On-line Load Balancing of Temporary Tasks *

Yossi Azar[†] Tel-Aviv Univ.

Bala Kalyanasundaram[‡] Univ. of Pittsburgh

Serge Plotkin[§] Stanford Univ.

Kirk R. Pruhs[¶] Univ. of Pittsburgh

Orli Waarts[||] IBM Almaden

Abstract

This paper considers the non-preemptive on-line load balancing problem where tasks have *limited duration* in time. Upon arrival, each task has to be immediately assigned to one of the machines, increasing the load on this machine for the duration of the task by an amount that depends on both the machine and the task. The goal is to minimize the maximum load.

Azar, Broder and Karlin studied the *unknown duration* case where the duration of a task is not known upon its arrival [4]. They focused on the special case in which for each task there is a subset of machines capable of executing it, and the increase in load due to assigning the task to one of these machines depends only on the task and not on the machine. For this case, they showed an $O(n^{2/3})$-competitive algorithm, and an $\Omega(\sqrt{n})$ lower bound on the competitive ratio, where n is the number of the machines. This paper closes the gap by giving an $O(\sqrt{n})$-competitive algorithm.

In addition, trying to overcome the $\Omega(\sqrt{n})$ lower bound for the case of unknown task duration, this paper initiates a study of the load balancing problem for tasks with *known duration* (*i.e.,* the duration of a task becomes known upon its arrival). For this case we show an $O(\log nT)$-competitive algorithm, where T is the ratio of the maximum possible duration of a task to the minimum possible duration of a task.

The paper explores also an alternative way to overcome the $\Omega(\sqrt{n})$ bound; it considers the *related machines* case with unknown task duration. In the related machines

---

*Preliminary version of this paper was presented at the 1993 Workshop on Algorithms and Data Structures. This paper merges the results of a paper by the first, third and fifth authors and a paper by the second and the forth authors.

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

‡Dept. of Computer Science, University of Pittsburgh. Supported in part by NSF under grants CCR-9009318 and CCR-9202158. E-Mail: kalyan@cs.pitt.edu

§Dept. of Computer Science, Stanford University. Research supported by U.S. Army Research Office Grant DAAL-03-91-G-0102, and by a grant from Mitsubishi Electric Laboratories. E-Mail: plotkin@cs.stanford.edu

¶Dept. of Computer Science, University of Pittsburgh. Supported in part by NSF under grant CCR-9209283. E-Mail: kirk@cs.pitt.edu

||IBM Almaden Research Center. E-Mail: orli@cs.stanford.edu

case, a task can be executed by any machine and the increase in load depends on the speed of the machine and the weight of the task. For this case we give a 20-competitive algorithm and show a lower bound of $3 - o(1)$ on the competitive ratio.

1 Introduction

This paper considers the load balancing problem defined as the problem of online assignment of tasks to n machines; assignment has to be done immediately upon arrival of the task, increasing the load on the machine the task is assigned to for the duration of the task. We consider non-preemptive load balancing, *i.e.*, reassignment of tasks is not allowed. The goal is to minimize the maximum load.

The online load balancing problem naturally arises in many applications involving allocation of resources. In particular, many cases that are usually cited as applications for bin packing become load balancing problems when one removes the assumption that the items, once "stored", remain in the storage forever (*e.g.*, storing food in warehouses). As a simple concrete example, consider the case where each "machine" represents a communication channel with bounded bandwidth. The problem is to assign each incoming request for bandwidth to one of the channels. Assigning a request to a certain communication channel increases the load on this channel, *i.e.*, increases the percentage of the used bandwidth. The load is increased for the duration associated with the request.

Formally, each arriving task j has an associated *load vector*, $\vec{p}_j = (p_{1j}, p_{2j}, \ldots, p_{nj})$, where p_{ij} defines the increase in the load of machine i if we were to assign task j to it. This increase in load occurs for the duration $d(j)$ of the task. Since the arriving tasks have to be assigned without knowledge of the future tasks, it is natural to evaluate the performance in terms of the *competitive ratio* [13], which in this case is the supremum, over all possible input sequences, of the maximum (over time and over machines) load achieved by the on-line algorithm to the maximum load achieved by the optimal off-line algorithm. The competitive ratio may depend, in general, on the number of machines n, which is usually fixed, and should not depend on the number of tasks that may be arbitrary large.

Similar to the way it is done for scheduling problems, it is natural to categorize load balancing problems according to the properties of the load vectors. The simplest case however is where the coordinates of each load vector are equal to some value that depends only on the task. It is easy to observe that the algorithm due to Graham [7], applied to this kind of load-balancing problem, leads to a $(2 - \frac{1}{n})$-competitive solution.

Azar, Broder, and Karlin [4] proposed to study a less restricted case, motivated by the problem of on-line assignment of network nodes to gateways (see also [5]). In this case, a task can represent a request of a network node to be assigned to a gateway; machines represent gateways. Since, in general, each node can be served by only a subset of gateways, this leads to a situation where each coordinate of a load vector is either ∞ or equal to a given value that depends only on the task. For this case, which we will refer to as *identical speed with assignment restriction case*, [4] shows an $\Omega(\sqrt{n})$ lower bound on the competitive ratio of any load balancing algorithm that deals with the *unknown duration* case, *i.e.* the case where the duration of a task becomes known only upon its termination. They also present an $O(n^{2/3})$-competitive algorithm.

