On-line Load Balancing of Temporary Tasks Revisited

Tak-Wah Lam Hing-Fung Ting Kar-Keung To Wai-Ha Wong

Department of Computer Science and Information Systems
The University of Hong Kong
Email: {twlam,hfting,kkto,whwong}@csis.hku.hk

Abstract. We study load balancing problems of temporary jobs (i.e., jobs that arrive and depart at unpredictable time) in two different contexts, namely, machines and network paths. Such problems are known as machine load balancing and virtual circuit routing in the literature. We present new on-line algorithms and improved lower bounds.

1 Introduction

In this paper we study on-line algorithms for load balancing of temporary jobs (i.e., jobs that arrive and depart at unpredictable time) in two different contexts, namely, machines and network paths. Such problems are referred to as machine load balancing and virtual circuit routing in the literature (see [4, 12, 16] for a survey). As for the former, we investigate a number of settings, namely, the list model, the interval model and the tree model. Our results show that these settings, though similar, cause the complexity of the load balancing problem to vary drastically, with competitive ratio jumping from $\Theta(1)$ to $\Theta(\log n)$ and to $\Theta(\sqrt{n})$, where n is the number of machines. We further extend the study of these settings to the more general *cluster-based* model. Regarding the virtual circuit routing problem, for networks comprising edges with same capacity, we give the first algorithm with a sub-linear competitive ratio of $O(m^{2/3})$, where m is the number of edges; for networks with arbitrary edge capacities, our algorithm is $O(W^{2/3})$-competitive, where W is the total edge capacity normalized to the minimum edge capacity. We also improve the lower bound from $\Omega(m^{1/4})$ [3, 17] to $\Omega(m^{1/2})$, which is valid even when randomization is allowed.

The remainder of this paper is organized as follows. In the rest of this section we give the background and state our results on machine load balancing and virtual circuit routing. In Section 2 we study the various settings of the machine load balancing problem. In Section 3 we extend our studies to a cluster-based model. In Section 4 we examine the virtual circuit routing problem. Finally, we discuss some open problems in Section 5.

1.1 On-line machine load balancing

We study the following on-line problem. There are n machines with identical speed. Jobs arrive and depart at unpredictable time. Each job comes with a positive load. When a job arrives, it must be assigned immediately to one of the machines for execution in a non-preemptive fashion. At any time, the load of a machine is defined to be the total load of jobs that are currently assigned to that machine and have not yet departed. The objective is to minimize the maximum load of any single machine over all time. As with previous work, we measure the performance of an on-line algorithm in terms of competitive ratio (see [12] for a survey), which is the worst-case ratio of the maximum load generated by the on-line algorithm to the maximum load generated by the optimal off-line algorithm.

The above on-line load balancing problem has been studied extensively in the literature (see e.g., [4–7, 17]). Existing results are distinguished by the presence of restrictions on machine assignment. In the simplest case, every job can be assigned to any machine. It has been known for long that Graham's greedy algorithm is $(2 - o(1))$-competitive [5, 15]. A matching lower bound was obtained recently by Azar and Epstein [6]. In the model with assignment restriction, each job specifies an arbitrary subset of the machines for possible assignment. For this model, Azar, Broder, and Karlin [5] proved that the competitive ratio of any on-line algorithm is $\Omega(\sqrt{n})$, and Ma and Plotkin [17] further simplified the proof. Recently, an algorithm with a matching upper bound was derived by Azar *et al.* [7]. This algorithm is referred to as ROBIN-HOOD.

Notably, the allowance of arbitrary assignment restriction makes the problem significantly harder. It is interesting to investigate the complexities of the settings where the assignment restriction is allowed in a more controllable manner. In particular, Bar-Noy *et al.* [10] initiated the study of the following hierarchical model. The machines are related in the form of a tree. Each job specifies a machine M, so that the algorithm is restricted to choose a machine among the ancestors of M. Bar-Noy *et al.* [10] showed that when the hierarchy is linear (i.e., the list model), $O(1)$ competitive ratio can be achieved. The complexity of the general tree model was left open. In this paper we adapt the result in [17] to show that the tree model actually admits an $\Omega(\sqrt{n})$ lower bound. In other words, the tree model, though more controllable, is not easier to handle than arbitrary assignment restriction in the worst case.

Intuitively, the list model orders the machines according to their capability, and a job specifies the least capable machine that can serve the job. A natural extension of the list model is that a job specifies both the least and the most capable machines (as in many applications, more capable machines would charge more). We call this model the interval model. The previous $O(1)$-competitive algorithm fails to work here. We find that there is an $\Omega(\log n)$ lower bound, and we obtain an $O(\log n)$-competitive algorithm. On the other hand, if a job is allowed to request two or more intervals, we show that every on-line algorithm is $\Omega(\sqrt{n})$-competitive. The lower bounds hold even when randomization is allowed.

This paper also initiates the study of cluster-based assignment restriction, which is a practical extension of machine-based assignment restriction. A *cluster* is a collection of machines with the same functionality. More formally, the cluster model states that each machine belongs to one of k clusters, and each job requests some clusters in which

Model	List	Interval	Two intervals, Tree, Arbitrary restriction
n machines	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\sqrt{n})$
k clusters	$\Theta(1)$	$\Theta(\log k)$	$\Theta(\sqrt{K})$

Table 1: Competitive ratios in different settings of assignment restriction.

any machine can be used to serve the job. Similar to machine-based models, we study clusters related in the form of lists, intervals and trees. The machine-based algorithms can be easily adapted to these settings, giving $O(1)$, $O(\log n)$ and $O(\sqrt{n})$ upper bounds respectively. However, it is more desirable to derive algorithms with competitive ratios depending on k instead of n since in reality, k is much smaller than n. For the list model and the interval model, we observe that the competitive ratios are $\Theta(1)$ and $\Theta(\log k)$, respectively. For the tree model, we obtain an algorithm with competitive ratio $O(\sqrt{K})$ where $K = n/s_{\min}$ and $s_{\min}$ is the number of machines in the smallest cluster. Intuitively, K is the total *normalized* number of machines. Note that $k \leq K \leq n$. If the clusters are of roughly the same size, then K is $O(k)$. This algorithm is actually a generalization of the algorithm Robin-Hood [7] and works even when a job can request any clusters arbitrarily. Table 1 shows a summary of these results. We conjecture that the competitive ratio for the tree model can be improved to $O(\sqrt{k})$. To support this conjecture, we prove that, for trees consisting of two levels, a simple algorithm suffices to be $O(\sqrt{k})$-competitive.

