# Path-Independent Load Balancing With Unreliable Machines

James Aspnes*† Yang Richard Yang*‡ Yitong Yin*§

**Abstract**

We consider algorithms for load balancing on unreliable machines. The objective is to optimize the two criteria of minimizing the makespan and minimizing job reassignments in response to machine failures. We assume that the set of jobs is known in advance but that the pattern of machine failures is unpredictable. Motivated by the requirements of BGP routing, we consider **path-independent** algorithms, with the property that the job assignment is completely determined by the subset of available machines and not the previous history of the assignments. We examine first the question of performance measurement of path-independent load-balancing algorithms, giving the measure of makespan and the normalized measure of reassignments cost. We then describe two classes of algorithms for optimizing these measures against an oblivious adversary for identical machines. The first, based on independent random assignments, gives expected reassignment costs within a factor of 2 of optimal and gives a makespan within a factor of $O(\log m/\log\log m)$ of optimal with high probability, for unknown job sizes. The second, in which jobs are first grouped into bins and at most one bin is assigned to each machine, gives constant-factor ratios on both reassignment cost and makespan, for known job sizes. Several open problems are discussed.

## 1 Introduction

Given a set of jobs $J = \{1 \dots n\}$ and machines $M = \{1 \dots m\}$, where each job $j$ has a **size** or **processing time** $p_j$, the problem of **load balancing** is to construct an assignment $f : J \to M$ that minimizes the **makespan**, defined as $C^{\max} = \max_{i \in M} \sum_{j \in f^{-1}(i)} p_j$.[1]

We consider a variant of load balancing in which the set of **available** machines $S$ changes over time. In effect, we are solving a sequence of $P||C^{\max}$ scheduling problems where the set of jobs is fixed but the set of machines varies, and our goal is to minimize both the makespan $C^{\max}$ and the **reassignment cost** of moving jobs from one machine to another as new machines become available and old machines leave. As a further complication, we restrict ourselves to **path-independent** algorithms, those that always assign the same jobs to the same machines given a particular set $S$ despite any previous history of assignments. This restriction simplifies the description of an algorithm, since we can just present an assignment for each nonempty set of machines, but it may dramatically increase reassignment costs since we cannot take previous assignments into account. Surprisingly, we show that randomization can lower the expected reassignment cost between any two states of a path-independent algorithm to within a constant factor of optimal, while maintaining a constant approximation ratio to the optimal makespan.

This variant of load-balancing is inspired by the problem of Internet routing using the Border Gateway Protocol (BGP) [12], the *de facto* interdomain routing protocol of the Internet. In BGP, the global Internet is divided into multiple autonomous systems (AS). The ASes exchange routes to reach destinations. Figure 1 shows that AS 0 has $m$ peering ASes 1 to $m$.

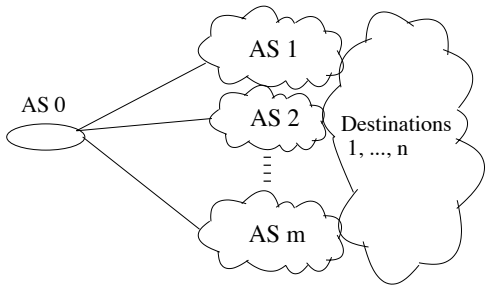


Figure 1: AS 0 learns routes to reach destination $j$ from neighboring ASes 1 to $m$.

Consider AS 0. It learns available routes to reach destinations in the Internet from its neighboring ASes and stores these routes in its routing cache called routing information base (RIB). Among the multiple routes in the RIB to reach the same destination, the AS selects only one and installs it as the route to forward traffic to that destination.

Because network connections are dynamic, the set

*Department of Computer Science, Yale University.

†Supported in part by NSF grants CNS-0305258 and CNS-0435201. Email: aspnes@cs.yale.edu.

‡Supported in part by NSF grants CNS-0435201 and ANI-0238038. Email: yry@cs.yale.edu.