The work in [4] opens several new research avenues. The first is the question of whether there exists an $O(\sqrt{n})$-competitive algorithm for the identical speed with assignment restriction case when the duration of a task becomes known only upon its termination. Secondly, the $\Omega(\sqrt{n})$ lower bound for the competitive ratio in [4] suggests considering natural variations of the problem for which this lower bound does not apply. One such fundamental candidate proposed here is the *known duration* case, where the duration of each task is known upon its arrival. Another important candidate is the *related machines* case. In the related machines case, the ith coordinate of each load vector is equal to w_j/s_i, where the "weight" w_j depends only on the task j and the "speed" s_i depends only on the machine i.

This work addresses the above three issues. The main results presented in this paper are as follows:

- A $(2\sqrt{n}+1)$-competitive algorithm, Robin-Hood, for the identical-speed with assignment restriction case studied by [4]. Their lower bound implies that this algorithm is optimal to within a $\sqrt{2}$ multiplicative factor.
- An $O(\log nT)$-competitive algorithm for the known-duration case with unrestricted load vectors, where T is the ratio of the maximum to minimum duration. We assume that the minimum possible task duration is known in advance.
- A 20-competitive algorithm, Slow-Fit, for the related machines case with unknown task duration. We also show that any algorithm for this case can't be better than $(3-o(1))$-competitive, even if the optimal load is known in advance.

More specifically, algorithm Robin-Hood is simple, deterministic, and runs in $O(n)$ time per task assignment. Interestingly, its decision where to assign a new task depends not only on the current load on each machine but also on the history of the machines.

Our algorithm for the known duration case is an application of the virtual circuit routing algorithm of [1]. Roughly speaking, we show how to use multiple concurrent instances of the *permanent* virtual circuit routing algorithm in order to solve a single instance of the load balancing problem with *temporal* tasks. Our approach, together with the results in [1], can be used to show an $O(\log nT)$-competitive algorithm for virtual circuit routing with known duration.

Algorithm Slow-Fit is essentially identical to the algorithm of Aspnes, Azar, Fiat, Plotkin and Waarts for assigning permanent tasks [1]. Roughly speaking, the idea (which originated in the paper by Shmoys, Wein, and Williamson [12]) is to assign the task to the least capable machine while maintaining that the load does not exceed the currently set goal. However, the analysis in [1] is inapplicable for the case where tasks have limited duration. Our analysis is thus different and shows that Slow-Fit is 5-competitive if the maximum load is known.[1] Similarly to algorithm Robin-Hood, also algorithm Slow-Fit is deterministic and runs in $O(n)$ time per task assignment.

The $(3-o(1))$ lower bound for the related machines case applies even if the optimal maximum load is known in advance. This lower bound stands in contrast to the 2-competitive algorithm in [1] for the case where tasks are permanent and the optimal off-line maximal load is known in advance, and the 2-competitive algorithm for the unknown-duration case where the machines are identical [7].

Other related work On-line load balancing of *permanent* tasks (*i.e.,* tasks never terminate) was studied extensively [7, 10, 5, 6, 1]. For the case where the machines are identical, Graham showed a $(2-\frac{1}{n})$ competitive solution [7]. This solution has been improved in [6] by Bartal, Fiat, Karloff and Vohra to $2-\epsilon$ for a small constant ϵ, and the value of ϵ was further improved in [9]. The identical speed with assignment restriction case was introduced by Azar, Naor, and Rom [5], who described an $O(\log n)$-competitive algorithm and a matching lower bound. The case of unrestricted load vectors (known also as the *unrelated machines* case), was considered by Aspnes, Azar, Fiat, Plotkin and Waarts [1], who showed an $O(\log n)$-competitive algorithm. They also showed an 8-competitive algorithm for the related machines case. Their algorithm is 2-competitive in the case that the optimal off-line maximum load is known. As the competitive ratios show, the fact that tasks have finite duration can make the task of the on-line load balancing algorithm significantly more difficult, especially when this duration is unknown upon the arrival of the task.

Both the idea introduced here of considering the case where the duration of the task is known upon its arrival, as well as the method used here to solve this case (*i.e.,* concurrently working with multiple copies of the graph, one copy for each time instance), can be adapted for the context of virtual circuit routing to provide an $O(\log nT)$-competitive algorithm for virtual circuit routing with known duration. Recently our approach was employed in the context of throughput-competitive on-line routing as well [2].

All the results in this paper, as well as in the papers mentioned above, concentrate on *non-preemptive* load balancing, *i.e.,* re-assignment of tasks is not allowed. Another, very different model is when re-assignment of existing tasks is allowed. After the time of this work, for the case where the coordinates of the load vector are restricted to be 1 or ∞, and a task duration is not known upon its arrival, Phillips and Westbrook [11] proposed an algorithm that achieves $O(\log n)$ competitive ratio with respect to load while making $O(1)$ amortized rassignments per job. The general case was later considered in [3], who designed an $O(\log n)$-competitive algorithm with respect to load that reroutes each circuit at most $O(\log n)$ times.

