On-Line Load Balancing of Temporary Tasks on Identical Machines

Yossi Azar[*] Tel-Aviv Univ.

Leah Epstein[†] Tel-Aviv Univ.

Abstract

We prove an exact lower bound of $2-\frac{1}{m}$ on the competitive ratio of any deterministic algorithm for load balancing of temporary tasks on m identical machines. We also show a lower bound of $2-\frac{1}{m}$ for randomized algorithms for small m and $2-\frac{2}{m+1}$ for general m. If in addition, we restrict the sequence to polynomial length, then the lower bound for randomized algorithms becomes $2-O(\frac{\log\log m}{\log m})$ for general m.

1. Introduction

We consider the problem of non-preemptive on-line load balancing of tasks on m identical machines. Tasks (jobs) arrive at arbitrary times, where each task has a weight and a duration. A task has to be assigned upon its arrival to exactly one of the machines, thereby increasing the *load* on this machine by its weight for the duration of the task. The duration of each task becomes known only upon its termination (this is called temporary tasks of unknown durations). Once a task has been assigned to a machine it cannot be reassigned to another machine. The goal is to minimize the maximum load over machines and time.

The problem of scheduling tasks on identical machines was first introduced by Graham [11, 12]. He gave a greedy algorithm "List Scheduling" which is $2-\frac{1}{m}$ competitive, where m is the number of machines. The upper bound was proved for permanent tasks, i.e., tasks that start at arbitrary times but continue forever. Nevertheless, his $2-\frac{1}{m}$ analysis of the upper bound holds also for temporary tasks.

In this paper we show that his algorithm is optimal by proving a matching lower bound. We show a lower bound of $2-\frac{1}{m}$ on the competitive ratio of any deterministic on-line algorithm for load balancing of temporary tasks on identical machines. We also show a lower bound of $2-\frac{2}{m+1}$ on the competitive ratio of any randomized on-line algorithm for the problem. In fact, for $m=2,3,4$ we can improve the lower bound to $2-\frac{1}{m}$, which implies that "List Scheduling" is also optimal in these cases. The randomized lower bound for general m requires a sequence of tasks of super-polynomial length in m. If we restrict the sequence to have a polynomial length we prove a lower bound of $2-O(\frac{\log\log m}{\log m})$ for any randomized algorithm.

Recall that Graham [11, 12] considered only permanent tasks. He showed that the greedy algorithm "List Scheduling" does not perform better than $2-\frac{1}{m}$. For $m=2,3$ the algorithm is optimal [9]. However, the algorithm of Graham is not optimal (for all $m \geq 4$) [10, 8]. Bartal et al. [5] were the first to show an algorithm whose competitive ratio is strictly below $c<2$ (for all m). More precisely, their algorithm achieves a competitive ratio of $2-\frac{1}{70}$. Later, the algorithm was generalized by Karger, Phillips and Torng [13] to yield an upper bound of 1.945. Very recently, Albers [1] designed 1.923 competitive algorithm and improved the lower bound to 1.852 (the previous lower bound for permanent tasks was 1.8370 [6]). The best lower bound known for randomized algorithms is 1.582 (for large m) [7, 14]. For $m=2$ the randomized competitive ratio is precisely $4/3$ [5]. We show that in contrast to permanent tasks, the simple algorithm of Graham turns out to be optimal for temporary tasks. Moreover, even randomization cannot reduce the competitive ratio below $2-o(1)$. Note that for $m=2$ our tight randomized lower bound is $3/2$. We also prove the same lower bound for the known duration case. This is in contrast to the competitive ratio for permanent jobs that is $4/3$.

To prove our randomized lower bound we introduce a new technique that converts a lower bound for deterministic algorithms to a lower bound for randomized algorithms. More precisely, we show that a lower bound for deterministic algorithms that maintains a fixed value for the optimal assignment is a lower bound for randomized algorithms.