§Supported by NSF grant CNS-0305258. Email: yitong.yin@yale.edu.

[1] In the classic $\alpha|\beta|\gamma$ notation of Graham *et al.*[5], we are considering primarily the $P||C^{\max}$ scheduling problem. Our use of notation largely follows that of [9].

of routes that are available at any time may vary. When the set of available routes changes, BGP will re-select routes from the RIB. If the forwarding route to a destination is changed, BGP will update its peers, and this update can propagate throughout the global Internet. Thus, it is important to minimize the number of reassignments, since they are expensive operations both in terms of bandwidth and router CPU processing [16].

Naturally, some reassignments are unavoidable: when a peering link goes down, all destination prefixes that previously used that link must be assigned elsewhere, and when new links become available, we cannot refuse to reassign destinations to them without violating our load-balancing criterion. So the number of reassignments performed by an algorithm will need to be measured against an estimate of how many reassignments are necessary, which will be obtained by considering the behavior of an ideal algorithm that maintains perfect load balance with minimal reassignment costs.

A further complication is that there are desirable properties on the algorithm to select the forwarding routes. To integrate the algorithm into current BGP implementation architecture, it is desirable to be a preference-based algorithm. Specifically, in current BGP, before a new route is stored in the RIB, it passes through an ingress filter and is assigned a local preference value depending only on the route [3, 15]. For example, a typical implementation technique is to match the route to a set of regular expressions to determine its preference value. Then for each destination, the route with the highest local preference value among all routes to that destination in the RIB is selected as the forwarding route. To conduct more effective routing, in particular load balancing, the AS may coordinate the selection of the forwarding routes for multiple destinations. In [17], the authors proposed guidelines to guarantee the stability of the Internet when ASes coordinate the selection of the forwarding routes for multiple destinations. A key requirement of the guidelines is that to avoid the instability that can be introduced by history dependent assignment and the interactions among the route selection of multiple ASes [6], the route selection algorithm depends only on the routes and not on past history, a property known as **path-independence**. Thus, we will consider only load-balancing algorithms in which the current assignment is determined only by the current set of available machines.

Formally, a path-independent load-balancing algorithm is just a function from $2^M \setminus \{\emptyset\}$ to $M^J$, where the argument represents the (always nonempty) set of available machines and the function value represents an assignment, with the constraint that no job is ever assigned to a machine that is not in the available set. Given such an algorithm $A$, we write $A_S$ for the assignment chosen by $A$ for available set $S$ and $A_S(j)$ for the machine to which job $j$ is assigned. With this notation, the constraint on $A$ is that $A_S(j) \in S$ for all $S$ and $j$.

Because the assignment of jobs to machines is determined by the set of available machines, this set completely determines the state of the system. We thus identify sets of available machines with states of the system and will refer to such sets simply as states hereafter.

For simplicity, we assume that traffic flows are uniform over time. Removing this assumption would be a useful extension of the present work.

**1.1 Our results** The paper contains several novel contributions:

- The new problem of **path-independent load balancing**, motivated by practical issues in BGP routing.
- A measure of **reassignment costs** that takes into account the difference between small and large changes in the set of available machines. This measure is motivated and described in Section 2. We also show that standard competitive analysis techniques [14] are not useful.
- A simple algorithm, based on independent random machine preferences, that achieves low reassignment costs and $O(\log m / \log \log m) \cdot \mathrm{OPT}$ makespan even with unknown job sizes. This algorithm, together with a more general class of **preference-based algorithms** of which it is a member, is described in Section 3.
- A more sophisticated algorithm *BinHash* for jobs of known sizes, based on consolidating jobs into a variable number of bins (depending on the number of available machines) and then hashing the bins. This algorithm achieves constant approximation ratios on both reassignment costs and makespan. It is described in Section 4.

Finally, in Section 5 we discuss possibilities for future work.