Finally, we note that the load balancing problem is different from the classical scheduling problem of minimizing the makespan of an on-line sequence of tasks with known running times (see [8, 12] for survey). Intuitively, in the load balancing context, the notion of

[1]It is interesting to note that the natural greedy algorithm, which tries to minimize the maximal load on any machine, is $\Theta(\log n)$-competitive [1]. (Their analysis for permanent tasks applies also for the case where tasks have limited duration .)

makespan corresponds to maximum load, and there is a new, orthogonal, notion of time. (See [1] for further discussion of the differences.)

2 Identical Speed with Assignment Restriction

In this section we describe a $(2\sqrt{n}+1)$-competitive algorithm for the identical speed with assignment restriction case, where the task duration is unknown upon its arrival. Task j must be assigned to one of the machines in a set $M(j)$; assigning this task to a machine i raises the load on machine i by w_j. This result complements the $\Omega(\sqrt{n})$ lower bound of [4].

The input sequence consists of task arrival and task departure events. Since the state of the system changes only as a result of one of these events, the event numbers can serve as time units, *i.e.* we can view the time as being *discrete*. We say that time t corresponds to the tth event. Initially the time is 0, and time 1 occurs when the first task arrives. Whenever we speak about the "state of the system at time t" we mean the state of the system *after* the tth event was already handled. In other words, the response to the tth event takes the system from "state at $t-1$" to "state at t".

Since we can assume that we are considering a fixed input sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$, we will use OPT for $OPT(\sigma)$. Let $\ell_g(t)$ denote the load on machine g at time t, *i.e.*, after the tth event. To simplify the notation, we use j to denote a task instead of σ_j, unless it might cause confusion.

At any time t we maintain an estimate $L(t)$ for OPT satisfying $L(t) \leq OPT$. A machine g is said to be *rich* at some point in time t if $\ell_g(t) \geq \sqrt{n}\ L(t)$, and is said to be *poor* otherwise. A machine may alternate between being rich and poor over time. If g is rich at t, its *windfall time* at t is the last moment in time it became rich. More precisely, g has windfall time t_0 at t if g is poor at time $t_0 - 1$, and is rich for all times t' $t_0 \leq t' \leq t$. Let g be a rich with windfall time t_0.

Algorithm Robin-Hood: Assign the first task to an arbitrary machine, and set $L(1)$ to the weight of the first task. When a new task j arrives at time t, set:

$$L(t) = \max\{L(t-1), w_j, \frac{1}{n}(w_j + \sum_g \ell_g(t-1))\}$$

The last quantity is the aggregate weight of the tasks currently active in the system divided by the number of machines. Note that the recomputation of $L(t)$ may cause some rich machines to be reclassified as poor machines. If possible, assign j to some poor machine g, *i.e.*, $\ell_g(t-1) < \sqrt{n}\ L(t)$. Otherwise, j is assigned to the rich machine g with the most recent windfall time.

Lemma 2.1 At all times t, Robin-Hood guarantees that $L(t) \leq OPT$.

Proof: The proof is by induction on the number of the assigned tasks. After the first task $L(t) \leq OPT$. For the inductive part, it suffices to consider only the cases where $L(t)$ is increased. The claim follows from the facts that $w_j \leq OPT$ and $\frac{1}{n}(w_j + \sum_g \ell_g(t-1)) \leq OPT$. ■

The following lemma is immediate from the fact that $nL(t)$ is an upper bound on the aggregate load of the currently active tasks.

Lemma 2.2 At most $\sqrt{n}$ machines can be rich at any point in time.

Theorem 2.1 The competitive ratio of Algorithm Robin-Hood is at most $2\sqrt{n}+1$.

Proof: We will show that the algorithm guarantees that at any point in time t, $\ell_g(t) \leq \sqrt{n}\ (L(t) + OPT) + OPT$ for any machine g. The claim is immediate if g is poor at t. Otherwise, let S be the set of tasks that were assigned to g since g's windfall time t_0 and are still active at time t. Consider the machines to which the optimal offline algorithm assigns the tasks in S. Let j be some task in S. Since g is rich throughout the time interval $[t_0, t]$, each of the machines in $M(j)$ must be rich when j arrives. Moreover, each machine in $M(j) - \{g\}$ must have been rich already before time t_0 since otherwise Robin-Hood would have assigned j to it. Let $k = |\cup_{j \in S} M(j)|$. Lemma 2.2 implies that $k \leq \sqrt{n}$.

Let q be the task assigned to g at time t_0 that caused g to become rich. Since $w(q) \leq OPT$, $\sum_{j \in S} w_j \leq kOPT$ and $k \leq \sqrt{n}$ we conclude that

$$\ell_g(t) \leq \sqrt{n}\ L(t) + w(q) + \sum_{j \in S} w_j \leq \sqrt{n}\ L(t) + OPT + \sqrt{n}OPT.$$

■

3 Tasks with Known Duration

This section considers the case where the duration of a task is known upon its arrival, *i.e.,* $d(j)$ is revealed to the online algorithm when task j arrives. We provide an algorithm whose competitive ratio is $O(\log nT)$ even for the unrelated machine case (i.e with unrestricted load vectors), where T is the ratio of the maximum to minimum duration (the minimum possible task duration is known in advance). This result is in contrast to the $\Omega(\sqrt{n})$ lower bound on the competitive ratio for the identical speed with restriction case when the duration of a task is not known upon its arrival [4].

