Online Scheduling with Bounded Migration

Peter Sanders, Naveen Sivadasan, and Martin Skutella

Max-Planck-Institut für Informatik, Saarbrücken, Germany, {sanders,ns,skutella}@mpi-sb.mpg.de

Abstract. Consider the classical online scheduling problem where 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 β times the size of the arriving job.

Our main result is a linear time 'online approximation scheme', that is, a family of online algorithms with competitive ratio $1 + \epsilon$ and constant migration factor $\beta(\epsilon)$, for any fixed $\epsilon > 0$. This result is of particular importance if considered in the context of sensitivity analysis: While a newly arriving job may force a complete change of the entire structure of an optimal schedule, only very limited 'local' changes suffice to preserve near-optimal solutions. We believe that this concept will find wide application in its own right.

We also present simple deterministic online algorithms with migration factors $\beta = 2$ and $\beta = 4/3$, respectively. Their competitive ratio $3/2$ beats the lower bound on the performance of any online algorithm in the classical setting without migration. We also present improved algorithms and similar results for closely related problems. In particular, there is a short discussion of corresponding results for the objective to maximize the minimum load of a machine. The latter problem has an application for configuring storage servers that was the original motivation for this work.

1 Introduction

A classical scheduling problem. 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, Lawler, Lenstra, & Rinnooy Kan [GLLR79], this scheduling problem is denoted by $\mathrm{P}\,||C_{\max}$ and it is well known to be strongly NP-hard [GJ78].

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.

A new online scheduling paradigm. 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.

Approximation algorithms. For an offline optimization problem, an *approximation algorithm* efficiently (in polynomial time) constructs schedules whose values are within a constant factor $\alpha \geq 1$ 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).

Competitive analysis. In a similar way, *competitive analysis* evaluates solutions computed in the online setting. An online algorithm achieves *competitive ratio* $\alpha \geq 1$ 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.

Known results and related work. For the online machine scheduling problem, Graham's *list scheduling* algorithm keeps the makespan within a facor $2 - 1/m$ of the offline optimum [Gra66]: Schedule a newly arriving job on the least loaded machine. It can also easily be seen that this bound is tight.

For the offline setting, Graham showed three years later that sorting the jobs in order of non-increasing sizes before feeding them to the list scheduling algorithm yields an approximation algorithm with performance ratio $4/3 - 1/(3m)$ [Gra69]. Later, exploiting the relationship between the machine scheduling problem under consideration and the binpacking problem, algorithms with improved approximation ratios have been obtained in a series of works [CGJ78,Fri84,Lan81].

Finally, polynomial time approximation schemes for a constant number of machines and for an arbitrary number of machines are given in [Gra69,Sah76] and by Hochbaum & Shmoys [HS87], 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., [Hoc96]): replace the dynamic program with an integer linear program on a fixed number of variables and constraints which can be solved in constant time [Len83].

In a series of papers, increasingly complicated online algorithms with better and better competitive ratios beating the Graham bound 2 have been developed [BFKV95,KPT96,Alb99]. The best result known to date is a 1.93-competitive algorithm due to Fleischer & Wahl [FW00]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin [Rud01]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$ [CvVW94,Sga97].

For more results on online algorithms and scheduling we refer to the recent survey articles by Albers [Alb03] and Sgall [Sga98].

New contributions. In Section 3 we describe a simple online algorithm which achieves approximation ratio $3/2$ using a moderate migration factor $\beta = 2$. Notice that already this result 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 $4/3$ while maintaining competitive ratio $3/2$. On the other hand we show that our approach does not allow for migration factor 1 and competitive ratio $3/2$. Furthermore, an improved competitive ratio $4/3$ can be achieved with migration factor 4. For two machines, we can achieve competitive ratio $7/6$ with a migration factor of one. This ratio is tight for migration factor one.

In Section 5 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 load. It is well known [AE98] 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 which is 2-competitive already for migration factor $\beta = 1$.

The main result of this paper can be found in Section 4. 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 one, i.e., optimality. We provide interpretations of these results in several different contexts:

Online algorithms. Online scheduling with bounded job migration is a relaxation of the classical online paradigm. Obviously, there is a tradeoff 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 tradeoff curve: Any desired quality can be guaranteed while relaxing the online paradigm only moderately by allowing for a constant migration factor.

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 impelling question in sensitivity analysis. As indicated above, for 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.

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.

The underlying details of the presented online approximation scheme have the same roots as the original PTAS by Hochbaum & Shmoys [HS87] and its refinements [Hoc96]. 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. Since 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 deep results from sensitivity analysis of integer programs in order to prove that the schedule of large jobs needs to be changed only slightly. For a detailed account of integer programming theory we refer to the books [Sch86,NW88].

Due to space limitations, many details have been moved to an appendix.

2 Preliminaries

Let the set of *machines* be denoted by $M = \{1, \dots, m\}$. The set of *jobs* is $\{1, \dots, n\}$ where job j arrives in round j. Let p_j denote the positive *processing time* or the *size* of job j. For a subset of jobs N, the *total processing time* of jobs in N is $p(N) := \sum_{j \in N} p_j$; let $p_{\max}(N) := \max_{j \in N} p_j$. For a subset of jobs N, let $\mathrm{opt}(N)$ denote the *optimal makespan*. If the subset of jobs N and a newly arrived job j are clear from the context, we sometimes also use the shorter notation $\mathrm{opt} := \mathrm{opt}(N)$ and $\mathrm{opt}' := \mathrm{opt}(N \cup \{j\})$. It is easy to observe that $\mathrm{lb}(N) := \max\{p(N)/m, p_{\max}(N)\}$ is a lower bound on $\mathrm{opt}(N)$ satisfying

$$\mathrm{lb}(N) \le \mathrm{opt}(N) \le 2\mathrm{lb}(N) . \tag{1}$$

The following well-known fact is used frequently in the subsequent sections.

Observation 1. *For a set of jobs N, consider an arbitrary schedule with makespan α. Assigning a new job j to the least loaded machine yields a schedule with makespan at most* $\max\{\alpha, \mathrm{opt}(N \cup \{j\}) + (1-1/m)p_j\}$.

3 Strategies with Small Migration Factor

We consider the problem of scheduling jobs arriving one after another on m parallel machines so as to minimize the makespan. We first show a very simple $3/2$-competitive algorithm with migration factor 2. The algorithm is as follows:

Procedure FILL_1 :
Upon arrival of a new job j, choose one of the following two options minimizing the resulting makespan.

Option 1: Assign job j to the least loaded machine.
Option 2: Let i be the machine minimizing the maximum job size. Repeatedly remove jobs from this machine; stop before the total size of removed jobs exceeds $2p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

Theorem 1. *Procedure* FILL_1 *is* $\left(\frac{3}{2} - \frac{1}{2m}\right)$*-competitive with migration factor* 2.

Proof. From the description of FILL_1 , it is clear that the migration factor is at most 2. In order to prove competitiveness, we consider an arbitrary $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate schedule for a set of jobs N and show that incorporating a new job j according to FILL_1 results in a new schedule which is still $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate. In the following, a job is called *small* if its processing time is at most $\mathrm{opt}'/2$, otherwise it is called *large*. If the new job j is small, then the first option yields makespan at most $\left(\frac{3}{2} - \frac{1}{2m}\right)\mathrm{opt}'$ by Observation 1. Thus, we can assume from now on that j is large.