**1.2 Related work** It has long been known that a simple greedy algorithm [4] achieves a makespan within a factor of 2 of optimal on identical machines. Much recent work on the problem has focused on on-line load-balancing, where jobs arrive one at a time; see [2] for an example of the current state of the art. Our work is distinguished from this work by the assumption that

jobs are known, but that the set of available machines changes over time.

Kalyanasundaram and Pruhs [7, 8] have considered models of fault-tolerant scheduling for parallel computers; here the issue is that process failure prevents any job assigned to it from being completed and the goal is to maximize the total value of completed jobs subject to release time and completion time constraints. Their results are based on redundant scheduling without preemption and are not directly applicable to our problem.

In the **load rebalancing problem** of Aggarwal *et al.*[1], the makespan of an existing suboptimal assignment of jobs to machines must be improved by moving a limited number of jobs. The paper shows that while the problem in general is NP-complete, good approximation algorithms exist. This work does not directly address the path-independence constraint we consider, but it does demonstrate what is possible without this constraint.

The technique of consolidating jobs in our algorithm for known job sizes is similar to an approach taken by Sibeyn [13] for load-balancing jobs with sizes drawn from a random distribution. Sibeyn's techniques are intended to reduce the variance in job sizes and do not have the low-reassignment-cost properties of our prefix-based method.

There has been substantial work on load-balancing mechanisms based on the **power of two choices**, where jobs pick two (or more) machines at random and choose the more lightly-loaded machine. (See [10] for a survey of these results.) We found through experiments that this approach, while producing excellent makespans, does not appear to yield low reassignment costs: the chains of displaced jobs that migrate to less loaded machines when a new machine becomes available are simply too long.

## 2 Measuring performance

We measure the performance of a path-independent load-balancing algorithm by two criteria: the **makespan** of $A_S$ for each set of available machines $S$, and the **reassignment cost** paid by $A$ when moving from one assignment $A_S$ to a different assignment $A_T$.

The makespan is defined using standard notation as $C_A^{\max}(S) = \max_{i \in S} \sum_{j \in A_S^{-1}(i)} p_j$, the maximum total load on any one machine.

The reassignment cost $r_A(S,T)$ is the number of jobs that move from one machine to another between $A_S$ and $A_T$. Formally, we define $r_A(S,T) = |\{j | A_S(j) \neq A_T(j)\}|$. Note that sizes are not used in computing the reassignment cost; we assume that all jobs incur an equal overhead when reassigned regardless of size. Note also that reassignment cost is symmetric: $r_A(S,T) = r_A(T,S)$ for all $S$ and $T$.

An **execution** of a path-independent load-balancing algorithm is specified as a sequence of states $S_0, S_1, S_2, \ldots$, which may or may not be finite. For a finite execution $S_0 \ldots S_t$, the **total reassignment cost** of an algorithm $A$ is $\sum_{i=0}^{t-1} r_A(S_i, S_{i+1})$. For an infinite execution, the **average reassignment cost** is defined as $\lim_{t\to\infty} \frac{1}{t} \sum_{i=0}^{t-1} r_A(S_i, S_{i+1})$. Note that because $r_A$ is bounded by $n$ this limit always exists.

Because both makespan and reassignment cost are parameterized by states, an algorithm's overall performance may depend strongly on the details of a specific execution. To be able to compare algorithms it will be useful to have summary statistics that describe the performance of an algorithm over all possible executions. Worst-case measures are of limited usefulness: the worst-case reassignment cost of any algorithm on two or more machines is $n$ jobs per step, since we may have only a single available machine at each step that alternates between steps, forcing us to always reassign jobs. Similarly, the worst-case makespan which arises when there is only one available machine does not distinguish algorithms in any way. So instead, we must consider measures that take into account the difficulty of particular executions.

We will show first that a traditional competitive analysis approach does not suffice for this purpose, and then propose an alternative method where the cost of an execution is normalized by a straw-man **reference cost** corresponding to the performance of a particular idealized load-balancing algorithm.

