Resource Augmentation in Load Balancing*

Yossi Azar[†] Leah Epstein[‡] Rob van Stee[§]

June 25, 2000

Abstract

We consider load balancing in the following setting. The on-line algorithm is allowed to use n machines, whereas the optimal off-line algorithm is limited to m machines, for some fixed $m < n$. We show that while the greedy algorithm has a competitive ratio which decays linearly in the inverse of n/m, the best on-line algorithm has a ratio which decays exponentially in n/m. Specifically, we give a deterministic algorithm with competitive ratio of $1 + 2^{-\frac{n}{m}(1-o(1))}$, and a lower bound of $1 + e^{-\frac{n}{m}(1+o(1))}$ on the competitive ratio of any randomized algorithm.

We also consider the preemptive case. We show an on-line algorithm with a competitive ratio of $1 + e^{-\frac{n}{m}(1+o(1))}$. We show that the algorithm is optimal by proving a matching lower bound.

We also consider the non-preemptive model with temporary tasks. We prove that for $n = m + 1$, the greedy algorithm is optimal. (It is not optimal for permanent tasks.)

---

*A preliminary version of this paper appears in the proceedings of the 7th Biennial Scandinavian Workshop on Algorithm Theory, SWAT 2000.

[†]Dept. of Computer Science, Tel-Aviv University. E-Mail: azar@math.tau.ac.il. Research supported in part by the Israel Science Foundation and by the United States-Israel Binational Science Foundation (BSF).

[‡]Dept. of Computer Science, Tel-Aviv University. E-Mail: lea@math.tau.ac.il. Part of the research was done while this author was visiting the Centre for Mathematics and Computer Science (CWI), supported by a grant from the Netherlands Organization of Scientific Research.

[§]Centre for Mathematics and Computer Science (CWI). E-Mail: Rob.van.Stee@cwi.nl. Research supported by the Netherlands Organization for Scientific Research (NWO), project number SION 612-30-002.

1 Introduction

Competitive analysis has been criticized for being too pessimistic. It has also been observed that this worst case analysis sometimes fails to differentiate between algorithms of which the performance is observed empirically to be very different. A general method to circumvent these shortcomings was introduced by KALYANASUNDARAM and PRUHS [13]: resource augmentation. For certain scheduling problems with unbounded competitive ratio, they show that it is possible to attain a good competitive ratio if the machines of the on-line algorithm are slightly faster than the machines of the off-line algorithm.

Resource augmentation has been applied to a number of problems. It was already used in the paper where the competitive ratio was introduced [20]. In that paper, the performance of some paging algorithms was studied, where they have more memory than the optimal off-line algorithm.

In several machine scheduling and load balancing problems [4, 8, 13, 14, 16, 18], the effect of adding more or faster machines has been studied.

We consider the following load balancing problem. Jobs arrive on-line, where job j has a certain weight w_j. The job has to be assigned immediately to a machine, adding w_j to the machine's load. The on-line algorithm has n identical machines, and it is compared to an optimal offline algorithm which has $m < n$ identical machines.

For a job sequence σ we write $A_n(\sigma)$ for the maximum load of A on n machines when it is given this job sequence. Analogously, we write $OPT_m(\sigma)$. We denote the competitive ratio of an online algorithm A with n machines relative to an optimal offline algorithm with m machines by $c_{m,n}(A)$. Specifically,

$$c_{m,n}(A) = \max_{\sigma} \frac{A_n(\sigma)}{OPT_m(\sigma)}.$$

The classical case of $n = m$ was considered in a series of papers [11, 12, 3, 15, 1]. The best upper bound is 1.923 due to ALBERS [1] and the best lower bound is 1.853 [10] based on [1]. The case $n > m$ was introduced by BREHOB et al [5]. They showed that no matter how many machines the on-line algorithm has, it can never perform optimally: $c_{m,n}(A) > 1$ for all $n > m \geq 2$. However, one would expect that for any reasonable algorithm A, $c_{m,n}(A)$ will approach 1 when $t = n/m$ tends to infinity.. In fact, [5] showed that the greedy algorithm has a competitive ratio which approaches 1 in a rate depending linearly on $1/t$.

