On-line Load Balancing for Related Machines [1]

Piotr Berman

The Pennsylvania State University, University Park, PA16802, USA
E-mail: berman@cse.psu.edu

and

Moses Charikar*

Stanford University, Stanford, CA 94305-9045
E-mail: moses@cs.stanford.edu

and

Marek Karpinski†

University of Bonn, 53117 Bonn, and
International Computer Science Institute, Berkeley
E-mail: marek@cs.uni-bonn.de

We consider the problem of scheduling permanent jobs on related machines in an on-line fashion. We design a new algorithm that achieves the competitive ratio of $3+\sqrt{8} \approx 5.828$ for the deterministic version, and $3.31/\ln 2.155 \approx 4.311$ for its randomized variant, improving the previous competitive ratios of 8 and $2e \approx 5.436$. We also prove lower bounds of 2.4380 on the competitive ratio of deterministic algorithms and 1.8372 on the competitive ratio of randomized algorithms for this problem.

Key Words: on-line algorithm, load balancing, related machines, competitive ratio

[1] A preliminary version of this paper appeared in Proceedings of WADS 97, LNCS 1272, Springer-Verlag, 1997, 116-125.

* Supported by Stanford School of Engineering Groswith Fellowship, an ARO MURI Grant DAAH04-96-1-0007 and NSF Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

† Partially supported by the DFG Grant KA 673/4-1, by the ESPRIT BR Grants 7097, 21726, EC-US 030, and by the Max–Planck Research Prize.

1. INTRODUCTION

The problem of on-line load balancing was studied extensively over the years (cf., e.g., [11], [3], [4], and [2]). In this paper we study the on-line load balancing problem for related machines (cf. [2]). We are given a set of machines that differ in speed but are related in the following sense: a job of size p requires time p/v on a machine with speed v. While we cannot compare structurally different machines using a single speed parameter, it is a reasonable approach when the machines are similar; in other cases it may be a good approximation.

Our task is to allocate a sequence of jobs to the machines in an on-line fashion, while minimizing the maximum load of the machines. This problem was solved with a competitive ratio 8 by Aspnes *et al.* [2]. Later, it was noticed by Indyk [8] that by randomizing properly the key parameter of the original algorithm the expected competitive ratio can be reduced to $2e$. A similar randomization idea has been used earlier by several authors in different contexts (cf. [5, 9, 12, 10, 6]).

For the version of the problem where the speeds of all the machines are the same, Albers [1] proved a lower bound of 1.852 on the competitive ratio of deterministic algorithms. Chen *et al.* [7] and independently Sgall [13] proved a lower bound of $e/(e-1) = 1.5819$ on the competitive ratio of randomized algorithms. No better lower bounds are known for on-line load balancing on related machines.

Adapting the notation of Aspnes *et al.*, we have n machines with speeds $v_1, \ldots, v_n$ and a stream of m jobs with sizes $p_1, \ldots, p_m$. A schedule s assigns to each job j the machine $s(j)$ that will execute it. We define $load(s,i)$, the load of a machine i in schedule s and $Load(s)$, the load of entire schedule s as follows:

$$load(s,i) = \frac{1}{v_i} \sum_{s(j)=i} p_j, \quad Load(s) = \max_i \; load(s,i)$$

It is easy to observe that finding an optimum schedule s^* is NP-hard off-line, and impossible on-line. We want to minimize the competitive ratio of our algorithm, i.e. the ratio $Load(s)/Load(s^*)$ where s is the schedule resulting from our on-line algorithm, and s^* is an optimum schedule.

In Section 2, we describe an on-line scheduling algorithm with competitive ratio $3 + \sqrt{8} \approx 5.828$ for the deterministic version, and $3.31/\ln 2.155 \approx 4.311$ for its randomized variant. In Section 3, we prove lower bounds of 2.4380 on the competitive ratio of deterministic algorithms and 1.8372 on the competitive ratio of randomized algorithms for on-line scheduling on related machines.

