Improved Bounds for On-line Load Balancing

Matthew Andrews* andrews@math.mit.edu

Michel X. Goemans† goemans@math.mit.edu

Y. Lisa Zhang‡ ylz@math.mit.edu

Department of Mathematics, MIT.

Abstract

We consider the following load balancing problem. Jobs arrive on-line and must be assigned to one of m machines thereby increasing the load on that machine by a certain weight. Jobs also depart on-line. The goal is to minimize the maximum load on any machine, the load being defined as the sum of the weights of the jobs assigned to the machine. The scheduler has also the option of preempting a job and reassigning it to another machine. Whenever a job is assigned or reassigned to a machine, the on-line algorithm incurs a reassignment cost depending on the job. For arbitrary reassignment costs, we present an on-line algorithm with a competitive ratio of 3.5981 against *current load*, i.e. the maximum load at any time is less than 3.5981 times the lowest achievable load at that time. Our algorithm also incurs a reassignment cost less than 6.8285 times the cost of assigning all the jobs. This is the first algorithm with a constant bound both on the competitive ratio and on the reassignment factor. For the special cases in which the reassignment costs are either 1 or proportional to the weights, we present several algorithms which improve upon Westbrook's recent 6-competitive algorithm against current load. Our best competitive ratios are $3 + \varepsilon$ and $2 + \varepsilon$ for the unit and proportional cases respectively.

*Supported by NSF contract 9302476-CCR.

†Supported by NSF contract 9302476-CCR and an NEC research grant.

‡Supported by an NSF graduate fellowship.

1 Introduction

In the on-line load balancing problem, jobs arrive in and depart from a system of m machines on-line and the scheduler has to assign each incoming job to one of the machines. The load on the machine selected is then increased by a certain amount until the job leaves the system. The general goal is to minimize the maximum load. Many variants can be considered: the departure time of a job may or may not be known when a job arrives, the increase in the load may depend on the job/machine pair, jobs may or may not be preempted, and so on.

Load balancing problems arise naturally in various settings in which jobs have to compete for the available resources. Machines may represent various kinds of communication channels with certain bandwidth, and jobs request for bandwidth [1, 2, 3, 4, 5, 6, 12]. Alternatively, machines may represent distributed database platforms, and the jobs may be application programs accessing the database [1, 12]; here, the increase in the load represents the time for an access. The on-line load balancing problem has been much studied in connection with the virtual circuit routing problem [2, 3, 12]. In this problem, the machines are the edges of a network, the jobs are requests for allocating a certain amount of capacity between two given endpoints, and the goal is to minimize the maximum congestion.

As usual, the performance of an on-line algorithm can be measured using the notion of competitive analysis [11]. In the context of load balancing, most analyses that have been performed compared the maximum load at time t of the on-line algorithm to the maximum load at any time between 0 and t, the *peak load*, of the best off-line algorithm presented with the same sequence of job arrivals and departures (see [3, 4, 5]). Westbrook [12], however, pointed out that a competitive analysis against peak load is somewhat unrealistic. Although there was a peak in cellular phone demand on June 17, 1994, one cannot be satisfied with being today within a certain ratio of that peak load. Westbrook instead proposed the more realistic notion of competitiveness against *current load*. An on-line algorithm is said to be α-competitive against current load if, for any time t, the maximum load at time t is at most α times the lowest achievable load at time t for the jobs then in the system. If the model does not allow for job departures, as in [2, 6], there is no difference between current load and peak load. However, in general, competitiveness against current load is a much stronger notion than competitiveness against peak load. Even though the arrival or departure of a job may greatly affect the optimum assignment of jobs to machines, a competitive algorithm against current load needs to ensure that its maximum load before and after the arrival (or departure) is still within a factor of α of the corresponding optimum. Phillips and Westbrook [10] were the first to present competitive analyses against current load, but the distinction between peak and current load was highlighted in Westbrook [12].

The notion of competitiveness against current load is too strong if we don't allow reassignments of jobs (also called job preemptions). Indeed, without reassignments, any algorithm is m-competitive against current load and no algorithm is better than m-competitive against current load [12]. On the other hand, if we allow an arbitrary amount of reassignment then the problem loses its on-line aspect: the scheduler can "simply" reconfigure all the machines at any departure or arrival to match (or closely match) the optimum configuration (although this is an NP-hard problem). There is thus a trade-off between the competitive ratio and the amount of reassignment performed.

The on-line load balancing we consider in this paper can formally be defined as follows. A set of jobs arrive in and depart from a system of m machines on-line, i.e. the arrival and departure times become known only when these events occur. Each job j has a *weight*, w_j, which measures the level of service needed for the job, and a *reassignment cost*, r_j, which measures the cost of assigning (or reassigning) the job to a machine. When job j arrives in the system, the on-line scheduler is informed of its weight and reassignment cost (but not of its departure time or, equivalently, its duration) and has to assign job j to some machine i at a cost of r_j. At any time, the scheduler is allowed to reassign some of the jobs currently in the system to other machines. The corresponding reassignment cost is paid for each reassigned job. At any time t, the *active* jobs are those that have arrived but not yet departed. The load $\lambda_i(t)$ on machine i at time t is defined as the sum of the weights of the active jobs on machine i (for notational convenience, we will later drop the time dependence). The maximum load $\lambda(t)$ at time t is the maximum of $\lambda_i(t)$ over all i. We say that an on-line algorithm is *α-competitive against current load and has a reassignment factor of* r if, for any time t, the maximum load $\lambda(t)$ is at most α times the lowest possible load for the active jobs at time t and the total reassignment cost (including the initial assignments) is at most r times the sum S of the reassignment costs of all jobs that have arrived in the system (regardless of whether or not they have departed). A reassignment factor of 1 therefore means that no reassignment is performed. In the special case in which the reassignment costs are either 1 or proportional to the weights ($r_j = cw_j$ for some constant c), Westbrook [12] was the first to present an on-line algorithm with a constant competitive ratio against current load and a constant reassignment factor. The competitive ratio he proves is 6 and his reassignment factor is 2 for unit reassignments and 3 for proportional reassignments. Westbrook's algorithm is based on partitioning the jobs into *levels* according to their weights and then load balancing each level separately.