In contrast, we design an algorithm with a competitive ratio which approaches 1 in a rate depending *exponentially* on t. More specifically, we give an algorithm of competitive ratio $1 + \frac{1}{2^{t(1-o(1))}}$. Moreover, we show that the competitive ratio of any on-line algorithm cannot decrease faster than exponentially in t by proving a lower bound of $1 + \frac{1}{e^{t(1+o(1))}}$ on the competitive ratio of any on-line algorithm. We also show for $n = 2m$ a lower bound of 5/4.

We also consider the preemptive case. Here we view load as time. Each job may be assigned to one or more machines and time slots, where the time slots have to be disjoint. The assignment has to be determined completely at the arrival of a job. Using similar techniques as in [6, 7, 19] we prove a lower bound of $1/(1 - (\frac{m-1}{m})^n) = 1 + \frac{1}{e^{t(1+o(1))}}$ on the competitive ratio of any randomized preemptive algorithm. We also show a matching upper bound by adapting the optimal preemptive algorithm of [7] to our problem.

We can also view time as a separate axis and not as the load axis. Here jobs arrive and depart at arbitrary times and the cost of an algorithm is the maximum load over time and machines. This model is called the temporary tasks model (the case where jobs never leave is called the permanent tasks model). It was proved in [2] that for $n = m$ the greedy algorithm, which is $(2 - 1/m)$-competitive, is optimal for this model. We show that if n is just slightly larger than m, i.e., $n = m + 1$, then the greedy algorithm which is $(2 - 2/(m + 1))$-competitive is also optimal. Note that the algorithm of [1], which implies that the greedy algorithm is not optimal for $n = m$ for permanent tasks, also implies that the greedy algorithm is not optimal for $n > m$ for permanent tasks.

2 Permanent tasks

In this section we investigate the growth of the competitive ratio as a function of $t = n/m$. We start with the competitive ratio of the greedy algorithm. This algorithm was first given by Graham [11], and assigns each new job to the least loaded machine. The following lemma is shown in [5] using a similar analysis as in [11]:

Lemma 1 *The competitive ratio of the greedy algorithm is* $1 + \frac{m-1}{n}$.

The above theorem implies a competitive ratio which is a linear function in $1/t$. Surprisingly, we can give an algorithm called *Buckets* which has a competitive ratio $1 + 2^{-t(1-o(1))}$.

2.1 Algorithm Buckets

For describing the algorithm *Buckets* we assume that $t > 3$. (If $t \leq 3$ we use the greedy algorithm.) Let $0 < \varepsilon < 1$ be some parameter to be fixed later. We partition all machines into buckets: $k = \lfloor t - \frac{2}{\varepsilon} \rfloor$ small buckets, each of which contains m machines, and one big bucket that contains all other machines. Note that the big bucket contains at least $\frac{2m}{\varepsilon}$ machines.

Algorithm Buckets maintains a value λ. Denote by λ_i the value of λ after the arrival of i jobs and by OPT_i the optimal load after i jobs. The algorithm consists of phases. During phase j, the algorithm can use only the big bucket and the small bucket number $j \bmod k$. We assign the first job to the first small bucket and initialize $\lambda_1 = w_1$. We modify λ only when a new phase starts while keeping the following two invariants on λ:

- $\max_{j \leq i} w_j \leq \lambda_i$
- $(2 - \varepsilon) OPT_i \geq \lambda_i$

On arrival of a job i (starting from $i = 2$), we do the following.