The problem of on-line load balancing of temporary

[*]Dept. of Computer Science, Tel-Aviv University. Research supported in part by Allon Fellowship and by a grant from the Israel Science Foundation. E-Mail: azar@math.tau.ac.il

[†]Dept. of Computer Science, Tel-Aviv University. E-Mail: lea@math.tau.ac.il

tasks was introduced by Azar, Broder and Karlin [2]. They studied the restricted assignment case, i.e. each task can be assigned only to a machine in a subset which may depend on the task. They showed an $\Omega(\sqrt{m})$ lower bound in contrast to the $\Theta(\log m)$ competitive ratio for permanent tasks [4]. A matching upper bound was given in [3]. Load balancing of temporary tasks was also studied for the related machines model. In this model the increase of the load on a machine is the ratio of the weight of the task and the speed of that machine. An algorithm which is 20 competitive and a lower bound of $3 - o(1)$ were given by [3].

2. Notations

We denote the input sequence by $\sigma = \sigma_1, ..., \sigma_r$. Each event σ_i is an arrival or a departure of a job (task). We view σ as a sequence of times, the time σ_i is the moment after the $i^{\underline{th}}$ event happened. We denote the weight of job j by w_j, its arrival time by a_j and its departure time (which is unknown until it departs) by d_j. An on-line algorithm has to assign a job upon its arrival without knowing the future jobs and the durations of jobs that have not departed yet. We compare the performance of on-line algorithms and the optimal off-line algorithm that knows the sequence of jobs and their durations in advance.

Let $J_i = \{j | a_j \leq \sigma_i < d_j\}$ be the active jobs at time σ_i. For a given algorithm A (on-line or off-line) let A_j be the machine on which job j is assigned. Let

$$l_k^A(i) = \sum_{\{j | A_j = k, j \in J_i\}} w_j$$

be the load on machine k at time σ_i, which is the sum of weights of all jobs assigned to k, and active at this time. The cost of an algorithm A is the maximum load ever achieved by any of the machines, i.e., $C_A = max_{i,k}\ l_k^A(i)$. The competitive ratio of A is r if for any sequence $C_A \leq r \cdot C_{opt}$ where C_{opt} is the cost of the optimal off-line algorithm.

3. Lower bounds for deterministic algorithms

We start with a simple lower bound of $2 - \frac{2}{m+1}$. The lower bound holds for the competitive ratio of any deterministic algorithm even if the optimal value is fixed and known in advance. Later we show the tight $2 - \frac{1}{m}$ lower bound.

Theorem 3.1 *Any deterministic on-line algorithm for load balancing of temporary tasks has a competitive ratio of at least $2 - \frac{2}{m+1}$. This is true even if the optimal value is known in advance (and is $m + 1$).*

Proof: We consider the following sequence. First arrive m^2 unit jobs ($w_j = 1$). Since there are m machines, there is at least one machine (say x) with a load of at least m, (see figure 1). Now all jobs depart, except m jobs on this machine, (figure 2).

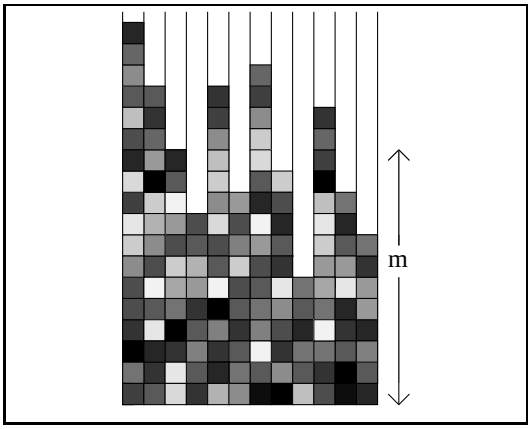

Figure 1. The on-line algorithm after the arrival of the unit jobs

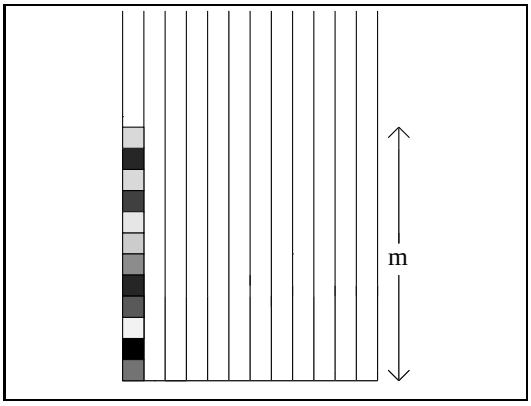

Figure 2. The on-line algorithm after the departure of the unit jobs except m jobs on one machine

Now m jobs of weight m arrive. Since machine x has load m, the on-line has to assign two jobs of weight m on one machine, or to assign one job of weight m on the machine x. In both cases, the maximum load of the on-line is at least $2m$. The off-line distributes the m large jobs and the m unit jobs that remained from the first phase evenly on the m machines, and thus has a load of $m + 1$. The other $m(m-1)$ unit jobs of the first phase are also distributed evenly on the m machines. The ratio between the costs of the two algorithms is at least $\frac{2m}{m+1} = 2 - \frac{2}{m+1}$. ■

The above lower bound can be improved as follows:

Theorem 3.2 *Any deterministic on-line algorithm for load balancing of temporary tasks has the competitive ratio of at least* $2 - \frac{1}{m}$.