The main results presented in this paper are the following.

- For unit reassignment costs, we present a different analysis showing that the competitive ratio of Westbrook's algorithm is in fact 5. We present an on-line algorithm, ZigZag, which treats even and odd levels differently and achieves a competitive ratio of 11/3 against current load. This analysis is shown to be tight. By defining the levels differently we reduce the competitive ratio still further to $1 + 3\sqrt{3}/2 < 3.5981$. ZigZag and its variant have a reassignment factor of 2, the same as Westbrook's algorithm. We also present a different algorithm, the Multi-Snakes algorithm, which for any $\varepsilon > 0$ achieves a competitive ratio of $3 + \varepsilon$ at the expense of a greater reassignment factor, $f(\varepsilon)$.

- When reassignment costs are proportional to the weights, ZigZag has a competitive ratio of 3.5981 against current load and a reassignment factor of 3, improving upon Westbrook's competitive ratio of 6. We also present an algorithm, Snake, which is $(2 + \varepsilon)$-competitive against current load and has a reassignment factor of $g(\varepsilon)$.

- Our most important result is for arbitrary reassignment costs. Our ZigZag-Snakes algorithm is 3.5981-competitive against current load and has a reassignment factor of 6.8285. This is the first algorithm which achieves a constant competitive ratio against current load while simultaneously having a constant reassignment factor. The ZigZag-Snakes algorithm is obtained by combining the ZigZag algorithm with the Snake algorithm. Within each level of the ZigZag algorithm, the jobs are balanced with respect to both their weights and reassignment costs by using the Snake algorithm.

- We also consider the *related* machine case in which every machine has a capacity c_i and the load on any machine is equal to the sum of the weights of the jobs assigned to it divided by its capacity. We present the first algorithm which can handle arbitrary reassignment costs, but our competitive ratio (against current load) is logarithmic in the ratio between the largest and the smallest capacity of any machine.

On-line load balancing is closely related to on-line scheduling in the case where jobs must be assigned immediately upon arrival. Graham's analysis of the greedy algorithm for scheduling [8] on identical machines can easily be extended to load balancing. If we assign any incoming job to the machine whose load is smallest then the resulting algorithm is $(2 - \frac{1}{m})$-competitive against peak load, even in the case where jobs depart [4]. Bartal et al. [7] and Karger et al. [9] give algorithms which are $(2-\varepsilon)$-competitive (ε a small constant) for the case where jobs never depart.

2 Unit Reassignment Costs

Our competitive analyses against current load will be performed by comparing the load of our on-line algorithm at time t to a lower bound on the lowest achievable load for the active jobs. Let J be the set of active jobs and let w_{max} be the maximum weight of a job in J. Then clearly w_{max} is a lower bound on the current load. Since all jobs are distributed to m machines, some machine has to have a load which is at least $\sum_{j\in J}\frac{w_j}{m}$. Hence $LB := \max\{w_{max}, \sum_{j\in J}\frac{w_j}{m}\}$ is a lower bound on the current load.

We first consider the trivial case in which every job has unit weight ($w_j = 1$) and unit reassignment cost ($r_j = 1$). Consider any algorithm which maintains the invariant that the number of jobs on any two machines differs by at most 1, i.e. the number of jobs on every machine is either p or $p+1$ for some p, with possibly no machine with $p+1$ jobs. It is clear that any such algorithm distributes the load optimally. Moreover, the invariant is easy to maintain. If a job arrives, simply assign it to any machine with p jobs. If a job leaves from a machine with $p+1$ jobs, the invariant is already maintained. On the other hand, if a job leaves from a machine with p jobs, say machine i, the algorithm moves any job from a machine with $p+1$ jobs to machine i. We pay one unit of cost for each arrival of a job and at most one unit for each departure. Hence the reassignment factor of this algorithm is at most 2.

For the case in which the jobs have arbitrary weights, Westbrook [12] gave the following generalization of this algorithm and showed that it has a competitive ratio of 6. The jobs are divided into *levels* according to their weights. Level ℓ consists of jobs whose weight w_j satisfies $2^\ell \le w_j < 2^{\ell+1}$. Let J_ℓ be the set of active jobs in level ℓ. We treat each job in J_ℓ as if it had weight 2^ℓ. For level ℓ, let $n^{(\ell)} = |J_\ell|$ be the number of jobs, $w^{(\ell)}_{max}$ be the maximum weight of a job, $w^{(\ell)}_{avg} = \sum_{j\in J_l} w_j/n^{(\ell)}$ be the average weight of the jobs, and $\lambda^{(\ell)}_i$ be the load on machine i due to these jobs. Consider one particular level ℓ (for notational simplicity, we shall drop the superscript denoting the level whenever possible). The algorithm guarantees that every machine has either p or $p+1$ jobs in this level. Since the weights of all these jobs differ by a factor less than two, the load λ_i on any machine of p jobs is at most twice the average load $\sum_{j\in J_\ell}\frac{w_j}{m}$, while the load of the potential $(p+1)$st job can be simply bounded by w_{max}. This implies that $\lambda^{(\ell)}_i \le 2\sum_{j\in J_\ell}\frac{w_j}{m} + w^{(\ell)}_{max}$ for any machine i. Summing over all levels, we derive that the load on any machine is at most,

$$2\sum_{j\in J}\frac{w_j}{m} + w_{max}\left(1+1+\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\ldots\right) = 2\sum_{j\in J}\frac{w_j}{m} + 3w_{max} \le 5LB,$$

showing that Westbrook's algorithm is in fact 5-competitive.