- If $w_i \leq \lambda_{i-1}/2$ assign i greedily to the least loaded machine in the big bucket.
- If $\lambda_{i-1}/2 < w_i \leq \lambda_{i-1}$, and there is a machine in the current small bucket which was not used in the current phase, assign i to this machine.
- Finally, if all m machines in the current small bucket were used in the current phase, or if $w_i > \lambda_{i-1}$, then a new phase begins: we define $\lambda_i = \max((2 - \varepsilon)\lambda_{i-1}, w_i)$ and the job is assigned to a machine in the next small bucket.

Theorem 1 *The algorithm Buckets is* $(1 + \frac{1}{2^{t(1-o(1))}})$*-competitive for an appropriate choice of* ε.

Proof. We start by showing that both invariants hold after the arrival of a job (and thus hold throughout the execution of Buckets). After the assignment of the first job, $\lambda_1 = OPT_1 = w_1$, and both invariants hold since $\varepsilon < 1$.

The first invariant always holds, since when a job which is larger than λ arrives, λ is modified. To show that the second invariant holds, we show that λ is increased only if the previous λ is smaller than the current OPT, and that λ is not increased too much. If λ is increased because $\lambda_{i-1} < w_i$, then $OPT_i \geq w_i$ and since $\lambda_i = \max((2-\varepsilon)\lambda_{i-1}, w_i)$ then $\lambda_i \leq (2-\varepsilon)w_i \leq (2-\varepsilon)OPT_i$. If λ is increased because all the machines in the small bucket were used in the current phase, then there are at least $m+1$ jobs of weight more than $\frac{\lambda_{i-1}}{2}$ and hence the optimal schedule has to assign two of them on one machine, yielding $OPT_i > \lambda_{i-1}$. Thus $\lambda_i \leq (2-\varepsilon)OPT_i$.

Next we show that the maximum load in the big bucket never exceeds OPT_i at step i (after arrival of job i). It is easy to see that the maximum load of running the greedy algorithm on αm machines is at most $\frac{OPT_i}{\alpha} + \max_{j \leq i} w_j$. Since $w_j \leq \frac{\lambda_{i-1}}{2}$ and $\lambda_{i-1}/(2-\varepsilon) \leq OPT_{i-1}$, the load is bounded by $(\frac{1}{\alpha} + \frac{2-\varepsilon}{2})OPT_{i-1} \leq (\frac{\varepsilon}{2} + \frac{2-\varepsilon}{2})OPT_i = OPT_i$.

Last, we bound the maximum load on the small bucket machines. When a new phase starts, the value of λ is multiplied by at least $2-\varepsilon$. Each machine in a small bucket is used at most once in each phase.

Consider a machine in a small bucket. Denote the last job assigned to this machine by i', and let $\lambda' = \lambda_{i'}$. The previous job assigned to the same machine is of weight at most $\lambda'/(2-\varepsilon)^k$. Moreover, a job that was assigned $r \geq 1$ jobs before i' to the same machine is of weight at most $\lambda'/(2-\varepsilon)^{rk}$. Thus the total weight of all jobs on this machine, except i', is at most $2\lambda'/(2-\varepsilon)^k$. Since $OPT \geq \frac{1}{(2-\varepsilon)}\lambda'$ we get that the total weight of jobs on this machine is at most

$$w(j') + \frac{4OPT}{(2-\varepsilon)^k} \leq (1 + \frac{4}{(2-\varepsilon)^k})OPT \leq (1 + \frac{4}{(2-\varepsilon)^{t-2/\varepsilon-1}})OPT.$$

Choosing an appropriate value of ε (for example $\varepsilon = \sqrt{3/t}$) yields a bound of $1 + 2^{-t+O(\sqrt{t})} = 1 + 2^{-t(1-o(1))}$ as required. □

2.2 Lower bounds

We begin by giving a simple exponential lower bound:

Theorem 2 *The competitive ratio of any deterministic on-line algorithm is at least* $1 + 2^{-2t+1}$.