Related Work: Other variants of the machine load balancing problem have also been studied extensively in the literature. They include models in which jobs never depart or have predetermined departure time, and in which jobs can be reassigned [1, 2, 8, 11, 13, 14, 19]. For details, readers can refer to the surveys of Azar [4] and Borodin and El-Yaniv [12].

1.2 On-line virtual circuit routing

The virtual circuit routing problem is a generalization of the machine load balancing problem to the context of high speed networks [2, 3]. The virtual circuit routing problem is defined as follows. We are given a directed graph with m edges. Every edge e is associated with a capacity c_e. A request, which asks for a route from a source to a destination, can arrive at any time and last for an unpredictable period. Each request carries a weight, denoted by w. When a request arrives, an on-line algorithm assigns the request to a path connecting the source to the destination, thereby increasing the load of every edge e along that path by w/c_e until the request terminates. The objective is to minimize the maximum load generated on any single edge over all time. The performance is again measured in terms of competitive ratio.

It is widely known that the $\Omega(\sqrt{n})$ lower bound on the competitive ratio of machine load balancing with assignment restriction [5] can lead to an $\Omega(m^{1/4})$ lower bound on the competitive ratio of the virtual circuit routing problem [3, 17]. This lower bound holds even when all edges have the same capacity. An interesting open problem in the literature is to determine the competitive ratio of the virtual circuit routing problem (see e.g., [4]). Prior to our work, the only related result is the work of Awerbuch *et al.* [3], who sidestepped the lower bound using limited re-routing, producing an $O(\log m)$-competitive algorithm.

In this paper we study the original virtual circuit routing problem and present the first algorithm which is $O(m^{2/3})$-competitive when all edges have the same capacity. The virtual circuit routing problem is more difficult than the machine load balancing problem in the sense that an inappropriate assignment may increase the total load of the network drastically. Our new algorithm is a novel extension of ROBIN-HOOD via a careful trade-off between the load and the length of the paths. We also observe that the previous $\Omega(m^{1/4})$ lower bound [3, 17] can be improved to $\Omega(\sqrt{m})$, which holds even when randomization is allowed. For networks with arbitrary edge capacities, the competitive ratio of our algorithm becomes $O(W^{2/3})$, where W is the total edge capacity normalized to the minimum edge capacity (i.e., $\sum_e c_e/c_{\min}$, where $c_{\min}$ is the minimum edge capacity).

2 Machine-based assignment restriction

In this section we study the competitiveness of the on-line load balancing problem in different models of assignment restriction. In the tree model, machines are nodes of a rooted tree. Each job specifies a machine, and the job can be assigned to any ancestor of the specified machine in the tree. The list model is a special case where machines are nodes of a list. Each job specifies a machine, and the algorithm can assign the job to any machine in the list between the list head and the specified machine. A 5-competitive algorithm for the list model has been known [10]. For the tree model, the best known algorithm is the $O(\sqrt{n})$-competitive algorithm inherited from arbitrary assignment restriction [7].

We define the interval model as an extension of the list model. Each job specifies two machines, and the algorithm may choose any machine in the list between these two machines to serve the job. In this section we show that this extension raises the competitive ratio to $\Theta(\log n)$. The lower bound result holds even if we add the assumptions that the jobs never depart, all jobs have the same load, and randomization may be used. We also show that all algorithms in the tree model are $\Omega(\sqrt{n})$-competitive, even if all jobs have the same load and randomization is allowed. A similar result is obtained when we further extend the interval model to allow two intervals per request. All the randomized lower bounds hold even when the adversary is oblivious [18], i.e. the adversary does not inspect the random choices made by the algorithm.

2.1 The interval model

We first show an $O(\log n)$-competitive algorithm INTERVAL for the interval model. Then we present an $\Omega(\log n)$ lower bound on the competitive ratio of algorithms under this model. To ease our discussion, we assume n is a power of two, and define a number of *aligned intervals* $I(i,j)$, where $i = 1, 2, \ldots, \log n$ and $j = 1, 2, \ldots, 2^{i-1}$. $I(1,1)$ is the interval $[1, n]$. Each interval $I(i,j)$, where $i < \log n$, is partitioned into two equal size

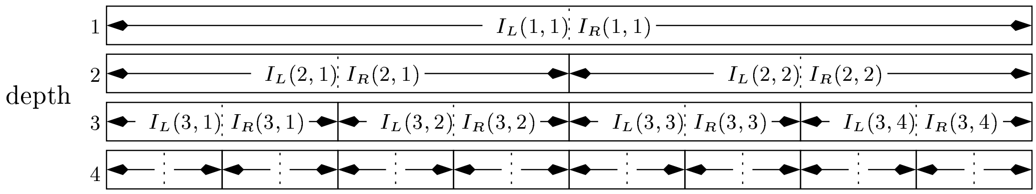

Figure 1: Left and right halves of aligned intervals when $n = 16$. Arrows represent the orientation of the copies of LINEAR running on them.

intervals $I(i+1, 2j-1)$ and $I(i+1, 2j)$. For example, $I(1,1)$ is partitioned into $I(2,1)$ and $I(2,2)$, defined as $[1, n/2]$ and $[n/2+1, n]$ respectively. There are $n-1$ aligned intervals of sizes ranging from 2^1 to $2^{\log n}$. An aligned interval $I(i,j)$ is said to have *depth* i. Note that the size of a depth i interval is $n/2^{i-1}$.