We now describe a more sophisticated variant of this algorithm with an improved performance guarantee. The idea is to take advantage of the flexibility one has in adding and reassigning jobs. The algorithm will be slightly different for even levels and odd levels. Assume that the machines are numbered from 1 to m. For even ℓ, we maintain that machine i has $p+1$ jobs for $i < z$ and p jobs for $z \le i \le m$ for some pointer $z \in \{1,\ldots,m\}$ (depending on ℓ). The initial value of the pointer is 1. When a job arrives it is assigned to machine z, and we increase z by 1. When a job leaves from machine i we reassign a job from machine $z-1$ to machine i and decrease z by 1. (We assume that $m+1 = 1$ and $1-1 = m$.) If ℓ is odd, the value of the pointer is initialized to m, decreased when a job arrives and increased when a job leaves. This means that for odd levels the machines with $p+1$ jobs are those for which $i > z$. We refer to this algorithm as the ZigZag algorithm. See Figure 1 for an illustration. The ZigZag algorithm has the following important feature: If one combines two consecutive levels, the number of jobs on any two machines still differs by at most one.

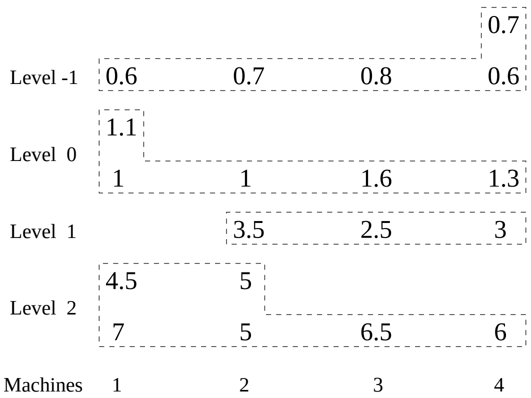


Figure 1: A typical state of the ZigZag algorithm for identical machines.

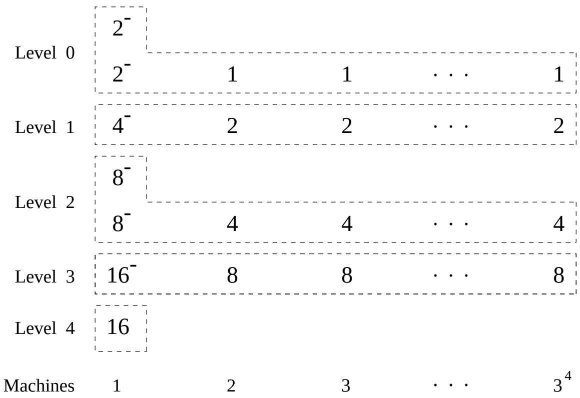


Figure 2: An example to demonstrate the tightness of the competitive ratio 11/3.

In Theorem 3 we show that the competitive ratio for the ZigZag algorithm is 11/3. To prove this result we need the following two lemmas. First, consider some arbitrary level ℓ (again, for simplicity, we drop the superscript). Assume that we have n jobs in level ℓ and every machine has p or $p+1$ jobs (of this level). Let $n = pm + k$, where $0 \le k < m$.

Lemma 1 *If machine i has p jobs, then λ_i is at most $2\sum_{j\in J_\ell} \frac{w_j}{m} - w_{max}\frac{k}{m}$. If machine i has $p+1$ jobs, then λ_i is at most $2\sum_{j\in J_\ell} \frac{w_j}{m} + w_{max}(1-\frac{k}{m})$.*

Proof: If machine i has p jobs then

$$\begin{aligned} \lambda_i &\le w_{max}p = w_{max}\left(p+\frac{k}{m}\right) - w_{max}\frac{k}{m} \le 2w_{avg}\left(p+\frac{k}{m}\right) - w_{max}\frac{k}{m} \\ &= 2\left(\sum_{j\in J_\ell}\frac{w_j}{n}\right)\left(p+\frac{k}{m}\right) - w_{max}\frac{k}{m} = 2\sum_{j\in J_\ell}\frac{w_j}{m} - w_{max}\frac{k}{m}. \end{aligned}$$

On the other hand, if machine i has $p+1$ jobs then $\lambda_i \le w_{max}(p+1)$, and thus λ_i is upper bounded by the above expression increased by w_{max}. □

We now consider what happens in two consecutive levels. This is where the ZigZag algorithm gains.

