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.) This problem is well known to be strongly NP-hard [8].

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

* This work was partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT) and by the EU Thematic Network APPOL II, Approximation and Online Algorithms, IST-2001-30012.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 1111–1122, 2004.

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.

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 [9]: 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 the order of non-increasing size before feeding them to the list scheduling algorithm yields an approximation algorithm with performance ratio $4/3 - 1/(3m)$. After a series of improvements, finally, a polynomial time approximation schemes for an arbitrary number of machines were given by Hochbaum and Shmoys [10].

In a series of papers, increasingly complicated online algorithms with better and better competitive ratios beating the Graham bound 2 have been developed [5,11,1]. The best result known to date is a 1.9201-competitive algorithm due to Fleischer and Wahl [7]. The best lower bound 1.88 on the competitive ratio of any deterministic online algorithm currently known is due to Rudin [12]. For randomized online algorithms there is a lower bound of $e/(e-1) \approx 1.58$ [6,

16]. For more results on online algorithms for scheduling we refer to the recent survey articles by Albers [2] and Sgall [17].

Strategies that reassign jobs were studied in the context of online load balancing, 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 [18] 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 u has size p_u and reassignment cost r_u. 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., r_u is proportional to p_u, and $2S$ for unit reassignments, i.e., $r_u = 1$ for all jobs. Later Andrews et al. [3] improved it to 3.5981 with the same reassignment factors. They also showed $3 + \epsilon$ and $2 + \epsilon$ competitive strategies respectively for the proportional and unit case, the reassignment factor depending 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. Job deletions is an aspect that we do not consider in our work, our focus is primarily on achieving competitive ratios close to 1. Our results can also be interpreted in this framework of online load balancing, with proportional reassignments and without job deletions. We show strategies with better competitive ratios, at the same time achieving reassignment factor strictly less than three. We also show $(1 + \epsilon)$–competitive strategies, for any $\epsilon > 0$, with constant reassignment factor $f(\epsilon)$. Our results are also stronger in the sense that a strategy with reassignment factor β ensures that when a job u arrives, the total reassignment cost incurred (for scheduling it) is at most βr_u. This is different from the more relaxed constraint that after t rounds, the total reassignment cost incurred is at most $\beta \sum r_u$ (summing over all jobs seen till round t). Most of our strategies are *robust*, they convert *any* α–competitive schedule to an α–competitive schedule after assigning the newly arrived job, whereas in [18,3] 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.

2 Our Contribution

In Section 4 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 com-

petitive 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.

Our main result can be found in Section 5. 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 and Shmoys [10].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 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 Section 6 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 [4] 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 $\beta = 1$.

Due to space limitation, proofs of some of the theorems are omitted. We refer the reader to [14] for a full version of the paper.

3 Preliminaries

Let the set of *machines* be denoted by $M = \{1, \ldots, m\}$. The set of *jobs* is $\{1, \ldots, n\}$ 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 κ. Assigning a new job j to the least loaded machine yields a schedule with makespan at most* $\max\{\kappa, \mathrm{opt}(N \cup \{j\}) + (1 - 1/m)p_j\}$.

4 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 $\text{opt}'/2$, otherwise it is called *large*. If the new job j is small, then the first option yields makespan at most $\left(\frac{3}{2} - \frac{1}{2m}\right)\text{opt}'$ 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 $\left(\frac{3}{2} - \frac{1}{2m}\right)\text{opt}'$. 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. (Break 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 the order of non-increasing size and 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.

Theorem 2. *Procedure* FILL_2 *is* $\frac{3}{2}$*-competitive with migration factor* $\frac{4}{3}$.

Robustness: Most of our scheduling strategies for minimizing the makespan discussed in this chapter 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 further remark that this is not an unreasonable requirement as even the simple list scheduling algorithm ensures this.

Negative Results

Theorems 1 and 2 raise the question of 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.*

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 $3/(2+\frac{2}{m})$ 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.

We conclude this section by stating two more results that improve the 3/2–competitive ratio.