Proof: We consider the following sequence. First $m(m-1)$ unit jobs arrive ($w_j = 1$). Since there are m machines, there is at least one machine (say x) with load at least $m-1$. Then all jobs depart, except $m-1$ jobs on this machine. Next $m-1$ jobs of load $m-1$ arrive. There are two possible cases:

1. The on-line algorithm keeps at least one empty machine. In this case, the load of the on-line algorithm on some machine is at least $2(m-1)$ since at least two jobs of load $m-1$ were assigned to one machine, or a job of load $m-1$ was assigned to the machine x, (see figure 3).

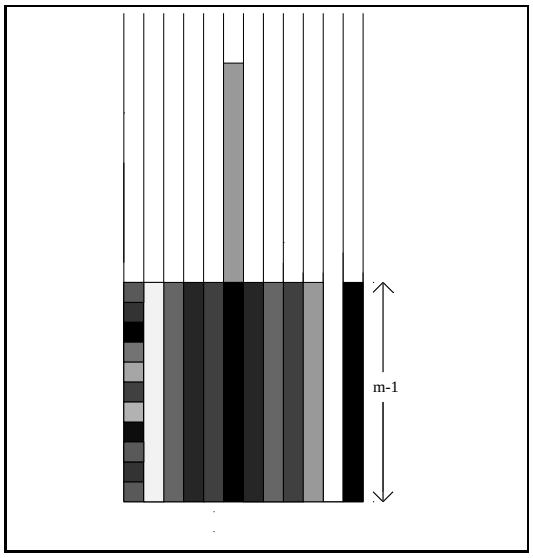

Figure 3. The on-line state if it assigns two jobs of weight $m-1$ on one machine

   At the first phase, the off-line assigns the $m-1$ unit jobs which will not depart on one machine and distributes the $(m-1)^2$ unit jobs that will depart evenly on the other $m-1$ machines. Then it assigns one job of weight $m-1$ on each of the other machines (see figure 4), to have the maximum load of $m-1$. In this case, the ratio between the costs of the on-line and the off-line algorithms is at least $2(m-1)/(m-1) = 2$.

2. The on-line algorithm does not keep an empty machine, i.e., there is now one job of weight $m-1$ on each machine except x, on which there are $m-1$ units jobs. Now one additional (and final) job of weight m arrives. The on-line must assign it on one of the machines (figure 5), which results in a load of $2m-1$. We show that the off-line can assign the jobs, having a maximum load of m. At the first phase, the unit jobs that do not depart, are assigned each to a different machine, the other jobs are distributed so that the load on each machine is exactly $m-1$. At the second phase only $m-1$ unit jobs, each on a different machine, have remained, and the jobs of load $m-1$ are added to those machines yielding load of m and keeping one machine empty. At last the job of load m is assigned to the empty machine (figure 6). In this case, the competitive ratio is at least $(2m-1)/m = 2 - 1/m$.

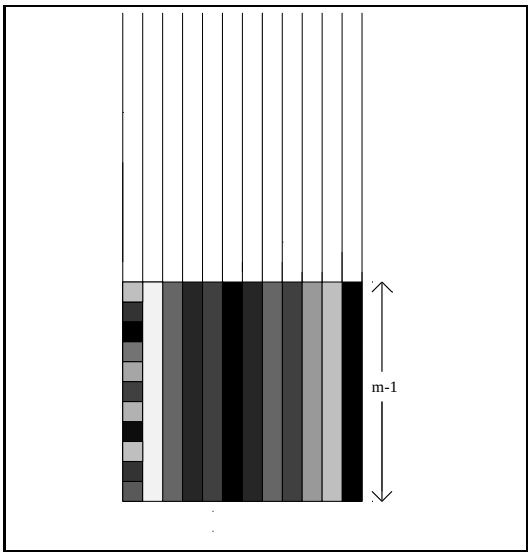

Figure 4. The off-line algorithm after the arrival of the $m-1$ jobs of load $m-1$