Proof. Consider an on-line algorithm A. We give a proof for even m and for integer t. It is easy to extend the proof for all cases. The sequence consists of $n + \frac{m}{2}$ jobs that arrive in $2t+1$ phases. Phase 1 consists of $\frac{m}{2}$ unit jobs, and phase i for $i > 1$ consists of $\frac{m}{2}$ jobs of weight 2^{i-2}. The sequence stops after a phase in which A schedules two jobs on one machine. (If A reaches the last phase, there are more jobs than on-line machines, therefore A has two jobs on one machine). The optimal off-line load after every phase is the weight of the last job. If A has two jobs on one machine, its load it at least $1+x$ where x is the weight of the last job. The minimum value of $\frac{1+x}{x}$ is $1+\frac{1}{2^{i-2}}$ where $i=2t+1$, hence $1+2^{-2t+1}$ is a lower bound on the competitive ratio. □

We can give a slightly better lower bound. This bound holds for deterministic and randomized algorithms. In fact, we show a lower bound on preemptive algorithms versus a non-preemptive optimal algorithm. Hence our lower bound holds both for the preemptive and non-preemptive models. The lower bound builds on the lower bounds given by Sgall [19] and independently by Chen, van Vliet and Woeginger [6, 7].

The main idea here is to use small jobs and a sequence of n big jobs J_i for $1 \leq i \leq n$ of increasing weight so that the optimal off-line load after job J_i, which we denote by OPT_i, is exactly equal to the weight of J_i. Hence, the weight of each big job is equal to the total weight of all previous jobs divided by $m-1$. Specifically, the sequence begins by very small jobs of total weight $m-1$ followed by n big jobs. The weight of J_i for $1 \leq i \leq n$ is μ^{i-1} where $\mu = \frac{m}{m-1}$.

Lemma 2 *The optimal off-line load for the above sequence is μ^{k-1} after the arrival of the job J_k, for $1 \leq k \leq n$.*

Proof. We consider an algorithm which assigns all jobs on off-line machines, and show that the resulting load is μ^{k-1}.

The algorithm assigns jobs to the off-line machines greedily, in non-increasing order (sorted according to weight). This is equivalent to using the LPT rule. We show that the loads never exceed μ^{k-1} after the assignment of a big job. Note that the total weight of all small jobs and the first j big jobs is $\mu^j(m-1) = \mu^{j-1}m$.

Assume that the assignment of job j causes the maximum load to exceed μ^{k-1}. This means that all other machines are loaded by more than $\mu^{k-1} - \mu^{j-1}$. Since the total weight of jobs smaller or equal to J_j is $\mu^{j-1}m$, we get that the total weight of jobs is more than $\mu^{k-1}m$, which is a contradiction. Hence, the assignment of the small jobs results in balanced machines, each with load of μ^{k-1}. □

The following lemma, adapted from [19, 9], is the key of lower bounding the competitive ratio.

Lemma 3 *For any deterministic or randomized, preemptive or non-preemptive algorithm for the sequence above the following holds: $r \geq \frac{W}{\sum_{i=1}^{n} OPT_i}$, where*

r is the competitive ratio and W is the total weight of the jobs.

Proof. Denote by $A(J_i)$ the maximum load of the on-line algorithm A after the assignment of the job J_i. Then

$$\frac{\sum_{i=1}^n E(A(J_i))}{\sum_{i=1}^n OPT_i} \leq \frac{\sum_{i=1}^n r \cdot OPT_i}{\sum_{i=1}^n OPT_i} = r.$$

Hence it is enough to show that $\sum_{i=1}^n E(A(J_i)) \geq W$.

