On-line Load Balancing for Related Machines

Piotr Berman *
Moses Charikar†
Marek Karpinski ‡

TR-97-007

January 1997

Abstract

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.

*Dept. of Computer Science & Eng., Pennsylvania State University, University Park, PA16802, USA Email:berman@cse.psu.edu

†Department of Computer Science, Stanford University, Stanford, CA 94305-9045. 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. E-mail: moses@cs.stanford.edu.

‡Dept. of Computer Science, University of Bonn, 53117 Bonn, and International Computer Science Institute, Berkeley. This research was partially supported by the DFG Grant KA 673/4-1, by the ESPRIT BR Grants 7097 and EC-US 030. Email:marek@cs.uni-bonn.de

1 Introduction

The problem of on-line load balancing was studied extensively over the years (cf., e.g., [7], [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 with 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 [6] that by randomizing properly the key parameter of the original algorithm the expected competitive ratio can be reduced to $2e$. 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 and Chen *et al.* [5] proved a lower bound of 1.5819 on the competitive ratio of randomized algorithms.

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 the load of a machine i and the load of entire schedule s as follows:

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

It is easy to observe that finding an optimum schedule s^* is NP-hard offline, 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 idea of the improvement is the following. 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.

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 from describing our method of estimating the necessary load.

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 these 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 observation is immediate:

Observation 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 reformulate it to make it more similar to the new algorithm. 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 accomodate the jobs in the 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 ← 1 to n do
    m_i ← c_i ← Λv_i
j ← 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

This algorithm shares the following property with the new one: the jobs are offered first to the machine 1 (the slowest), then to machine 2 etc., so each time the first 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 $1 \leq i < n$ this 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 reasonings we define $V_i = \{0, v_i, \ldots, v_n\}$ and $Cap_i(v)$ which is the sum of speeds from V_i that exceed v.

Observation 2. If there exists a schedule s^* with $Load(s^*) = \Lambda$, then for every $i = 0, \ldots, n$ and every $v \in V_i$

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

Proof. By induction on i. For $i = 0$ the claim is equivalent to Observation 1. For the inductive step, after assuming the claim for i, we have to show that

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

Observe that for any $v \in V_{i+1} - \{0\}$, $OnlyFor(v, \Lambda, J_i) \leq OnlyFor(v, \Lambda, J_{i-1})$ and $Cap_{i+1}(v) = Cap_i(v)$. Thus it suffices to show that $OnlyFor(0, \Lambda, J_i) \leq \Lambda Cap_{i+1}(0)$.

First observe that $Cap_{i+1}(0) = Cap_i(0) - v_i \geq Cap_i(v_i)$. We consider two cases according to the final value of c_i in the execution of the algorithm. If it is positive, then 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 $\Lambda Cap_i(v_i)$. Because $Cap_i(v_i) \leq Cap_{i+1}(0)$, the claim follows.

To finish the proof, we consider the case when the final value of c_i is negative or 0. Then total size of the jobs accepted by machine i is at least Λv_i, the initial value of c_i, hence

$OnlyFor(0, \Lambda, J_i) \leq OnlyFor(0, \Lambda, J_{i-1}) - \Lambda v_i$, while $Cap_{i+1}(0) = Cap_i(0) - v_i$. Because one of our assumption is $OnlyFor(0, \Lambda, J_{i-1}) \leq \Lambda Cap_i(0)$, the claim follows. □

Observation 2 implies that if a schedule with load Λ exists, then $OnlyFor(0, \Lambda, J_n) \leq \Lambda Cap_{n+1}(0) = 0$. 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. While it is correct for any value of the parameter $r > 1$, we will later find the optimum r's (they are different in the deterministic and randomized versions).

```
(* initialize *)
Λ ← something very small
for i ← 1 to n do
    m_i ← c_i ← 0
j ← 0, J ← empty string
(* online processing *)
repeat
    read(p)
    j ← j + 1, p_j ← p, append J with p_j
    (* start a new phase if needed *)
    while OnlyFor(v, Λ, J) > ΛCap(v) for some v ∈ V do
        Λ ← rΛ,    m_i ← Λv_i,    c_i ← c_i + m_i
    (* schedule p_j *)
    s(j) ← min{i| c_i + m_i > p_j}
    c_s(j) ← c_s(j) − p_j
forever
```

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. We will say that computing this minimum schedules p_j (even though, for the sake of argument, we admit the case that the set of machines with sufficient capacity is empty).

Let Λ_0 be the value of Λ when the first job was scheduled. We view the execution as consisting of phases numbered from 0 to k, where 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-1}^l to be the stream of jobs that in phase l machine i received or machine $i-1$ passed over. Now the correctness will mean that the stream J_n^l are empty for every phase l.