In both possible cases, the competitive ratio is at least $2 - \frac{1}{m}$, and thus any on-line algorithm has at least this ratio. ■

4. Lower bounds for randomized algorithms

In this section we prove lower bounds for randomized on-line algorithms. We start by proving a general theorem that converts lower bounds for deterministic algorithms with a fixed value for the optimal load to general lower bounds for randomized algorithms. We represent any lower bound for a deterministic algorithm by a tree. Each path in the tree is one possible lower bound sequence. Each node in the tree is a subsequence, and a child node of a node is one possible way to continue the sequence (see figure 7). The

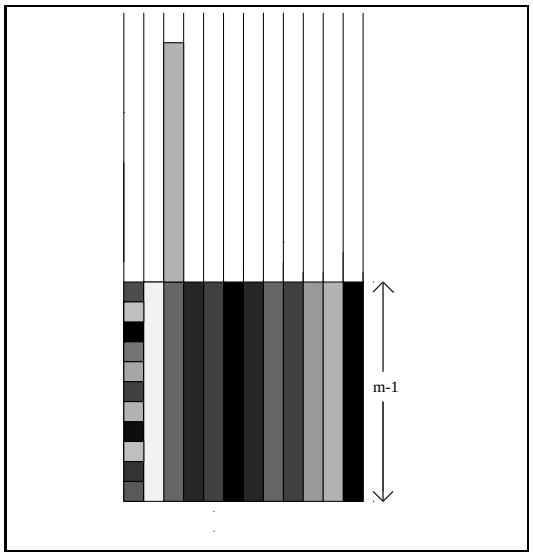

Figure 5. The on-line assigns the job of weight m above a job of weight $m-1$

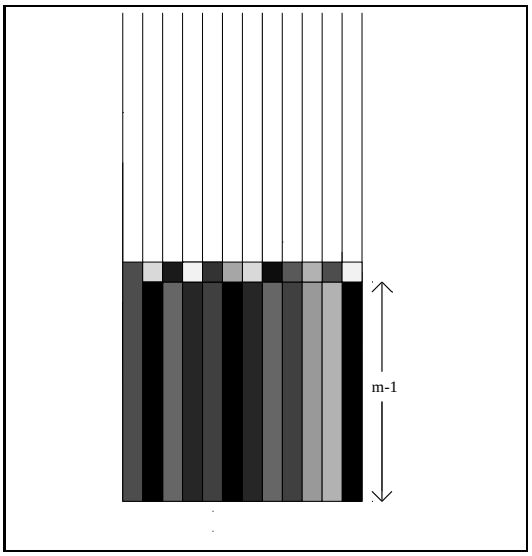

Figure 6. The off-line assignment if the job of weight m arrives

size $|T|$ of a tree T is defined to be the number of leaves in T (the number of possible sequences). We consider both the unknown duration case and the known duration case. In the first case the duration of a job is known only when it departs where in the second one the duration of a job is known upon its arrival.

Theorem 4.1 *Let r_1 (r_2, respectively) be a deterministic lower bound for temporary jobs with unknown (known, respectively) durations, when the optimal value of the load is known in advance. Then, r_1 (r_2, respectively) is also a lower bound for randomized algorithms for temporary jobs with unknown (known, respectively) durations.*