2. ALGORITHM

2.1. Preliminaries

The previous algorithm for our problem, due to Aspnes *et al.* [2] uses the following idea. There exists a simple algorithm that achieves competitive ratio 2 if we know exactly the optimum load Λ: we simply assign each job to the slowest machine that would not increase its load above 2Λ. Because we do not know Λ, we make a safely small initial guess and later double it whenever we cannot schedule a job within the current load threshold. In the worst case, the final guess is almost twice the optimum, and thus the load created by the jobs scheduled in that phase can be almost four times the optimum; it is easy to see that the jobs scheduled in the previous phases can add load of the same magnitude. This way, converting from known Λ to unknown increases the resulting load at most four times. In this paper, we will show more efficient methods of such conversion.

Our innovation is to double (or rather, increase by a fixed factor r) the guess as soon as we can prove that it is too small, without waiting for the time when we cannot schedule the subsequent job. Intuitively, we want to avoid wasting the precious capacity of the fast machines with puny jobs that could be well served by the slow machines. Therefore we start with describing our method of estimating the necessary load, i.e. computing lower bounds on the optimal load for the sequence of jobs seen so far.

Let $V = \{0, v_1, \ldots, v_n\}$ (for later convenience, we assume that the sequence of speeds is nondecreasing). For $v \in V$ we define $Cap(v)$ as the sum of speeds of these machines that have speed larger than v. (*Cap* stands for capacity, note that $Cap(0)$ is the sum of speeds of all the machines and $Cap(v_n) = 0$.) For a set of jobs J and a load threshold Λ we define $OnlyFor(v, \Lambda, J)$ as the sum of sizes of those jobs that have $p_j/v > \Lambda$. (*OnlyFor* stands for the work that can be performed only by the machines with speed larger than v if the load cannot exceed Λ.) The following lemma is immediate:

LEMMA 2.1. *For a set of jobs J, there exists a schedule s with $Load(s) \leq \Lambda$ only if $OnlyFor(v, \Lambda, J) \leq \Lambda\, Cap(v)$ for every $v \in V$.*

Before we formulate and analyze our algorithm, we will show how to use the notions of *Cap* and *OnlyFor* to analyze the already mentioned algorithm that keeps the load under 2Λ if load Λ is possible off-line. We first reformulate it to make it more similar to the new algorithm which will be presented later. Machine i has *capacity* $c_i = \Lambda v_i$ equal to the amount of work it can perform under Λ load, and the *safety margin* m_i to assure that we will be able to accommodate the jobs in an on-line fashion. In this algorithm the capacity and the safety margin are given the same value, in the new one they will be different.

(* initialize *)
for $i \leftarrow 1$ **to** n **do**
  $m_i \leftarrow c_i \leftarrow \Lambda v_i$
$j \leftarrow 0$
(* online processing *)
repeat
  read(p)
  $j \leftarrow j + 1$

$s(j) \leftarrow \min\{i |\, c_i + m_i > p\}$
$c_{s(j)} \leftarrow c_{s(j)} - p$
forever

Note that m_i remains fixed while c_i could decrease during the execution of the algorithm. In fact, c_i could become negative; however the value of c_i is always at least $-m_i$.

This algorithm shares the following property with the new one: the jobs are offered first to machine 1 (the slowest), then to machine 2 etc., so each time the slowest possible machine accepts the new job. Given a stream of jobs J, we can define J_i as the stream of jobs that are passed over by machine i or that reach machine $i + 1$ (for $1 \leq i < n$ these two conditions are equivalent, for $i = 0$ only the latter and for $i = n$ only the former applies). The correctness of the algorithm is equivalent to the fact that the stream J_n is empty—it consists of the jobs passed over by all the machines. From the correctness the load guarantee follows easily, because the sum of sizes of jobs assigned to machine i is less than the initial capacity plus the safety margin, i.e. $\Lambda v_i + \Lambda v_i$, and so the load is less than $2\Lambda v_i / v_i = 2\Lambda$.

For the inductive reasoning we define $V_i = \{0, v_i, \ldots, v_n\}$ and $Cap_i(v)$ to be the sum of speeds from V_i that exceed v.