Since there can be at most m large jobs in $N \cup \{j\}$, all jobs on the machine chosen in the second option are small. Thus, after removing jobs from this machine as described above, the machine 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 this machine cannot increase its load above $\left(\frac{3}{2} - \frac{1}{2m}\right)\mathrm{opt}'$. Thus, using the same argument as above, assigning the removed small jobs successively to the least loaded machine yields again a $\left(\frac{3}{2} - \frac{1}{2m}\right)$-approximate schedule. □

Next we show that the migration factor can be decreased to $4/3$ without increasing the competitive ratio above $3/2$. This result is achieved by carefully modifying FILL_1 .

Procedure FILL_2 :
Upon arrival of j, choose the one of the following $m + 1$ options that minimizes the resulting makespan. (Brake ties in favor of option 0.)

Option 0: Assign job j to the least loaded machine.
Option i [for $i \in \{1, \ldots, m\}$]: Ignoring the largest job on machine i, consider the remaining jobs in order of non-increasing size and repeatedly remove them from the machine; stop before the total size of removed jobs exceeds $\frac{4}{3}p_j$. Assign job j to machine i. Assign the removed jobs successively to the least loaded machine.

The proof of the following result can be found in Appendix B.

Theorem 2. *Procedure* FILL_2 *is* $3/2$*-competitive with migration factor* $4/3$.

Robustness: All scheduling strategies for minimizing the makespan discussed in this paper 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 FILL_2 , the schedule should additionally satisfy that, on any machine, the load excluding the largest job in it is at most the optimum makespan. We

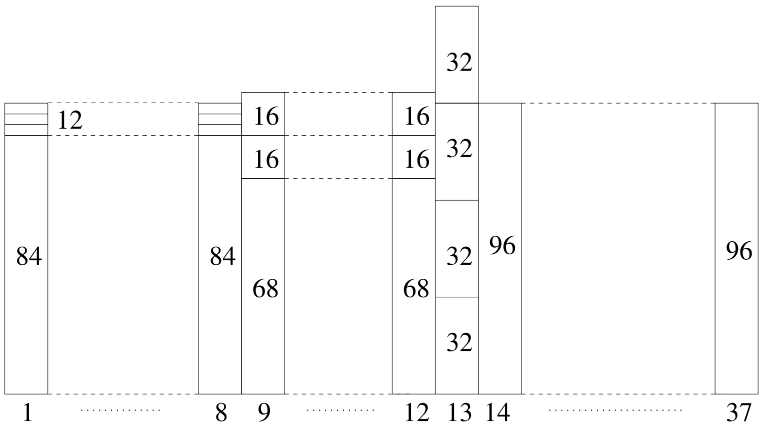

Fig. 1. A $3/2$-approximate schedule ($m = 37$). If a new job of size 86.1 arrives, jobs of total size at least 96 have to be moved in order to construct a schedule which is still $3/2$-approximate.

further remark that this is not an unreasonable requirement as even the simple list scheduling algorithm ensures this.

Theorems 1 and 2 raise the question which migration factor is really necessary to achieve competitive ratio $3/2$. We can prove that any robust strategy needs migration factor greater than 1 in order to maintain competitive ratio $3/2$.

Lemma 1. *There exists a* $3/2$*-approximate schedule such that, upon arrival of a particular job, migration factor* 1.114 *is needed to achieve* $3/2$*-competitiveness. Moreover, migration factor* 1 *only allows for competitive ratio* 1.52 *in this situation.*

Proof. The situation is depicted in Figure 1. There are 37 machines. Machines 1 to 8 are identically packed. Each of them has one job of size 84 and three jobs of size 4. Machines 9 to 12 are also identical. Each of them has one job of size 68 and two jobs of size 16. Machine 13 contains four jobs of size 32. Machines 14 to 37 contain one job of size 96 each. The size of the newly arriving job is 86.1. The optimal makespan is 100. To achieve a makespan of at most 150, it is necessary to migrate at least 3 jobs of size 32 from machine 13 to other machines. Hence, the migration factor is at least $96/86.1 > 1.114$. To show the lower bound on the competitiveness (the second part of the lemma), we set the size of the newly arriving job to 88 instead of 86.1. It is straightforward to check that the final optimal makespan is 100 and the best possible makespan achievable for any strategy with migration factor $\beta \leq 1$ is 152. □

Remark: An additional feature of FILL_1 and FILL_2 is that they are *local* in the sense that they migrate jobs only from the machine where the newly arrived job is assigned to. There is a class of optimal schedules for which, upon arrival of a new job, it is not possible to achieve a better competitive ratio than $3/2$ using only local migration. This holds even if an arbitrary migration factor is allowed. 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 migration. This bound converges to $3/2$ for large m. The example looks as follows: Machines 1 and 2 each contain one job of size $1/2$ and $m/2$ jobs of size $1/m$. All other machines contain a single job of size 1. The newly arriving job has size 1. The optimum makespan is $1 + 1/m$ and the makespan achievable by any local strategy is $3/2$ (by scheduling the new job on say machine 1 and migrating all small jobs to other machines).

We conclude this section with two more results. An improved competitive ratio of $4/3$ can be achieved by a more sophisticated algorithm with migration factor 4. Details can be found in Appendix C.

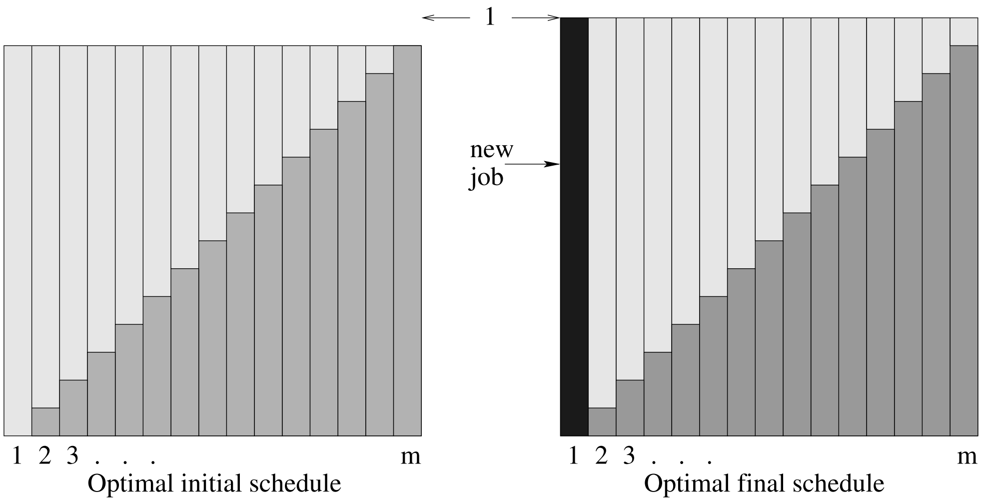

Fig. 2. An instance where all machine configurations have to change to maintain optimality.

Theorem 3. *There exists a* $4/3$*–competitive strategy with migration factor* 4.

Even better results are possible for two machines. Appendix E gives a specialized algorithm with competitive ratio $7/6$ and migration factor of one. We also prove that this ratio is tight for any deterministic strategy with migration factor one.

4 An Online Approximation Scheme with Constant Migration

The results presented in the last section raise the question how far the competitive ratio for online algorithms with 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 $1 + \epsilon$.

Lemma 2. *Any online algorithm computing optimal solutions needs migration factor* $\beta = \Omega(m)$.

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 2. The optimum makespan is $(m-1)/m$. 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 2. A short computation shows that the minimum total size of jobs that have to be migrated is $\lfloor m^2 \rfloor/(4m) = \Omega(m)$. □

In the following, $\epsilon > 0$ is a fixed constant. We assume without loss of generality that $\epsilon \leq 1$. The following observation belongs by now to the folklore in the field of scheduling; see, e.g., [ABC+99].