Let LINEAR denote the 5-competitive algorithm shown in [10] for the list model. INTERVAL works by running multiple copies of LINEAR on different subsets of machines, and each interval request gets translated into a list request for one of these copies. More precisely, INTERVAL runs two copies of LINEAR for each aligned interval $I(i,j)$, one on the left half $I_L(i,j)$ of $I(i,j)$ and one on the right half $I_R(i,j)$. Both copies of LINEAR treat the element closest to the middle of $I(i,j)$ as the list head. Figure 1 shows an example. Note that each of the n machines can be assigned a job by $\log n$ copies of LINEAR. Two machines of $I_L(i,j)$ and $I_R(i,j)$ are said to be *mirror-image* of each other if they have the same distance from their list heads.

Suppose an interval request $[l,r]$ arrives. INTERVAL finds the deepest aligned interval $I(i,j)$ which contains the interval $[l,r]$. Since $I(1,1)$ contains all machines, such an aligned interval always exists. Let $L = I_L(i,j) \cap [l,r]$ and $R = I_R(i,j) \cap [l,r]$. Note that, since $I(i,j)$ is the deepest aligned interval, the heads of $I_L(i,j)$ and $I_R(i,j)$ are always included in L and R respectively. (In the boundary case when $i = \log n$, one of L and R may be empty.) INTERVAL translates the interval request $[l,r]$ into the list request L for the copy of LINEAR on $I_L(i,j)$ if $|L| \geq |R|$, and into the list request R for the copy of LINEAR on $I_R(i,j)$ otherwise. INTERVAL then assigns the job to the machine assigned by LINEAR.

To analyse INTERVAL, consider a sequence of interval requests for which the optimal off-line algorithm generates a load of OPT. Recall that INTERVAL translates these interval requests into list requests. Lemma 1 shows that the list requests for any particular copy of LINEAR are not demanding, i.e. admit a good off-line assignment and hence a good assignment under LINEAR. Then we derive the competitive ratio of INTERVAL in Theorem 2.

Lemma 1. *For the list requests for any copy of* LINEAR*, there exists an off-line assignment $\mathcal{S}$ which generates at most $2\,OPT$ load on any machine on which this copy of* LINEAR *runs.*

Proof. Consider the copy of LINEAR running on $I_L(i,j)$ (the case for $I_R(i,j)$ is symmetric). A list request in this interval is translated from an interval request $[l,r]$. Note that, by the way INTERVAL chooses between the left and the right intervals,

$|[l,r] \cap I_L(i,j)| \geq |[l,r] \cap I_R(i,j)|$. In other words, if a machine is in $[l,r] \cap I_R(i,j)$, its mirror image is in $[l,r] \cap I_L(i,j)$.

Let $\mathcal{A}$ denote the optimal off-line algorithm for INTERVAL. $\mathcal{S}$ assigns the list requests for $I_L(i,j)$ as follows. For each list request, $\mathcal{S}$ finds the machine M to which $\mathcal{A}$ assigns the corresponding interval request $[l,r]$. If $M \in I_L(i,j)$, $\mathcal{S}$ assigns the list request to M. Otherwise, $\mathcal{S}$ assigns it to the mirror image of M, which is allowed since that mirror image must be in $[l,r] \cap I_L(i,j)$. Note that any machine in $I(i,j)$ only receives jobs which $\mathcal{A}$ assigns to either the same machine or its mirror image. Since $\mathcal{A}$ generates at most OPT load on any machine, $\mathcal{S}$ generates at most $2\,OPT$ load on any machine. □

Theorem 2. INTERVAL *is* $(10 \log n)$*-competitive.*

Proof. By Lemma 1, for the list requests translated by INTERVAL for a copy of LINEAR, there exists an off-line assignment which generates at most $2\,OPT$ load on any machine. Since LINEAR is 5-competitive, the load of a machine in that copy of LINEAR is at most $10 OPT$. Each machine can receive requests from $\log n$ copies of LINEAR, so the maximum load of any machine is $(10 \log n) OPT$. □

Now we turn to the lower bound, which shows INTERVAL is asymptotically optimal.

Theorem 3. *No on-line algorithm for the interval model is better than* $(\log n/2)$*-competitive. This holds even if (1) jobs never depart, (2) all jobs have the same load, and (3) randomization is allowed.*

Proof. We establish the randomized lower bound using an oblivious adversary. The deterministic bound then follows. Given any randomized on-line algorithm, we construct a sequence of jobs, executing in $\log n$ phases, such that at the end of the p-th phase, there exists an aligned interval $\mathcal{I}_p$ of depth p which has an expected average load of at least p. Therefore, after $\log n$ phases, $\mathcal{I}_{\log n}$ has an expected average load of at least $\log n$, so that one machine has an expected load of at least $\log n$.

In the first phase, n jobs are released, requesting the interval $\mathcal{I}_1 = I(1,1)$. At the end of this phase, the average load of this interval is 1. Inductively, for $1 \leq p < \log n$, suppose $\mathcal{I}_p = I(p,j)$. That is, $I(p,j)$ has an expected average load of at least p after phase p. Since $I(p,j)$ is partitioned into $I(p+1, 2j-1)$ and $I(p+1, 2j)$, one of them has an expected average load of at least p. Let this interval be $\mathcal{I}_{p+1}$. In the $(p+1)$-st phase, $n/2^p$ jobs are released, requesting $\mathcal{I}_{p+1}$. The expected average load of $\mathcal{I}_{p+1}$ after the $(p+1)$-st phase is thus at least $p+1$.