LEMMA 2.2. *If there exists a schedule s^* with $Load(s^*) = \Lambda$, then for every $i = 1, \ldots, n+1$ and every $v \in V_i$*

$$OnlyFor(v, \Lambda, J_{i-1}) \leq \Lambda\, Cap_i(v).$$

Proof. By induction on i. For $i = 1$ the claim is equivalent to Lemma 2.1. For the inductive step, after assuming the claim for i, we have to show that

$$OnlyFor(v, \Lambda, J_i) \leq \Lambda\, Cap_{i+1}(v) \text{ for } v \in V_{i+1}.$$

For $v \neq 0$, the inequality follows from the inductive hypothesis directly: the left hand side can only become smaller (because J_i is a subsequence of J_{i-1}), while the right hand side remains unchanged. Thus it suffices to show that $OnlyFor(0, \Lambda, J_i) \leq \Lambda\, Cap_{i+1}(0)$.

We consider the following two cases according to the final value of c_i in the execution of the algorithm.

Case 1: $c_i > 0$.

In this case machine i accepted all jobs with size at most $m_i = \Lambda v_i$ from the stream J_{i-1}, hence $OnlyFor(0, \Lambda, J_i)$, which is the sum of job sizes in J_i, is at most $OnlyFor(v_i, \Lambda, J_{i-1})$, which in turn is less or equal to $\Lambda\, Cap_i(v_i)$. Since $Cap_i(v_i) \leq \sum_{j=i+1}^{n} v_j = Cap_{i+1}(0)$, the claim follows.

Case 2: $c_i \leq 0$.
In this case, the total size of the jobs accepted by machine i is at least Λv_i, the initial value of c_i, hence $\mathit{OnlyFor}(0, \Lambda, J_i) \leq \mathit{OnlyFor}(0, \Lambda, J_{i-1}) - \Lambda v_i$, while

$$\mathit{Cap}_{i+1}(0) = \sum_{v \in V_{i+1}} v = \left(\sum_{v \in V_i} v \right) - v_i = \mathit{Cap}_i(0) - v_i.$$

By the inductive hypothesis, $\mathit{OnlyFor}(0, \Lambda, J_{i-1}) \leq \Lambda\, \mathit{Cap}_i(0)$. Hence, the claim follows. ▮

Lemma 2.2 allows us to conclude that if a schedule with load Λ exists, then $\mathit{OnlyFor}(0, \Lambda, J_n) \leq \Lambda\, \mathit{Cap}_{n+1}(0) = 0$. Thus the stream J_n of unscheduled jobs is empty, which means that the algorithm is correct.

2.2. The New Algorithm

The next algorithm is similar, but it proceeds in phases, each phase having a different value of Λ. Note that the algorithm presented in the previous section has to be combined with a doubling procedure to guess the optimum load Λ; thus it can be viewed as a single phase in the complete algorithm for load balancing. The new algorithm we describe incorporates the guessing of the optimal load Λ as an integral part of the algorithm. Unlike the previous algorithm, it is a complete algorithm and not just a single phase. Instead of losing a safety margin for each phase, the safety margin is shared among all phases, thus ensuring a better usage of the available space.

The algorithm uses a parameter r. While it is correct for any value of the parameter $r > 1$, we will later find the optimum values for r (the values are different in the deterministic and randomized versions). The algorithm maintains a guess Λ of the optimum load as well as variables c_i and m_i for the capacity and safety margin of machine i (as in the previous algorithm). Each time a new job is received, the algorithm checks if its guess Λ needs to be updated. The guess Λ is too low if $\mathit{OnlyFor}(v, \Lambda, J) > \Lambda\, \mathit{Cap}(v)$ for some $v \in V$. (Here J is the sequence of jobs received so far). In such a case, Λ is multiplied by r and the values of c_i and m_i are updated. m_i is set to Λv_i and c_i is incremented by m_i. The complete algorithm is as follows.

(* initialize *)
$\Lambda \leftarrow$ something very small
for $i \leftarrow 1$ **to** n **do**
  $m_i \leftarrow c_i \leftarrow 0$
$j \leftarrow 0$, $J \leftarrow$ empty string
(* online processing *)
repeat
  read(p)
  $j \leftarrow j + 1$, $p_j \leftarrow p$, append J with p_j

(* start a new phase if needed *)
while $\mathit{OnlyFor}(v, \Lambda, J) > \Lambda\, \mathit{Cap}(v)$ for some $v \in V$ **do**
$\Lambda \leftarrow r\Lambda, \quad m_i \leftarrow \Lambda v_i, \quad c_i \leftarrow c_i + m_i$
(* schedule p_j *)
$s(j) \leftarrow \min\{i|\, c_i + m_i > p_j\}$
$c_{s(j)} \leftarrow c_{s(j)} - p_j$
forever

We will say that executing "$s(j) \leftarrow \min\{i|\, c_i + m_i > p_j\}$" schedules p_j (even though, for the sake of argument, we admit the case that the set of machines with sufficient capacity is empty). We need to prove that the algorithm is correct, i.e. that we never apply min to an empty set; in other words, for every job we can find a machine with sufficient remaining capacity.