Theorem 3. *There exists a* $\frac{4}{3}$*–competitive strategy with factor* 4 *migration.*

Theorem 4. *For the case of two machines, there exists a* $\frac{7}{6}$*–competitive strategy with factor 1 migration. Moreover, let A be any deterministic algorithm that is c–competitive with factor 1 migration, for two machines. Then* $c \geq \frac{7}{6(1+\epsilon)}$ *for any sufficiently small positive* $\epsilon \in \mathbb{R}^+$.

5 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* $\Omega(m)$.

Proof (Sketch). 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,

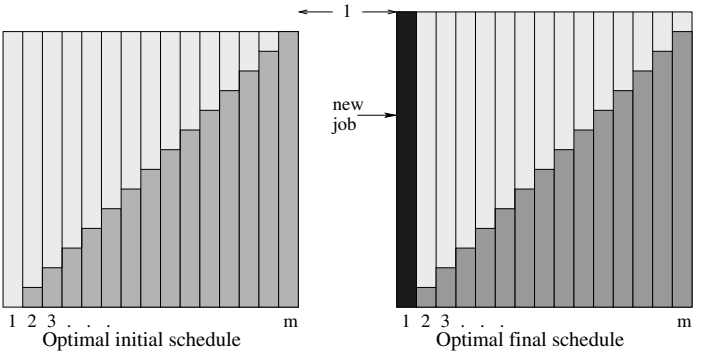

Fig. 1. An instance where all machine configurations have to change to maintain optimality.

any optimum schedule has the structure depicted in the left part of Figure 1. The optimum makespan is $(m-1)/m$. When a new job of size 1 arrives, the optimum makespan increases to 1; see the right hand side of Figure 1. A short calculation shows that the minimum total size of the jobs to be migrated is $\Omega(m)$. □

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.

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 $I := \{i \in \mathbb{Z} \mid \epsilon \operatorname{lb}(N) \leq (1+\epsilon)^i \leq p_{\max}(N)\}$ 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:

$$|I| \;\leq\; 1 + \log_{1+\epsilon} \frac{p_{\max}(N)}{\epsilon \operatorname{lb}(N)} \;\leq\; 1 + \log_{1+\epsilon} \frac{1}{\epsilon} \;\leq\; \frac{2}{\epsilon} \log\left(\frac{1+\epsilon}{\epsilon}\right) \;. \qquad (2)$$

Given an assignment 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

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

Up to permutations of machines, an arbitrary schedule for N_L can be described by specifying, for each $k \in \mathbf{K}$, 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

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

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

We denote the set of vectors $y \in \mathbb{N}_0^{\mathbf{K}}$ satisfying (3) and (4) by $\mathbf{S}$. Thus, $\mathbf{S}$ 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 $k \in \mathbf{K}$ let

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

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

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

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

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

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

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

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

where $A(\mu)$ is a matrix in $\mathbb{N}_0^{(1+|I|)\times|\mathbf{K}(\mu)|}$ and b 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 $p_j \geq \epsilon\, \text{lb}(N)$. Any schedule for N_L with makespan $\mu \leq (1+\epsilon)\text{opt}(N)$ 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* $\max\{\mu, \text{opt}(N_L \cup \{j\})\}$.

Proof. We distinguish two cases. If $p_j \geq 2\mu$, then it is easy to observe that $\text{opt}(N_L \cup \{j\}) = p_j$ 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 $\mathrm{opt}(N_L \cup \{j\}) \le 2\mu$. Let $y \in \mathbf{S}(\mu)$ denote the given schedule for N_L. Then, y satisfies

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

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

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

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

Moreover, y' should be 'similar' to y. In order to compare the two vectors, we first 'lift' y to a vector in $\mathbb{N}_0{}^{\mathbf{K}'(\mu')}$ as follows. A configuration $k \in \mathbf{K}(\mu)$ can be interpreted as an element of $\mathbf{K}'(\mu')$ by defining $k(i) := 0$ for all $i \in I' \setminus I$. We then define $y_k \;:=\; 0$ for all $k \in \mathbf{K}'(\mu') \setminus \mathbf{K}(\mu)$. It follows from (5) that the extended vector y satisfies

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

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., [15, Corollary 17.2a]), there exists a solution y' to (6) satisfying

$$\|y - y'\|_\infty \;\le\; 3\left|\mathbf{K}'(\mu')\right| \Delta \;, \tag{8}$$

where Δ is an upper bound on the absolute value of any sub-determinant of the matrix $A'(\mu')$. To complete the proof, we have to show that the right hand side of (8) is constant.