Lemma 2 *The load on machine i due to jobs in J_ℓ and $J_{\ell+1}$ is at most $2\sum_{j\in J_\ell\cup J_{\ell+1}} \frac{w_j}{m} + w_{max}^{(\ell+1)}$.*

Proof: We consider two cases. First, assume that machine i has simultaneously $p^{(\ell)}+1$ jobs of level ℓ and $p^{(\ell+1)}+1$ jobs of level $\ell+1$. Then it must be the case that $k^{(\ell+1)} + k^{(\ell)} \ge m$ since the pointer for level $\ell+1$ travels in the opposite direction to the pointer for level ℓ. Hence by Lemma 1 we obtain:

$$\begin{aligned} \lambda_i^{(\ell+1)} + \lambda_i^{(\ell)} &\le 2\sum_{j\in J_\ell\cup J_{\ell+1}}\frac{w_j}{m} + w_{max}^{(\ell+1)}\left(1-\frac{k^{(\ell+1)}}{m}\right) + w_{max}^{(\ell)}\left(1-\frac{k^{(\ell)}}{m}\right) \\ &\le 2\sum_{j\in J_\ell\cup J_{\ell+1}}\frac{w_j}{m} + w_{max}^{(\ell+1)} + w_{max}^{(\ell)} - \frac{w_{max}^{(\ell)}\left(k^{(\ell)}+k^{(\ell+1)}\right)}{m} \end{aligned}$$

$$\begin{aligned}&\leq 2\sum_{j\in J_\ell\cup J_{\ell+1}}\frac{w_j}{m}+w_{max}^{(\ell+1)}+w_{max}^{(\ell)}-w_{max}^{(\ell)}\\&= 2\sum_{j\in J_\ell\cup J_{\ell+1}}\frac{w_j}{m}+w_{max}^{(\ell+1)}.\end{aligned}$$

In the second case, machine i has either $p^{(\ell)}$ jobs of level ℓ or $p^{(\ell+1)}$ jobs of level $\ell+1$. Lemma 1 implies that

$$\lambda_i^{(\ell)}+\lambda_i^{(\ell+1)}\leq 2\sum_{j\in J_\ell\cup J_{\ell+1}}\frac{w_j}{m}+w_{max}^{(\ell+1)}.$$

□

Theorem 3 *The* ZigZag *algorithm is* $(11/3)$*-competitive against current load. For unit reassignment costs, the reassignment factor is* 2.

Proof: The bound on the reassignment cost follows from the same bound for the unit weight algorithm. Let $L=\max\{\ell : J_\ell\neq\emptyset\}$. Using Lemma 2, aggregating two levels at a time (starting from level L) and summing over these pairs of levels, we derive that the total load on any machine is at most,

$$\begin{aligned}\lambda_i &\leq \sum_\ell 2\sum_{j\in J_\ell}\frac{w_j}{m}+\sum_{q=0}^{\infty}w_{max}^{(L-2q)}\leq 2\sum_{j\in J}\frac{w_j}{m}+w_{max}^{(L)}\left(1+\frac{1}{2}+\frac{1}{2^3}+\frac{1}{2^5}+\ldots\right)\\&= 2\sum_{j\in J}\frac{w_j}{m}+\frac{5}{3}w_{max}^{(L)}\leq\frac{11}{3}LB.\end{aligned}$$

The first inequality holds because $w_{max}^L\geq 2^L$ and $w_{max}^{(L-2q)}\leq 2^{L-2q+1}$. □

We give the following example to show that the ratio of $11/3$ is tight for the ZigZag algorithm. Fix a parameter s (s even) and suppose that we have $m=3^s$ machines. Consider the following sequence of arrivals: $\{2^s;(m-1)\times 2^{s-1},(2^s)^-;(2^{s-1})^-,\ (m-1)\times 2^{s-2},(2^{s-1})^-;\ldots (m-1)\times 2,4^-;2^-,(m-1)\times 1,2^-\}$. By $(m-1)\times 2^{s-1}$ we mean $m-1$ separate jobs of weight 2^{s-1} and by $(2^s)^-$ we mean a job of weight slightly less than 2^s. The case $s=4$ is illustrated in Figure 2. Using the ZigZag algorithm the load on machine 1 is $2^s\left(2\left(1+\frac{1}{2}+\frac{1}{2^3}+\frac{1}{2^5}+\ldots+\frac{1}{2^{s-1}}\right)\right.$ $\left.+\left(\frac{1}{2^2}+\frac{1}{2^4}+\frac{1}{2^6}+\ldots+\frac{1}{2^{s-2}}\right)\right)$. This expression approaches $\frac{11}{3}2^s$ as s tends to infinity. The jobs can however be placed on the machines so that the current load is 2^s.

We can actually get a competitive ratio better than $11/3<3.6667$ by defining the levels differently. Suppose level ℓ consists of jobs whose weight w_j satisfies $\alpha^\ell\leq w_j<\alpha^{\ell+1}$. By using an analysis identical to that used in proving Theorem 3, we obtain a competitive ratio of $\alpha+1+\frac{1}{\alpha}+\frac{1}{\alpha^3}+\frac{1}{\alpha^5}+\ldots=1+\alpha+\frac{\alpha}{\alpha^2-1}$. We minimize this expression by setting $\alpha=\sqrt{3}$ to give a ratio of $1+3\sqrt{3}/2<3.5981$.

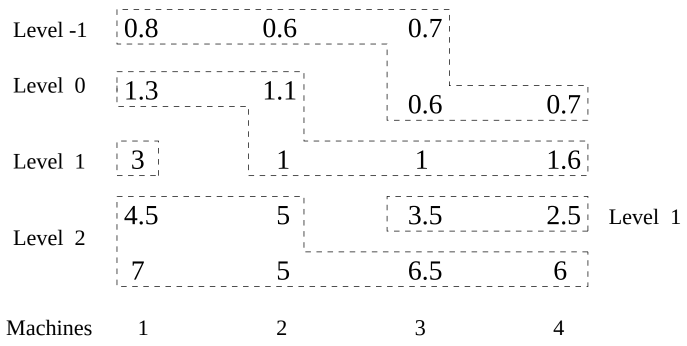
Theorem 4 *For* $r_j=1$*, the* ZigZag *algorithm with* $\alpha=\sqrt{3}$ *is* 3.5981*-competitive against current load and has a reassignment factor of* 2.

