approximate schedule with a migration factor less than* $m-2$.

PROOF. Consider the following m-machine schedule for $3m-1$ jobs of unit size. Machine 1 contains $2m$ jobs whereas all remaining machines contain only one job. The optimal minimum load for this instance is 2 and the considered schedule is $\frac{1}{2}$-approximate. When a new job of size 1 arrives, the new optimal minimum load is 3. But to achieve minimum load greater than 1, any strategy has to move at least $m-2$ jobs. □

We show that the following simple strategy, which is very similar to Procedure 1, leads to a $\frac{1}{2}$-competitive online algorithm with migration factor 1.

PROCEDURE 5.
Input: *A schedule for a set of jobs* N *and a new job* $j \notin N$.
Output: *A schedule for* $N \cup \{j\}$.
Repeatedly remove jobs from the least loaded machine in any order; *stop before the total size of removed jobs exceeds* p_j. *Assign job* j *to this machine. Assign the removed jobs successively to a least loaded machine.*

With respect to a given schedule, a machine is called a *multijob machine* if at least two jobs are assigned to it. In the analysis of Procedure 5, we make use of a special structural property of schedules. A schedule for a set of jobs N has property $(**)$ if the load of any multijob machine is at most twice the minimum load. For example, a schedule constructed by list scheduling in order of nonincreasing processing times always has property $(**)$. We show that Procedure 5 maintains property $(**)$.

LEMMA 7.2. *Given a schedule for N satisfying property $(**)$ and a new job $j \notin N$, Procedure 5 outputs a schedule for $N \cup \{j\}$ satisfying property $(**)$.*

PROOF. The proof is based on the following claim.

CLAIM. *Given a schedule satisfying property $(**)$ and a new job whose size is at most the minimum machine load, assigning the new job to a least loaded machine maintains property $(**)$.*

This claim is true because incorporating the new job does not decrease the minimum machine load and the load of the machine the new job is assigned to increases by at most a factor of 2.

Let Machine i be a least loaded machine in the given schedule S for N. As a consequence of the claim, reassigning the removed jobs in Procedure 5 successively to a least loaded machine maintains property $(**)$. It therefore remains to show that property $(**)$ holds in the intermediate schedule immediately after assigning job j to machine i (before assigning the removed jobs). Notice that the minimum machine load in this intermediate schedule is at least the minimum machine load of the given schedule for N. If $p_j \leq p(S^{-1}(i))$, then property $(**)$ obviously holds for the intermediate schedule. Otherwise, if $p_j > p(S^{-1}(i))$, then all jobs are removed from machine i such that machine i is not a multijob machine in the intermediate schedule. Thus, property $(**)$ holds in this case as well. □

THEOREM 7.1. *For the objective to maximize the minimum machine load, any schedule satisfying property $(**)$ is $\frac{1}{2}$-approximate. In particular, scheduling every newly arriving job according to Procedure 5 is a $\frac{1}{2}$-competitive algorithm with migration factor 1 for this objective.*

PROOF. The proof is based on the following claim.

CLAIM. *Consider an arbitrary schedule. If there is at most one job on every machine, the schedule is optimal. Otherwise, the maximum load of a multiple-job machine is an upper bound on the optimum.*

The first part of the claim is clear. Consider an arbitrary schedule S with at least one machine containing at least two jobs. Let L denote the maximum load of a multiple-job machine in S. In the following we call a job *large* if its size is strictly larger than L. Notice that S schedules each large job on a machine of its own.

Consider an optimum schedule S^* and modify S^* as follows. For each machine i: If machine i contains a large job, then delete machine i and all jobs scheduled on it. The resulting schedule is denoted by $\overline{S}^*$. We apply exactly the same procedure to schedule S and denote the resulting schedule by $\overline{S}$. Notice that $\overline{S}^*$ schedules a subset of the jobs scheduled by $\overline{S}$ because each large job occupies a machine of its own in S. Moreover, for the same reason, $\overline{S}^*$ uses at least as many machines as $\overline{S}$. Thus, the minimum machine load in $\overline{S}^*$ is at most the maximum machine load in $\overline{S}$ that is equal to L. In particular, there exists a machine in $\overline{S}^*$, and thus in S^*, whose load is at most L. This concludes the proof of the claim.

As an immediate consequence of the claim, any schedule satisfying property $(**)$ is $\frac{1}{2}$-approximate. Together with Lemma 7.2, this yields a bound of $\frac{1}{2}$ on the competitiveness. Finally, the bound on the migration factor is clear from the description of Procedure 5. □

We close this section with an interesting open problem. Woeginger [33] presents a PTAS for the problem to maximize the minimum machine load on identical parallel machines. This raises the question of whether an online PTAS with constant migration factor $\beta(\epsilon)$ as in §6 can be obtained for the max-min objective function. Lemma 7.1 seems to indicate that this is more tricky than for the makespan objective. At least, it is not enough to only take care of the objective function in each iteration, but also the overall structure of the schedule has to be "reasonable" such that it can be easily adapted in future iterations without too much migration.