Assume that A is deterministic. For $1 \leq l \leq n$, let T_l be the load on the l^{th} machine at the end of the sequence, after sorting the machines by non-increasing load. Removing any $l-1$ jobs still leaves a machine with a load of at least T_l (even if the schedule is preemptive), and thus $A(J_l) \geq T_{n-l+1}$. Since $W = \sum_{i=1}^n T_i$ we conclude that

$$\sum_{i=1}^n A(J_i) \geq \sum_{i=1}^n T_{n-l+1} = W$$

as needed. If A is randomized, we average over deterministic algorithms and conclude again that $\sum_{i=1}^n E(A(J_i)) \geq W$. □

Theorem 3 *The competitive ratio of an on-line algorithm, deterministic or randomized, preemptive or non-preemptive, is at least $1/(1-(\frac{m-1}{m})^n) = 1 + e^{-\frac{n}{m}(1+o(1))}$.*

Proof. We use the above job sequence and apply Lemma 3. We have

$$W = \mu^n(m-1)\ ,$$

$$\sum_{i=1}^n OPT_i = \sum_{i=1}^n \mu^{i-1} = \frac{\mu^n - 1}{\mu - 1}$$

and

$$r \geq \frac{\mu^n(m-1)}{(\mu^n - 1)}(\mu - 1) = \frac{\mu^n}{\mu^n - 1} = \frac{1}{1 - \frac{1}{\mu^n}} = \frac{1}{1 - (\frac{m-1}{m})^n}$$

as needed. □

We can improve the bound for the special case $t = 2$ for the non-preemptive deterministic case.

Claim 1 *The competitive ratio of any non-preemptive deterministic on-line algorithm with $n = 2m$, where $m \geq 8$, is at least $\frac{5}{4}$.*

m mod 6	0	1	2	3	4	5
Number of jobs	$2m+2$	$2m+1$	$2m+1$	$2m+1$	$2m+1$	$2m+1$

Proof. We use a job sequence consisting of four phases:

- m jobs of weight 1
- $\lfloor \frac{m}{2} \rfloor$ jobs of weight 3/2
- $\lfloor \frac{m}{3} \rfloor + 1$ jobs of weight 3
- $\lfloor \frac{m+1}{6} \rfloor + 1$ jobs of weight 4.

The sequence stops after a phase in which the on-line algorithm schedules two jobs on one machine. Note that the sequence contains more than $2m$ jobs.

We show that the optimal load in phase i is i. This is clear for phases 1 and 2. In phase 3, if the machines are packed to a maximum load of 3, at most 2.5 of space can be lost: 2 if a job of weight 1 has to be assigned to its own machine, and 0.5 if there is an odd number of jobs of weight 1.5. The total weight is at most $m + \frac{3m}{4} + (m+3) = \frac{11m}{4} + 3$, which is at most $3m - 2.5$ for $m \geq 22$. This implies that the machines can be packed with a maximum load of 3 for $m \geq 22$. By inspection, the machines can be packed for $8 \leq m \leq 21$ too.

In phase 4, the total weight is at most $\frac{11m}{4} + 3 + \frac{4m}{6} + \frac{14}{3} = \frac{41}{12}m + \frac{23}{3}$. In the optimal packing, at most 3.5 of space is lost. We have $\frac{41}{12}m + \frac{23}{3} \leq 4m - 3.5$ for $m \geq 20$. Therefore the optimal algorithm can maintain a load of 4 in phase 4, if $m \geq 20$. By inspection, the machines can be packed for $8 \leq m \leq 19$ as well.

As an example, we give the optimal schedules for phases 3 and 4 when $m = 8$ and $m = 9$ (see Figure 1).

Depending on the phase in which the on-line algorithm puts two jobs on the same machine, we find competitive ratios of $2, \frac{5}{4}, \frac{4}{3}$ and $\frac{5}{4}$. Hence the competitive ratio is at least 5/4. □

2.3 An optimal preemptive algorithm

The last part of this section presents an optimal preemptive on-line algorithm. The algorithm is similar to the algorithm in [7].

