Theoretical Computer Science 130 (1994) 73–84
Elsevier

On-line load balancing*

Yossi Azar**
Computer Science Department, Tel Aviv University, Tel Aviv 69978, Israel

Andrei Z. Broder and Anna R. Karlin
Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA

Abstract

Azar, Y., A.Z. Broder and A.R. Karlin, On-line load balancing, Theoretical Computer Science 130 (1994) 73–84.

The setup for our problem consists of n servers that must complete a set of tasks. Each task can be handled only by a *subset* of the servers, requires a different level of service, and once assigned cannot be reassigned. We make the natural assumption that the level of service is known at arrival time, but that the duration of service is not. The on-line load balancing problem is to assign each task to an appropriate server in such a way that the maximum load on the servers is minimized. In this paper we derive matching upper and lower bounds for the competitive ratio of the on-line greedy algorithm for this problem, namely, $[(3n)^{2/3}/2](1+o(1))$, and derive a lower bound, $\Omega(n^{1/2})$, for any other deterministic or randomized on-line algorithm.

1. Introduction

Consider an idealized local area network that links multimedia workstations, computers, I/O devices, etc. Each device is directly connected to one or more gateways (bridges) to the net. Communication tasks arrive and disappear at arbitrary times. Upon arrival, each task requests a certain guaranteed bandwidth (e.g. low for file transfers, medium for graphic applications, high for video) and must be assigned to one of the gateways directly connected to the device, for the duration of the task. Service must begin immediately. The problem is to assign each task in such a way that no bridge is overloaded.

Correspondence to: A. Broder, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA. Email addresses of the authors: azar@cs.stanford.edu, broder@src.dec.com and karlin@src.dec.com.

* A preliminary version of this paper appeared in the *Proc. 33rd Symp. on Foundations of Computer Science*, 1992.

** Part of this work was done while the author was at Digital Systems Research Center, Palo Alto.

0304-3975/94/$07.00
SSDI 0304-3975(93)E0128-O

The above is an example of the following general problem. Consider n servers that must complete a set of tasks. Each task can be handled only by a *subset* of the servers, and requires a different level of service, called the *weight* of the task. Tasks arrive and depart one by one. Service must start as soon as the task arrives. A task, once assigned, cannot be reassigned. At assignment time only the weight is known, but not the duration of service. The on-line load balancing problem is to assign each task to an appropriate server in such a way that the maximum load on the servers is minimized.

As usual nowadays, we evaluate the performance of the on-line algorithm by its competitive ratio, which is the ratio between its maximum load and the maximum load of the optimal off-line algorithm for the worst-case sequence of tasks.

Previous analyses (e.g. [2–4, 10]) were restricted to models where either the tasks continued forever, or where the tasks can be delayed for arbitrarily long periods of time; thus, forsaking real-time service.

If the arrival and departure times are known in advance, the servers are called machines, and the tasks are called jobs, the question becomes an instance of the well-known class of static load balancing problems, studied in a myriad of variants. (See [6] and references therein.)

There are two independent dichotomies that characterize the tasks, and this leads to four possible load balancing problems. The first is according to how long a task runs: We call tasks which start at arbitrary times but continue forever *permanent*, while tasks that begin and end are said to be *temporary*. The second is according to the set of servers on which a task can run. If a task can be assigned to any server, we say it is *unrestricted*, whereas if it can only be assigned to a proper subset of the servers, we say it is *restricted*. Three of the four possible problems were considered in the past. Here we address the fourth one.

Graham [5] showed that for permanent, unrestricted tasks, the greedy algorithm achieves a competitive ratio of $2-1/n$. (There is a recent improvement of this result due to Bartal et al. [3] to $2-\varepsilon$, using a more complicated algorithm.) It is straightforward to check that Graham's analysis of the greedy algorithm extends to temporary tasks and that the competitive ratio is still $2-1/n$.

For permanent, restricted tasks, Azar et al. [2] showed that the competitive ratio of the greedy algorithm is precisely $\log n$ and that no algorithm can do better. Although the analysis of [2] does not generalize to temporary tasks, it is natural to ask whether the $\log n$ competitive ratio holds for restricted temporary tasks, the way the $2-1/n$ competitive ratio holds for unrestricted tasks, both permanent and temporary.