Observation 2. *Rounding up each job's processing time to the nearest integer power of* $1 + \epsilon$ *increases the makespan of an arbitrary schedule at most by a factor* $1 + \epsilon$. *In particular, in specifying a* $(1 + O(\epsilon))$*-competitive algorithm we can assume that all processing times are integer powers of* $1 + \epsilon$.

The current set of jobs is denoted by N. A job in N is called *large* if its processing time is at least $\epsilon \operatorname{lb}(N)$; otherwise, it is called *small*. The subset of large and small jobs is denoted by N_L and N_S, respectively. We partition N into classes N_i, $i \in \mathbb{Z}$, with

$$N_i := \{j \in N \mid p_j = (1+\epsilon)^i\} .$$

Let $I := \{i \in \mathbb{Z} \mid \epsilon \operatorname{lb}(N) \le (1+\epsilon)^i \le p_{\max}(N)\}$ such that $N_L = \bigcup_{i \in I} N_i$. Thus, the number of different job sizes for large jobs is bounded by $|I|$ and therefore constant:

$$|I| \;\le\; 1 + \log_{1+\epsilon} \frac{p_{\max}(N)}{\epsilon \operatorname{lb}(N)} \;\le\; 1 + \log_{1+\epsilon} \frac{1}{\epsilon} \;=\; O\left(\frac{\log(1/\epsilon)}{\epsilon}\right) \quad . \tag{2}$$

Given an assigment of jobs N_L to machines, we say that a particular machine obeys *configuration* $k : I \to \mathbb{N}_0$ if, for all $i \in I$, exactly $k(i)$ jobs from N_i are assigned to this machine. The set of configurations that can occur in any schedule for N_L is

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

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

$$\sum_{k \in K} y_k \;=\; m \qquad\qquad \text{and} \tag{3}$$

$$\sum_{k \in K} k(i)\, y_k \;=\; |N_i| \qquad\qquad \text{for all } i \in I. \tag{4}$$

We denote the set of vectors $y \in {\mathbb{N}_0}^K$ satisfying (3) and (4) by S. Thus, S represents the set of all schedules (up to permutations of machines and up to permutations of equal size jobs) for N_L. For a configuration $k \in K$ let

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

denote the load of a machine obeying configuration k. The makespan of a schedule $y \in S$ is equal to $\max\{\operatorname{load}(k) \mid y_k > 0\}$. For $\mu \ge 0$, let

$$K(\mu) \;:=\; \{k \in K \mid \operatorname{load}(k) \le \mu\} \;\; .$$

The set of all schedules with makespan at most μ is denoted by

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

In the following, we usually interpret a schedule $y \in S(\mu)$ as a vector in ${\mathbb{N}_0}^{K(\mu)}$ by ignoring all zero-entries corresponding to configurations not contained in $K(\mu)$.

The minimum makespan for N_L can be obtained by determining the minimum value μ with $S(\mu) \neq \emptyset$. Checking whether $S(\mu)$ is empty and, otherwise, finding a schedule $y \in S(\mu)$ can be done by finding a feasible solution to an integer linear program. We can write

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

where $A(\mu)$ is a matrix in ${\mathbb{N}_0}^{(1+|I|)\times|K(\mu)|}$ and b is a vector in ${\mathbb{N}_0}^{1+|I|}$. The first row of the linear system $A(\mu)\, y = b$ corresponds to constraint (3); the remaining $|I|$ rows correspond to constraints (4).

Lemma 3. *Let N be a set of jobs and let j be a new job of size $p_j \ge \epsilon \operatorname{lb}(N)$. Any schedule for N_L with makespan $\mu \le (1+\epsilon)\operatorname{opt}(N)$ can be turned into a schedule for $N_L \cup \{j\}$ by touching only a constant number of machines such that the makespan of the resulting schedule is at most* $\max\{\mu, \operatorname{opt}(N_L \cup \{j\})\}$.

Proof. We distinguish two cases. If $p_j \geq 2\mu$, then it is easy to observe that $\mathrm{opt}(N_L \cup \{j\}) = p_j$ and an optimal schedule for $N_L \cup \{j\}$ can be obtained by assigning job j to an arbitrary machine and moving all jobs that are currently on this machine to any other machine.

In the remainder of the proof we can assume that $p_j = (1+\epsilon)^{i'} < 2\mu$ and therefore $\mathrm{opt}(N_L \cup \{j\}) \leq 2\mu$. Let $y \in S(\mu)$ denote the given schedule for N_L. Then, y satisfies

$$A(\mu)\, y \;=\; b \;, \qquad y \in {\mathbb{N}_0}^{K(\mu)} \;. \tag{5}$$

Let $I' := I \cup \{i'\}$ and let K' denote the set of configurations $k : I' \to \mathbb{N}_0$ that can occur in any schedule for $N_L \cup \{j\}$. Then, $K'(\mu)$, $S'(\mu)$, $A'(\mu)$, and b' are defined analogously to $K(\mu)$, $S(\mu)$, $A(\mu)$, and b, respectively, with K replaced by K' and I replaced by I'.

Let $\mu' := \max\{\mu, \mathrm{opt}(N_L \cup \{j\})\} \leq 2\mu$. We are looking for a schedule $y' \in S'(\mu')$, that is, y' must satisfy

$$A'(\mu')\, y' \;=\; b' \;, \qquad y' \in {\mathbb{N}_0}^{K'(\mu')} \;. \tag{6}$$

Moreover, y' should be 'similar' to y. In order to compare the two vectors, we first 'lift' y to a vector in ${\mathbb{N}_0}^{K'(\mu')}$ as follows. A configuration $k \in K(\mu)$ can be interpreted as an element of $K'(\mu')$ by defining $k(i) := 0$ for all $i \in I' \setminus I$. We then define

$$y_k \;:=\; 0 \qquad\qquad \text{for all } k \in K'(\mu') \setminus K(\mu) \;.$$

It follows from (5) that the extended vector y satisfies

$$A'(\mu')\, y \;=\; \hat{b} \;, \qquad y \in {\mathbb{N}_0}^{K'(\mu')} \;. \tag{7}$$

The right hand side $\hat{b} \in {\mathbb{N}_0}^{1+|I'|}$ is defined as follows: If $I' = I$, then $\hat{b} = b$; otherwise, $I' = I \cup \{i'\}$ and we define the entry of vector $\hat{b}$ corresponding to i' to be zero and all other entries as in vector b.

Thus, y and y' are solutions to essentially the same integer linear program ((7) and (6), respectively) with slightly different right hand sides $\hat{b}$ and b', respectively. More precisely, the right hand sides are equal for all but one entry (the one corresponding to i') where they differ by 1. Using a sensitivity result from integer linear programming (see, e.g., [Sch86, Corollary 17.2a]), there exists a solution y' to (6) satisfying

$$\|y - y'\|_\infty \;\leq\; 3\,\bigl|K'(\mu')\bigr|\,\Delta \;, \tag{8}$$

where Δ is an upper bound on the absolute value of any subdeterminant of the matrix $A'(\mu')$. To complete the proof, we have to show that the right hand side of (8) is constant. This follows if we can prove that the dimension and all entries of the matrix $A'(\mu')$ are constant.