Let Λ_0 be the initial value of Λ. We view the execution as consisting of phases numbered from 0 to k, where the l-th phase schedules jobs with $\Lambda = \Lambda_l = \Lambda_0 r^l$. Let J^l be the stream of jobs scheduled in phase l. Using the same convention as in the analysis of the previous algorithm, we define J_i^l to be the stream of jobs that in phase l machine $i+1$ received or machine i passed over. Now the correctness will mean that the stream J_n^l is empty for every phase l. Λ_0 is chosen to be small enough so that the stream J^0 is empty. Setting Λ_0 to be any value strictly less than p_1/v_n achieves this, where p_1 is the size of the first job and v_n is the speed of the fastest machine.

Because the initial estimate for Λ may be too low, machines may receive more work than in the previous algorithm. This is due to the fact that in the initial phases the slower machines needlessly refuse to pick jobs that they would gladly accept later, thus increasing the load on the faster machines. Nevertheless, as we shall show, this increase is limited.

As a preliminary, we need to analyze the consequences of the test that triggers a new phase as soon as Λ is not appropriate for the stream of jobs received so far. First of all, the test implies that every Λ_l is appropriate for the stream $J^0 \cdots J^l$, and in particular, for the substream J^l. Therefore

$$\mathit{OnlyFor}(v, \Lambda_l, J^l) \le \Lambda_l\, \mathit{Cap}(v) \text{ for every phase } l \text{ and every } v \in V. \quad (\#)$$

This allows us to prove the following lemma by induction:

LEMMA 2.3. *For every $i = 0, \ldots, n$ and every phase l*

$$\sum_{t=0}^{l} \mathit{OnlyFor}(0, \Lambda_t, J_i^t) \;\le\; \left(\sum_{t=0}^{l} \Lambda_t\right)\left(\sum_{j=i+1}^{n} v_j\right).$$

Proof. For $i = 0$ this follows simply from the fact that for every phase $t \leq l$

$$\mathit{OnlyFor}(0, \Lambda_t, J_0^t) = \mathit{OnlyFor}(0, \Lambda_t, J^t) \leq \Lambda_t \, \mathit{Cap}(0) = \Lambda_t \left(\sum_{j=1}^{n} v_j \right).$$

We now show that the claim holds for $l = 0$. Note that J^0 is empty. Hence J_i^0 is empty and $\mathit{OnlyFor}(0, \Lambda_0, J_i^0)$ is 0 for all i. Thus the claim holds for $l = 0$.

Therefore we may assume that the claim is true for $(i, l-1)$ and $(i-1, l)$. We will prove the claim for (i, l). We consider two cases, according to the value of c_i at the end of phase l.

Case 1: $c_i > 0$.
Subtract formally from both sides of the claim for i and l the respective sides of the claim for i and $l - 1$. This way we see that it suffices to show that

$$\mathit{OnlyFor}(0, \Lambda_l, J_i^l) \ \leq \ \Lambda_l \left(\sum_{j=i+1}^{n} v_j \right)$$

Because the final value of c_i is positive, in phase l machine i accepted all jobs from the stream J_{i-1}^l that had size bounded by $\Lambda_l v_i$, and therefore the stream J_i^l consists only of the jobs that must be executed on machines faster than v_i. Thus the sum of sizes of all jobs in this stream, $\mathit{OnlyFor}(0, \Lambda_l, J_i^l)$, is at most $\mathit{OnlyFor}(v_i, \Lambda_l, J^l)$, which by (#) is at most $\Lambda_l \, \mathit{Cap}(v_i)$. Lastly, $\mathit{Cap}(v_i) \leq \sum_{j=i+1}^{n} v_j$.