### 2.1 Impossibility of bicriteria-competitive algorithms

The technique of **competitive analysis** [14] compares the cost of a candidate on-line algorithm (which receives the input only step-by-step) in a given history against the cost of an optimal off-line algorithm (which sees the entire input in advance and is typically not computationally bounded). A candidate algorithm is said to have **competitive ratio** $c$ or be $c$-competitive if its cost is at most $c$ times the cost of the optimal algorithm, plus a constant.[2]

For the path-independent load-balancing problem, we are measuring two quantities at each step: the makespan in the new state, and the reassignment cost between the old state and the new state. A natural way to apply competitive analysis to this situation might be to adopt a bicriteria approach and insist that our candidate algorithm be $(\alpha, \beta)$-competitive, where

[2]The constant allows excluding perverse effects of very short executions.

$\alpha$ bounds the ratio between the candidate's makespan in each state and the optimal algorithm's makespan in its corresponding state, and $\beta$ bounds the ratio between the candidate's total reassignment cost over a finite execution and the optimal algorithm's total reassignment cost.[3] We show below that no such algorithm exists for any finite $\alpha$ and $\beta$ in any system with $n \geq 2$ jobs and $m \geq 2$ machines, even if the jobs have identical sizes.

THEOREM 2.1. *Let $A$ be a path-independent load-balancing algorithm for a system with $n \geq 2$ jobs and $m \geq 2$ machines. Then $A$ is not $(\alpha, \beta)$-competitive for any $\alpha$ and $\beta$.*

*Proof.* Without loss of generality, let $m = 2$ (we can always make any additional machines permanently unavailable).

Let $A$ be some candidate algorithm. We will show that $A$ is not competitive with respect to reassignment costs regardless of makespan. Let $i \in \{1, 2\}$ be such that some job is assigned to machine $i$ in $A_{\{1,2\}}$; let $i'$ be the other machine. Now consider an execution in which $S_t = \{1, 2\}$ when $t$ is even and $S_t = \{i'\}$ when $t$ is odd. Because there is some job that is assigned to machine $i$ at even times and not odd times, $A$ pays a reassignment cost of at least 1 per step of the execution. However, an optimal $A^*$ that assigns all jobs to machine $i'$ in all states pays 0. No finite $\beta$ nor additive constant is large enough to overcome in the limit the infinite ratio between $A$ and $A^*$'s reassignment costs.

**2.2 Normalized reassignment costs** We define the **normalized reassignment cost** relative to the costs of an ideal algorithm that always maintains perfect balance. We consider first the case of $n = km!$ identical jobs, where $k$ is some integer; the $m!$ factor ensures that the jobs can be equally divided over any subset of the machines. In this case we can directly calculate the reassignment cost when moving from state $S$ to state $T$.

THEOREM 2.2. *Consider a system of $m$ machines and $n = km!$ identical jobs. Let $A$ be an algorithm that assigns exactly $n/|S|$ jobs to each machine in each state $S$. Then the number of reassignments performed by $A$ going from $S$ to $T$ is at least*

$$r^*(S,T) = n \cdot \left(1 - \frac{|S \cap T|}{\max(|S|, |T|)}\right). \tag{2.1}$$

*Proof.* Consider the set of jobs assigned to machines in $S \setminus T$ by $A_S$; since $A$ assigns exactly $n/|S|$ jobs to each machine in $S \setminus T$, there are $n \cdot |S \setminus T|/|S|$ such jobs total. All of these jobs must be reassigned going from $S$ to $T$.

Conversely, any job assigned to a machine in $T \setminus S$ by $A_T$ must also have been reassigned going from $S$ to $T$. Though these two sets of jobs may overlap, $A$ must reassign at least

$$\begin{aligned} & n \cdot \max\left(\frac{|S \setminus T|}{|S|}, \frac{|T \setminus S|}{|T|}\right) \\ = \; & n \cdot \max\left(\frac{|S \setminus (S \cap T)|}{|S|}, \frac{|T \setminus (S \cap T)|}{|T|}\right) \\ = \; & n \cdot \max\left(1 - \frac{|S \cap T|}{|S|}, 1 - \frac{|S \cap T|}{|T|}\right) \\ = \; & n \cdot \left(1 - \frac{|S \cap T|}{\max(|S|, |T|)}\right). \end{aligned}$$

We will take $r^*$ as the ideal reassignment cost, and measure the reassignment cost of any particular algorithm $A$ as the maximum over all $S$ and $T$ of the ratio of its reassignment cost $r_A(S,T)$ to $r^*(S,T)$.