We answer this question negatively. More precisely, we obtain matching upper and lower bounds show that the competitive ratio of the greedy algorithm for restricted temporary tasks is a surprisingly high $[(3n)^{2/3}/2](1+o(1))$. (The function implied by $o(1)$ is different for the upper and lower bound, but the leading term is exactly the same!) Moreover, we show a lower bound of $\Omega(n^{1/2})$ for any deterministic or randomized algorithm.

A variation on the model described so far is when the weight of a task can be split among its allowed servers. For this case we show that the competitive ratio for the

natural, "water-level", algorithm is also $\Theta(n^{2/3})$, and for any other algorithm (deterministic or randomized) the lower bound is again $\Omega(n^{1/2})$.

Recently, our lower bound of $\Omega(n^{1/2})$ was matched by an algorithm of Azar et al. [1]; and a related problem, where preemptive rescheduling is allowed was considered by Philips and Westbrook [9], who give algorithms that exhibit different trade-offs between the maximum on-line load and the number of preemptions.

Our problems can be recast in the dynamic-graph framework. Previous results on on-line algorithms for dynamic graphs (e.g. graph coloring [8, 11], on-line matching [7]) allowed only vertex additions. Our results pertain to the more challenging fully dynamic case, where arbitrarily long sequences of vertex additions and deletions are allowed. However, since the tasks-on-servers framework is more natural, this is the one we adopt here.

1.1. Formal definition of the problem

Let M be a set of servers (or machines) that is supposed to run a set of tasks that arrive and depart in time. Each task j has associated to it a weight, or load, $w(j) \geq 0$, an arrival time $\tau(j)$, and a set $M(j) \subset M$ of servers capable of running it.

As soon as it arrives, each task must be assigned to exactly one of the servers capable of running it, and once assigned, it cannot be transferred to a different server. The assigned server starts to run the task immediately, and continues to run it until the task departs.

When task j arrives, an *assignment algorithm* must select a server $i \in M(j)$, and assign task j to it.

The *load* on server i at time t, denoted $L_i(t)$, is the sum of the weights of all the tasks running on server i at time t.

Let σ be a sequence of task arrivals and departures, and let $|\sigma|$ be the time of the last arrival. Then the cost $C_A(\sigma)$ of an assignment algorithm A on the sequence σ, is defined as

$$C_A(\sigma) = \max_{0 \leq t \leq |\sigma|;\, i \in M} L_i(t)$$

An *on-line assignment algorithm* must assign an arriving task j at time $\tau(j)$ to a server in $M(j)$ knowing only $w(j)$, $M(j)$, the current state of the servers, and the past – the decision is made without any knowledge about future arrivals or departures. An *optimal off-line assignment algorithm*, denoted OPT, assigns arriving tasks knowing the entire sequence of task arrivals and departures and does so in a way that minimizes its cost.

The competitive ratio of an on-line algorithm A is defined as the supremum over sequences σ of $C_A(\sigma)/C_{\text{OPT}}(\sigma)$. Our goal is to find an on-line assignment algorithms with minimum competitive ratio.

Note that computing the optimal off-line solution is NP-complete even when all the tasks are permanent, unrestricted, and there are only two processors. (An easy reduction from PARTITION.)

1.2. Notation

Before plunging into proofs, we summarize the notations we already used, or plan to use soon.

- M is the set of all servers (machines). $|M|=n$.
- Task (job) j has a weight $w(j)$, an arrival time $\tau(j)$, and a set $M(j)\subset M$ of servers capable of running it.
- $T_i(t)$ is the set of tasks run (by the on-line algorithm under consideration) on server i at time t, and $T(t)=\bigcup_{1\leq i\leq n} T_i(t)$.
- $L_i(t)$ is the load on server i at time t, i.e. $L_i(t)=\sum_{j\in T_i(t)} w(j)$.
- $m(j)\in M(j)$ is the server on which OPT (the optimum off-line algorithm) runs task j. Thus, for $S\subset M$, $m^{-1}(S)$ is the set of tasks assigned by OPT to servers in S.
- $T_i^S(t)$, for $i\in M$ and $S\subset M$, is defined at $T_i(t)\backslash m^{-1}(S)$, i.e. $T_i^S(t)$ is the set of tasks being run on server i except for those tasks being run by OPT on servers in S.
- $L_i^S(t)=\sum_{j\in T_i^S(t)} w(j)$.