Case 2: $c_i \leq 0$.
Suppose the final value of c_i in phase l equals some $c \leq 0$. This time subtract from both sides of the claim for i and l the respective sides of the claim for $i - 1$ and l. By the inductive hypothesis, the claim is true for $i - 1$ and l. Thus,

$$\sum_{t=0}^{l} \mathit{OnlyFor}(0, \Lambda_t, J_{i-1}^t) \ \leq \ \left(\sum_{t=0}^{l} \Lambda_t \right) \left(\sum_{j=i}^{n} v_j \right).$$

Subtracting this from the claim for i and l, we get

$$\sum_{t=0}^{l} (\mathit{OnlyFor}(0, \Lambda_t, J_i^t) - \mathit{OnlyFor}(0, \Lambda_t, J_{i-1}^t)) \ \leq \ - \left(\sum_{t=0}^{l} \Lambda_t \right) v_i.$$

It suffices to show that the above inequality holds, as adding this to the statement of the claim for $i - 1$ and l (which we know to be true), we will get the statement of the claim for i and l. Equivalently, we need to prove that

$$\sum_{t=0}^{l} (\mathit{OnlyFor}(0, \Lambda_t, J_{i-1}^t) - \mathit{OnlyFor}(0, \Lambda_t, J_i^t)) \ \geq \ \left(\sum_{t=0}^{l} \Lambda_t \right) v_i. \quad (\#\#)$$

On the left hand side this inequality has the difference between the sum of jobs sizes that reach machine i and the sum of the job sizes that are passed over by machine i to the subsequent machines (during the phases from 0 to l). In other words, this is the sum of sizes of the jobs accepted by machine i during these phases. This sum, say s, is related in the following manner to c:

$$0 \geq c = \left(\sum_{t=0}^{l} \Lambda_t v_i \right) - s \text{ which implies } s \geq \left(\sum_{t=0}^{l} \Lambda_t \right) v_i \equiv (\#\#).$$

∎

Lemma 2.3 implies that

$$\sum_{t=0}^{k} \mathit{OnlyFor}(0, \Lambda_t, J_n^t) \leq 0$$

This means that, for every phase $t \leq k$,

$$\mathit{OnlyFor}(0, \Lambda_t, J_n^t) = 0$$

Observe that $\mathit{OnlyFor}(0, \Lambda_t, J_n^t)$ is simply the sum of the sizes of all jobs in J_n^t. Thus J_n^t is empty for every phase t, implying the correctness of the algorithm.

To analyze the competitive ratio, we may assume that $Load(s^*) = 1$. Then the penultimate value of Λ must be smaller than 1 and the final one smaller than r. Consider a machine with speed 1. The work accepted by a machine is smaller than the sum of all Λ's up to that time (additions to the capacity) plus the last Λ given for the safety margin. Together it is $(r + 1 + r^{-1} + \ldots) + r = r(1/(1 - r^{-1}) + 1)$ $= r(2r - 1)/(r - 1)$. To find the best value of r, we find zeros of the derivative of this expression, namely of $(2r^2 - 4r + 1)/(r - 1)^2$, and solve the resulting quadratic equation. The solution is $r = 1 + \sqrt{1/2}$ and the resulting competitive ratio is $3 + \sqrt{8} \approx 5.8284$.

One can observe that the worst case occurs when our penultimate value of Λ is very close to 1 (i.e. to the perfect load factor). We will choose the initial value of Λ to be of the form r^{-N+x} where N is a suitably large integer and x is chosen, uniformly at random, from some interval $< -y, 1 - y >$ (we shifted the interval $< 0, 1 >$ to compensate for the scaling that made $Load(s^*) = 1$). Therefore we can replace the factor r with the average value of the last Λ. For negative x this value is r^{x+1}, for positive it is r^x. The average is

$$\int_{-y}^{0} r^{1+x} dx + \int_{0}^{1-y} r^x dx = \int_{1-y}^{1} r^x dx + \int_{0}^{1-y} r^x dx = \int_{0}^{1} r^x dx = \frac{r - 1}{\ln r}.$$

Therefore the expected competitive ratio is

$$\frac{r-1}{\ln r}\frac{2r-1}{r-1} = \frac{2r-1}{\ln r}.$$

The equation for the minimum value of this expression does not have a closed form solution, but nevertheless we can approximate it numerically. The minimum is achieved for r close to 2.155, and approximately equals 4.311.