The number of rows of $A'(\mu')$ is at most $2 + |I| = O\bigl(\log(1/\epsilon)/\epsilon\bigr)$ by (2). Next, we give an upper bound on the number of columns $|K'(\mu')|$, i.e., on the number of machine configurations with load at most μ'. Since each job has size at least

$$\epsilon\, \mathrm{lb}(N) \;\overset{(1)}{\geq}\; \frac{\epsilon}{2}\, \mathrm{opt}(N) \;\geq\; \frac{\epsilon}{2(1+\epsilon)}\, \mu \;\geq\; \frac{\epsilon}{4(1+\epsilon)}\, \mu' \;,$$

there are at most $\gamma := \lfloor 4(1+\epsilon)/\epsilon \rfloor$ jobs in any configuration $k \in K'(\mu')$. In particular, $k(i) \leq \gamma$ for all $i \in I'$. This yields

$$\bigl|K'(\mu')\bigr| \;\leq\; \gamma^{|I'|} \;\leq\; \gamma^{|I|+1} \;\overset{(2)}{=}\; \left(\frac{1}{\epsilon}\right)^{O\left(\frac{\log(1/\epsilon)}{\epsilon}\right)} \;.$$

Finally, all entries in the first row of $A'(\mu')$ are 1 and the remaining entries are of the form $k(i) \leq \gamma$. This concludes the proof. $\square$

Theorem 4. *Let N be a set of jobs and let j be a new job not contained in N. Any $(1+\epsilon)$-approximate schedule for N can be turned into a $(1+\epsilon)$-approximate schedule for $N \cup \{j\}$ such that the total size of jobs that have to be moved is bounded by a constant $\beta(\epsilon)$ times p_j.*

Proof. We distinguish two cases. If the newly arrived job is small, i.e., $p_j < \epsilon \operatorname{lb}(N)$, then j can simply be assigned to the least loaded machine by Observation 1 and no job in N has to be moved.

It remains to consider the case $p_j \geq \epsilon \operatorname{lb}(N)$. The given schedule for N induces a schedule for N_L with makespan $\mu \leq (1+\epsilon)\operatorname{opt}(N) \leq (1+\epsilon)\operatorname{opt}(N \cup \{j\})$. By Lemma 3, the latter schedule can be turned into a schedule for $N_L \cup \{j\}$ with makespan at most

$$\max\{\mu, \operatorname{opt}(N_L \cup \{j\})\} \;\leq\; (1+\epsilon)\operatorname{opt}(N \cup \{j\}) \;\;,$$

by touching only a constant number of machines. In the following, this subset of machines of constant size is denoted by M'. We construct a schedule for $N \cup \{j\}$ as follows:

i) Start with the schedule for $N_L \cup \{j\}$ discussed above.
ii) The jobs in N_S that were assigned, by the given schedule for N, to one of the machines in $M \setminus M'$ are assigned to the same machine again.
iii) The remaining jobs in N_S are assigned one after another to the least loaded machine.

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 makespan of the schedule for $N_L \cup \{j\}$. It is thus bounded by $(1+\epsilon)\operatorname{opt}(N \cup \{j\})$. Assigning small jobs greedily to the least loaded machine in step iii) therefore results in a $(1+\epsilon)$-approximate schedule for $N \cup \{j\}$ by Observation 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 at most

$$(1+\epsilon)\operatorname{opt}(N)|M'| \;\overset{(1)}{\leq}\; 2(1+\epsilon)\operatorname{lb}(N)|M'| \;\leq\; 2(1+\epsilon)\frac{p_j}{\epsilon}|M'|$$

and therefore bounded by a constant times p_j. This concludes the proof. ◻

Theorem 5. *There exists a $(1+\epsilon)$-competitive online algorithm with constant migration factor $\beta(\epsilon)$ such that the running time for scheduling a newly arrived job is constant.*

In particular, it follows from the last property mentioned in the theorem that the algorithm has linear running time.

Proof. The result on the competitive ratio follows from Theorem 4. It remains to show that upon arrival of a new job j the schedule can be updated in constant time. We assume that for the current set of jobs N the following information is given:

– The total size of jobs $p(N)$, the maximum job size $p_{\max}(N)$, and the lower bound $\operatorname{lb}(N)$.
– A schedule $y \in S\big((1+\epsilon)\operatorname{opt}(N)\big)$ for the subset of large jobs N_L.
– The small jobs N_S are grouped into batches of size at most $\epsilon \operatorname{lb}(N)$. These batches are assigned to the machines such that the resulting schedule for N is still $(1+\epsilon)$-approximate. The number of batches on each machine is constant.
– For each machine, its load rounded up to the next integer power of $1+\epsilon$.

We argue that this information can be updated in constant time for the new set of jobs $N \cup \{j\}$. It is certainly easy to compute $p(N \cup \{j\}) := p(N) + p_j$, $p_{\max}(N \cup \{j\}) := \max\{p_{\max}(N), p_j\}$, and $\operatorname{lb}(N \cup \{j\}) := \max\{p(N \cup \{j\})/m, p_{\max}(N \cup \{j\})\}$.

If the new job j is small, we simply assign it to a machine whose approximate load is below average. This can be achieved using a priority queue exploiting integer keys [vEB77]. To be more precise, we assign j to an existing batch of small jobs on this machine. If none such batch exists or if all such batches are 'full', we create a new batch for job j.

If the new job j is large, a schedule $y' \in S'\big((1+\epsilon)\mathrm{opt}(N \cup \{j\})\big)$ for the jobs in $N_L \cup \{j\}$ can be obtained in constant time by completely enumerating all vectors y' with constant distance $\|y - y'\|_\infty$ (see proof of Lemma 3). The batches of small jobs on the machines in M' whose schedule has been changed are successively reassigned to a machine whose approximate load is below average.

Notice that some classes of jobs in $N_L \cup \{j\}$ that were previously large (i.e., of size at least $\epsilon\,\mathrm{lb}(N)$) might have become small with respect to the new lower bound $\mathrm{lb}(N \cup \{j\})$. These job classes are deleted from schedule y' and each such job is considered to form a batch of its own on its current machine.

In order to ensure that the number of batches on each machine remains constant, we do the following. Whenever the schedule on a machine is changed, we check all batches of small jobs on this machine and, if possible, merge groups of 'tiny' batches to form a new small batch. Notice that this can be done in constant time since only a constant number of machines is touched in every step.

Finally, whenever the schedule on a machine is changed, its actual load and its approximate load are updated. □

5 Discussions and Further Results

Although small changes of the input can radically change optimal solutions, this may not be necessary when we just want to maintain approximate solutions. This obvious intuition is quite trivial, yet we are not aware of previous theoretical studies of this kind of sensitivity analysis which we believe is a rewarding topic in approximation algorithms far beyond scheduling. Even in basic scheduling problems, there are many interesting open question. Below, we look at the objective to maximize the minimum machine load and an application. In future work, we intend to look at departures of jobs also.

Maximizing the Minimum Machine Load

An alternative, yet less frequently used objective for machine scheduling is to maximize the minimum load. However, we have 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 [BSS02,San04]. A simple scheme with many nice properties implements this idea if we manage to partition the SAN into several sub-servers [San04] of about equal size. Each sub-server stores the same amount of data. For example, if we have two sub-servers, each of them stores all the data to achieve a fault tolerance comparable to mirroring in ordinary RAID level 0 arrays [PGK88]. More subservers allow for a more flexible tradeoff between fault tolerance, redundancy, and access granularity. In any case, the capacity of the virtual disk is determined by the *minimum* capacity of a sub-server. 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). In Appendix D we show that the following simple strategy, which is very similar to FILL_1 , is 2–competitive already for migration factor $\beta = 1$.

Procedure FILL_5:
Upon arrival of a new job j, do the following. Repeatedly remove jobs from the least loaded machine $i_{\min}$; stop before the total size of removed jobs exceeds p_j. Assign job j to machine $i_{\min}$. Assign the removed jobs successively to the least loaded machine.

Theorem 6. FILL_5 *is* 2*–competitive with migration factor* 1.