A justification for this approach is that the $r^*$ lower bound continues to hold in expectation for *any* path-independent algorithm—regardless of whether it distributes jobs evenly or not—provided the machine names are randomly permuted before the algorithm is used and such renaming is undone afterwards. This fact is formally stated in the following theorem. The random permutation of machine names, which is easily implemented by an oblivious adversary, prevents an algorithm from getting lucky by placing all of its jobs on machines that stay available in both $S$ and $T$.

THEOREM 2.3. *For any algorithm $A$ that maps states to assignments, choose a permutation $\rho$ uniformly at random. Let $\rho^{-1}A\rho$ be the algorithm that constructs an assignment for $S$ by applying $A$ to $\rho S$ and then undoing the machine renaming. Then the algorithm $\rho^{-1}A\rho$ reassigns at least $n \cdot \left(1 - \frac{|S \cap T|}{\max(|S|, |T|)}\right)$ jobs in expectation when moving from state $S$ to state $T$.*

*Proof.* Fix some particular job $j$. Let $A_S(j)$ be the machine to which $A$ assigns job $j$ in state $S$. Note that $A_S(j) \in S$. Observe that:

- For any $\langle U, V \rangle$ where $|U| = |S|$, $|V| = |T|$ and $|U \cap V| = |S \cap T|$, there exists some permutation $\rho$ such that $\langle U, V \rangle = \langle \rho S, \rho T \rangle$.

[3] The reason for using a worst-state ratio for makespan but a total ratio for reassignment costs is that the algorithm could spend an arbitrarily long time in a bad state, while reassignments costs accumulate more straightforwardly over transitions. We could also demand that the candidate's reassignment cost is at most $\beta$ times the optimal algorithm's reassignment for each transition, but this only makes life harder for the candidate algorithm and excludes algorithms that depend on amortizing reassignment costs.

- There are $\binom{m}{|S|}\binom{|S|}{|S\cap T|}\binom{m-|S|}{|T|-|S\cap T|}$ equivalence classes after applying permutations to $\langle S,T\rangle$, each of which contains $|S\cap T|!(|S|-|S\cap T|)!(|T|-|S\cap T|)!(m-|S\cup T|)!$ permutations. We denote this value as $e(S,T)$.
- Fix $S$ and $T$. For any $\rho S$, the number of $\rho T$ which contain $A_{\rho S}(j)$, i.e., $|\{\rho T \mid A_{\rho S}(j)\in\rho T\}|$ equals $\binom{|S|-1}{|S\cap T|-1}\binom{m-|S|}{|T|-|S\cap T|}$.

Then the probability that job $j$ stays while the state going from $S$ to $T$

$$\begin{aligned}
&\Pr\rho[A_{\rho S}(j)=A_{\rho T}(j)]\\
=\;& \frac{1}{m!}\sum_{\rho}|\{\rho \mid A_{\rho S}(j)=A_{\rho T}(j)\}|\\
=\;& \frac{1}{m!}\sum_{\rho S}|\{\rho T \mid A_{\rho S}(j)=A_{\rho T}(j)\}|\cdot e(S,T)\\
\le\;& \frac{1}{m!}\sum_{\rho S}|\{\rho T \mid A_{\rho S}(j)\in\rho T)\}|\cdot e(S,T)\\
=\;& \frac{1}{m!}\sum_{\rho S}\binom{|S|-1}{|S\cap T|-1}\binom{m-|S|}{|T|-|S\cap T|}\cdot e(S,T)\\
=\;& \frac{|S\cap T|}{|S|}.
\end{aligned}$$