First we give an upper bound on the number of columns $|\mathbf{K}'(\mu')|$, i.e., on the number of machine configurations with load at most μ'. Since each job has size at least

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

there are at most $\gamma := \lfloor 4(1+\epsilon)/\epsilon \rfloor \le \frac{8}{\epsilon}$ jobs in any configuration $k \in \mathbf{K}'(\mu')$. In particular, $k(i) \le \gamma$ for all $i \in I'$. This yields

$$\left|\mathbf{K}'(\mu')\right| \;\le\; \gamma^{|I'|} \;\le\; \gamma^{|I|+1} \le \left(\frac{8}{\epsilon}\right)^{|I|+1} \;\overset{(2)}{\le}\; \left(\frac{8}{\epsilon}\right)^{\frac{3}{\epsilon}\log(\frac{1+\epsilon}{\epsilon})} \;\le\; \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{3}{\epsilon}\log(\frac{8}{\epsilon})} .$$

Finally, all entries in the first row of $A'(\mu')$ are 1 and the remaining entries are of the form $k(i) \leq \gamma$. Hence we bound Δ as follows. The maximum dimension of a square sub-matrix inside $A'(\mu')$ is at most the number of rows, i.e., $2 + |I|$ and, each entry in it has value at most γ. Hence the value of its determinant is upper-bounded by $((2 + |I|)\gamma)^{2+|I|}$. Thus

$$\Delta \leq \left((2 + |I|)\,\gamma\right)^{2+|I|} \overset{(2)}{\leq} \left(\frac{8}{\epsilon^2}\right)^{2+|I|} \cdot \left(\frac{8}{\epsilon}\right)^{2+|I|} \overset{(2)}{\leq} \left(\frac{1+\epsilon}{\epsilon}\right)^{\frac{12}{\epsilon}\log(\frac{4}{\epsilon})}$$

Thus the number of machines touched, which is at most $3\left|\mathbf{K}'(\mu')\right|\Delta$ by (8), is a constant. This concludes the proof. □

Theorem 5. *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.*

Theorem 6. *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.

6 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. A simple scheme with many nice properties implements this idea if we manage to partition the SAN into several sub-servers of about equal size [13]. Mapping to the scheduling paradigm, the contents of disks correspond to jobs and the sub-servers correspond to machines. 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. More sub-servers allow for a more flexible tradeoff between fault tolerance, redundancy, and access granularity. In any case, the capacity of the server 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. For the case without disk failures (job departures) we obtain the following result using a simple greedy strategy.

Theorem 7. *There exists a* 2*–competitive strategy with migration factor* 1*.*

Acknowledgments. We thank Gerhard Woeginger for interesting discussions and helpful comments on the topic of this paper. We also thank the referees for pointing us to [3,18].

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29:459–473, 1999.
2. S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97:3–26, 2003.
3. M. Andrews, M.X. Goemans, and L. Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23:278–301, 1999.
4. Y. Azar and L. Epstein. On-line machine covering. *Journal of Algorithms*, 1:67–77, 1998.
5. 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.
6. B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51:219–222, 1994.
7. R. Fleischer and M. Wahl. Online scheduling revisited. *Journal of Scheduling*, 3:343–353, 2000.
8. M. R. Garey and D. S. Johnson. Strong np-completeness results: Motivation, examples and implications. *Journal of the ACM*, 25:499–508, 1978.
9. R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.
10. 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.
11. D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.
12. J. F. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.
13. P. Sanders. Algorithms for scalable storage servers. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932, pages 82–101, 2004.
14. P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. Research Report MPI-I-2004-1-004, Max-Planck-Institut für Informatik, Saarbrücken, Germany, April 2004.
15. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Chichester, 1986.
16. J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 14 July 1997.
17. 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.
18. J. Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000.