Note that the optimal off-line algorithm assigns all requests of phase p evenly to $\mathcal{I}_p - \mathcal{I}_{p+1}$, so that each machine gets a load of at most 2. Therefore, the competitive ratio against the oblivious adversary is at least $\log n/2$. □

2.2 The tree model

We show that the $O(\sqrt{n})$-competitive algorithm ROBIN-HOOD introduced in [7] is asymptotically optimal for the tree model. Note that if jobs never depart, $O(1)$ competitive ratio can be achieved [10].

Lemma 4. *No on-line algorithm for the tree model (where jobs may depart) is* $(\sqrt{n}-1)$*-competitive. This is true even if all jobs have the same load.*

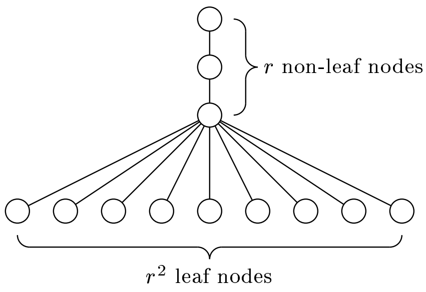

Figure 2: A worst case scenario for the tree model.

Proof. The proof is adapted from the lower bound proof for arbitrary assignment restriction presented in [17]. Consider the tree in Figure 2 containing $r^2 + r$ nodes, with r non-leaf nodes forming a list, and r^2 leaf nodes being children of the tail of this list. The following job sequence ensures that any on-line algorithm assigns at least r jobs to one of the nodes. The job sequence consists of r^2 phases. In the p-th phase, r jobs are released, requesting an ancestor of the p-th leaf. If the on-line algorithm assigns all these jobs to the leaf, we are done. Otherwise, we retain a job assigned to a non-leaf node, and let all other jobs depart. After r^2 phases, the non-leaf nodes must be serving at least r^2 jobs, so one of them must be serving at least r jobs.

In an off-line assignment, only non-departing jobs are assigned to leaf nodes. The maximum load generated is 1. So the on-line algorithm is no better than r-competitive. Since $r > \sqrt{n} - 1$, the lemma follows. □

In the following variant, we show that the competitive ratio cannot be significantly improved even if we allow randomization.

Lemma 5. *No randomized on-line algorithm for the tree model (where jobs may depart) is $(\sqrt{n} - 1)/2$-competitive. This is true even if all jobs have the same load.*

Proof. Again we establish this with an oblivious adversary. Consider the tree constructed in the proof of Lemma 4. The following job sequence ensures that any on-line algorithm has a node with an expected load of $r/2$. The job sequence consists of r^2 phases. In the p-th phase, r jobs are released, requesting an ancestor of the p-th leaf. If the expected load of the p-th leaf is $r/2$, we are done. Otherwise, the expected number of jobs assigned to a non-leaf node is at least $r/2$, so a job randomly selected from those released in this phase has a probability at least $1/2$ to be assigned to a non-leaf node. The adversary randomly selects a job to retain, and lets all other jobs depart. The expected total load of the non-leaf nodes increases by at least $1/2$. After r^2 phases, the expected total load of the non-leaf nodes is at least $r^2/2$, so one non-leaf node must have expected load of at least $r/2$.

In an off-line assignment, only non-departing jobs are assigned to leaf nodes. The maximum load generated is 1. So the on-line algorithm is no better than $r/2$-competitive. Since $r > \sqrt{n} - 1$, the lemma follows. □

In the above arguments, if we number the non-leaf nodes from 1 to r, and the leaf nodes from $r + 1$ to $r^2 + r$, all the jobs have assignment restriction in the form

$\{1, 2, \cdots, r, j\}$. Therefore, if we extend the interval model so that each job can specify two intervals, the same lower bounds hold.

Corollary 6. *For the two-interval model (where jobs may depart), no deterministic on-line algorithm is* $(\sqrt{n} - 1)$*-competitive, and no randomized on-line algorithm is* $(\sqrt{n} - 1)/2$*-competitive. They are true even if all jobs have the same load.*

3 Cluster-based assignment restriction

In reality, assignment restriction is usually used to model the requirement of jobs for some discrete capabilities possessed only by some machines. The number of the distinct sets of capabilities possessed by the machines is usually much smaller than the number of machines. This motivates us to study cluster-based assignment restriction models, an extension in which each machine belongs to one of k clusters. Each job specifies some clusters, so that only machines in these clusters can serve the job. We define the list, interval, two-interval, tree and arbitrary restriction models analogous to the machine-based models we studied in Section 2.

Here are some simple observations. In the extreme case in which each cluster contains only one machine, the cluster-based models reduce to the machine-based models. As a result, all the lower bounds in Section 2 still apply, replacing n by k in the respective bounds. As mentioned earlier, k may be much smaller than n and thus it is more interesting to see whether we can provide better upper bounds in terms of k.

For the list model, the algorithm by Bar-Noy *et al.* [10] is $O(1)$-competitive, independent of n and k. For the interval model, the algorithm in Section 2.1 can be extended to produce an $O(\log k)$-competitive algorithm, matching the lower bound. For the tree model and the arbitrary restriction model, a trivial algorithm which always assigns a job to the most-lightly-loaded machine in the largest cluster specified by the job is $(k + 1)$-competitive. However, it is not clear how we can provide better upper bounds.