Our algorithm is based on the on-line algorithm of [1], which solves the following "route allocation" problem: We are given a directed graph $G = (V, E)$ with $|V| = N$. Request i is specified as tuple $(s_i, t_i, \{p_{i,e} | e \in E\})$, where $s_i, t_i \in V$ and for all $e \in E$, $p_{i,e} \geq 0$. Upon arrival of request i, the route allocation algorithm has to assign it to a path (route) P_i from s_i to t_i in G; the route assignments are permanent. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ be the routes assigned to requests 1 through k by the on-line algorithm, and let $\mathcal{P}^* = \{P_1^*, P_2^*, \ldots, P_k^*\}$

be the routes assigned by the off-line algorithm. Given a set of routes $\mathcal{P}$, the *load on edge* e after the first j requests are satisfied is defined by:

$$\ell_e(j) = \sum_{i \leq j : e \in P_i} p_{i,e} \tag{1}$$

Let $\lambda(j) = \max_{e \in E} \ell_e(j)$ define the maximum load. Similarly, define $\ell_e^*(j)$ and $\lambda^*(j)$ to be the corresponding quantities for the routes produced by the off-line algorithm. For simplicity we will abbreviate $\lambda(k)$ as λ and $\lambda^*(k)$ as λ^*. The goal of the online algorithm is to produce a set of routes $\mathcal{P}$ that minimizes λ/λ^*.

The online route allocation algorithm of [1] is $O(\log N)$-competitive, where N is the number of vertices in the given graph. Roughly speaking, we will reduce our problem of online load balancing of *temporary* tasks with known duration to several concurrent instances of the online route allocation for *permanent* routes problem, where $N = nT$. Then we will apply the algorithm of [1] to achieve $O(\log N) = O(\log nT)$ competitive ratio.

We assume that the minimum task duration is known in advance and measure time in terms of this minimum duration interval. Let $a(j)$ denote the arrival time of task j, and assume $a(1) = 0$. Let $T' = (\max_{1 \leq j \leq |\sigma|} a(j) + d(j))$ be the total duration of all the tasks. First we make several simplifying assumptions and then show how to eliminate them:

1. T' is known in advance to the online algorithm.
2. T is known in advance to the online algorithm.
3. Arrival times and task durations are integers, *i.e.,* time is discrete.

We now describe an $O(\log nT')$-competitive algorithm Assign1, which we later use as a subroutine in our final algorithm.

Algorithm Assign1: The idea is to translate each task into a request for allocating a route. We construct a directed graph G consisting of T' layers, each of which consists of $n+2$ vertices, numbered $1, \ldots, n+2$. We denote vertex i in layer k by $v(i,k)$. For each layer $1 \leq k \leq T'$, and for $1 \leq i \leq n$, we refer to vertices $v(i,k)$ as *common* vertices. Similarly, for each layer k, $v(n+1,k)$ is referred to as the *source* vertex and $v(n+2,k)$ is referred to as the *sink* vertex. For each layer, there is an arc from the source $v(n+1,k)$ to each of the n common vertices $v(i,k)$. In addition, there is an arc from each common vertex $v(i,k)$ to the sink $v(n+2,k)$ in each layer. Finally, there is an arc from each common vertex $v(i,k)$ to the corresponding common vertex $v(i,k+1)$ in the next layer. The arc from $v(i,k)$ to $v(i,k+1)$ will represent machine i during the time interval $[k,k+1)$, and the load on the arc will correspond to the load on machine i during this interval.

We convert each new task j arriving at $a(j)$ into a request for allocating a route in G from the source of layer $a(j)$ to the sink of layer $a(j)+d(j)$. The loads $p_{j,e}$ are defined as follows: for arcs $v(i,k)$ to $v(i,k+1)$ for $a(j) \leq k \leq a(j)+d(j)-1$, we set $p_{j,e} = p_i(j)$; we set $p_{j,e}$ to 0 for the arcs out of the source of layer $a(j)$ and for the arcs into the sink of layer $a(j)+d(j)$, and to ∞ for all other arcs. Clearly the only possible way to route the task j is through the arcs $v(i,k)$ to $v(i,k+1)$ for $a(j) \leq k \leq a(j)+d(j)-1$ for some i. This route raises the load on the participating arcs by precisely $p_i(j)$ which corresponds to assigning the task to machine i. Thus, minimizing the maximum load on an arc in the online route allocation on the directed graph G corresponds to minimizing the maximum machine load in the load balancing problem.

We now show how to construct an $O(\log nT)$-competitive algorithm. Partition the tasks into groups according to their arrival times. Group m contains all tasks that arrive in the time interval $[(m-1)T, mT)$. Clearly each task in group m must depart by time $(m+1)T$, and the overall duration of tasks in any group is at most $2T$. For each group invoke a separate copy of Assign1 with $T' = 2T$. That is, assign tasks belonging to a certain group independently of the assignments of tasks in other groups. Using the route allocation algorithm of [1], we get that for each group, the ratio between maximal online load to the maximal off-line load is at most $O(\log nT)$. Moreover, at each instance of time active tasks can belong to at most 2 groups. Thus, the maximal online load at any given time is at most twice the maximal online load of a single group. The off-line load is at least the largest load of the off-line algorithms over all groups, and hence the resulting algorithm is $O(\log nT)$-competitive.