2. Upper bounds

The greedy algorithm is formally defined as follows:

Algorithm GREEDY: Upon arrival of a task j assign it to the server in $M(j)$ with the current minimum load (ties are broken arbitrarily).

Lemma 2.1. *Consider an execution of the algorithm* GREEDY. *Suppose that the optimal off-line cost is λ. If there is a server (without loss of generality, server 1), and a time t such that $L_1(t)=\mu$, then for every $k\geq 1$, such that*

$$\left(\binom{k+1}{2}+k\right)\lambda<\mu,$$

there exists a time $t_k\leq t$, and a set of k servers S (without loss of generality, $S=\{1,2,\ldots,k\}$) such that

$$\begin{aligned}
&L_1^S(t_k)\geq \mu-(1+k)\lambda,\\
&L_2^S(t_k)\geq \mu-((1+2)+k)\lambda,\\
&L_3^S(t_k)\geq \mu-((1+2+3)+k)\lambda,\\
&\qquad\vdots\\
&L_k^S(t_k)\geq \mu-\left(\binom{k+1}{2}+k\right)\lambda.
\end{aligned}$$

Proof. The proof is by induction on k. The base case is trivial: Take $t_1=t$. Since $S=\{1\}$, $L_1^S(t)\geq\mu-\lambda$. (The total load that may be excluded is the load that OPT has on server 1, which is at most λ.)

For the induction step, assume that the hypothesis holds for k. Without loss of generality, assume that $S=\{1,2,\ldots,k\}$. Consider the set of tasks $T=\bigcup_{i\in S} T_i^S(t_k)$. Let $t_{k+1}=\max_{j\in T}\tau(j)$. Let $j^*\in T$ be the task that started at time t_{k+1}. Note that the servers in S may work at time t_{k+1} on tasks that are no longer present at time t_k, but that all the tasks in T are present at time t_{k+1}.

By definition of T_i^S, the server $m(j^*)\notin S$. Without loss of generality, assume that $m(j^*)=k+1$. Therefore, at time t_{k+1}, GREEDY could have placed task j^* on server $m(j^*)$, but did not. Hence,

$$\begin{aligned} L_{k+1}(t_{k+1}) &\geqslant \min_{1\leqslant i\leqslant k}(L_i^S(t_k))-w(j^*) \\ &\geqslant \mu-\left(\binom{k+1}{2}+k\right)\lambda-w(j^*) \\ &\geqslant \mu-\binom{k+2}{2}\lambda, \end{aligned}$$

since $w(j^*)\leqslant\lambda$. Similarly, for any set of servers S, we have $\sum_{j\in m^{-1}(S)} w(j)\leqslant|S|\cdot\lambda$, thus

$$L_{k+1}^{S\cup\{k+1\}}(t_{k+1})\geqslant\mu-\binom{k+2}{2}\lambda-(k+1)\lambda.$$

Finally, for $1\leqslant i\leqslant k$,

$$\begin{aligned} L_i^{S\cup\{k+1\}}(t_{k+1}) &\geqslant L_i^S(t_k)-\lambda \\ &\geqslant \mu-\left(\binom{i+1}{2}+(k+1)\right)\lambda, \end{aligned}$$

since every task in $T_i(t_k)$ contributes to $T_i(t_{k+1})$ with the possible exception of j^*. □

Theorem 2.2. *For any sequence σ of task arrivals and departures, the greedy on-line assignment algorithm is $[(3n)^{2/3}/2](1+o(1))$ competitive.*

Proof. By Lemma 2.1, if the maximum load GREEDY ever has on some server is μ and the optimal off-line algorithm reaches a maximum load of λ, then there is a time when the sum of the loads on all the servers is at least

$$k\mu-\left(k^2+\sum_{1\leqslant i\leqslant k}\binom{i+1}{2}\right)\lambda$$

for any k such that

$$\left(\binom{k+1}{2}+k\right)\lambda<\mu.$$

But by hypothesis, the sum of all loads is at most $n\lambda$. Taking k to be $(\sqrt{2\mu/\lambda}-2)$ completes the proof. □

3. Lower bounds

3.1. Lower bound for the greedy algorithm

Theorem 3.1. *The competitive ratio of the greedy on-line assignment algorithm is at least*

$$\frac{(3n)^{2/3}}{2}(1+o(1)).$$

Proof. We assume that all tasks have unit weight, and allow GREEDY to break ties in any deterministic way.