Acknowledgments. We would like to thank Gerhard Woeginger for interesting discussions and helpful comments on the topic of this paper.

References

[ABC+99] F. Afrati, E. Bampis, C. Chekuri, D. Karger, C. Kenyon, S. Khanna, I. Milis, M. Queyranne, M. Skutella, C. Stein, and M. Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–43, New York City, NY, 1999.

[AE98] Yossi Azar and Leah Epstein. On-line machine covering. *Journal of Algorithms*, 1:67–77, 1998.

[Alb99] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.

[Alb03] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, 2003.

[BFKV95] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *Journal of Computer and System Sciences*, 51:359–366, 1995.

[BSS02] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, adaptive placement schemes for non-uniform requirements. In *14th ACM Symposium on Parallel Algorithms and Architectures*, pages 53–62, 2002.

[CGJ78] E. G. Coffman Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7:1–17, 1978.

[CvVW94] Bo Chen, Andre van Vliet, and Gerhard J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51:219–222, 1994.

[Fri84] D. K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13:170–181, 1984.

[FW00] R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.

[GJ78] M. R. Garey and D. S. Johnson. Strong np-completeness results: Motivation, examples and implications. *Journal of the ACM*, 25:499–508, 1978.

[GLLR79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17:263–269, 1969.

[Hoc96] D. S. Hochbaum. Various notions of approximation: Good, better, best, and more. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 9, pages 346–398. Thomson, 1996.

[HS87] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[KPT96] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[Lan81] M. A. Langston. *Processor scheduling with improved heuristic algorithms*. PhD thesis, Texas A&M University, 1981.

[Len83] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

[PGK88] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). *Proceedings of ACM SIGMOD'88*, pages 109–116, 1988.

[Rud01] J. F. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, The University of Texas at Dallas, 2001.

[Sah76] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23:116–127, 1976.

[San04] P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932, pages 82–101, 2004.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

[Sga97] Jiří Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 14 July 1997.

[Sga98] J. Sgall. On-line scheduling — a survey. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer, Berlin, 1998.

[vEB77] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. *Information Processing Letters*, 6(3):80–82, 1977.

A Additional Notation

For a schedule on the set of jobs N, let $S(i)$ denote the set of *jobs scheduled on machine* i. For a subset of machines $Y \subseteq M$, let $S(Y) := \bigcup_{i \in Y} S(i)$.

B $3/2$–Competitive Strategy with Factor 4/3 Migration — Proof of Theorem 2

Proof. The migration factor is clear from the description of FILL_2 . To show the competitive ratio, we consider any arbitrary schedule on the set of jobs N that is 3/2–approximate with the following additional property. We also refer to this schedule as the *input schedule*. The total load in any machine excluding the largest job in it, is at most opt. We show that incorporating the new job j results in a 3/2–approximate schedule. The resulting schedule also satisfies the above property as shown by the following fact.

Fact 1. *Consider any schedule on a set of jobs N such that the total load on any machine excluding the largest job in it is at most* opt. *Incorporating the new job j using* FILL_2 *still preserves this property with respect to the new optimum* opt*'.*

Proof. Let M' be the subset of machines 'touched' by FILL_2 for scheduling job j. For any machine in M', consider the 'last' job assigned to it during the execution of FILL_2 . If it is not j, then that job was assigned to it as part of the re-distribution phase. Since re-distribution is always performed on the current least loaded machine, the load in this machine excluding this last job is at most opt'. If the last job is j then, as option 0 is always preferred, the total load of this machine can only be smaller than the load in the least loaded machine together with the load of job j. Hence in both cases, the fact is ensured. □

We call a job *small* if its size is at most $\mathrm{opt}'/2$ and call it *medium* if its size is between $\mathrm{opt}/2$ and $(3/4)\mathrm{opt}$ and call it *large* otherwise. If j is a small job, then option 0 already yields a schedule of makespan at most (3/2)opt' by Observation 1. Hence from now on, we only distinguish two cases; the job is either medium or large.

The medium job case: $\mathrm{opt}'/2 < p_j \leq (3/4)\mathrm{opt}'$: Let

$$p_j = \mathrm{opt}'/2 + \delta \quad \text{where} \quad 0 \; < \; \delta \; \leq \; \mathrm{opt}'/4 \tag{9}$$

We will show that there is a good option $k \in [1, m]$ that yields a 3/2–approximate schedule after inserting job j. The rest of the arguments are to first define this k and then to show that option k would suffice.

We use the notation $S(i)$ to denote the set of jobs scheduled on machine i in the input schedule (ref. Appendix A). With respect to the input schedule, partition the set of machines M as machines with *small* jobs, denoted as M_s and the rest of the machines as M_l, where,

$$M_s = \{i \mid p_{\max}(S(i)) \; \leq \; \mathrm{opt}' - \delta\} \quad \text{and} \quad M_l = M - M_s \tag{10}$$

Observe that on any (3/2)–approximate schedule of jobs in N, a machine can only have at most one job of size more than $\mathrm{opt}' - \delta$. Otherwise the total load in that machine exceeds $(3/2)\mathrm{opt}'$ as $\delta \leq \mathrm{opt}'/4$. This is obviously true even for any optimal schedule of jobs in N. Hence, we fix an optimal schedule of N in such a way that, exactly the machines in M_l have jobs with size more than $\mathrm{opt}' - \delta$. That is, in the optimal schedule as well as input schedule, on each machine in M_l, exactly one job has size more than $\mathrm{opt}' - \delta$. And the rest of the machines, i.e, M_s, do not contain any such job (on both schedules).

We now compare the input schedule with this optimal schedule. Except for the large jobs in M_l, other jobs might be scheduled differently in the optimal schedule when compared to the input schedule. The same holds for the jobs in M_s also. As shown in figure 3, we identify these differences as follows:

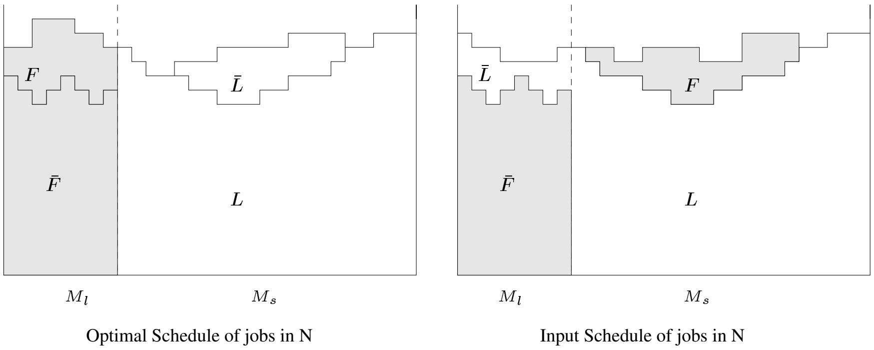

Fig. 3. Figure showing the differences between the optimal schedule and the input schedule of jobs in N. Each machine in M_l has exactly one job of size greater than $\mathrm{opt}' - \delta$. Moreover, in both schedules, such jobs are present only in $\bar{F}$. Hence all jobs in F have size at most δ.

– Set L: Set of jobs scheduled on M_s in both optimal and input schedule.
– Set $\bar{F}$: Set of jobs scheduled on M_l in both optimal and input schedule.
– Set F: Set of jobs scheduled on M_s in input schedule but not in optimal schedule.
– Set $\bar{L}$: Set of jobs scheduled on M_l in input schedule but not in optimal schedule.