In this section we show that the algorithm of Azar *et al.* [7] can be generalized to the cluster-based arbitrary assignment restriction model, producing an $O(\sqrt{K})$-competitive algorithm, where $K = n/s_{\min}$ and $s_{\min}$ is the size of the smallest cluster. Thus, in the case when all clusters are of similar sizes, it is $O(\sqrt{k})$-competitive. Without the assumption of size, we also derive an $O(\sqrt{k})$-competitive algorithm that works in the special case where the clusters are organized as a two-level tree.

In the rest of this paper, we assume that an on-line algorithm for machine load balancing or virtual circuit routing is given in advance the value of OPT, the maximum load generated by the optimal off-line algorithm for the coming inputs. The following lemma states that this algorithm can be converted into one that does not know OPT in advance (instead, it approximates OPT dynamically). This conversion employs the doubling technique [2] and degrades the competitive ratio by only a multiplicative factor of four. Readers can refer to [12] for details.

Lemma 7. [2, 12] *Suppose we have an on-line algorithm for the load balancing problem (or virtual circuit routing problem) that is given* OPT *in advance and is* c*-competitive. Then we can construct an on-line algorithm* $\mathcal{A}$ *that does not know* OPT *in advance and is* $4c$*-competitive.*

3.1 Arbitrary assignment restriction model

In this section we study the cluster-based arbitrary assignment restriction model in which each machine belongs to one of k clusters and each job can request an arbitrary subset of clusters. Denote s_i as the size of the i-th cluster and $s_{\min}$ as the size of the smallest cluster. Let $K = n/s_{\min}$. The lower bound result given in [5] implies an $\Omega(\sqrt{K})$ lower bound for the cluster model. We extend the algorithm ROBIN-HOOD to work in the cluster model, resulting in an $O(\sqrt{K})$-competitive algorithm, called CLUSTER below.

By Lemma 7, we assume that CLUSTER is given OPT in advance. At any time, a cluster is said to be *overloaded* if its average load is greater than $\sqrt{K}\,OPT$ under CLUSTER. For any overloaded cluster, define its *windfall time* to be the last moment it became overloaded.

When a job arrives, CLUSTER assigns the job to a cluster that is not overloaded if possible. Otherwise, CLUSTER assigns it to the cluster with the greatest windfall time. Whenever a job is assigned to a cluster, it is assigned to the machine with the smallest load in that cluster.

Now we analyse the competitive ratio of CLUSTER. The analysis is based on the characteristics of CLUSTER that the average load of any cluster does not increase drastically after its windfall time.

Lemma 8. *The increase of average load on a cluster after its windfall time is at most $\sqrt{K}\,OPT$.*

Proof. Suppose C is a cluster that is overloaded. Let t be its windfall time and let $\mathcal{O}$ be the set comprising C and other overloaded clusters at t. At t, the total load of jobs assigned to clusters in $\mathcal{O}$ is at least $\sum_{i\in\mathcal{O}} s_i \sqrt{K}\,OPT$. All these jobs must be accommodated by the optimal off-line algorithm and thus the total load is at most $n\,OPT = K\,OPT\,s_{\min}$. Therefore, we have $\sum_{i\in\mathcal{O}} s_i \sqrt{K}\,OPT \leq K\,OPT\,s_{\min}$ which implies that $\sum_{i\in\mathcal{O}} s_i \leq \sqrt{K}\,s_{\min}$.

Let $\mathcal{J}$ be the set of jobs that are currently assigned to C by CLUSTER and have arrival time after t. Consider a job in $\mathcal{J}$. Since CLUSTER assigns the job to an overloaded cluster, all feasible clusters, including the cluster assigned by the off-line algorithm, must be overloaded at t. This implies that the total load of jobs in $\mathcal{J}$ is not greater than the total load of jobs assigned to $\mathcal{O}$ by the optimal off-line algorithm, which is at most $\sum_{i\in\mathcal{O}} s_i\,OPT \leq \sqrt{K}\,OPT\,s_{\min}$. The increase in the average load of C after t is at most $\sqrt{K}\,OPT$ since the size of C is at least $s_{\min}$. $\square$

Theorem 9. *With* CLUSTER*, the load of any machine at any time is at most $(2\sqrt{K}+2)OPT$. Thus,* CLUSTER *is $(2\sqrt{K}+2)$-competitive.*

Proof. Consider the moment just after some job is assigned to a machine M by CLUSTER. Suppose C is the cluster containing M. Since CLUSTER assigns the job to the machine with the smallest load in C, it suffices to show that the average load of C is at most $(2\sqrt{K}+1)OPT$. This is immediate when C is not overloaded.

Consider the case when C is overloaded. Before its windfall time, C has an average load of at most $\sqrt{K}\,OPT$. The increase of load due to the job making it overloaded is at most OPT. By Lemma 8, the increase in average load after its windfall time is at most $\sqrt{K}\,OPT$ and the theorem follows. $\square$

3.2 Two-level trees

In this section we present an $O(\sqrt{k})$-competitive algorithm for the model in which the clusters form a tree consisting of two levels. More precisely, the set of machines is partitioned into $k-1$ leaf clusters S_i $(1 \le i \le k-1)$ and a root cluster S_0. Each job specifies one of S_i containing machines which can be used to serve the job, but the algorithm may instead use a machine in S_0. The lower bound results in Section 2.2 can be adapted to this model, giving an $\Omega(\sqrt{k})$ lower bound on the competitive ratio. Here, we prove that a simple algorithm called TwoLevel can achieve a matching upper bound.

By Lemma 7, we assume that TwoLevel is given OPT in advance. If a job arrives which requests a leaf cluster containing a machine with a load less than $\sqrt{k}\,OPT$, TwoLevel assigns the job to that machine. Otherwise, TwoLevel assigns it to a machine in S_0 with the smallest current load.