3. LOWER BOUNDS

In this section, we will prove deterministic and randomized lower bounds for load balancing on related machines.

As with the algorithm in the previous section, we will use very similar constructions for the deterministic and the randomized case. The set of machines and the sequences of jobs being considered are defined in terms of a parameter $\alpha > 1$. Different values of the parameter will be used in the deterministic and randomized cases.

We will consider a set of machines $\{0, 1, \ldots, n\}$, where i-th machine has speed

$$v_i = \begin{cases} \alpha^{-i} & \text{if } i < n \\ \frac{\alpha}{\alpha-1}\alpha^{-n} & \text{if } i = n \end{cases} \tag{1}$$

Note that $v_n = \sum_{i=n}^{\infty} \alpha^{-i}$, consequently $\sum_{i=0}^{n} v_i = \sum_{i=0}^{\infty} \alpha^{-i}$. In the deterministic case the only requirement on n will be that $v_n \leq v_3 = \alpha^{-3}$. In the randomized case we will consider $n = 6$. In both settings, a lower bound for some particular value of n implies the same result for all higher values.

The sequences of job sizes that we will consider will have the form $J_i = (j_0, \ldots, j_i)$, where $j_k = \alpha^k$. These sequences have two salient properties. Since all of them are prefixes of the same infinite sequence, after processing k initial jobs, all sequences of length at least k remain equally possible. Moreover, j_i, the size of the last job of J_i, is also the optimum load: on one hand, the last job alone requires load j_i, as the largest speed equals 1; on the other, we can achieve this load by assigning j_k to machine $\min(i-k, n)$.

3.1. Deterministic Lower Bound

The idea of the deterministic lower bound is to formulate a necessary condition for the existence of a scheduling algorithm that has a competitive ratio below α^3; finally arriving at a condition that can be effectively tested by a computer program. Then to show a lower bound of α^3 it will be sufficient to check that α fails this test. We start from the following lemma.

LEMMA 3.1. *Suppose that a scheduling algorithm A achieves a competitive ratio lower than α^3. Then for every sequence of job sizes of the form J_i all jobs are scheduled on machines 0, 1, and 2.*

Proof. Suppose an algorithm A schedules the ith job on machine $m > 2$ for some i. Then for the sequence J_i the algorithm will lead on machine m to the load $j_i/v_m \geq j_i/\alpha^{-3} = \alpha^3 j_i$ (remember that $v_m \leq \alpha^{-3}$). On the other hand the optimum load for J_i is j_i, hence the competitive ratio is at least α^3, a contradiction. ▌

The second step in developing our necessary condition is forming a representation of the configuration achieved by an algorithm after processing a sequence J_i. Obviously, we can represent this configuration by recording for each machine the sum of sizes of jobs scheduled on this machine. By Lemma 3.1, we may assume that jobs are scheduled only on machines 0, 1, and 2, and so it suffices to record the sums for these machines only, say that they form a vector $S^i = (S^i_0, S^i_1, S^i_2)$. By dividing the coefficients of this vector by the optimum load we obtain the vector of *relative loads* $R^i = \alpha^{-i}S^i$. This way the necessary condition for the competitive ratio to be below α^3 is that $R^i \prec U$ for every $i > 0$. Here the upper bound vector U is $(\alpha^3, \alpha^2, \alpha)$ and $X \prec Y$ means that for every coordinate the entry in X is lower than the entry in Y.

Let $E_0 = (1, 0, 0)$, $E_1 = (0, 1, 0)$ and $E_3 = (0, 0, 1)$. Suppose that the $(i+1)$st job is scheduled on machine m. Then $R^{i+1} = \alpha^{-1}R^i + E_m$. Obviously, $R^0 = (0, 0, 0)$. Consider the following infinite graph G_0: the set of vertices is $\mathbf{V}$ = $\{V \in \mathbf{R}^3 | \ V \prec U\}$ and the edges are of the form $(V, \alpha^{-1}V + E_m)$. Then if there exists a deterministic scheduling algorithm with competitive ratio below α^3, we have an infinite path in the graph G_0 that starts at the node $(0, 0, 0)$.