It is easy to see that these subsets define a partition of jobs in both input schedule and optimal schedule. As we discussed earlier, in both the schedules, each machine in M_l contain one job of size greater than $\mathrm{opt}' - \delta$. Hence, such jobs are only present in the common subset $\bar{F}$. This in particular means that the remaining jobs that are scheduled in M_l in the optimal schedule, i.e, F, is such that,

$$\text{for every job } b \in F, \quad p_b \leq \delta \tag{11}$$

Otherwise, the load in a machine in M_l will exceed opt' in the optimal schedule, which is not possible. Consider the jobs scheduled in M_s in the input schedule,i.e, $F \cup L$. Since the set L is scheduled on M_s in both optimal as well as input schedule, any arbitrary schedule of L onto M_s ensures that the load of the least loaded machine is at most opt. This in particular means that for the input schedule, there exists a machine in M_s such that, if the jobs belonging to F are removed from it, the total remaining load on it (i.e, $p(S(k)-F)$) is at most opt. Let that machine be k. That is,

$$p(S(k) - F) \leq \mathrm{opt} \tag{12}$$

For later purpose, we need to view the jobs in k in the input schedule (i.e, $S(k)$) as being partitioned into two sub-sets, K_δ and K_r. The subset K_δ contains all jobs in $S(k)$ of size at most δ. The set K_r contain the remaining jobs. That is,

$$K_\delta = \{\text{All jobs in machine } k \text{ with size at most } \delta\}; \quad K_r = S(k) - K_\delta$$

Since F is a subset of K_δ (by definition), (12) implies that

$$p(K_r) \leq \mathrm{opt} \tag{13}$$

As promised earlier, we are now in a position to fix the 'good' option. In the following, we show that "option k" yields a 3/2–approximate schedule. While performing option K, let the set of jobs that are removed from machine k before assigning job j, be denoted by R. To show that this option is good, it suffices to show that

i) The resulting load on machine k after removing jobs in R from it and assigning job j there, is at most $(3/2)\mathrm{opt}'$. That is, $p((S(k) - R) \cup \{j\}) \leq (3/2)\mathrm{opt}'$.
ii) Each removed job has size at most $\mathrm{opt}'/2$ (Redistributing them is fine by Observation 1).

It is easy to see that the second condition is met. Recall that the largest job from machine m is never removed in option k. This job belongs to set K_r (by definition). Since the total size of K_r is at most opt, all jobs in K_r except (possibly) this largest job, have size at most $\mathrm{opt}'/2$. Moreover, since all jobs in k_δ have size at most $\mathrm{opt}'/4$ as $\delta \leq \mathrm{opt}'/4$, every removed job has size at most $\mathrm{opt}'/2$.

It remains to show that condition i) also hold. First consider the simpler sub-case that, before assigning job j, all jobs in machine k except the largest job (this job is never touched) were removed. Since the largest job's size cannot exceed $\mathrm{opt}' - \delta$ (by definition, as machine k belongs to M_s. See def. (10)), the resultant load is at most $(3/2)\mathrm{opt}'$, as the size of job j is $\mathrm{opt}'/2 + \delta$.

Hence from now on we assume that, there were unremoved jobs from machine k in addition to the largest job. If such an unremoved job is from K_δ, then also condition i) holds. This is because of the following. Since there is one job in K_δ that is unremoved, the total removed amount is at least $(4/3)p_j - \delta$. Otherwise, this job should have been removed with out exceeding the removal limit as this job has size at most δ, by definition of K_δ. Hence the load after assigning job j is no more than $Original\ load - ((4/3)p_j - \delta) + p_j$. It is easy to verify that this is at most $Original\ load$ (at most $(3/2)\mathrm{opt}$) as $p_j = \mathrm{opt}'/2 + \delta$ and $\delta \leq \mathrm{opt}'/4$.

Now the only remaining case is that there are unremoved jobs, in addition to the largest job, and, all of them are from K_r. Observe that jobs from K_r are considered first by the greedy removal, as all of them have larger size than jobs in K_δ. This already implies that at least one job from K_r was removed. This is because; (a) the removal limit, i.e, $(4/3)p_j$ is at least $\mathrm{opt}'/2$, and, (b) all jobs in K_r, except the largest job, have size at most $\mathrm{opt}'/2$ (as $p(K_r) \leq \mathrm{opt}$). Thus, the total removed size is no less than $p(K_\delta) + \delta$. The δ is for the removed job from K_r. The original load is $P(K_\delta) + p(K_r)$. Hence the load after assigning job j in machine k is at most

$$p(K_\delta) + p(K_r) - p(K_\delta) - \delta + p_j \ = \ p(K_r) + \mathrm{opt}'/2 \ \leq \ (3/2)\mathrm{opt}' \quad \text{by (13).}$$

The large job case: $p_j > (3/4)\mathrm{opt}'$, let

$$p_j \ = \ (3/4)\mathrm{opt}' + \delta \quad \text{where} \quad \delta \ \leq \ (1/4)\mathrm{opt}' \tag{14}$$

Since there can be at most m large jobs in $N \cup \{j\}$, there is a machine k in the input schedule, where, in fact all jobs in it have size at most $\mathrm{opt}'/2$. In the following, we show that "option k" yields a $3/2$–approximate schedule. First observe that all the removed jobs from machine k have size at most $\mathrm{opt}'/2$. Hence re-distributing them is fine by Observation 1. Recall that by the input requirement, in the input schedule, on any machine, the load excluding the largest job in it is at most opt. Hence all jobs except the largest job (this is untouched) will be removed by option k as the removal limit, i.e, $(4/3)p_j$, is at least opt'. Hence after removing the jobs greedily, only the largest job remains and, assigning job j there do not increase the load above $(3/2)\mathrm{opt}'$.

□

C A 4/3–Competitive Strategy with Factor 4 Migration – Proof of Theorem 3

Consider the following procedure FILL_3. In this section, we show that FILL_3 is 4/3–competitive with factor 4 migration. We use the notation $S(.)$ as defined in Appendix A.

Procedure FILL_3:

Upon arrival of j, choose the one of the following $m + 1$ options that minimizes the resulting makespan.

Option 0: Assign job j to the least loaded machine.

Option i [for $i \in \{1, \dots, m\}$]: There are two main phases.

If $2p_j \geq p(S(i))$ and $p_j \geq p_{\max}(S(i))$ then perform phase one, else directly proceed to phase two.

Phase one:

Let ℓ denote the largest job in machine i. Remove all jobs from machine i and schedule job j there. Except job ℓ, assign the removed jobs successively in the least loaded machine.

Phase two:

We assign the unassigned job ℓ in this phase. If phase one was skipped then ℓ is simply job j. After phase one, let $\tilde{S}(x)$ denote the new set of jobs assigned to machine x. If phase one is skipped then $\tilde{S}(\cdot) = S(\cdot)$. To assign job ℓ, we further consider $m + 1$ sub-options that looks as follows. Finally choose the sub-option that yields the minimum final makespan.

Sub-option 0: Assign job ℓ to the least loaded machine.

Sub-option k [for $k \in \{1, \dots, m\}$]: The set of jobs in machine k is $\tilde{S}(k)$. Ignoring the largest job in machine k, consider the remaining jobs in order of non-increasing size and repeatedly remove them; stop before the total size of removed jobs exceeds $2p_j$. Assign job ℓ to machine k and assign the removed jobs successively to the least loaded machine.

Lemma 4. FILL_3 *is* $4/3$*–competitive with factor* 4 *migration.*

Proof. The proof of migration factor is clear from the description of FILL_3. In both phases of any option $i \in [1, m]$, migration factor is 2. Hence together, the migration factor is 4. To show the competitive ratio, we consider any arbitrary schedule on the set of jobs N that is 4/3–approximate. We also refer to this schedule as the *input schedule*. We show that FILL_3 yields a 4/3–approximate schedule.