Proof: Consider a lower bound tree T' for deterministic algorithms with a fixed optimal load which is known in advance. We show how to convert it into a lower bound for randomized algorithms. A lower bound for temporary jobs with unknown durations, is converted into a lower bound for randomized unknown durations, and a lower bound for known durations is converted into a lower bound for randomized known durations. We first slightly modify the lower bound tree as follows: each possible sequence $\sigma \in T'$, is followed by the events that all the existing jobs depart. This can be done for unknown durations and also for known durations. Note that the new tree T satisfies $|T| = |T'|$. Next we recall the adaptation of Yao's theorem for on-line algorithms. It states that a lower bound for the competitive ratio of deterministic algorithms on any distribution on the input is also a lower bound for randomized algorithms and is given by $E(C_{on}/C_{opt})$. The main idea of the proof is to construct sequences in which on one hand the new value C'_{opt} is the same as the known optimal value C_{opt} of the lower bound of the original tree T and on the other hand with high probability the new value of C'_{on} is also the same as the value C_{on} for T. To construct the lower bound we choose uniform at random a leaf of the tree T that corresponds to a sequence. Define this short sequence as a segment. Repeat the choice of segments $|T|k$ times, and concatenate the sequences to one long sequence. This defines a distribution on the set of possible long sequences. Since the off-line costs of all possible segments are the same, the off-line cost of every resulting sequence is C_{opt} as well. For any deterministic algorithm, there exists a leaf in T that has the on-line cost C_{on}. With probability at least $\frac{1}{|T|}$, the cost of the on-line algorithm on a specific segment (and thus for the whole sequence) is C_{on}. The probability that the cost C_{on} would not be achieved in one segment is at most $(1-\frac{1}{|T|})^{|T|k} \leq e^{-k}$ and thus with probability at least $1-e^{-k}$ the competitive ratio is C_{on}/C_{opt}, and otherwise it is at least 1. We calculate $E\left(\frac{C'_{on}}{C'_{opt}}\right)$ where C'_{opt},and C'_{on}, are respectively, the off-line and on-line costs of the long sequence.

$$E\left(\frac{C'_{on}}{C'_{opt}}\right) \geq (1-e^{-k})\frac{C_{on}}{C_{opt}} + e^{-k} \ .$$

Since this is true for every k,

$$E\left(\frac{C'_{on}}{C'_{opt}}\right) \geq \frac{C_{on}}{C_{opt}} \ ,$$

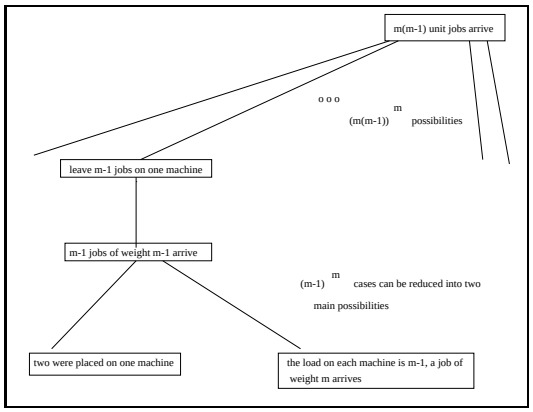

Figure 7. The tree for the lower bound in Theorem 3.2

which is exactly the competitive ratio of the lower bound of the tree T. ■

Corollary 4.2 *Any randomized on-line algorithm for load balancing of temporary tasks has a competitive ratio of at least* $2 - \frac{2}{m+1}$.

Proof: The proof follows from Theorems 3.1 and 4.1. In this case, there are $\left(m^2\right)^m = m^{2m}$ different possibilities for the m^2 unit jobs to be placed, and there are m^{2m} leaves in the lower bound tree. ■

We now improve the lower bound for small numbers of machines.

Theorem 4.3 *Any randomized on-line algorithm for load balancing of temporary tasks on two machines, has a competitive ratio of at least* $2 - \frac{1}{2} = \frac{3}{2}$.

Proof: We show a lower bound for deterministic algorithms, for which the value of the optimal load is 2. Theorem 4.1 implies the same lower bound for randomized algorithms. We consider the following sequence. First three unit jobs arrive ($w_j = 1$). The possible behaviors of the algorithm can be divided into two cases.

1. All jobs are on one machine. In this case $C_{on} = 3$ and thus the competitive ratio is at least 1.5.
2. There is at least one job on each machine. In this case, there is one machine with two jobs. We pick one of these jobs and let it depart. Thus, we are left with one job on each machine. Now a final job of weight 2 arrives, and the current load on the machine on which it is assigned is 3. Again the competitive ratio is 1.5 .

■

We can extend this lower bound for the known duration case, i.e., the duration of each job is known upon its arrival.