Let $r = 1/(1 - \frac{1}{\mu^n})$. We denote the load on machine i at time T by L_i^T. The algorithm maintains three invariants, which hold at any step T:

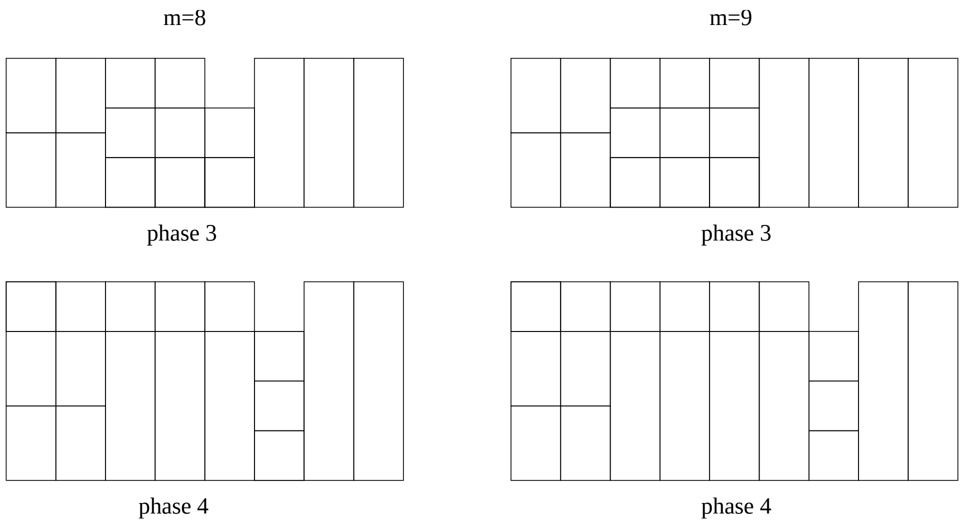

Figure 1: The last phases for $m = 8, 9$

- $L_1^T \le L_2^T \le \ldots \le L_n^T$.
- $L_n^T \le r \cdot OPT^T$.
- For $1 \le k \le n$,
$$\sum_{i=1}^{k} L_i^T \le \frac{\mu^k - 1}{\mu^n - 1} W^T,$$
where W^T is the total weight of jobs which arrived till time T.

Similar to the algorithm in [7], we try to maintain a ratio of $\frac{m}{m-1}$ between machine loads. We show how to assign a new job j with weight w_j, arriving at time $T+1$, to n machines. First, compute the new optimal load, which is $\max(W^{T+1}/m, \max_{1\le i\le T+1} w_i)$ [17], and then reserve the following intervals for j: for $1 \le l \le n-1$, reserve $[L_l^T, L_{l+1}^T]$, and for $l = n$, reserve $[L_n^T, r \cdot OPT^T]$. Note that these intervals are disjoint. Next, for $j = n$ down to 1, assign a portion out of w_j of size equal to the size of the reserved interval. Continue assigning portions until job j is completely assigned. (The last portion assigned might be smaller than the interval.)

It is easy to follow the proof in [7], replacing the number of machines used by the on-line algorithm from m to n. The proof shows that each job is completely distributed to the machines and that the invariants hold. By that we conclude that the algorithm is r-competitive as required.

3 Temporary tasks

Recall that for $n = m$ the greedy algorithm is $(2 - 1/m)$-competitive for permanent tasks as well as for temporary tasks. The greedy algorithm is not optimal for permanent tasks, but is optimal for temporary tasks. Also for $n > m$, it is easy to see that the greedy algorithm has the same competitive ratio for temporary tasks as for permanent tasks, which is $1 + (m-1)/n$. However, in contrast to the case $n = m$, the greedy algorithm is not optimal for temporary tasks, since algorithm Buckets (defined on temporary tasks) achieves a better competitive ratio for large n.

Claim 2 *For temporary tasks, the algorithm Buckets is* $(1+\frac{1}{2^{t(1-o(1))}})$*-competitive for an appropriate choice of* ε.

Proof. The analysis of Buckets for both the big bucket and the small buckets in the proof of Theorem 1 also holds for temporary tasks. □

However, we show that if the online algorithm has one more machine than the optimal offline algorithm, then the greedy algorithm is still optimal.

Theorem 4 *The greedy algorithm is optimal for temporary tasks for* $n = m + 1$.