Let $S(t)=(S_1(t), S_2(t),\ldots, S_n(t))$ be the sorted (nonincreasing) n-tuple of the loads that GREEDY has on servers at time t, and let $s_i(t)$ be the server whose load is $S_i(t)$. Write $S(t)>S(t')$ if the tuple $S(t)$ is lexicographically greater than $S(t')$.

We prove the lower bound by showing that there is a finite sequence of requests σ for which the following three conditions hold:

(1) For all j such that $S_j(|\sigma|)>0$, $S_{j-1}(|\sigma|)-S_j(|\sigma|)\geqslant j-1$.

(2) There are n tasks running at time $|\sigma|$.

(3) The maximum off-line load at all times t, $0\leqslant t\leqslant|\sigma|$, is 1.

If such a sequence σ can be found, the theorem follows, since at $|\sigma|$ GREEDY's maximum load μ must satisfy

$$\mu+(\mu-1)+(\mu-(1+2))+\cdots+(\mu-(1+2+\cdots+(q-1)))\geqslant n,$$

where q is the number of busy servers at time t and $\binom{q}{2}<\mu$. Thus,

$$\mu\geqslant\frac{(3n)^{2/3}}{2}(1+o(1))$$

and the competitive ratio is at least $[(3n)^{2/3}/2](1+o(1))$.

We build the sequence σ via a two-step process. First, we suppose that we can construct a sequence of requests ρ such that at time $|\rho|$ the following properties are satisfied:

(1) At time $|\rho|$, the number of active tasks $|T(|\rho|)|$ is n.

(2) At every time t, $0\leqslant t\leqslant|\rho|$, every server in the optimal off-line assignment has load at most 1. (Thus, at time ρ, every server in the optimal off-line assignment has load *exactly* 1.)

(3) If at time $|\rho|$, GREEDY uses server i at all (i.e. $L_i(|\rho|)>0$) then GREEDY runs on i the task that OPT runs on i, for the simple reason that it cannot be run anywhere else. (Symbolically, $M(m^{-1}(i))=\{i\}$.)

The following sequence ρ of n requests satisfies these conditions: for each server $i\in M$, a new task v_i arrives, with $M(v_i)=\{i\}$. Clearly, both GREEDY and OPT must assign v_i to server i. Properties 1–3 are trivially satisfied at time $|\rho|$.

The second step is to show that any sequence ρ satisfying these three properties either has the property

(P1) For all j, such that $S_j(|\rho|)>0$,

$$S_{j-1}(|\rho|)-S_j(|\rho|)\geqslant j-1,$$

or it can be extended by a subsequence ρ' such that at time $|\rho\rho'|$, properties 1–3 are satisfied, and $S(|\rho\rho'|)>S(|\rho|)$.

Indeed, assume that we have constructed a sequence ρ that satisfies properties 1–3. If it satisfies property (P1), we stop and set $\sigma=\rho$. If not, then we can extend the sequence ρ with the following subsequence ρ'.

Let $j\geqslant 2$ be the smallest value for which $S_j(|\rho|)>0$ and $S_{j-1}(|\rho|)-S_j(|\rho|)=d\leqslant j-2$.

The sequence ρ' is constructed as follows:

(1) For $1\leqslant i\leqslant d$, task $m^{-1}(s_i)$ departs. This results in a decrease in the load on each of the servers $s_1,\ldots,s_d$ by 1.

(2) Tasks $m^{-1}(s_j)$ and $m^{-1}(s_{j-1})$ depart.