We now show how to remove the remaining simplifying assumptions. To remove the restriction that T is known in advance, let $T = d(1)$ when the first task arrives. If we are currently considering the mth group of tasks, and task j arrives with $d(j) > T$, we set $T = d(j)$ and $T' = 2d(j)$. Observe that this does not violate the invariant that at any point in time the active tasks belong to at most two groups. We use the fact that the route allocation algorithm of [1] is scalable in the sense that the current assignment of tasks in group m is consistent with the assignment if Assign1 had used the new value of T' instead of the old one. Thus the argument of $O(\log nT)$-competitiveness goes through as before.

Finally, to eliminate the assumption that events coincide with the clock (that is for all j, $a(j)$'s and $d(j)$'s are integral multiples of time unit), Assign 1 approximates $a(j)$ by $\lfloor a(j) \rfloor$ and $d(j)$ by $\lceil a(j)+d(j) \rceil - \lfloor a(j) \rfloor$. Since for all j, $d(j) \geq 1$, this approximation increases the maximal off-line load by at most a factor of 2. Thus, we have proved the following claim:

Theorem 3.1 The above online load balancing algorithm for unrelated machines with known tasks duration is $O(\log nT)$-competitive.

4 Related Machines

In this section we consider the related machines case when the duration of a task is not known upon its arrival. In the related machines case, there exists a notion of *machine speed* and each task can be assigned to any machine. In other words, we assume that each task j has weight w_j, and that $\forall i,\ p_i(j) = w_j/s_i$, where s_i is the speed of machine i. For convenience, we assume that $s_i \leq s_j$ for $i < j$. If G is a set of machines we define $S(G) = \sum_{k\in G} s_k$. If T is a set of tasks then $w(T) = \sum_{j\in T} w_j$. The definitions of time, OPT, and $\ell_g(t)$ are as described in the beginning of Section 2.

We first give an algorithm SLOW-FIT that is 5-competitive for this problem if OPT is known in advance and 20-competitive otherwise. Similarly to the result presented in Section 3, also this result is in contrast to the $\Omega(\sqrt{n})$ lower bound on the competitive ratio for the identical speed with restriction case when the duration of a task is not known upon its arrival.

Next we give a $3 - o(1)$ lower bound on the competitive ratio achievable by any on-line algorithm. Note that since our upper bounds apply for the case of tasks with unknown duration, they apply therefore also for the case of tasks with known duration.

4.1 Algorithm SLOW-FIT

We first describe a simplified version of SLOW-FIT that assumes that the value of OPT is known in advance. Let c be an integer constant (we will later choose $c = 5$ to minimize the resulting competitive ratio). Assume that a task j arrived at time t. We say j is *assignable* to machine g if and only if $w_j/s_g \leq OPT$ and $\ell_g(t-1) + w_j/s_g \leq c \cdot OPT$.

Algorithm SLOW-FIT (simplified version): Assign the new task to the slowest *assignable* machine. Break ties by assigning to the machine with the smallest index.

Theorem 4.1 Provided $c \geq 5$, the simplified version of SLOW-FIT guarantees that every task is assignable, which implies that the simplified SLOW-FIT algorithm is c-competitive.

Proof: Suppose that at some point in time t_0, a task q arrives that is not *assignable*. We then reach a contradiction to the supposed value of OPT by showing that at some time in the past the aggregate weight of the tasks in the system was more than $\sum_{i=1}^{n} s_i \cdot OPT$. Starting from the current time t_0, we construct a sequence of times $t_0 > t_1 > \ldots > t_k$. At time $t_i - 1$ we observe the load on set G_i of machines. We define $J(G_i)$ to be the collection of tasks that are active at time $t_i - 1$ and that were assigned to machines in G_i by the on-line algorithm.

Initially, $G_0 = \{n\}$. After the end of the ith iteration of our argument (during which we defined G_i and t_i) the following four invariants hold:

1. The times $t_0, \ldots, t_i$ are defined and $t_i < \ldots < t_1 < t_0$.
2. The G_j's, $0 \leq j \leq i$, are disjoint sets of consecutive machines, *i.e.*,

$$G_j = \{m_j, m_j + 1, \ldots m_{j-1} - 1\}$$

for some m_j's. (Note that since $G_0 = \{n\}$, we define $m_0 = n$ and $m_{-1} = n+1$. Also, G_j could be empty.)

3. At time $t_i - 1$, each machine in G_i has a load that exceeds $(c-1)$ OPT.

4. For $0 < j \le i$, $S(G_j) \ge (c-3)S(G_{j-1})$.

Initially, t_0 is the time at which task q arrived. Since task q can not be assigned to the fastest machine n, and $OPT \ge w(q)/s_n$, we have $\ell_n(t_0 - 1) > (c-1)$ OPT, which means that the cumulative weight of the tasks assigned to machine n that are active at $t_0 - 1$ exceeds $(c-1)$ s_n $OPT = (c-1)$ $S(G_0)$ OPT. Hence, the third invariant initially holds. The remaining invariants hold trivially.