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 machines from the beginning of the sequence needlessly refuse to pick jobs that they would gladly accept later, thus increasing the load of the end of the sequence. 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, this implies that every Λ_i is appropriate for the stream $J^1 \cdots J^l$, and in particular, for the substream J^l. Therefore

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

This allows to prove, by induction on i, the following

Observation 3. For every $i = 0, \ldots, n$ and every phase l

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

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

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

For $l = 0$ the follows from Observation 2, as the phase 0 is identical to the first algorithm with $\Lambda = \Lambda_0$.

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. Assume first that this value is positive. 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, $OnlyFor(0, \Lambda_l, J_i^l)$, equals to $OnlyFor(v_i, \Lambda_l, J^l)$, which by (#) is at most $\Lambda_l\, Cap(v_i)$. Lastly, $Cap(v_i) \leq \sum_{j=i+1}^{n} v_j$.

Now assume that the final value of c_i in phase l equals some $c \leq 0$. This time subtract from both sides of the claim the respective sides of the claim for $i-1$ and l, this way we can see that it suffices to show that

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

equivalently,

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

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:

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

Observation 3 implies that

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

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

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

Observe that $OnlyFor(0, \Lambda_t, J_n^t)$ 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 componsate 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 average competitive ratio is

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

The equation for the minimum value is kind of ugly, but nevertheless 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.

Fix a parameter $\alpha > 1$. This is used to choose a set of machine speeds and the job sequence. We will specify α later with different values depending on whether we wish to obtain deterministic or randomized lower bounds.

Consider n machines $M_0 \ldots M_{n-1}$ and let v_i be the speed of M_i. The machine speeds are chosen as follows: $v_i = \frac{1}{\alpha^i}$, $0 \leq i \leq n-2$ and $v_{n-1} = \frac{\alpha}{\alpha-1} \cdot \frac{1}{\alpha^{n-1}}$. Note that $v_{n-1} = \sum_{i=n-1}^{\infty} \frac{1}{\alpha^i}$. We consider the job sequence $\rho_i = j_1, j_2 \ldots j_i$, where $j_i = \alpha^i$. Observe that ρ_i is a prefix of ρ_{i+1}.

For the job sequence ρ_i, the optimal assignment of jobs is to assign the $n-1$ biggest jobs to the machines $M_0, \ldots M_{n-2}$ and place all the remaining jobs on machine M_{n-1}. In other words, the optimal assignment for ρ_i is to assign job j_{i-k} to machine M_k, $0 \leq k \leq n-2$ and assign jobs $j_1 \ldots j_{i-n+1}$ to machine M_{n-1}. The optimal load for ρ_i is α^i, which is the size of the largest job in ρ_i.

3.1 Deterministic Lower Bound

We will focus on the load on the 3 fastest machines and demonstrate that, for a sufficiently small α, the competitive ratio of any deterministic online algorithm is at least α^3. Assume

that α and n satisfy the condition that

$$v_{n-1} = \frac{\alpha}{\alpha - 1} \cdot \frac{1}{\alpha^{n-1}} \leq \frac{1}{\alpha^3} \tag{1}$$

Let A be any online algorithm for load balancing. Suppose the competitive ratio of A is less than α^3.

Claim 3.1 *A schedules the largest job of ρ_i on one of the 3 fastest machines.*

Proof: Suppose A schedules the largest job j on some machine other than the 3 fastest machines. Then j must be scheduled on some machine of speed at most $1/\alpha^3$. The optimal load is j, but the maximum load in A's schedule is at least $\alpha^3 \cdot j$, contradicting the fact that the competitive ratio of A is less than α^3. □

Hence assume that the algorithm schedules the largest job only on one of the 3 fastest machines. This must be true for each sequence ρ_i. Hence, all the jobs are scheduled on the 3 fastest machines.

Suppose j is the last job in the request sequence seen so far. Recall that the optimal load is j. Let l_i be the sum of the jobs on machine M_i and define the *relative load vector* to be $R = (R_0, R_1, R_2)$ where $R_i = l_i/j$. For brevity, we will use the term 'load vector' to mean 'relative load vector'.

Note that the load on machine M_i is $\alpha^i \cdot l_i$. Since the competitive ratio of A is less than α^3, $\alpha^i \cdot l_i \leq \alpha^r \cdot j$, i.e. $l_i/j \leq \alpha^{3-i}$, i.e. $R_i \leq \alpha^{3-i}$.

Thus the components of the load vector are bounded. Call a load vector *legal* if it satisfies $R_i \leq \alpha^{3-i}, 0 \leq i \leq 2$.