If job j is small, i.e, $p_j \leq \text{opt}'/3$, option 0 yields a 4/3–approximate schedule by Observation 1.

Consider the remaining case $p_j > \text{opt}'/3$. With respect to the input schedule, we partition the set of machines M as M_L and M_S, where, M_L refers to machines containing 'large' jobs and machines in M_S contain only 'small' jobs. More precisely,

$$M_L = \left\{ i \mid \text{there exists a job } b \in S(i) \text{ with } p_b > (2/3)\text{opt}' \right\} \text{ and } M_S = M - M_L$$

In other words, M_L is the set of all machines such that, in each such machine, there is at least one 'large' job of size greater than $(3/2)\text{opt}'$. Clearly not all machines can have such large jobs as $p_j > \text{opt}'/3$. Hence $M_S \neq \emptyset$.

Observation 3. *If* $p_j > \text{opt}'/3$ *then there exists a machine* $z \in M_S$ *such that, there is at most one job in it, i.e, in* $S(z)$*, with size more than* $\text{opt}'/3$.

Otherwise there is no optimal schedule on $N \cup \{j\}$ with makespan no more than opt', as each machine in M_L already contains at least one job with size greater than $(2/3)\text{opt}'$.

Choose the z that conforms to Observation 3. In the following, we show that "option z" yields a 4/3–approximate schedule. We show this by showing the following two conditions.

Condition 1: After phase one of option z, the resulting intermediate makespan is at most $(4/3)\text{opt}'$.

Condition 2: If condition 1 holds then after phase two of option z, the final makespan is at most $(4/3)\text{opt}'$.

Condition 1 is easily seen as follows. Recall that the largest job in machine z is denoted as ℓ (see FILL_3 description). After removing all jobs from machine z and assigning job j there, the total load there is at most $p_j \leq \text{opt}'$. By Observation 3, we know that, all jobs in machine z except possibly the largest job, i.e, ℓ, have size at most $\text{opt}'/3$. Hence by Observation 1, redistributing the removed jobs except ℓ still maintains Condition 1.

It remains to show condition 2 having shown condition 1. Recall that in the beginning of phase two, the job to be scheduled is ℓ and the subset of jobs assigned to any machine x is $\tilde{S}(x)$. Also recall that if phase one is skipped then $\ell = t$ and $\tilde{S}(\cdot) = S(\cdot)$. Since phase two of Option z is entered either

– By skipping phase one. Hence $p_\ell = p_j < p(S(z))/2 \leq (2/3)\mathrm{opt}'$ or $p_\ell = p_j < p_{\max}(S(z)) \leq (2/3)\mathrm{opt}'$ (as $z \in M_S$).
– Or after phase one. Hence $p_\ell = p_{\max}(S(z)) \leq (2/3)\mathrm{opt}'$ (as $z \in M_S$).

In either case, it implies that

$$p_\ell \leq (2/3)\mathrm{opt}' \quad \text{and} \quad p_\ell \leq p_j \tag{15}$$

We complete the proof for condition 2 by splitting the analysis into two cases, namely, $p_\ell \leq \mathrm{opt}'/3$ and $\mathrm{opt}'/3 < p_\ell \leq (2/3)\mathrm{opt}'$.

Consider the **case** $p_\ell \leq \mathrm{opt}'/3$. Since the intermediate schedule after Phase 1 is (4/3)–approximate, in Phase two, the Sub-option 0 yields (4/3)–approximate schedule by Observation 1.

Consider the remaining **case** $\mathrm{opt}'/3 < p_\ell \leq (2/3)\mathrm{opt}'$. Recall that in the input schedule, no job in machine z has size greater than $(2/3)\mathrm{opt}'$ as $z \in M_S$. Hence, during phase one (if at all it is performed) of Option z, no job of size greater than $(2/3)\mathrm{opt}$ is 'touched'; and hence their positions (machine where it is assigned) are left unchanged. Hence if we re-look at the definition of M_S and M_L with respect to $\tilde{S}(\cdot)$ instead of $S(\cdot)$, the machines that fall into these subsets are exactly the same. It is straightforward to verify that observation 3 still holds with respect to ℓ, M_S and $\tilde{S}(\cdot)$. Let y be a machine satisfying this modified observation and let the largest job in y be b. By Observation 3,

$$p_b \leq (2/3)\mathrm{opt}' \quad \text{and} \quad \text{for all } j \in \tilde{S}(y) \setminus b,\ p_j \leq \mathrm{opt}'/3 \tag{16}$$

The first inequality is simply because y belongs to M_S. We show that Sub-option y maintains the invariant after Phase two. Since the largest job, in machine y, i.e b, is never removed, all the removed jobs have size at most $\mathrm{opt}'/3$. Hence by Observation 1, the schedule after re-assigning the removed job is 4/3–approximate if the schedule before that step is 4/3–approximate. Hence all that remains is to show that the schedule obtained after removing jobs from machine y and assigning job ℓ there, the schedule is $4/3$–approximate. If all jobs are removed (except b), then clearly this is true as the final load is $p_j + p_b \leq (4/3)\mathrm{opt}'$ by (15) and (16). Now assume that there is at least one unremoved job other than b. Since such a job has size no more than $\mathrm{opt}'/3$ by (16), the total size of removed jobs is at least $2p_j - \mathrm{opt}'/3$, where $2p_j$ is the removal limit. Hence the load after assigning job ℓ is at most $Original\ load - (2p_j - \mathrm{opt}'/3) + p_\ell$ which is no more than the original load by (15) and the current case assumption. □

D Maximizing the Minimum Load – Proof of Theorem 6

Proof. The migration factor is clear from the description of FILL_5. In the input schedule, consider the machine with the maximum load such that it *contains more than one job*. If there is no such machine (i.e, no machine contain more than one job), then clearly FILL_5 schedule of $N \cup \{j\}$ is 1–approximate. Hence the interesting case is that such a machine exists. We call machines with more than one job as *multi-job* machine.

We assume that the input schedule has the following property;

$$\text{Maximum load of a multi-job machine} \leq 2 \cdot \text{minimum load}$$

We show later that the above property is preserved on the output schedule. If the above property is preserved, then it is a 2–approximate schedule, as the optimal minimum load is at most the maximum load of a multi-job machine. It remains to show that the above property holds for the output schedule. For the case $p_j \leq$

$p(S(i_{\min}))$, it is straightforward to see that the above property holds. Now for the remaining case, i.e, $p_j > p(S(i_{\min}))$, all jobs from the least loaded machine will be removed first and then j will be assigned to $i_{\min}$. Consider this intermediate schedule (before assigning the removed jobs). Clearly this schedule satisfies the property. Observe that each of the removed job has size at most the current minimum load. Hence re-assigning them still preserves the invariant as shown by the first case analysis. □

E The Two Machine Case

In this section we show a tight competitive ratio for the two machine case. Consider the following procedure FILL_4.

Procedure FILL_4:
Upon arrival of j, choose the option from the following options that minimizes the new makespan.

For each fixed machine $i \in \{1, 2\}$, we define more options in the following way. Let L be the largest seven jobs in machine i. Set L could possibly have less than seven jobs. Let the remaining jobs in machine i be B. That is $B = S(i) - L$. Let $\mathcal{L} = \{L_1, \ldots, L_s\}$ be set of all possible subsets (including ϕ) of L such that for each $L_k \in \mathcal{L}$, $p(L_k) \le p_j$. Each set $L_k \in \mathcal{L}$ gives rise to a new

Option $(i.k)$: Remove the set of jobs L_k from machine i and assign them to the other machine. Consider the jobs in B in non-increasing order order of size and repeatedly remove them; stop before the total size of removed jobs exceeds $p_j - p(L_k)$. Let B_k denote these removed jobs. Assign job j to machine i and assign the removed jobs successively to the least loaded machine.
Option 0: Assign job j to the least loaded machine.