Now assume that the above invariants hold at the end of the ith iteration. We will show how to construct t_{i+1} and G_{i+1} so that the invariants hold for $i+1$ as well. Using the 3rd invariant, we know that the load on each machine in G_i exceeds $(c-1)$ OPT at time $t_i - 1$. Hence, the cumulative weight of tasks in $J(G_i)$ is at least $(c-1)S(G_i)OPT$.

Consider the machines to which the optimal offline algorithm assigns the tasks in $J(G_i)$, and let m_{i+1} be the slowest machine that gets a task in $J(G_i)$. We let

$$G_{i+1} = \{m_{i+1}, m_{i+1}+1, \ldots, m_i - 1\}.$$

(Notice that G_{i+1} is empty if $m_{i+1} \ge m_i$.) Since all the tasks in $J(G_i)$ are active at time $t_i - 1$, the cumulative weight of the tasks from $J(G_i)$ assigned to machines in $\bigcup_{0 \le j \le i} G_j$ by the offline algorithm is bounded by $\sum_{0 \le j \le i} S(G_j)$ OPT. If $i = 0$, then this sum is at most $S(G_0)OPT$. Otherwise ($i > 0$), the fourth invariant guarantees that it is at most 2 $S(G_i)$ OPT. Thus the rest of the tasks in $J(G_i)$ have to be assigned by the offline algorithm to machines in G_{i+1}, and their cumulative weight is at least $(c-3)$ $S(G_i)$ OPT. Since the load on any machine never exceeds OPT in the offline assignment, it must be the case that $S(G_{i+1})$ $OPT \ge (c-3)$ $S(G_i)$ OPT. As a consequence we get

$$S(G_{i+1}) \ge (c-3)\ S(G_i).$$

Now consider the task q_i with the smallest weight in $J(G_i)$. Define time t_{i+1} to be the time at which q_i arrived. Since q_i was not assigned to any machine in G_{i+1}, all these machines were unassignable at t_{i+1}. Since $w(q_i)/s_g \le OPT$ for any machine $g \in G_{i+1}$,

$$\forall g \in G_{i+1}: \quad \ell_g(t_{i+1} - 1) > (c-1)\ OPT.$$

This proves that the invariants now hold for $i+1$.

Consider the last iteration, say k, where the set G_k includes machine 1. We know that the cumulative weight of tasks in $J(G_k)$ is at least $(c-1)S(G_k)OPT$ at time $t_k - 1$. Recall that $\sum_{0 \le i \le k} S(G_i) \le 2S(G_k)$. The contradiction follows from the fact that at any point in time, the maximum task weight that can be handled by the system is at most

$$\sum_{0 \le i \le k} S(G_i)\ OPT \le 2S(G_k)OPT < (c-1)S(G_k)OPT.$$

■

We now show how to modify SLOW-FIT so that it will not require advance knowledge of OPT.

Algorithm SLOW-FIT: After the arrival of the first task q set $L = w(q)/s_n$. Note that L is our estimate of OPT. We then repeat the following: (1) Apply the simplified version of SLOW-FIT, which uses L as the value of OPT, until it encounters an unassignable task q. (2) Set $L = 2L$, and return to step 1, ignoring any tasks encountered up to this point. In other words, when the simplified version of SLOW-FIT tries to assign q, it behaves as if the load on every machine is zero.

Theorem 4.2 The competitive ratio of SLOW-FIT is at most $4c$, provided $c \geq 5$.

Proof: Notice that at any point in time $OPT \geq L/2$. During the last iteration of the simplified version of SLOW-FIT the maximum load on any machine is at most cL, and hence at most $2c\ OPT$. Since L was doubled on each iteration, the cumulative load on any machine for all of the prior iterations is at most cL, or at most $2c\ OPT$. Therefore, the load on any machine is at most $4c\ OPT$. ■

4.2 Lower Bound

In this section we show that any on-line algorithm has competitive ratio of at least $3 - o(1)$ for the related machine case even if the value of OPT is known in advance. This lower bound stands in contrast to the 2-competitive algorithm in [1] for the case where tasks are permanent and the optimal off-line maximal load is known in advance, as well as to the 2-competitive algorithm for the unknown-duration case where the machines are identical [7].

Theorem 4.3 The competitive factor c of any on-line algorithm for the related machines case satisfies $c \geq 3 - o(1)$.

Proof: Assume that there exists an on-line algorithm whose competitive ratio is at most $c < 3$. We define a sequence z_i based on which we define a set of machines and a sequence of task arrivals and departures that will contradict the claim that the competitive ratio is bounded by c. Let $z_0 = 0$, $z_1 = 1/(1+c)$ and

$$z_{i+1} = z_i + \frac{1 - z_{i-1}}{1+c} .$$

Lemma 4.1 For $c < 3$ there exists a positive k such that $z_k \geq 1$.