Proof. We need to show a lower bound of $\frac{2m}{m+1}$ on the competitive ratio of any on-line algorithm. The proof consists of two parts: one for odd m and one for even m. In the proof we mention the value of the optimal load only when the value increases.

Case A. m is odd. We start the sequence with $(m-1)m^2$ unit-weight jobs. The optimal load is $m(m-1)$. We distinguish between two cases:

Case A1. The online algorithm places at least $m(m-1)$ jobs on one machine, say machine x.

In this case, all the jobs leave except $m(m-1)$ jobs on x. Then, $m(m-1)$ jobs of weight $m-1$ arrive. Since the optimal load is again $m(m-1)$, at most $m-2$ of them can be assigned to x. Otherwise the load would be $(2m-1)(m-1)$ on x, and $\frac{(2m-1)(m-1)}{m(m-1)} > \frac{2m}{m+1}$. So $(m-1)^2+1$ of these jobs must be assigned to the m empty machines. We distinguish between two sub-cases:

Case A1a. One machine (not x) has at least m jobs of weight $m-1$.

All jobs of weight $m-1$ leave except m jobs of weight $m-1$ on one machine, and $m-1$ jobs of weight $m(m-1)$ arrive. The new optimal load is $(m+1)(m-1)$. Therefore all these jobs must be assigned to different machines. Finally, a job of weight $m(m+1)$ arrives. The optimal load is $m(m+1)$: the last job has it own machine, the other machines have one job of weight $m(m-1)$, one or two jobs of weight $m-1$ and some jobs of weight 1, so that the load is precisely $m(m+1)$. This completes the proof, since the online load is $2m^2$.

Case A1b. All machines (except machine x) have at least one job of weight $m-1$.

All jobs of weight $m-1$ leave except m jobs, one such job remains on each machine except on machine x. Next, $\frac{m^2-2m-1}{2}$ jobs of weight $2(m-1)$ arrive. The optimal load is again $m(m-1)$. At most $\frac{m-3}{2}$ jobs are assigned to machine x, otherwise the load there becomes too large. There are $\frac{m-3}{2}+\frac{1}{m}$ jobs on average on the other machines, so there is at least one machine (not x) with at least $\frac{m-1}{2}$ jobs of this weight and a load of at least $m(m-1)$, say machine y. All jobs leave except the unit jobs on x and jobs of total weight precisely $m(m-1)$ on machine y.

Finally, $m-1$ jobs of weight $m(m-1)$ arrive and one job of weight $m(m+1)$. Clearly, the online algorithm must assign each job of weight $m(m-1)$ to an empty machine and hence its final load is $2m^2$. The optimal algorithm can balance its jobs and have a load of $m(m+1)$ since there are at least $2(m-1)$ jobs of weight 1, which completes the proof.

Case A2. All machines now have load at least $m-1$.

All jobs leave except $m-1$ jobs on each machine, and m^2-m-1 jobs of weight $m-1$ arrive. The average number of jobs of weight $m-1$ on the machines is $m-2+\frac{1}{m+1}$, and hence there is a machine with $m-1$ jobs of weight $m-1$ and a load of $m(m-1)$. By letting some jobs leave, we can get the loads to be the same as in Case A1b just before the arrival of the jobs of weight $2(m-1)$. Hence, we can continue as in that case.

Case B. m is even. We start the sequence with $(m-1)m^2$ unit jobs. The optimal load is $m(m-1)$. We distinguish between two cases:

Case B1. One machine, say x, has at least $m(m-1)$ jobs. All jobs leave except $m(m-1)$ jobs on x, and $(m-1)^2$ jobs of weight m arrive. The optimal load is again $m(m-1)$. At most $m-2$ jobs can be assigned to x. We distinguish between two sub-cases:

Case B1a. Another machine (not x) has load at least $m(m-1)$. Then all jobs of weight m leave except $m-1$ jobs on that machine, and $m-1$ jobs of weight $m(m-1)$ arrive followed by a job of weight $m(m+1)$. Clearly, the online load is at least $2m^2$, while the optimal load is $m(m+1)$ which completes the proof.