Theorem 7. FILL_4 *is* $7/6$*–competitive with factor 1 migration.*

Proof. The migration factor is clear from the FILL_4 description. To show the competitive ratio, we consider any arbitrary schedule on the set of jobs N that is 7/6–approximate. We also refer to this schedule as the *input schedule*. We show that FILL_4 yields a 7/6–approximate schedule. First we need the following two observations.

In the input schedule, since the load in any machine i is at most $(7/6)\mathrm{opt}$, and since the largest seven jobs are not included in set B defined for machine i (see FILL_4), we have,

Observation 4. *For any machine i, consider the set B as defined in* FILL_4*. Every job in this set has size at most* $\mathrm{opt}'/6$.

If job j is 'small', i.e, $p_j \le \mathrm{opt}'/3$, then Option 0 yields 7/6–competitive schedule by Observation 1.
It remains to handle the *case* $p_j > \mathrm{opt}'/3$.

Observation 5. *From any machine, it is sufficient to migrate jobs of total size at most* $p_j - \mathrm{opt}'/6$.

Proof. W.l.o.g let $p(S(1)) \le p(S(2))$. Let $p(S(2)) = \mathrm{opt}' - p_j/2 + y$. This implies that $p(S(1)) \le \mathrm{opt}' - p_j/2 - y$. Otherwise the average makespan exceeds opt'. If $y \ge p_j/2 - \mathrm{opt}'/6$, then scheduling job j in machine 1 without migration would suffice. For the remaining case where $p_j > \mathrm{opt}'/3$ and $y < p_j/2 - \mathrm{opt}'/6$, it is sufficient to have a migration of total size either $p_j/2 - y - \mathrm{opt}'/6$ from machine 1 to 2 or $p_j/2 + y - \mathrm{opt}'/6$ from machine 2 to 1. In both cases the migration amount is at most $p_j - \mathrm{opt}'/6$. □

Though above observation is true, it is possible that every feasible set of jobs that needs to be migrated have total size more than $p_j - \mathrm{opt}'/6$.

Fact 2. *There is a subset of jobs of total size at most p_j residing in a machine such that scheduling job j here and migrating this subset to the other machine maintains the invariant.*

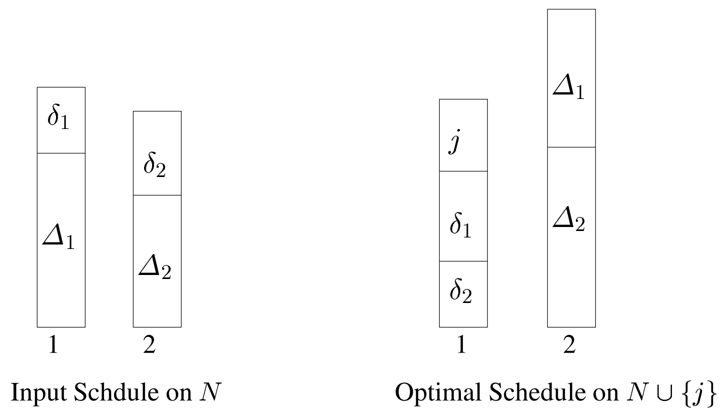

Fig. 4. Comparison of input schedule on N and optimal schedule on $N \cup \{j\}$ for two machines.

Though fact 2 alone is sufficient to devise a strategy that yields a $7/6$–approximate schedule, it might take exponential time in the worst case to identify these subsets that needs to be migrated. Using the above two, we now show that our polynomial time strategy does the job.

Under the assumption that Fact 2 holds, the proof is completed as follows. W.l.o.g assume that, by fact 2, migrating a set of jobs from machine 1 to 2 and scheduling job j in machine 1 is enough. Let X denote the *set with the smallest total size* that needs to be migrated from machine 1. Let the jobs in X that are among the largest seven jobs in machine 1 be the set $L_k \in \mathcal{L}$. We show that "Option $1.k$" maintains the invariant. The total size of jobs removed from machine 1 by option $1.k$ initially, i.e, $p(L_k \cup B_k)$ is at least $p_j - \mathrm{opt}'/6$ unless $B_k = B$ as size of any job in B is at most $\mathrm{opt}'/6$ by Observation 4 (standard argument). Since it is enough to migrate a total size of at most $p_j - \mathrm{opt}'/6$ by Observation 5 and since X is a feasible set to be migrated with the minimum total size, in any case the total size of jobs removed initially is at least $p(X)$. Hence after assigning job j, the total load in machine 1 is at most $(7/6)\mathrm{opt}'$. The re-assignment of jobs (they belong to B_k and hence have size at most $\mathrm{opt}'/6$) is also fine by Observation 1.

□

Proof. (Fact 2) We now show the existence of a set of jobs to be migrated such that, they all lie in one machine before migration. Hence for the remainder of the proof, we do not bother about its total size, as there is no need to migrate such a set with total size more than p_j, since we only have two machines (simply assign p_j on the destination machine instead). Consider any fixed optimal schedule of $N \cup \{j\}$. We capture the difference between this optimal schedule and the input schedule on N by sets $\delta_1, \delta_2, \Delta_1$ and Δ_2. As shown in figure 4 (the input schedule), the above four sets define a partition of N. The set δ_1 is the set of all jobs assigned to machine 1 on both schedule. Similarly, set Δ_2 is the set of all jobs assigned to machine 2 on both schedules. The remaining two sets capture the differences between these two schedules except for job j, which is only present in the optimal schedule.

We only consider the interesting case that assigning j to any machine with out migration fails. For convenience, from now on, for any set X, we denote its size also as X. For job j we denote its size as j. We also normalize opt' to 1. Because of the optimal schedule, (figure 4) and the fact that assigning j without migration fails, we have,

$$\begin{aligned}
\delta_1 + \delta_2 + j &\leq 1 \\
\Delta_1 + \Delta_2 &\leq 1 \\
\delta_1 + \Delta_1 + j &> 7/6 \\
\delta_2 + \Delta_2 + j &> 7/6
\end{aligned}$$

It is straightforward to verify that these four inequalities yield, $j > 2/3$ and $\delta_1 + \delta_2 \leq 1/3$ (Inequalities 1 and 3 imply $\Delta_1 > 1/6 + \delta_2$. Inequalities 2 and 4 imply $\delta_2 + j > 1/6 + \Delta_1$). W.l.o.g let $\delta_1 \leq 1/6$. Hence the migration strategy is; migrate Δ_2 to machine 1 and schedule j on machine 2. □

Tight Lower Bound For Two Machines

Theorem 8. *Let A be any deterministic algorithm which is c–competitive on two machines with migration factor at most one. Then $c \geq \frac{7}{6(1+\epsilon)}$ for any sufficiently small positive $\epsilon \in \mathbb{R}^+$.*

Proof. The adversary initially issues four jobs with the following size: $1/6 + \epsilon/2, 1/6 + \epsilon/2, 1/2$ and $1/2$. It is easy to verify that the only 7/6–approximate way of scheduling them is to assign in each machine one job of size $1/6 + \epsilon/2$ and one job of size $1/2$.

Assume otherwise. That is both machines contain one job of size $1/2$ and one job of size $1/6 + \epsilon/2$. Now adversary issues a new job of size $2/3$. The optimal makespan is $1 + \epsilon$. But if the migration factor is restricted to 1, then the best possible makespan by A is $7/6$. Hence $c = 7/(6(1 + \epsilon))$. □