Proof: Let $x_t = 1 - z_t$. Thus, $x_0 = 1$, $x_1 = 1 - 1/(1+c)$ and $x_{t+1} = x_t - x_{t-1}/(1+c)$. We need to show that there exists a positive k such that $x_k \leq 0$. Notice that since $c < 3$ the roots of the quadratic equation $x^2 = x - 1/(c+1)$ are complex and have values ϕ_1 and ϕ_2 equal to

$$1/2 \pm \sqrt{1/4 - 1/(1+c)} = \rho e^{\pm i\phi}$$

where $\rho > 0$ and $0 < \phi < \pi/2$. Thus, it follows (formally by induction) that

$$x_t = \alpha_1 \phi_1^t + \alpha_2 \phi_2^t$$

where α_1, α_2 satisfy $\alpha_1 + \alpha_2 = 1$ and $\alpha_1\phi_1 + \alpha_2\phi_2 = 1 - 1/(1+c)$. Hence $\alpha_1, \alpha_2 = \rho_0 e^{\pm i\xi}$ where $\rho_0 > 0$ and $-\pi/2 < \xi < \pi/2$. That implies that

$$x_t = \rho_0 \rho^t (e^{i(\phi t+\xi)} + e^{-i(\phi t + \xi)})$$

and hence $x_k \leq 0$ for $k = \lceil (\pi/2 - \xi)/\phi \rceil$. ∎

Notice that for $0 \leq i \leq k$ the sequence z_i is a strictly monotonic increasing sequence, and hence the differences $z_{i+1} - z_i$ for $0 \leq i \leq k-1$ are strictly greater than 0.

Let M be the set of machines and G be any subset of M. Recall that, by definition, $S(G) = \sum_{i \in G} s_i$ and $S(M) = 1$. For $0 \leq \alpha \leq 1$, we define $f(\alpha) = j$, where $j \geq 1$ is the smallest integer such that $S(\{1, \ldots, j\}) \geq \alpha$. Correspondingly, we define $F(\alpha) = \{1, \ldots, f(\alpha)\}$. Denote by $OFF(G)$ the set of the current tasks that the offline algorithm assigns to machines in G.

We choose the speeds of the machines as follows. Without loss of generality assume that c is a rational number. Since c is rational, the z_i's and the difference between any two of them are rational.

Choose s_1 as the largest rational such that for any $x \in \{1\} \cup \{z_i/3^{k-1} | 0 \leq i \leq k-1\}$, the ratio x/s_1 is an integer. Next if $j = f(z_i)$ for some i, $1 \leq i \leq k-1$, then $s_{j+1} = 3s_j$, else $s_{j+1} = s_j$. We will take $(1 - z_{k-1})/(3^{k-1}s_1) \leq (z_k - z_{k-1})/(3^{k-1}s_1)$ machines with speed $s_k = 3^{k-1}s_1$. (These are the fastest machines.) Observe that the aggregate speed of all the machines is:

$$\sum_{i=1}^{n} s_i = (z_1/s_1)s_1 + ((z_2 - z_1)/(3s_1))(3s_1) + \cdots + ((1 - z_{k-1})/(3^{k-1}s_1))(3^{k-1}s_1) = 1.$$

Notice that $S(\{1, \ldots, f(z_i)\}) = z_i$ for each $i < k$. We remark that k grows as c approaches 3, and hence the number of machines required for the above construction increases. This is the reason that our lower bound is $3 - o(1)$ instead of 3.

We now construct, in phases, a sequence of tasks that satisfy the following properties:

- At the end of phase i the offline load on each machine is precisely $OPT = 1$. At any time the offline load on any machine is at most 1.

- Let R_i be the subset of $OFF(F(z_i))$ of tasks which are assigned by the online algorithm to machines in the complement of $F(z_i)$ at the end of any phase i. Then $w(R_i) \geq z_i - z_{i-1}$. Intuitively, R_i is the set of tasks that must be assigned to "faster" machines by the online algorithm in order to maintain a load of at most c on each machine.

At the beginning of the first phase, $1/s_1$ tasks of weight s_1 arrive. (Note that by choice of s_1, $1/s_1$ is an integer.) The online algorithm assigns them while maintaining maximum load $\leq c$. Label these tasks in increasing order starting with the tasks on the slowest machine. The offline algorithm assigns the s_n/s_1 lowest labeled tasks to the fastest machine n, the next s_{n-1}/s_1 lowest labeled tasks to machine $n-1$ and so on. (Note that all these numbers are integers.) Thus, at the end of phase 1 the offline load is precisely 1, and hence the first property holds.

For the second property, observe that (i) the aggregate weight of jobs assigned by the online algorithm to machines in $F(z_1)$ is at most cz_1, since it maintains maximum load $\leq c$; (ii) the online algorithm assigns tasks in $OFF(F(z_1))$ to higher numbered machines than it assigns the tasks in the complement of $OFF(F(z_1))$; (iii) the aggregate weight of tasks in the complement of $OFF(F(z_1))$ is $1 - z_1$. Therefore, the aggregate weight of tasks of $OFF(F(z_1))$ that are assigned by the online algorithm to machines in $F(z_1)$ is at most $cz_1 - (1 - z_1) = 0$. Hence all tasks in $OFF(F(z_1))$ are assigned by the online algorithm to machines in the complement of $F(z_1)$, and thus $w(R_1) = z_1 = z_1 - z_0$.

Assume that the above two properties hold at the end of phase i where $1 \leq i \leq k-2$. In phase $i+1$, all the tasks in the complement of $OFF(F(z_i))$ depart. Using the first property, the cumulative weight of tasks removed is $1 - z_i$. Next, $(1 - z_i)/s$ tasks of weight $s = s_{f(z_i)+1}$ arrive. Note that $s = 3s_{f(z_i)}$. Hence, the online algorithm must assign the new arriving tasks to machines whose speed is at least s, otherwise by the choice of speeds the load would be at least 3. Again label these new tasks in increasing order starting with the tasks on the slowest machine. The offline algorithm assigns the tasks with the s_n/s lowest labels to machine n, the tasks with the next s_{n-1}/s lowest labels to machine $n-1$ and so on satisfying the first property.