(3) For $1\leqslant i\leqslant d$, a new task c_i with $M(c_i)=\{s_i,s_j\}$ arrives. GREEDY assigns all of these tasks to s_j, since it is always less loaded than the alternative machine. The result is that L_j becomes equal to L_{j-1}. (OPT assigns c_i to s_i.)

(4) Two additional tasks c_a and c_b arrive with $M(c_a)=M(c_b)=\{s_j,s_{j-1}\}$. GREEDY assigns one of them, say c_a, to s_{j-1}, and the other to s_j. (OPT assigns c_a to s_j and c_b to s_{j-1}, i.e. the opposite of GREEDY.)

(5) Tasks c_i, $1\leqslant i\leqslant d$ depart, and then arrive again renamed c_i', this time with $M(c_i')=\{s_i\}$, and task c_b departs and then arrives again, this time renamed c_b' with $M(c_b')=s_{j-1}$.

(6) Each task h run by GREEDY on server s_j departs and then arrives again, renamed h', this time with $M(h')=m(h)$.

It is straightforward to check that properties 1–3 hold at time $|\rho\rho'|$, and that $S(|\rho\rho'|)>S(|\rho|)$.

Clearly, the sequence can be extended only a finite number of times because each extension increases the lexicographic value of its state $S(|\rho|)$ and the state S can take only a finite number of different values since it consists of n nonnegative integers whose sum is n. But as long as property (P1) does not hold, the sequence can be further extended. Therefore, after a finite number of extensions, property (P1) must hold. □

The proof can be extended to deal with a randomized variant of the greedy on-line assignment algorithm RGREEDY, whereby ties are broken using random bits.

Theorem 3.2. *The competitive ratio of the randomized greedy on-line assignment algorithm, is at least*

$$\frac{(3n)^{2/3}}{4}(1+o(1)).$$

Proof. The proof is almost identical to the proof of Theorem 3.1. The idea is to modify the sequence σ such that RGREEDY has no random choices to make. The modified sequence satisfies the following:

(1) For all j such that $S_j(|\sigma|)>0$, $S_{j-1}(|\sigma|)-S_j(|\sigma|)\geqslant j-1$.

(2) There are n tasks running at time $|\sigma|$.

(3) The maximum off-line load at all times t, $0\leqslant t\leqslant|\sigma|$, is 2.

We need to make two modifications to the construction of σ:

(a) At every time t, $0\leqslant t\leqslant|\rho|$, every server in the optimal off-line assignment is allowed a load up to 2. But at time ρ, every server in the optimal off-line assignment has load *exactly* 1.

(b) Steps 4 and 5 in the construction of the extension sequence ρ' are modified as follows:

(4′) Three additional tasks c_a, c_b, and c_c arrive with $M(c_a)=\{s_j\}$, $M(c_b)=\{s_j, s_{j-1}\}$, and $M(c_c)=\{s_{j-1}\}$. RGREEDY assigns c_a to s_j and c_b and c_c to s_{j-1}, since there are no ties. OPT assigns c_a and c_b to s_j, and c_c to s_{j-1}. (At this point, and only at this point, OPT has load 2 on s_j.)

(5′) Tasks c_i, $1\leqslant i\leqslant d$, depart, and then arrive again renamed c_i', this time with $M(c_i')=\{s_i\}$. Task c_a departs. (At this point OPT has again load 1 on every server.)

With these modifications, the required properties are clearly satisfied and the bound follows. □

3.2. The general lower bound

Theorem 3.3. *The competitive ratio of any randomized on-line assignment algorithm is at least* $\Omega(n^{1/2})$.

Proof. We follow the outline of the proof of Theorem 3.1. As before, we assume that all tasks have unit weight. Let A be any randomized on-line algorithm. We use the notion of an *adjusted load* $L_i'(t)$. The adjusted loads have the following properties:

(1) Each task j contributes to the adjusted load of exactly one server in $M(j)$, say k, and the value of j's contribution to $L_k'(t)$ is p_k, the probability that A assigns task j to server k. If $|M(j)|>1$, the choice of $k\in M(j)$ depends on the randomized algorithm A but not on the outcome of its coin flips. Note that if $|M(j)|=1$, task j always contributes exactly 1 to its server.