Theorem 10. TwoLevel *is* $(\sqrt{k}+2)$*-competitive.*

Proof. Since no job creates a load of more than OPT, each machine in a leaf cluster has a load of at most $(\sqrt{k}+1)OPT$. The remainder of the proof establishes that no machine of the root cluster gets a load of more than $(\sqrt{k}+2)OPT$.

Let s_i denote the number of machines in S_i. Consider any particular time t. Denote $n_i(t)$ and $f_i(t)$ as the total load of machines in S_0 at t due to jobs requesting S_i under TwoLevel and the optimal off-line algorithm respectively.

Note that, at the last time when $n_i(t)$ increases, all machines in S_i must have load at least $\sqrt{k}\,OPT$. The off-line algorithm must accommodate all the load of these machines. That is,

$$s_i\sqrt{k}\,OPT + n_i(t) \le (s_0+s_i)OPT \qquad \text{for } 1 \le i \le k-1.$$

On the other hand, the off-line algorithm assigns a load of at least $n_i(t)$ to machines in S_0 or S_i.

$$n_i(t) \le s_i OPT + f_i(t) \qquad \text{for } 1 \le i \le k-1.$$

Eliminating s_i, summing over all i and adding the equality $n_0(t) = f_0(t)$, we have

$$\sqrt{k}\sum_{i=0}^{k-1} n_i(t) \le s_0(k-1)OPT + (\sqrt{k}-1)\sum_{i=0}^{k-1} f_i(t).$$

Note that $\sum_{i=0}^{k-1} f_i(t) \le s_0 OPT$. As a result, $\sqrt{k}\sum_{i=0}^{k-1} n_i(t) \le (k+\sqrt{k}-2)s_0 OPT$; thus, $\sum_{i=0}^{k-1} n_i(t) \le (\sqrt{k}+1)s_0 OPT$. Since TwoLevel always assigns a job to a machine with the smallest load when using S_0, the load of any machine in S_0 at t is less than $(\sqrt{k}+2)OPT$. □

4 Virtual Circuit Routing

In this section we study the virtual circuit routing problem. We are given a directed graph with a set E of m edges. Each edge e is associated with a capacity c_e. Requests arrive and terminate at unpredictable time, each requesting a route from a source to a destination. We need to find a path from the source to the destination, thereby increasing the load of each edge e in the path by w/c_e, where w is the weight specified by the request. Our objective is to minimize, over all time, the maximum load on any edge.

In [2], Aspnes *et al.* studied the problem with the assumption that requests never terminate. With this assumption, they gave an $O(\log m)$-competitive on-line algorithm and proved that no on-line algorithm can have competitive ratio better than $O(\log m)$. We present an on-line algorithm called VC-ROUTING for the more general case where requests may terminate at unpredictable time. It is $O(W^{2/3})$-competitive, where $W = \sum_{e\in E} c_e/c_{\min}$ and $c_{\min}$ is the minimum capacity of the edges. When all edges have identical capacity, the competitive ratio is equivalent to $O(m^{2/3})$. We also prove that any on-line algorithm for the problem is $\Omega(\sqrt{m})$-competitive, even if all the edges have identical capacity and randomization is allowed. Again, the randomized lower bound holds even when the adversary is oblivious.

4.1 The algorithm

VC-ROUTING is a novel adaptation of ROBIN-HOOD, the algorithm which achieves optimal competitive ratio for the machine load balancing problem with arbitrary assignment restriction. The main challenge in the virtual circuit routing problem is that the length of the path chosen by the off-line algorithm may be different from that of the path chosen by the on-line algorithm. As a result, the aggregate load generated by the on-line algorithm can be very different from that generated by the optimal off-line algorithm. In order to control the difference of the aggregate load, VC-ROUTING takes into account the length of paths when assigning a request. Roughly speaking, it prefers relatively short paths, and applies a strategy similar to ROBIN-HOOD only to those short paths. The details are as follows.

By Lemma 7, we assume that VC-ROUTING is given OPT in advance. At any particular time, we say an edge is *overloaded* if its current load is greater than $W^{2/3}\ OPT$. An overloaded path is a path which contains an overloaded edge. For an overloaded edge, we define its *windfall time* as the last moment it became overloaded. The windfall time of an overloaded path is the minimum windfall time of the overloaded edges on the path. We classify the paths into three categories, namely Short, Medium, and Long, depending on whether their length is in the ranges $[1, W^{1/3}]$, $(W^{1/3}, W^{2/3}]$, and $(W^{2/3}, W]$ respectively. For every request j with weight $w(j)$, we say a path is *eligible* if every edge on it satisfies $w(j)/c_e \le OPT$. Since the maximum load generated by the optimal off-line algorithm is at most OPT, an eligible path always exists.

When a new request arrives, VC-ROUTING selects a path among the eligible paths in the order of Short, Medium, Long. Ties among paths in Short are broken as follows. VC-ROUTING selects one that is not overloaded if it exists; otherwise, it selects one with maximum windfall time. For the other two categories, ties are broken arbitrarily.

We now analyse the competitive ratio of VC-ROUTING, by bounding the load on any edge. For edges which are not overloaded, this is trivially bounded by $W^{2/3}OPT$. To bound the load on an overloaded edge e_o, we partition the requests currently assigned to e_o into three sets, depending on the arrival time of the requests and on how the off-line algorithm assigns the requests. Let $\mathcal{J}_1(e_o)$ be the partition containing requests which are assigned to e_o at or before the windfall time of e_o, let $\mathcal{J}_2(e_o)$ be the partition containing those remaining requests which the off-line algorithm assigns to paths in either Medium or Long, and let $\mathcal{J}_3(e_o)$ be the partition containing the other requests. Lemmas 11 and 14 bound the total weight of requests in $\mathcal{J}_2(e_o)$ and $\mathcal{J}_3(e_o)$ respectively, allowing us to conclude the competitive ratio of VC-ROUTING in Theorem 15. To ease the discussion, we denote, at any particular time, $w(e)$ and $w^*(e)$ as the total weight of requests assigned to an edge $e \in E$ by VC-ROUTING and the optimal off-line algorithm, respectively.