Consider the change in the load vector when the next job $j' = \alpha \cdot j$. arrives. Let $R' = (R'_0, R'_1, R'_2)$ be the new load vector. Suppose A schedules j' on machine M_k. Then $R'_i = R_i/\alpha, i \neq k$, and $R'_k = R_k/\alpha + 1$.

We will choose α such that, starting from the zero load vector, A will certainly reach a state where the load vector is not *legal*. This will contradict the assumption on the competitive ratio of A.

We discretize the space of load vectors and construct a directed graph as follows. Fix $\delta > 0$. We shall lower bound the load vector R by the load vector R' where $R'_i = \lfloor \frac{R_i}{\delta} \rfloor \cdot \delta$.

The vertices of the graph are 3-tuples $(c_0 \cdot \delta, c_1 \cdot \delta, c_2 \cdot \delta)$ where the c_i's are non-negative integers. The vertex set consists of all tuples which are legal load vectors. Since legal load vectors have bounded components, the vertex set is finite. Each vertex has 3 edges, an

edge corresponding to placing a job on one of the 3 fastest machines. In particular, for each k, $0 \leq k \leq 2$, we have an edge from the tuple $t = (c_0 \cdot \delta, c_1 \cdot \delta, c_2 \cdot \delta)$ to the tuple $\tau_k(t) = (c'_0 \cdot \delta, c'_1 \cdot \delta, c'_2 \cdot \delta)$ where $c'_i = \lfloor \frac{c_i}{\alpha} \rfloor$, $i \neq k$ and $c'_k = \lfloor \frac{c_k}{\alpha} + \frac{1}{\delta} \rfloor$. The edge $(t, \tau_k(t))$ is present only if $\tau_k(t)$ is a legal tuple.

We will consider the connected component of the graph containing the tuple (0,0,0). Call this $G(\alpha, \delta)$.

We shall say that a tuple $t = (t_0, t_1, t_2)$ is a lower bound for a load vector $R = (R_0, R_1, R_2)$ if $t_i \leq R_i$, $0 \leq i \leq 2$. Note that if R is legal then any lower bound tuple for R is also legal.

Lemma 3.2 *Let L be the load vector at any point and let L' be the load vector obtained by scheduling the next job on machine k. If tuple t is a lower bound for L, then the tuple $\tau_k(t)$ is a lower bound for L'.*

Consider the sequence of load vectors of A while servicing the job sequence. Let this sequence be $R_0, R_1, \ldots$, where $R_0 = (0, 0, 0)$. Construct a sequence of tuples $t_0, t_1, \ldots$, where $t_0 = (0, 0, 0)$ and $t_{i+1} = \tau_k(t_i)$ where R_{i+1} is obtained from R_i by scheduling the last job on machine M_k. Using Lemma 3.2, we can show inductively that tuple t_i is a lower bound for R_i. If A has a competitive ratio less than α^3 then each of the R_i's is legal. But this implies that the each of the t_i's is legal. The sequence of t_i's is an infinite path in the finite graph $G(\alpha, \delta)$. Hence $G(\alpha, \delta)$ must have a directed cycle.

We have verified by a computer program that for $\alpha = 1.3459$, $\delta = 0.0008$, $G(\alpha, \delta)$ has no cycle. Recall that α and n had to satisfy the inequality (1). For $\alpha = 1.3459$, this implies that $n \geq 8$. This proves a lower bound of $\alpha^3 \geq 2.4380$ on the competitive ratio of any deterministic online algorithm for load balancing with $n \geq 8$ related machines.

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 the loads on the fastest 2 machines over the last 3 jobs of the job sequence ρ_i as $i \to \infty$. We omit the details.

3.2 Randomized Lower Bound

Fix a constant m. We consider the distribution over the m job sequences $\rho_1, \ldots \rho_m$ where the sequence ρ_i, $1 \leq i \leq m$ is given with probability $\frac{1}{m}$. In other words, we give the job sequence ρ_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 ρ_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 ρ_m. Note that any schedule for ρ_m induces a schedule for ρ_i, $1 \leq i \leq m$. ¿From this, we can compute the expected load incurred by A for the chosen distribution of job sequences.

We compute all possible schedules for ρ_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, 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 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 Acknowledgements

We would like to thank Yossi Azar, Amos Fiat, Piotr Indyk and Rajeev Motwani for valuable discussions and encouragement to write this paper.

References

[1] S. Albers, *Better bounds for online scheduling*, to appear in Proc. 29th ACM STOC (1997).

[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.

[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] 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.

[6] P. Indyk, personal communication.

[7] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Technical Journal 45 (1966), pp. 1563-1581.