(2) The sequence of requests generated by the adversary for the lower bound will ensure that task j contributes to server k's adjusted load only if $p_k\geqslant 1/3$. Therefore, if r tasks are running at time t, then $\sum_{i\in M}L_i'(t)\geqslant r/3$.

The first property implies that $L_i'(t)$ is a lower bound on the expected load on server i at time t, i.e. $L_i'(t)\leqslant E(L_i(t))$.

Let $S'(t)=(S_1'(t), S_2'(t), \ldots, S_n'(t))$ to be the sorted (nonincreasing) n-tuple of adjusted loads that A has at time t. Define a partial order on S' as follows: $S'(t)>S'(t')$ if there is a j such that for all $1\leqslant i<j$, $S_i'(t)\geqslant S_i'(t')$, and $S_j'(t)-S_j'(t')\geqslant 1/3$.

We prove the lower bound by showing that there is a sequence of requests σ, and a way to define the adjusted loads L' so that the following three conditions hold:

(1) For all j such that $S'_j(|\sigma|)>0$, $S'_{j-1}(|\sigma|)-S'_j(|\sigma|)\geqslant 1/3$.

(2) There are n tasks running at time $|\sigma|$. Therefore, $\sum_{i\in M} L'_i(|\sigma|)\geqslant n/3$.

(3) The maximum off-line load at all times is at most 1.

If such a σ can be found, the theorem follows, since at time $|\sigma|$, maximum adjusted load μ of A (and, hence, maximum expected load) must satisfy

$$\mu+(\mu-1/3)+(\mu-2/3)+\cdots+(\mu-(q-1)/3)\geqslant \sum_{i\in M} L'_i(|\sigma|)\geqslant n/3,$$

where $q<3\mu+1$ is the number of servers with a nonzero adjusted load at time $|\sigma|$. Hence,

$$\mu\geqslant \frac{(2n)^{1/2}}{3}(1+o(1))$$

and the competitive ratio is at least $[(2n)^{1/2}/3](1+o(1))$.

We build the sequence σ via a two-step process. First, we suppose that we can construct a sequence of requests ρ, and define the adjusted loads L' such that at time $|\rho|$ the following properties are satisfied:

(1) The number of active tasks $|T(|\rho|)|$ is n. Thus, $\sum_{i\in M} L'_i(|\rho|)\leqslant n$.

(2) For any t, $0\leqslant t\leqslant|\rho|$, every server in the optimal off-line assignment has a load of at most 1.

(3) For any server i with $L'_i(|\rho|)>0$, the algorithm A runs on i the task that OPT runs on i, for the simple reason that it cannot be run anywhere else. (Symbolically, $M(m^{-1}(i))=\{i\}$). Therefore, this task contributes precisely 1 to i's adjusted load. We conclude that if $L'_i(|\rho|)>0$, then $L'_i(|\rho|)\geqslant 1$.

The following sequence ρ of n requests satisfies these conditions: for each server $i\in M$, a new task v_i arrives, with $M(v_i)=\{i\}$. Clearly, both A and OPT must assign v_i to server i. Since we must have $L'_i(|\rho|)=1$ for all $i\in M$, we observe that properties 1–3 are trivially satisfied.

The second step is to show that any sequence ρ satisfying these three properties either has the property

(P2): For all j such that $S'_j|\rho|>0$,

$$S'_{j-1}|\rho|-S'_j|\rho|\geqslant 1/3$$

or it can be extended by a subsequence ρ' such that at time $|\rho\rho'|$, properties 1–3 are satisfied, $S'(|\rho\rho'|)>S'(|\rho|)$.

Indeed, assume that we have constructed a sequence ρ that satisfies properties 1–3. If it satisfies property (P2), we stop and set $\sigma=\rho$. If not, then we can extend the sequence ρ with the following subsequence ρ':

(1) Let a and b be two distinct servers such that $1/3>L'_a-L'_b\geqslant 0$ and $L'_b>0$. Tasks $m^{-1}(a)$ and $m^{-1}(b)$ depart. By Property 3, this results in a decrease in L'_a and L'_b by 1.

(2) A new task j with $M(j)=\{a,b\}$ arrives. Let p_a (resp. p_b) be the probability (conditioned on the entire sequence of requests up to now) that A assigns j to a (resp. to b). Clearly, $p_a+p_b=1$.