3 Proportional Reassignment Costs

In the case of reassignment costs r_j of the form cw_j for some constant c, Westbrook's algorithm [12] as well as the ZigZag algorithm still apply. Since reassignments occur only within a level, the weight of a reassigned job is at most twice the weight of a departing job. As a result, both algorithms incur a reassignment cost upper bounded by $3S$. Theorem 4 thus implies:

Theorem 5 *When $r_j = cw_j$, the* ZigZag *algorithm with $\alpha = \sqrt{3}$ is 3.5981-competitive against current load and has a reassignment factor of 3.*

In this section we present an algorithm with an improved competitive ratio for the case of proportional reassignment costs, at the expense of a somewhat weaker bound on the reassignment cost. As for the ZigZag algorithm, level ℓ consists of jobs whose weight w_j satisfies $\alpha^\ell \leq w_j < \alpha^{\ell+1}$ for a fixed parameter $\alpha > 1$, and each level has a pointer, $z^{(\ell)}$. The difference is in the way assignments and reassignments are performed. The algorithm, which we will call the Snake algorithm, maintains the following property: for any ℓ, there exists a threshold y and a parameter q such that all machines whose index is less than y have a total of $q+1$ jobs from levels $\hat{\ell} > \ell$ while all machines whose index is at least y have a total of q jobs from levels greater than ℓ. This is maintained in the following way. When a level ℓ job arrives, we assign it to machine $i = z^{(\ell)}$ and increase $z^{(\ell)}$ by 1. If level ℓ was empty before the arrival, the pointer $z^{(\ell)}$ would first be initialized to the value of the pointer $z^{(\hat{\ell})}$ of the next non-empty level $\hat{\ell} > \ell$, or to 1 if ℓ is greater than any existing level. Once the job has been assigned to machine i, we find a job of level $\max\{\ell' < \ell : J_{\ell'} \neq \emptyset\}$ on machine i and reassign it by treating it as a new arrival (it will thus be placed on machine $z^{(\ell')}$). Consequently, this will cause a job of level ℓ'' on machine $z^{(\ell')}$ to be reassigned, and so on. When a level ℓ job leaves machine i, we reassign a job of level ℓ on machine $z^{(\ell)}$ to machine i and decrease $z^{(\ell)}$ by 1. We then reassign jobs in levels $\ell' < \ell$ in a manner that is the reverse of that carried out for job arrivals. Thus the arrival or the departure of a level ℓ job will cause all pointers $z^{(\ell')}$ of levels $\ell' \leq \ell$ to be incremented or decremented respectively. We thus maintain a structure similar to the one shown in Figure 3.



If a level 1 job arrives, it will be added to machine 2; this will cause a level 0 job from machine 2 to be reassigned to machine 3, and a level -1 job from machine 3 to be reassigned to machine 4. If a level 0 job on machine 3 leaves, a level 0 job from machine 2 will be moved to machine 3, and a level -1 job from machine 3 will be moved to machine 2.

Figure 3: The Snake algorithm.

Lemma 6 *When $r_j = cw_j$, the reassignment factor of the* Snake *algorithm is $\frac{\alpha^2+2\alpha-1}{\alpha-1} = 3 + \alpha + \frac{2}{\alpha-1}$.*

Proof: When a job of level ℓ arrives, say job j, a job from each level smaller than ℓ will be reassigned. Thus, the reassignment cost caused by the arrival of job j is at most

$$cw_j + c\alpha^\ell + c\alpha^{\ell-1} + c\alpha^{\ell-2} + \ldots = cw_j + c\frac{\alpha^\ell}{1-1/\alpha} \leq cw_j + c\frac{w_j}{1-1/\alpha} = \frac{2\alpha-1}{\alpha-1}r_j.$$

Similarly, the reassignment cost caused by the departure of job j is at most

$$c\alpha^{\ell+1} + c\alpha^\ell + c\alpha^{\ell-1} + \ldots = c\frac{\alpha^{\ell+1}}{1-1/\alpha} \leq \frac{\alpha^2}{\alpha-1}r_j.$$

Hence the total reassignment factor is $\frac{\alpha^2+2\alpha-1}{\alpha-1} = 3 + \alpha + \frac{2}{\alpha-1}$. □

Lemma 7 *The* SNAKE *algorithm is* $(\alpha+1)$*-competitive against current load.*

Proof: Given the way the reassignments are done in the SNAKE algorithm, there exists $p^{(\ell)}$ such that each machine contains either $p^{(\ell)}$ or $p^{(\ell)}+1$ jobs of J_ℓ. Moreover, the machines with $p^{(\ell)}+1$ jobs of level ℓ are precisely those with indices $z^{(\ell+1)}, z^{(\ell+1)}+1, \ldots, z^{(\ell)}-2, z^{(\ell)}-1$. If $z^{(\ell+1)} > z^{(\ell)}$, this sequence has to be interpreted as $z^{(\ell+1)}, \ldots, m, 1, \ldots, z^{(\ell)}-1$, while if $z^{(\ell+1)} = z^{(\ell)}$ there are no machine with $p^{(\ell)}+1$ jobs of level ℓ. Let C denote the set of "crossing" levels, i.e. the levels ℓ for which $z^{(\ell+1)} > z^{(\ell)}$. The key observation we will exploit is the fact that the sequence of machines with $p^{(\ell)}+1$ jobs of level ℓ is immediately followed by the corresponding sequence for level $\ell-1$ and so on. This implies that if machine i has $p^{(\ell)}+1$ jobs of level ℓ and $p^{(\ell')}+1$ jobs of level $\ell' < \ell$, then there must exist a crossing level among $\{\ell', \ell'+1, \cdots, \ell\}$.