To reduce the graph to a finite size we *discretize* the relative load vector. Intuitively, for some small δ, we want to map every vertex V in the infinite graph (V is a vector in $\mathbf{R}^3$), to the vector obtained by replacing every coordinate of V by the smallest multiple of δ that does not exceed it. More precisely, we perform discretization as follows. Consider the operation $'$ defined on vectors, such that V' is a vector obtained from V by replacing each coordinate x with $\lfloor x \rfloor$. Let $n > 0$ be a parameter of discretization (the value of δ in the previous discussion is $1/n$). We define a new graph G_1 with the set of nodes $(n\mathbf{V})'$ and edges of the form $(V, (\alpha^{-1}V + nE_m)')$. Obviously, if there exists an infinite path starting at $(0, 0, 0)$ in G_0, we also have such a path in G_1. Moreover, the set of nodes of G_1 consist of vectors with integer coordinates that satisfy $(0, 0, 0) \prec V \prec nU$. Therefore, this set is finite, and if there exists an infinite path starting at (0,0,0) in G_0, then there exists a cycle in G_1.

To simplify the problem further, we define $G(\alpha, n)$ to be a subgraph of G_1 consisting of nodes reachable from (0,0,0). Now we can phrase our necessary condition: for every $n > 0$ the graph $G(\alpha, n)$ contains a cycle. Consequently, to show a lower bound of α^3 it suffices to compute $G(\alpha, n)$ for some $n > 0$ and check that it is acyclic.

We have verified exactly that for $\alpha = 1.3459$ and $n = 1250$, thus obtaining a lower bound of $\alpha^3 > 2.4380$.

We mention that it is possible to give a purely analytic proof of a weaker lower bound of 2.25. The idea is to fix $\alpha = 1.5$ and consider a similar setting as before, except that we will prove that one cannot have a better competitive ratio than $\alpha^2 = 2.25$. By reasoning similarly as in Lemma 4, one can show that if we achieve a better ratio, then all jobs are scheduled on machines 0 and 1. As before we define vectors of relative loads, which must satisfy $(0,0) \prec V \prec (2.25, 1.5)$. When we schedule a job on machine 0, the vector of relative loads changes from V to $\frac{2}{3}V + (1,0)$, and if we schedule this job on machine 1, V changes to $\frac{2}{3}V + (0,1)$.

Now observe that we can never schedule two jobs in a row on machine 1, because this would change the (relative) load vector to $\frac{4}{9}V + (0, \frac{5}{3})$ and $\frac{5}{3} > \frac{3}{2}$. Suppose now that $V = (v_0, v_1)$ and we can schedule two jobs in a row on machine 0. Then the load vector changes to $\frac{4}{9}V + (\frac{5}{3}, 0)$, hence $\frac{4}{9}v_0 + \frac{5}{3} \leq \frac{9}{4}$, which implies $v_0 \leq \frac{21}{16}$. Consequently, $v_0 + v_1 \leq \frac{21}{16} + \frac{3}{2} = 3(1 - \frac{1}{16})$. On the other hand, after scheduling of the first k jobs, $v_0 + v_1 = 3(1 - (\frac{2}{3})^k)$. Thus, once we schedule the first seven jobs, we cannot ever schedule two jobs in a row at machine 0.

According to our last two observations, once we schedule the first seven jobs, we must strictly alternate between scheduling on machine 0 and on machine 1. However, if we schedule a job on machine 1 three times in such a way, the resulting relative load will be at least $1 + \frac{4}{9} + \frac{16}{81} = \frac{133}{81} > \frac{3}{2}$, a contradiction.

A more careful argument will succeed if $\alpha^4 - \alpha^3 - \alpha^2 + \alpha - 1 < 0$, which allows to increase α from 1.5 to 1.5128, and the proven ratio from 2.25 to 2.288. One can apply a similar technique to improve our better lower bound, namely 2.438, but the gain is much smaller.

3.2. Randomized Lower Bound

Fix a constant m. We consider the distribution over the m job sequences $J_1, \ldots J_m$ where the sequence J_i, $1 \leq i \leq m$ is given with probability $\frac{1}{m}$. In other words, we give the job sequence J_m and stop the job sequence after i jobs where i is chosen uniformly and at random from the set $\{1, 2, \ldots m\}$. As noted before, the optimal load for J_i is α^i, hence the expected value of the optimal load is $\frac{1}{m} \cdot \sum_{i=1}^{m} \alpha^i$.