(3) The rest of ρ' depends on the value of p_a.

- $p_a \geq 1/3$: OPT assigns j to b. Task j contributes p_a to the adjusted load on a, and nothing to the adjusted load on b. A new task k with $M(j)=a$ arrives. All tasks T contributing to the adjusted load on b depart and then arrive again, this time with $M(v)=m(v)(v\in T)$.
- $p_a < 1/3$: OPT assigns j to a. Task j contributes p_b to the adjusted load on b, and nothing to the adjusted load on a. A new task k with $M(j)=b$ arrives. All tasks T contributing to the adjusted load on a depart and then arrive again, this time with $M(v)=m(v)$ $(v\in T)$.

It is straightforward to verify that properties 1–3 hold at time $|\rho\rho'|$.

Lastly, we show that $S'(\rho\rho')>S'(\rho)$: In case 3(a), the subsequence of adjusted loads $(L_a(|\rho|), L_b(|\rho|), 0, 0, \ldots, 0, 0)$ was replaced by $(L_a(|\rho|)+p_a, 0, 1, 1, \ldots, 1, 1)$, where $L_a(|\rho|)\geq 1$ and $p_a \geq 1/3$. In case 3(b), the subsequence of adjusted loads $(L_a(|\rho|), L_b(|\rho|), 0, 0, \ldots, 0, 0)$ was replaced by $(L_a(|\rho|)+p_b, 0, 1, 1, \ldots, 1, 1)$ where $p_b \geq 2/3$ and $L_b(|\rho|)+p_b \geq L_a(|\rho|)+1/3$, since $L_a(|\rho|)-L_b(|\rho|)<1/3$. (The 0's and 1's in these vectors correspond to servers running tasks in T).

Finally, note that after a finite number of extensions of the sequence ρ, property (P2) will hold since the number of extensions is bounded by the length of the maximum chain in the partial order on S', which is finite, since every difference is bounded from below, and the sum of all components is bounded from above. □

For a general deterministic algorithm, we can simplify this proof and obtain a lower bound of $[\sqrt{2n}]$.

4. Split assignments

In this section we consider a variant of the load balancing problem where the weight of an incoming task can be split among its allowed servers. The weights can be split into arbitrary positive real portions, and, as before, once assigned cannot be reassigned.

Algorithm WATER-LEVEL: Upon arrival of a task j, split its weight among the servers in $M(j)$ so that $\min_{i\in M(j)} L_i$ is maximized.

The algorithm is called "water-level" since it raises the load equally on the least loaded servers in $M(j)$. Azar et al. [2] showed that for permanent tasks its competitive ratio is $O(\log n)$ and this is tight. For the general case (i.e. temporary tasks are allowed), we obtain the following theorem.

Theorem 4.1. *The competitive ratio of the water-level algorithm for the split assignment problem is* $[(3n)^{2/3}/2](1+o(1))$. *This is tight. Any other deterministic or randomized on-line algorithm for this problem has competitive ratio* $\Omega(n^{1/2})$.

Proof. We can easily derive a $[(3n)^{2/3}/4](1+o(1))$ lower bound by following the proof of Theorem 3.2: since RGREEDY never encounters a tie, WATER-LEVEL on the same sequence never splits any job and never produces nonintegral loads, while at the same time OPT can keep its maximum load under 2. The bound can be pushed to $[(3n)^{2/3}/2](1+o(1))$ via an intricate modification of the sequence used in Theorem 3.1. We omit this argument in the interest of brevity.

Similarly, the $\Omega(n^{1/2})$ lower bound of Theorem 3.3 can be extended to the split-assignment problem by replacing the probability that the algorithm assigns a task to one of two machines with the expected fraction of a job assigned to that machine. The proof goes through, mutatis mutandis.

Finally, we show that a $[(3n)^{2/3}/2](1+o(1))$ upper bound for WATER-LEVEL follows from Theorem 2.2 and the following lemma.

Lemma 4.2. *The competitive ratio of* WATER-LEVEL *for the split-assignment problem is not larger than the competitive ratio of* GREEDY *for the nonsplit assignment problem.*

Proof. There are two steps to the proof.