Theorem 4.4 *Any randomized on-line algorithm for load balancing of temporary tasks on two machines has a competitive ratio of at least* $2 - \frac{1}{2} = \frac{3}{2}$ *even if the durations of the jobs are known upon its arrival.*

Proof: We again show a lower bound for deterministic algorithms with fixed optimal load which is 2. The randomized lower bound follows from Theorem 4.1. We consider the following sequence. At time 0 a unit job of duration 8 arrives. At time 1 a unit job of duration 3 arrives. Now there are several cases. If the two jobs are on different machines, a job of weight 2 and duration 1 arrives at time 2. Between time 2 and time 3 all the three jobs are present and the on-line load is 3, which yields the competitive ratio of $3/2$. Otherwise, the two jobs are on one machine. In this case a third unit job of duration 5 arrives at time 2. If it is assigned to the same machine, the on-line load is already 3. If it is assigned to the other machine, we wait to time 5 that one unit job has already departed, and we have one unit job on each machine. At that time a final job of weight 2 and duration 1 arrives, and the on-line load is 3. It is easy to check that the optimal load in each case is 2 and thus the competitive ratio is $3/2$. ■

We can extend the lower bound of Theorem 4.3 for 3 and 4 machines.

Theorem 4.5 *Any randomized on-line algorithm for load balancing of temporary tasks on* $m = 3, 4$ *machines has a competitive ratio of at least* $2 - \frac{1}{m}$, *i.e.,* $5/3$ *for* 3 *machines, and* $7/4$ *for* 4 *machines.*

Proof: We show lower bounds for deterministic algorithms with fixed optimal load as before. Let $m = 3$. Consider the following sequence that maintains optimal off-line load of 3. First eight unit jobs arrive.

1. If there is a machine with at least five jobs then the on-line load is at least 5. Thus we can assume that there are at most four jobs on each machine.
2. If there are at least two jobs on each machine then we continue the sequence by a departure of two jobs so that exactly two jobs remain on each machine. Now one final job of size 3 arrives and the on-line load is 5.
3. If there is at least one machine with at most one job then there must be at least three jobs on each of the other two machines, since there are at most four jobs on each machine. Now two jobs depart, so that there is one empty machine, and two machines, each with three jobs. Now one job of size 2 arrives.

- If it is assigned to one of the non-empty machines then the on-line load is 5.
- If it is assigned to the empty machine then two unit jobs, one from each of the other two machine, depart. This results in a load 2 on all three machines. One final job of weight 3 causes the on-line the load of 5.

In each case the optimal algorithm can maintain maximum load 3 and thus the competitive ratio is at least $\frac{5}{3}$.

We use the same idea to prove the lower bound for $m = 4$. Consider the following sequence that maintains an optimal off-line load of 4. First 16 unit jobs arrive. There are several cases:

1. If there is a machine with at least seven jobs then the on-line load is at least 7.
2. If there are at least three jobs on each machine then four jobs depart, so that there are exactly three jobs on each machine. Then a final job of weight 4 arrives, which causes the load of 7 to the on-line.
3. If there is at least one job on each machine then we continue as follows. First note that there is at least one machine with four jobs or more. Four jobs on this machine and one job on each of the other machines remain and all other jobs depart. Now three jobs of weight 3 arrive.
   - If two jobs of weight 3 assigned to one machine, or a job of weight 3 assigned to the machine with load 4 then the on-line load is at least 7.
   - Otherwise, there are three machines with the same configuration: two jobs, one of which has weight 3, and the other has weight 1. The fourth machine contains four unit jobs. Now four unit job depart, one from each machine, which results in load 3 to all machines. A final job of weight 4 arrives and creates on-line load of 7.
4. If there is one empty machine and there are at most six jobs on each machine then there are three machines each with at least four jobs. Next, four jobs depart so that there are exactly four jobs on each of the three machines. Now one job of weight 3 arrives. It should be assigned to the empty machines in order not to create load 7. Now three unit jobs depart, one from each of the machines with load 4. Consequently, every machine has load 3. Finally, one job of weight 4 arrives and creates load 7 for the on-line algorithm.

One can easily see that the optimal off-line algorithm can maintain load 4 in all the cases and thus the competitive ratio is at least $\frac{7}{4}$. ■

5. Sequences of polynomial length

Theorem 4.1 provides lower bounds that require $|T| \cdot k$ large sequences. To get short sequences, we use the same methods, but we examine the tree more carefully.

We first consider a base sequence. Let h be an integer $h \leq m$.

1. mh unit jobs ($w_j = 1$) arrive.
2. $mh - m$ unit jobs chosen uniformly at random depart. Thus a random set S of m exactly unit jobs remains. Note that all $\binom{mh}{m}$ possible sets of m remaining jobs are equally likely.
3. Now m jobs of weight h arrive.
4. All jobs depart.