We need some more notation. Assume $|J_\ell| = n^{(\ell)} = p^{(\ell)}m + k^{(\ell)}$, where $0 \leq k^{(\ell)} < m$. Fix machine i, and let $D_0 = \{\ell :$ machine i has $p^{(\ell)}$ jobs $\}$ and $D_1 = \{\ell :$ machine i has $p^{(\ell)}+1$ jobs $\}$. By Lemma 1, the load on machine i is at most

$$\begin{aligned}
\lambda_i &\leq \alpha \sum_{j \in J} \frac{w_j}{m} - \sum_{\ell \in D_0} \frac{k^{(\ell)}}{m} w_{max}^{(\ell)} + \sum_{\ell \in D_1} \left(1 - \frac{k^{(\ell)}}{m}\right) w_{max}^{(\ell)} \\
&= \alpha \sum_{j \in J} \frac{w_j}{m} - \sum_{\ell} \frac{k^{(\ell)}}{m} w_{max}^{(\ell)} + \sum_{\ell \in D_1} w_{max}^{(\ell)}.
\end{aligned}$$

Observe that $k^{(\ell)} = z^{(\ell)} - z^{(\ell+1)}$ for $\ell \notin C$ and $k^{(\ell)} = z^{(\ell)} - z^{(\ell+1)} + m$ for $\ell \in C$. We can thus rewrite the upper bound on λ_i as

$$\begin{aligned}
\lambda_i &\leq \alpha \sum_{j \in J} \frac{w_j}{m} - \sum_{\ell} \frac{z^{(\ell)} - z^{(\ell+1)}}{m} w_{max}^{(\ell)} - \sum_{\ell \in C} w_{max}^{(\ell)} + \sum_{\ell \in D_1} w_{max}^{(\ell)} \\
&= \alpha \sum_{j \in J} \frac{w_j}{m} - \frac{1}{m} \sum_{\ell} z^{(\ell)} (w_{max}^{(\ell)} - w_{max}^{(\ell-1)}) - \sum_{\ell \in C} w_{max}^{(\ell)} + \sum_{\ell \in D_1} w_{max}^{(\ell)} \\
&\leq \alpha \sum_{j \in J} \frac{w_j}{m} - \sum_{\ell \in C} w_{max}^{(\ell)} + \sum_{\ell \in D_1} w_{max}^{(\ell)}.
\end{aligned}$$

The term involving $z^{(\ell)}$ has disappeared since $w_{max}^{(\ell)} - w_{max}^{(\ell-1)} > 0$. Now we use the crucial observation that the levels in C (the crossing levels) and in D_1 must alternate. Indeed, if we go through the sequences of machines with $p^{(\ell)}+1$ jobs of level ℓ in order of decreasing levels, we will alternate between machine i and crossings from machine m to machine 1. Therefore $\sum_{\ell \in D_1} w_{max}^{(\ell)} - \sum_{\ell \in C} w_{max}^{(\ell)} \leq w_{max}$ and thus,

$$\lambda_i \leq \alpha \sum_{j \in J} \frac{w_j}{m} + w_{max} \leq (\alpha+1)LB.$$

□

Theorem 8 *When* $r_j = cw_j$*, the* SNAKE *algorithm with parameter* $\alpha > 1$ *is* $(1+\alpha)$*-competitive against current load and has a reassignment factor of* $3 + \alpha + \frac{2}{\alpha-1}$.

We can see that there is a tradeoff between the competitive ratio and the reassignment factor. For example, for $\alpha = 2$, we have a competitive ratio of 3 (better than the ZIGZAG algorithm) and a reassignment factor of 7. The competitive ratio can be made arbitrarily close to two if we are willing to carry out a lot of reassignments. Note that using our lower bound LB we cannot get a competitive ratio better than two. Consider an input which consists of $m+1$ unit weight jobs arriving. In this case $LB = 1 + \frac{1}{m}$ but the optimum current load is 2.

We can also use a variation of the SNAKE algorithm for unit reassignment costs. Fix an integer parameter $\mu \geq 1$ and for all integers s form a snake out of levels $s\mu, \cdots, (s+1)\mu - 1$. The reassignment factor of the resulting MULTI-SNAKES algorithm now becomes $\mu+1$. Using an analysis similar to that of Lemma 7 we obtain,

$$\lambda_i \leq \alpha \sum_{j \in J} \frac{w_j}{m} + w_{max}\left(1 + 1 + \frac{1}{\alpha^\mu} + \frac{1}{\alpha^{2\mu}} + \cdots\right) \leq \left(2 + \alpha + \frac{1}{\alpha^\mu - 1}\right) LB.$$

Theorem 9 *When $r_j = 1$, the* MULTI-SNAKES *algorithm with parameters μ and α is $(2+\alpha+\frac{1}{\alpha^\mu-1})$-competitive against current load and has a reassignment factor of $1+\mu$.*

Notice that the competitive ratio can be made arbitrarily close to 3 but with a corresponding increase in the reassignment factor.