Lemma 11. *The total weight of requests in $\mathcal{J}_2(e_o)$ is at most $W^{2/3} OPT\, c_{\min}$.*

Proof. For each edge $e \in E$, the weight of requests assigned by the optimal off-line algorithm is at most $c_e\, OPT$. Therefore, $\sum_{e \in E} w^*(e) \leq \sum_{e \in E} c_e OPT = W\ OPT\, c_{\min}$. Since the length of paths in Medium or Long is greater than $W^{1/3}$, the total weight of requests assigned to paths in Medium or Long by the optimal off-line algorithm is at most $W^{2/3} OPT\, c_{\min}$. The lemma follows since $\mathcal{J}_2(e_o)$ is a subset of these requests. □

Lemma 12. *Every request is assigned by* VC-ROUTING *to a path of length no more than $W^{1/3}$ times of the length of the path assigned by the optimal off-line algorithm.*

Proof. Suppose the optimal off-line algorithm assigns a request to a path in category Short (Medium or Long respectively). The path must be eligible, so VC-ROUTING considered the path during the assignment of the request. Due to the priority of path assignment, VC-ROUTING selects a path in category with priority not less than Short (Medium or Long respectively). The lemma follows from that the ratio of the length of the longest path to that of the shortest path in any category is at most $W^{1/3}$. □

Corollary 13. *At any time t, for any $E' \subseteq E$, $\sum_{e \in E} w^*(e) \geq \sum_{e \in E'} w(e)/W^{1/3}$.*

Proof. Let X be the set of all the requests existing at time t. For any such request j, let $p(j)$ and $p^*(j)$ be the paths to which j is assigned by VC-ROUTING and the optimal off-line algorithm respectively. Note that $\sum_{e \in E} w^*(e) = \sum_{j \in X} w(j)|p^*(j)|$, where $|p^*(j)|$ denotes the number of edges in $p^*(j)$. By Lemma 12, we have $\sum_{j \in X} w(j)|p^*(j)| \geq \sum_{j \in X} w(j)|p(j)|/W^{1/3}$. The latter is equal to $\sum_{e \in E} w(e)/W^{1/3}$. Therefore, $\sum_{e \in E} w^*(e) \geq \sum_{e \in E} w(e)/W^{1/3}$, which is at least $\sum_{e \in E'} w(e)/W^{1/3}$ as E' is a subset of E. □

Lemma 14. *The total weight of requests in $\mathcal{J}_3(e_o)$ is at most $W^{2/3} OPT\, c_{\min}$.*

Proof. Let t be the windfall time of e_o. Denote $\mathcal{O}$ as the set comprising e_o and other overloaded edges at t. Denote J as the set of requests that are assigned to some edges in $\mathcal{O}$ by the optimal off-line algorithm at t. To prove Lemma 14, we first show that the total weight of J is at most $W^{2/3} OPT\, c_{\min}$, and then argue that $\mathcal{J}_3(e_o)$ has a total weight no greater than that of J.

The requests assigned by the optimal off-line algorithm to any edge e have a total weight at most $c_e\, OPT$. Thus, the total weight of J is at most $\sum_{e \in \mathcal{O}} c_e\, OPT$. The latter quantity is bounded as follows. By Corollary 13, $\sum_{e \in E} w^*(e) \geq \sum_{e \in \mathcal{O}} w(e)/W^{1/3}$. Since all edges in $\mathcal{O}$ are overloaded, $\sum_{e \in \mathcal{O}} w(e) > \sum_{e \in \mathcal{O}} c_e\, W^{2/3}\, OPT$. Thus, we have

$$\sum_{e \in \mathcal{O}} c_e\, OPT < W^{-\frac{1}{3}} \sum_{e \in E} w^*(e).$$

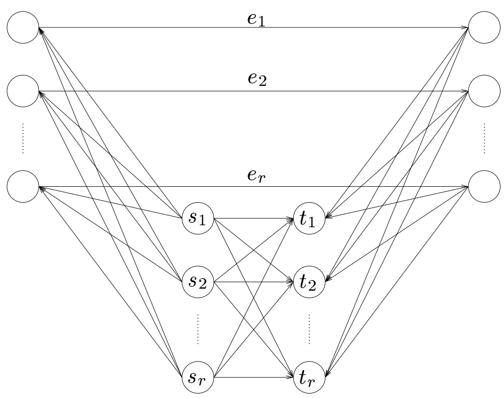

Figure 3: A bad scenario for virtual circuit routing.

On the other hand, $\sum_{e\in E} w^*(e) \le \sum_{e\in E} c_e OPT = W\, OPT\, c_{\min}$. Combining with the previous inequality, we have $\sum_{e\in\mathcal{O}} c_e\, OPT \le W^{2/3}\, OPT\, c_{\min}$. Therefore, the total weight of J is at most $W^{2/3}\, OPT\, c_{\min}$.