Consider the schedule produced by a deterministic algorithm for the job sequence J_m. Note that any schedule for J_m induces a schedule for J_i, $1 \leq i \leq m$. From this, we can compute the expected load incurred by the algorithm for the chosen distribution of job sequences.

We compute all possible schedules for J_m and for each schedule, compute the expected load for the distribution of job sequences. From this, we obtain the minimum expected load for any deterministic algorithm. By Yao's principle (cf. [14]), the ratio of the minimum expected load to the expected value of the optimal load gives us a lower bound for randomized algorithms versus oblivious adversaries.

A computer program tested all possible schedules for $n = 6$, $m = 14$ and $\alpha = 1.6$ and computed a lower bound of 1.8372. Note that this implies a randomized lower bound of 1.8372 for any $n \geq 6$ machines. For $n > 6$, we consider n machines with speeds v_i chosen as before. For the purpose of analysis, we group the slowest $n-5$ machines into a single machine, i.e. pretend that any job scheduled on the $n-5$ slowest machines is scheduled on a single machine of speed $\sum_{i=5}^{n-1} v_i = \frac{\alpha}{\alpha-1} \cdot \frac{1}{\alpha^5}$. The load on this single machine is also a lower bound for the maximum load on the slowest $n-5$ machines. Observe that this gives us 6 machines whose speeds are the same as the speeds of the machines we use for the case $n = 6$. Hence the analysis for $n = 6$ applies and so does the lower bound of 1.8372.

4. CONCLUSIONS

We have designed new on–line algorithms for scheduling permanent jobs on related machines achieving the best to date deterministic and randomized competitive ratios of 5.828 and 4.311, respectively. We have also proved lower bounds of 2.4380 and 1.8372 for the corresponding competitive ratios. A challenging problem that remains is to close the gaps between the upper and lower bounds. Can some variants of our methods lead to further improvements in the competitive ratios? It seems that some new proof techniques are needed for improving our lower bounds, especially for the randomized case.

5. ACKNOWLEDGEMENTS

We would like to thank Yossi Azar, Amos Fiat, Piotr Indyk and Rajeev Motwani for valuable discussions and encouragement, and Susanne Albers for letting us read her paper before its publication.

REFERENCES

1. S. Albers, *Better bounds for online scheduling*, Proc. 29th ACM STOC (1997), pp. 130-139.
2. J. Aspnes, Y. Azar, A. Fiat, S. Plotkin and O. Waarts, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, Proc. 25th ACM STOC(1993), pp. 623-631, see also: *On-line routing of virtual circuits with applications to load balancing and machine scheduling*, J. ACM, 44:486-504, 1997.
3. Y. Azar, A. Broder, A. Karlin, *On-line load balancing*, Proc. 33rd IEEE FOCS (1992), pp. 218-225.
4. Y. Azar, J. Naor, R. Rom,*The competitiveness of on-line assignment*, Proc. 3rd ACM-SIAM SODA (1992), pp. 203-210.
5. A. Beck and D. Newman, *Yet more on the linear search problem*, Israel Journal of Math., 8:419–429, 1970.
6. S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein, and J. Wein, *Improved scheduling algorithms for minsum criteria*, Proc. 23rd ICALP, Springer, 1996.
7. B. Chen, A. van Vliet and G. J. Woeginger, *A lower bound for randomized on-line scheduling algorithms*, Information Processing Letters, vol.51, no.5, pp. 219-22, 1994.
8. P. Indyk, personal communication.

9. S. Gal, *Search Games*, Academic Press, 1980.
10. M. Goemans and J. Kleinberg, *An improved approximation ratio for the minimum latency problem*, Proc. 7th ACM-SIAM SODA, (1996), pp. 152–157.
11. R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal 45 (1966), pp. 1563-1581.
12. R. Motwani, S. Phillips and E. Torng, *Non-clairvoyant scheduling*, Proc. 4th ACM-SIAM SODA (1993), pp. 422–431, see also: Theoretical Computer Science, 130 (1994), pp. 17–47.
13. J. Sgall, *A lower bound for randomized on-line multiprocessor scheduling*, Information Processing Letters, vol.63, no.1, pp. 51–5, 1997.
14. A.C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, Proc. 17th IEEE FOCS (1977), pp. 222–227.