4 Arbitrary Reassignment Costs

In this section we generalize the 3.5981-competitive ZIGZAG algorithm from unit reassignment costs to arbitrary reassignment costs. Once again jobs are divided into levels according to their weights, level ℓ consisting of jobs whose weight w_j satisfies $\sqrt{3}^\ell \leq w_j < \sqrt{3}^{\ell+1}$. As for the ZIGZAG algorithm, two kinds of pointers, depending on the parity of the level number, are used to maintain the structure of Figure 1. However, what happens within a level is different. Indeed, we need to make sure that reassignments within a level are not too costly. For this purpose, within every level, we "load balance" the jobs according to their reassignment costs using the SNAKE algorithm. More precisely, within each level, jobs are divided into *blocks*. For a fixed parameter β, block b consists of jobs in level ℓ whose reassignment cost r_j satisfies $\beta^b \leq r_j < \beta^{b+1}$. We use a pointer for each block to maintain a block structure identical to the level structure of the SNAKE algorithm. (See Figure 4.)

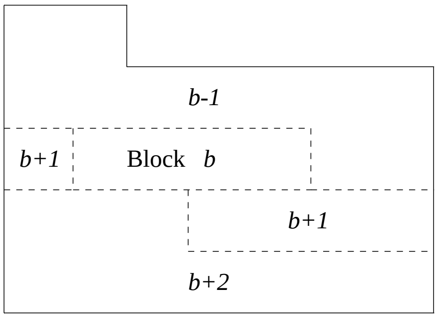


Figure 4: The block structure for level ℓ.

When a level ℓ job j arrives/departs, we use the SNAKE algorithm (using the reassignment costs as weights) to make appropriate reassignments between the blocks of level ℓ. To be more precise, we use the SNAKE algorithm for even levels, and a reverse SNAKE algorithm (in which the pointers go in the opposite direction) for odd levels. The resulting algorithm, which we call the ZIGZAG-SNAKES algorithm, is still 3.5981-competitive. Indeed, the SNAKE algorithm has the important feature that the number of jobs on any two machines differs by at most one and that the machines with one additional job are the lowest indexed machines. Using the analysis of Lemma 6, we also obtain that the reassignment factor is $3 + \beta + \frac{2}{\beta-1}$. The minimum value of this expression is $4 + 2\sqrt{2} < 6.8285$ and is achieved by setting $\beta = 1 + \sqrt{2}$.

Theorem 10 *For arbitrary reassignment costs and arbitrary weights, the* ZIGZAG-SNAKES *algorithm is* 3.5981*-competitive against current load and has a reassignment factor of* 6.8285.

5 Related Machines

In this section we examine the related machine problem in which each machine i has a capacity c_i and the load on machine i is equal to the total weight assigned to it divided by c_i. Without loss of generality we assume that $c_1 \geq c_2 \geq \cdots \geq c_m = 1$. When the reassignment costs are proportional to the weights, Westbrook [12] has derived a 24-competitive algorithm against current load with a reassignment factor of 4. We shall consider arbitrary reassignment costs and propose a competitive algorithm against current load whose competitive ratio depends logarithmically on the largest capacity c_1.

We first present a Greedy algorithm for the special case of unit weight jobs. When a job arrives, Greedy assigns it to the machine whose resulting load is minimized. Thus, if there are n jobs present in the system and $T_i(n)$ jobs on machine i, the $(n+1)$st job will be assigned to the (or any) machine i minimizing $(T_i(n)+1)/c_i$. This defines a sequence of machines $\{s_1, s_2, s_3, \cdots\}$ on which jobs are assigned; the ith job is assigned to machine i. If we have n jobs in the system and a job leaves from machine i, Greedy then reassigns a job from machine s_n to machine i to fill in the gap. This ensures that the number of jobs on any machine is as if $n-1$ jobs have arrived and no departure has occurred. Thus, the number $T_i(n)$ of jobs on machine i only depends on the total number n of active jobs but not on the sequence of arrivals and departures. Observe that the reassignment factor of Greedy is 2.

Theorem 11 *For unit weight jobs,* Greedy *is optimal.*

Proof: Without loss of generality we assume that n jobs have arrived and no jobs have left. The Greedy algorithm assigns them to machines $s_1, s_2, \cdots, s_n$. Let the number of jobs on machine i be $T_i(n)$ (or simply T_i for notational convenience). We show that the following two identities hold for $1 \leq i \leq m$:

$$\frac{T_i}{c_i} \leq \frac{T_{s_n}}{c_{s_n}}, \qquad (1)$$

$$\frac{T_i+1}{c_i} \geq \frac{T_{s_n}}{c_{s_n}}. \qquad (2)$$

To see that identity (1) holds, we consider the latest time at which a job, say j, was assigned to machine i. The construction of the sequence $\{s_1, s_2, \cdots\}$ guarantees that the resulting load (including j) on machine i is no bigger than that on machine s_n, i.e. $\frac{T_i}{c_i} \leq \frac{T_{s_n}}{c_{s_n}}$, otherwise j would have been assigned to machine s_n. If identity (2) did not hold for some i, i.e. $\frac{T_i+1}{c_i} < \frac{T_{s_n}}{c_{s_n}}$, then the nth job would have been assigned to machine i by the Greedy algorithm. Hence identity (2) holds for all i.