It is left to prove the second property. Define $H_i = F(z_{i+1}) - F(z_i)$. Let T_i be the subset of R_i of tasks assigned by the online algorithm to machines in H_i. (Observe that these tasks have not departed in the beginning of the phase.) Similarly to the above, observe that (i) since the online algorithm keeps the load of any machine at most c, the cumulative weight of the new tasks it assigns in this phase to machines in H_i is at most $c(z_{i+1} - z_i) - w(T_i)$; (ii) tasks in $OFF(H_i)$ (i.e., tasks assigned in this phase by the offline to machines in H_i) are assigned by the online to higher numbered machines than the other tasks arriving in this phase (i.e., tasks in the complement of $OFF(F(z_{i+1}))$; (iii) the aggregate weight of tasks in the complement of $OFF(F(z_{i+1}))$ is $1 - z_{i+1}$. Therefore, the cumulative weight of the tasks of $OFF(H_i)$ that the online algorithm could assign to machines in H_i is at most $\max\{0, (c(z_{i+1} - z_i) - w(T_i)) - (1 - z_{i+1})\}$. The last expression simplifies to $\max\{0, z_i - z_{i-1} - w(T_i)\}$, since by the definition of z_i it follows that for all $1 \leq i \leq k-1$,

$$z_i - z_{i-1} = c(z_{i+1} - z_i) - (1 - z_{i+1}) \tag{2}$$

Since the total weight of tasks of $OFF(H_i)$ is $z_{i+1} - z_i$, it follows from the above that the aggregate weight of tasks of $OFF(H_i)$ that are assigned by the online to machines in the complement of $F(z_{i+1})$ is at least $(z_{i+1} - z_i) - \max\{0, z_i - z_{i-1} - w(T_i)\}$. On the other hand, the cumulative weight of tasks of $OFF(F(z_i))$ which are assigned to machines in the complement of $F(z_{i+1})$ is clearly $w(R_i) - w(T_i)$ which by the inductive hypothesis is at least $\max\{0, z_i - z_{i-1} - w(T_i)\}$. Thus,

$$w(R_{i+1}) \geq \max\{0, z_i - z_{i-1} - w(T_i)\} + ((z_{i+1} - z_i) - \max\{0, z_i - z_{i-1} - w(T_i)\})) = z_{i+1} - z_i,$$

which completes the proof of the inductive assumption.

Now assume that $z_{k-1} < 1$ and $z_k \geq 1$. Consider the end of phase $k-1$. During phase $k-1$, tasks with aggregate weight $1 - z_{k-1}$ were assigned to machines in the complement of F_{k-1} by the off-line algorithm. Therefore, the total weight of tasks currently assigned by the online algorithm to the complement of F_{k-1} is at least $z_{k-1} - z_{k-2} + (1 - z_{k-1})$. By Equation 2 we get that

$$z_{k-1} - z_{k-2} + (1 - z_{k-1}) = (1 + c)(z_k - z_{k-1}) > c(1 - z_{k-1}).$$

Thus we reach a contradiction to the competitive ratio $c < 3$, since $1 - z_{k-1}$ is the aggregate speed of the machines in the complement of F_{k-1}. ■

5 Open Problems

This paper raises several open problems and we would like to mention a few of them. The first open problem is to get an on-line algorithm for the case of unrelated machines with tasks with unknown duration. It is clear that the competitive ratio of such an algorithm would be $\Omega(\sqrt{n})$ but no algorithm with $o(n)$ competitive ratio is known. Secondly, it would be interesting to close the gap between the upper and lower bounds on the competitive ratio for the related machines case with unknown tasks duration. The upper bound presented here for the case when the value of OPT is known is 5 while the lower bound is 3. Moreover, do we need to lose the additional factor of 4 when we do not know the value of OPT? A third and maybe most important problem is to find an on-line algorithm with small competitive ratio for the assignment restriction case or even for the unrelated machine case whose competitive ratio does *not* depend on T, the ratio between the maximum to the minimum duration; or alternatively, find a lower bound that does depend on T. This would be interesting when T is superpolynomial in n. The result for unknown duration implies an $O(\sqrt{n})$ competitive algorithm for this case. Is this the best when the durations are known but unbounded?

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 623–631, May 1993.

[2] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of 34th IEEE Annual Symposium on Foundations of Computer Science*, pages 32–40, November1993.

[3] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pp. 321-327, January 1994.

[4] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. In *Proceedings of 33rd IEEE Annual Symposium on Foundations of Computer Science*, pages 218–225, 1992.

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

[6] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1992.

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

[8] 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. *Annals of Discrete Mathematics*, 5:287–326, 1979.

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

[10] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[11] S. Phillips and J. Westbrook. Online load balancing and network flow. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 402–411, 1993.

[12] D. Shmoys, J. Wein, and D.P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the 32nd IEEE Annual Symposium on Foundations of Computer Science*, pages 131–140, 1991.

[13] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.