The off-line can distribute evenly the m unit jobs of S and the m larger jobs on the m machines. Thus it maintains load of $h+1$ at the end of phase 3 and can easily maintain a load of h at the end of phase 1. Let us calculate the expected maximum load of the on-line algorithm. In the first step, there are mh unit jobs, thus there is (at least) one machine that contains at least h jobs. Denote the set of the first h jobs on that machine as T. If $S \supseteq T$ the load for the on-line algorithm is at least $2h$, at the end of phase 3. In this case, the ratio between the maximum load of the on-line and the off-line is $\frac{2h}{h+1} = 2 - \frac{2}{h+1}$. Note that S is a random variable and T is a fixed set. Clearly, the probability of the event $S \supseteq T$ is $\binom{mh-h}{m-h} / \binom{mh}{m}$.

We use this sequence to prove the polynomial length lower bound for randomized algorithms.

Theorem 5.1 *Any randomized on-line algorithm for load balancing of temporary tasks has a competitive ratio of at least $2 - O(\frac{\log\log m}{\log m})$ even when the input sequence is of polynomial length in m.*

Proof: We choose $h = \Theta(\frac{\log m}{\log\log m})$ such that $2h^h \leq m$ and repeat the base sequence mk times.

Let us calculate the probability that S never contains T. The probability that S does not contain T in one base sequence is

$$\begin{aligned} 1 - \frac{\binom{mh-h}{m-h}}{\binom{mh}{m}} &= 1 - \frac{(mh-h)!}{(m-h)!} \cdot \frac{m!}{(mh)!} \\ &\leq 1 - \frac{\left(1-\frac{h}{m}\right)^h}{h^h} \\ &\leq 1 - \frac{1-\frac{h^2}{m}}{h^h} \\ &\leq 1 - \frac{1}{2h^h} . \end{aligned}$$

The probability that S never contains T is at most

$$(1-\frac{1}{2h^h})^{mk} \leq (1-\frac{1}{m})^{mk} \leq e^{-k} \ .$$

In this case the ratio is at least 1, otherwise the ratio is $\frac{2h}{h+1} = 2 - \frac{2}{h+1}$. This implies as before that

$$E\left(\frac{C_{on}}{C_{opt}}\right) \geq 2 - O\left(\frac{\log\log m}{\log m}\right) \ .$$

Note that the length of the sequence is polynomial in m as required. ■

6. Concluding remarks

We proved $2 - o(1)$ lower bounds on the competitive ratio of load balancing of temporary tasks. Note that there is a small gap between the randomized lower bound and the optimal deterministic one. One would like to know the exact bound for randomized algorithms or at least if randomization helps at all to reduce the competitive ratio in these problems. It follows from our results that randomization may help slightly only for $m \geq 5$. It is open if knowing the durations of the tasks can help in reducing the competitive ratio strictly below 2 both for deterministic and randomized algorithms.

7. Acknowledgments

We would like to thank Allan Borodin for many helpful discussions.

References

[1] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th ACM Symp. on Theory of Computing*, 1997. To appear.

[2] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proc. 33rd IEEE Symp. on Found. of Comp. Science*, pages 218–225, Oct. 1992.

[3] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Proc. Workshop on Algorithms and Data Structures*, pages 119–130, Aug. 1993.

[4] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignment. In *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, pages 203–210, 1992.

[5] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proc. 24th ACM Symposium on Theory of Algorithms*, pages 51–58, 1992. To appear in *Journal of Computer and System Sciences*.

[6] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Processing Letters*, 50:113–116, 1994.

[7] B. Chen, A. van Vliet, and G. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51:219–222, 1994.

[8] B. Chen, A. van Vliet, and G. Woeginger. New lower and upper bounds for on-line scheduling. *Operations Research Letters*, 16:221–230, 1994.

[9] U. Faigle, W. Kern, and G. Turan. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[10] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. *Siam Journal on Computing*, 22(2):349–355, 1993.

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

[12] R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:263–269, 1969.

[13] D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[14] J. Sgall. On-line scheduling on parallel machines. Technical Report Technical Report CMU-CS-94-144, Carnegie-Mellon University, Pittsburgh, PA, USA, 1994.