In the first step, for any input instance I to WATER-LEVEL, we create another input I' (to WATER-LEVEL) such that I and I' have the same on-line and off-line costs and the optimal off-line solution OPT on I' does not split any weights.

To do this, suppose that the weight $w(j)$ of task j of I was split by OPT into portions $w_1(j), \ldots, w_{d_j}(j)$. Then in I', the arrival of task j is replaced by the consecutive arrival of d_j tasks, where the ith task has weight $w_i(j)$, $1 \leqslant i \leqslant d_j$, and they all have the same set of allowed processors, namely, $M(j)$. Similarly, the departure of task j of I is replaced in I' by the consecutive departure of these d_j tasks.

It is easy to verify that this transformation has the two properties stated above. Indeed, after the arrival (or departure) of the replacement tasks of a task in I, WATER-LEVEL is the same state in both I and I', and hence has the same costs. For the second property, we observe that, by definition, further splitting cannot improve the performance of OPT.

In the second step, we transform the input instance I' into an input I'' such that the cost of GREEDY on I'' is equal to the cost of WATER-LEVEL on I', and OPT's costs on I' and I'' are the same.

Suppose that WATER-LEVEL split the weight $w(j)$ of task j into several portions, $w_1(j), \ldots, w_{d_j}(j)$. In I'', the arrival of task j is replaced by the arrival of d_j tasks, where the ith has weight $w_i(j)$, $1 \leqslant i \leqslant d_j$, and they all have the same set of allowed processors, namely $M(j)$. Furthermore, in I'' these d_j tasks arrive in order of nonincreasing weight.

It is easy to verify that both GREEDY and WATER-LEVEL have the same loads after the arrival (or departure) in I'' of all the replacement tasks for a task t from I' as WATER-LEVEL had after the arrival (or departure) of t in I'. The optimal off-line costs of I' and I'' are the same, since OPT on I'' can emulate OPT on I', and vice versa.

This concludes the proof of the lemma. □

Proof of Theorem 4.1 (*conclusion*). Since the competitive ratio of GREEDY is $[(3n)^{2/3}/2](1+o(1))$, the proof of the theorem is complete. □

References

[1] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs and O. Waarts, Online load balancing of temporary tasks, in: *Workshop on Algorithms and Data Structures (WADS)*, Montreal, Canada, 1993, 119–130.

[2] Y. Azar, J. Naor and R. Rom, The competitiveness of on-line assignment, *Proc. 3rd Ann. ACM–SIAM SODA* (1992) 203–210.

[3] Y. Bartal, A. Fiat, H. Karloff and R. Vohra, New algorithms for an Ancient Scheduling Problem, in: *Proc. 24th Ann. ACM Symp. on Theory of Computing* (1992) 51–58.

[4] S. Baruah, G. Koren, B. Mishra, A. Raghunthan, L. Rosier and D. Shasta, On-line scheduling in the presence of overload, in: *Proc. 32nd IEEE Symp. on Foundations of Computer Science* (1991) 100–110.

[5] R.L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Tech. J.* **45** (1966) 1563–1581.

[6] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Ann. Discrete Math.* **5** (1979) 287–326.

[7] R. Karp, U. Vazirani and V. Vazirani, An optimal algorithm for on-line bipartite matching, in: *Proc. 22nd Ann. ACM Symp. on Theory of Computing*, Baltimore, Maryland (1990) 352–358.

[8] L. Lovász, M. Saks and W. Trotter, An on-line graph coloring algorithm with sublinear performance ratio, *Discrete Math.* **75** (1989) 319–325.

[9] S. Phillips and J. Westbrook, Online load balancing and network flow, in: *Proc. 25th Ann. ACM Symp. on Theory of Computing* (1993) 402–411.

[10] D. Shmoys, J. Wein and D.P. Williamson, Scheduling parallel machines on-line, in: *Proc. 32nd IEEE Symp. on Foundations of Computer Science* (1991) 131–140.

[11] S. Vishwanathan, Randomized on-line graph coloring, in: *Proc. 31st Ann. Symp. on Foundations of Computer Science* (1990) 464–469.