According to the priority of path assignment, VC-ROUTING assigns requests in $\mathcal{J}_3(e_o)$ to overloaded paths in Short with maximum windfall time. Therefore, the off-line algorithm must assign requests in $\mathcal{J}_3(e_o)$ to paths which are overloaded at t (as defined in VC-ROUTING); otherwise, VC-ROUTING would assign the requests to those paths instead of the assigned overloaded paths that contain e_o. As a result, the total weight of requests in $\mathcal{J}_3(e_o)$ is no greater than the total weight of J, which is at most $W^{2/3}\, OPT\, c_{\min}$. The lemma follows. □

Theorem 15. *With* VC-ROUTING, *the load of any edge at any time is at most* $(3W^{2/3}+1)OPT$. *Thus* VC-ROUTING *is* $(3W^{2/3}+1)$*-competitive.*

Proof. The theorem holds for those edges that are not overloaded. Consider any overloaded edge e_o. The total weight of requests assigned to e_o before its windfall time is at most $W^{2/3}c_{e_o}OPT$. The weight of the request making it overloaded is no more than $c_{e_o}OPT$. Thus the total weight of requests in $\mathcal{J}_1(e_o)$ is at most $(W^{2/3}+1)c_{e_o}OPT$. Together with Lemmas 11 and 14, the total weight of requests assigned to e_o is bounded by $(W^{2/3}+1)OPT\, c_{e_o} + 2W^{2/3}OPT\, c_{\min}$. Since $c_{e_o} \ge c_{\min}$, the load of e_o is at most $(3W^{2/3}+1)OPT$ and the theorem follows. □

4.2 Lower bound

We show an $\Omega(\sqrt{m})$ lower bound on the competitive ratio of any on-line algorithm for the virtual circuit routing problem.

Lemma 16. *No on-line algorithm for the virtual circuit routing problem is* $(\sqrt{m/3}-1)$*-competitive. This is true even if all edges have the same capacity and all requests have the same weight.*

Proof. Consider the graph in Figure 3, which has $4r$ nodes and $3r^2+r$ edges with identical capacity. The set of source nodes and the set of destination nodes form a complete bipartite graph. Each of the two sets form a bipartite graph with a distinct set of r intermediate nodes. The two sets of intermediate nodes form a bipartite matching. Let E' be the set of edges connecting the intermediate nodes.

We construct a sequence of requests with r^2 phases so that any on-line algorithm assigns at least r requests to one of the edges. In each phase, r requests of unit weight are released with a distinct pair of source and destination. If the on-line algorithm assigns all these requests to the edge connecting the source and the destination, we are done. Otherwise, at least one of these requests is assigned to a path containing an edge in E'. At the end of this phase, all requests except this one terminate. After r^2 phases, r^2 requests remain, each increases the load of one of the edges in E'. Since there are r edges sharing the load, at least one of them has a load of r.

On the other hand, in an off-line assignment, a request is assigned to the edge connecting the source and the destination involved only if it never terminates. The maximum load is 1 at all time. So the on-line algorithm is no better than r-competitive. Since $r > \sqrt{m/3} - 1$, the lemma follows. □

Using the same argument in Lemma 5, the adversary in Lemma 16 can be converted to an oblivious adversary for any randomized algorithm and we have the following corollary.

Corollary 17. *No randomized on-line algorithm for the virtual circuit routing problem is* $(\sqrt{m/12} - 1/2)$*-competitive. This is true even if all edges have the same capacity and all requests have the same weight.*

5 Open problems

This paper leaves several open problems. In Section 3, we give an $O(\sqrt{K})$-competitive algorithm for the cluster-based load balancing problem where K is the total normalized number of machines. Note that K can be much larger than k, the total number of clusters. Thus, it is desirable to design algorithm whose competitive ratio depends only on k. As a first step, we give an $O(\sqrt{k})$-competitive algorithm for the problem when the clusters form a tree of two levels. We conjecture that $O(\sqrt{k})$-competitive algorithm exists for general trees. In Section 4, we give an $O(W^{2/3})$-competitive algorithm for the virtual circuit routing problem where W is the total normalized edge capacity. Also, we derive an $\Omega(\sqrt{m})$ lower bound on the competitive ratio where m is the total number of edges. Note that an $\Omega(\sqrt{W})$ lower bound follows immediately. An open problem is to close the gap between the $O(W^{2/3})$ upper bound and the $\Omega(\sqrt{W})$ lower bound.

Due to practical motivation, we think it is interesting to investigate the variants of the load balancing problem in which the execution of jobs can be delayed. On the other hand, measuring on-line algorithms with respect to their maximum load might not be accurate enough in some cases. It might be desirable to also measure, say, the amount of time when the load of the machines are close to their maximum.

References

[1] M. Andrews, M. X. Goemans, and L. Zhang. Improved bounds for on-line load balancing. In *Proceedings of the Second Annual International Computing and Combinatorics Conference*, pages 1–10, 1996.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

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

[4] Y. Azar. On-line load balancing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442, Springer, pages 178–195, 1998.

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

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

[7] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22:93–110, 1997.

[8] Y. Azar, J. S. Naor, and R. Rom. Competitiveness of on-line assignments. *Journal of Algorithms*, 18:221–237, 1995.

[9] Y. Azar and O. Regev. Off-line temporary tasks assignment. In *Proceedings of the Seventh Annual European Symposium on Algorithm*, pages 161–171, 1999.

[10] A. Bar-Noy, A. Freund, and J. Naor. Online load balancing in a hierarchical server topology. In *Proceedings of the Seventh Annual European Symposium on Algorithm*, pages 77–88, 1999.

[11] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. In *Workshop on Algorithms and Data Structures*, pages 116–125, 1997.

[12] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[13] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, 9(1):91–103, 1980.

[14] G. Galambos and B. J. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993.

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

[16] S. Irani and A. R. Karlin. Online computation. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard problems*, PWS Publishing Company, chapter 13.6, pages 550–555, 1997.

[17] Y. Ma and S. Plotkin. An improved lower bound for load balancing of tasks with unknown duration. *Information Processing Letters*, 62(6):301–303, 1997.

[18] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[19] S. Phillips and J. Westbrook. Online load balancing and network flow. In *Proceedings of the Twenty-Fifth Annual Symposium on Theory of Computing*, pages 402–411, 1993.