Acknowledgments. The authors thank the anonymous referees for their careful reading and their numerous suggestions, which led to an improved presentation of the paper. They would also like to thank Gerhard Woeginger for interesting discussions and helpful comments on the topic of this paper. Finally, the authors are much indebted to Jiří Sgall for pointing out the result presented in §4. Part of this work was done while all three authors were at Max-Planck-Institut für Informatik in Saarbrücken. The last author is supported by DFG Research Center MATHEON in Berlin.

References

[1] Afrati, F., E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, et al. 1999. Approximation schemes for minimizing average weighted completion time with release dates. *Proc. 40th Annual IEEE Sympos. Foundations Comput. Sci.*, New York, 32–43.
[2] Albers, S. 1999. Better bounds for online scheduling. *SIAM J. Comput.* **29** 459–473.
[3] Albers, S. 2003. Online algorithms: A survey. *Math. Programming* **97** 3–26.

[4] Andrews, M., M. X. Goemans, L. Zhang. 1999. Improved bounds for on-line load balancing. *Algorithmica* **23** 278–301.
[5] Azar, Y., L. Epstein. 1998. On-line machine covering. *J. Algorithms* **1** 67–77.
[6] Bartal, Y., A. Fiat, H. Karloff, R. Vohra. 1995. New algorithms for an ancient scheduling problem. *J. Comput. System Sci.* **51** 359–366.
[7] Brinkmann, A., K. Salzwedel, C. Scheideler. 2002. Compact, adaptive placement schemes for non-uniform requirements. *14th ACM Sympos. Parallel Algorithms and Architectures*. ACM, New York, 53–62.
[8] Chen, B., A. van Vliet, G. J. Woeginger. 1994. Lower bounds for randomized online scheduling. *Inform. Processing Lett.* **51** 219–222.
[9] Coffman, Jr., E. G., M. R. Garey, D. S. Johnson. 1978. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.* **7** 1–17.
[10] Epstein, L., A. Levin. 2006. A robust APTAS for the classical bin packing problem. M. Bugliesi, B. Preneel, V. Sassone, I. Wegener, eds. *Automata, Languages and Programming, Lecture Notes in Computer Science*, Vol. 4051, Springer, Berlin, 214–225.
[11] Fleischer, R., M. Wahl. 2000. Online scheduling revisited. *J. Scheduling* **3** 343–353.
[12] Friesen, D. K. 1984. Tighter bounds for the multifit processor scheduling algorithm. *SIAM J. Comput.* **13** 170–181.
[13] Garey, M. R., D. S. Johnson. 1978. Strong np-completeness results: Motivation, examples and implications. *J. ACM* **25** 499–508.
[14] Graham, R. L. 1966. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* **45** 1563–1581.
[15] Graham, R. L. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* **17** 263–269.
[16] Graham, R. L., E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.* **5** 287–326.
[17] Hochbaum, D. S. 1996. Various notions of approximation: Good, better, best, and more. D. S. Hochbaum, ed. *Approximation Algorithms for NP-Hard Problems*. Thomson, PWS Publishing Company, Boston, 346–398.
[18] Hochbaum, D. S., D. B. Shmoys. 1987. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J. ACM* **34** 144–162.
[19] Karger, D. R., S. J. Phillips, E. Torng. 1996. A better algorithm for an ancient scheduling problem. *J. Algorithms* **20** 400–430.
[20] Langston, M. A. 1981. Processor scheduling with improved heuristic algorithms. Ph.D. thesis, Texas A&M University, College Station.
[21] Lenstra, H. W. 1983. Integer programming with a fixed number of variables. *Math. Oper. Res.* **8** 538–548.
[22] Patterson, D., G. Gibson, R. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). *Proc. ACM SIGMOD'88*, ACM, New York, 109–116.
[23] Rudin, J. F., III. 2001. Improved bounds for the on-line scheduling problem. Ph.D. thesis, The University of Texas at Dallas, Richardson.
[24] Rudin, J. F., III, R. Chandrasekaran. 2003. Improved bounds for the online scheduling problem. *SIAM J. Comput.* **32** 717–735.
[25] Sahni, S. 1976. Algorithms for scheduling independent tasks. *J. ACM* **23** 116–127.
[26] Sanders, P. 2004. Algorithms for scalable storage servers. *SOFSEM 2004: Theory and Practice of Computer Science, LNCS*, Vol. 2932. Springer, Berlin, 82–101.
[27] Sanders, P., N. Sivadasan, M. Skutella. 2004. Online scheduling with bounded migration. J. Diaz, J. Karhumäki, A. Lepistö, D. Sannella, eds. *Automata, Languages and Programming, Lecture Notes in Computer Science*, Vol. 3142, Springer, Berlin, 1111–1122.
[28] Schrijver, A. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, UK.
[29] Sgall, J. 1997. A lower bound for randomized on-line multiprocessor scheduling. *Inform. Processing Lett.* **63**(1) 51–55.
[30] Sgall, J. 1998. On-line scheduling—A survey. A. Fiat, G. J. Woeginger, eds. *Online Algorithms: The State of the Art, Lecture Notes in Computer Science*, Vol. 1442. Springer, Berlin, 196–231.
[31] Sgall, J. 2008. Personal communication. November.
[32] Westbrook, J. 2000. Load balancing for response time. *J. Algorithms* **35** 1–16.
[33] Woeginger, G. J. 1997. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.* **20** 149–154.