Case B1b. Each machine except x has one job of weight m. All jobs of weight m leave except m jobs, one remains on each machine except on machine x. Next $\frac{m^2-3m}{2}$ jobs of weight $2m$ arrive. At most $\frac{m-2}{2}$ can be assigned to machine x. Hence, the average number of jobs of weight $2m$ on all machines besides x is $\frac{m}{2}-2+\frac{1}{m}$. Thus, one machine y must have $\frac{m}{2}-1$ jobs of weight $2m$ and a load of at least $m(m-1)$. All jobs leave except the unit jobs on x and jobs of total weight $m(m-1)$ on y. Finally, $m-1$ jobs of weight $m(m-1)$ arrive and one job of weight $m(m+1)$. Again, the online load is at least $2m^2$, while the optimal load is $m(m+1)$ which completes the proof.

Case B2. There are at least m jobs on each machine. (This can happen only for $m \geq 4$).

All jobs leave except m jobs on each machine. Next, $\frac{m^2(m-2)-m}{2}$ jobs of weight 2 arrive. If there is a machine with load at least $m(m-1)$, we continue as in Case B1. Otherwise, each machine has load at least $2m$. Then, some jobs of weight 2 leave in such a way that the load on each machine is $2(m-1)$. Next, m^2-2m-2 jobs of weight $m-1$ arrive and one machine will have a load of at least $m(m-1)$. Jobs of weight $m-1$ on that machine leave such that the load becomes $m(m-1)$. All non-unit jobs on the other machines leave. We continue as in Case B1b. □

4 Conclusions

We have examined the effects of resource augmentation for several load balancing problems. For the problem of scheduling jobs on identical machines, we have shown an algorithm with a competitive ratio which decreases exponentially in n/m, while greedy has a competitive ratio that is linear in n/m.

An open question is whether it is possible to close the gap between the lower bound and the upper bound on identical machines. Both bounds are decreasing exponentially, and we conjecture that the true value of the competitive ratio is closer to the lower bound.

5 Acknowledgments

The authors wish to thank Han La Poutré for helpful discussions.

References

[1] S. Albers. Better bounds for on-line scheduling. In *Proc. 29th ACM Symp. on Theory of Computing*, pages 130–139, 1997.

[2] Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *5th Israeli Symp. on Theory of Computing and Systems*, pages 119–125, 1997.

[3] 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.*

[4] P. Berman and C. Coulston. Speed is more powerful than clairvoyance. In *Nordic Journal of Computing*, pages 181–193, 1999.

[5] M. Brehob, E. Torng, and P. Uthaisombut. Applying extra-resource analysis to load balancing. Manuscript, 1999.

[6] B. Chen, A. van Vliet, and G. J. Woeginger. Lower bounds for randomized online scheduling. *Information Processing Letters*, 51:219–222, 1994.

[7] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18:127–131, 1995.

[8] J. Edmonds. Scheduling in the dark. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 179–188, 1999.

[9] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *To appear in Oper. Res. Lett.*, 2000.

[10] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, 2000.

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

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

[13] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *Proceedings of 36th IEEE Symposium on Foundations of Computer Science*, pages 214–221, 1995.

[14] Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. In *European Symposium on Algorithms*, pages 235–246, 1998.

[15] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 132–140, 1994.

[16] Tak Wah Lam and Kar Keung To. Trade-offs between speed and processor in hard-deadline scheduling. In *ACM/SIAM Symposium on Discrete Algorithms*, pages 623–632, 1999.

[17] R. McNaughton. Scheduling with deadlines and loss functions. *Management Sci.*, 6:1–12, 1959.

[18] Cynthia A. Philips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.

[19] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Inf. Process. Lett.*, 63(1):51–55, 1997.

[20] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.