We now show that the optimum load λ_{opt} is equal to $\frac{T_{s_n}}{c_{s_n}}$. This is all that we need since identity (1) implies that the load due to Greedy is $\frac{T_{s_n}}{c_{s_n}}$. If $\lambda_{opt} < \frac{T_{s_n}}{c_{s_n}}$, then in an optimum assignment machine s_n must have fewer than T_{s_n} jobs and so some machine $i \neq s_n$ must have at least $T_i + 1$ jobs. But identity (2) implies that $\frac{T_i+1}{c_i} \geq \frac{T_{s_n}}{c_{s_n}} > \lambda_{opt}$. Hence we have a contradiction. □

In the case of arbitrary job weights, Aspnes et al. [2] have shown that the generalization of Greedy is $\Theta(\log m)$-competitive if jobs never depart. Considering our model in which jobs depart, we divide the jobs into levels according to their weights. As in the identical machine case, level ℓ consists of jobs whose weight w_j satisfies $2^\ell \leq w_j < 2^{\ell+1}$. Let J_ℓ be the set of jobs in level ℓ, $n^{(\ell)} = |J_\ell|$ and $L = \max\{\ell : J_\ell \neq \emptyset\}$. We treat jobs in J_ℓ as if they had weight 2^ℓ and assign them to machines using Greedy. We call this algorithm the Greedy ZigZig algorithm. The reassignment factor of Greedy ZigZig is 2 if $r_j = 1$ for all j, and 3 if r_j is proportional to w_j. If we have arbitrary reassignment costs then we can modify the algorithm by dividing each level into blocks based on reassignment cost as for the ZigZag-Snakes algorithm. The highest block will have jobs on machines $s_1, s_2, \cdots, s_k$, the next block on machines $s_{k+1}, \cdots$, and so on. We omit the details. We now bound the competitive ratio (of either version of the algorithm). Let $\tau = \lceil \log c_1 \rceil + 1$.

Theorem 12 GREEDY ZIGZIG *is* $(6+2\tau)$*-competitive against current load. Its reassignment factor is* 2 *for unit reassignments,* 3 *for proportional reassignments, and* 6.8285 *for arbitrary reassignments.*

Proof: The reassignment factors follow from the identical machine case. Let $C = \sum_{1 \le i \le m} \lfloor c_i \rfloor$ and $n^{(\ell)} = p^{(\ell)} C + k^{(\ell)}$, where $0 \le k^{(\ell)} \le C - 1$. Let $\lambda_{opt}(k)$ be the (optimum) load resulting from the assignment of k unit weight jobs by GREEDY. Observe that $\lambda_{opt}(k) \le 1$ for $k \le C$. Since the weight of any level ℓ job is at least 2^ℓ, $\max_{\ell \le L}\{2^\ell \lambda_{opt}(k^{(\ell)})\}$ is a lower bound on the optimum load. Two other valid lower bounds are $(\sum_j w_j)/(\sum_i c_i)$ and $2^L/c_1$ (since the jobs in the last level must be assigned). Hence

$$LB := \max\left\{ \max_{\ell \le L}\{2^\ell \lambda_{opt}(k^{(\ell)})\}, \frac{\sum_j w_j}{\sum_i c_i}, \frac{2^L}{c_1} \right\}$$

is a lower bound on the current optimum load.

The load on each machine due to the first $p^{(\ell)} C$ jobs is at most $w_{max}^{(\ell)} p^{(\ell)}$ and the load due to the remaining $k^{(\ell)}$ jobs is at most $2^{\ell+1} \lambda_{opt}(k^{(\ell)})$. Hence summing over all levels, we obtain that the load on machine i, λ_i, satisfies,

$$\begin{aligned}
\lambda_i &\le \sum_{\ell \le L} \left(w_{max}^{(\ell)} p^{(\ell)} + 2^{\ell+1} \lambda_{opt}(k^{(\ell)}) \right) \\
&\le \sum_{\ell \le L} 2 w_{avg}^{(\ell)} \frac{n^{(\ell)}}{C} + \sum_{\ell = L-\tau+1}^{L} 2^{\ell+1} \lambda_{opt}(k^{(\ell)}) + \sum_{\ell \le L-\tau} 2^{\ell+1} \lambda_{opt}(k^{(\ell)}) \\
&\le 2 \frac{\sum_j w_j}{C} + 2\tau LB + \sum_{\ell \le L-\tau} 2^{\ell+1} \\
&\le 4LB + 2\tau LB + \frac{2^L}{c_1} \left(1 + \frac{1}{2} + \frac{1}{2^2} + \dots \right) \\
&\le 4LB + 2\tau LB + 2LB \\
&= (6 + 2\tau) LB,
\end{aligned}$$

where we have repeatedly used the definition of the lower bound LB, and the fact that $\sum_i c_i < 2C$ since $c_i < 2\lfloor c_i \rfloor$. □

Using different techniques, [1] obtains an algorithm with a competitive ratio of 32 and a reassignment factor of 79.5. Note that this only improves on our algorithm when $c_1 > 2^{12}$.

6 Conclusions

We have presented the first on-line algorithm for load balancing on identical machines which has a constant competitive ratio against current load and a constant reassignment factor. We have also improved upon Westbrook's competitive ratios for the cases of unit and proportional reassignment costs.

References

[1] M. Andrews. Constant factor bounds for on-line load balancing on related machines. Unpublished manuscript, 1995.

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

[3] B. Awerbuch, Y.Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 321–330, 1994.

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

[5] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, and O. Waarts. Online load balancing of temporary tasks. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures (WADS 93), Lecture Notes in Computer Science 709*, pages 119–130. Springer-Verlag, 1993.

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

[7] 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*, pages 51–58, 1992.

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

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

[10] 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.

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

[12] J. Westbrook. Load balancing for response time. In *Proceedings of the 3rd Annual European Symposium on Algorithms*, pages 355–368, 1995.