Symmetrically,

$$\Pr\rho[A_{\rho S}(j)=A_{\rho T}(j)] \;\le\; \frac{|S\cap T|}{|T|}.$$

Therefore, we have $\Pr\rho[A_{\rho S}(j)=A_{\rho T}(j)]\le\frac{|S\cap T|}{\max(|S|,|T|)}$. The probability that job $j$ is reassigned from $S$ to $T$ is thus lower bounded by $(1-\frac{|S\cap T|}{\max(|S|,|T|)})$. The expected total cost is obtained by summing over all $n$ jobs.

## 3 Preference-based algorithms

A **preference-based algorithm** is one in which each job is assigned a permutation of the machines (which may depend on the set of other jobs), and always moves to the first available machine in its permutation. We can think of the operation of a preference-based algorithm as choosing for each job $j$ and state $S$ the machine $i$ in $S$ that minimizes $\sigma_j(i)$, where $\sigma_j$ is the permutation for job $j$.

We consider first the reassignment costs of preference-based algorithms in general and then the makespan of the simple preference-based algorithm where each $\sigma_j$ is a random permutation.

**3.1 Reassignment costs** Preference-based algorithms have the desirable property that they achieve close to the minimum reassignment cost against an oblivious adversary, provided the preferences are permuted randomly before the algorithm is used. This fact is stated formally in the following theorem.

THEOREM 3.1. *For each job $j$, let $\sigma_j$ be a permutation of the machines. Choose a permutation $\rho$ uniformly at random. Then the preference-based algorithm using preferences $\sigma_j\rho$ reassigns an expected $n\cdot\left(1-\frac{|S\cap T|}{|S\cup T|}\right)$ jobs when moving from state $S$ to state $T$.*

*Proof.* Fix some particular job $j$. Going from $S$ to $T$, job $j$ stays put just in case $\min_{i\in S}\sigma_j\rho(i)=\min_{i\in T}\sigma_j\rho(i)$. This occurs precisely when $\min_{i\in S\cup T}\sigma_j\rho(i)$ is achieved by some $i$ in $S\cap T$, i.e. when neither $S$ nor $T$ provides a machine that $j$ prefers to the best machine in their intersection. Since $\sigma_j\rho$ is a random permutation, the probability that $j$ does *not* move is thus precisely $|S\cap T|/|S\cup T|$. The expected number of moves is obtained by taking one minus this probability and summing over all $n$ jobs.

COROLLARY 3.1. *For any preference-based algorithm $A$ and any states $S$ and $T$, $\mathrm{E}[r_A(S,T)]\le 2r^*(S,T)$, where $r^*$ is defined as in Theorem 2.2.*

*Proof.* Observe that

$$\begin{aligned}
\mathrm{E}[r_A(S,T)] &= n\cdot\left(1-\frac{|S\cap T|}{|S\cup T|}\right)\\
&= n\cdot\left(\frac{|S\setminus T|+|T\setminus s|}{|S\cup T|}\right)\\
&= n\cdot\left(\frac{|S\setminus T|}{|S\cup T|}+\frac{|T\setminus S|}{|S\cup T|}\right)\\
&\le n\cdot\left(\frac{|S\setminus T|}{|S|}+\frac{|T\setminus S|}{|T|}\right)\\
&\le 2n\cdot\max\left(\frac{|S\setminus T|}{|S|},\frac{|T\setminus S|}{|T|}\right)\\
&= 2r^*(S,T).
\end{aligned}$$

**3.2 Makespan for random preferences** A typical case of preference-based algorithms is the random preference algorithm, in which a fixed random permutation of machines is picked independently for each job as its preference list.

This randomization implicitly subsumes the random permutation of machine names assumed in Corollary 3.1; thus the corollary applies and random preferences yield an expected $2r^*(S,T)$ reassignments going from state $S$ to $T$. For *identical* jobs, the makespan of random preference when all machines are available is $\Theta(\log m/\log\log m)\cdot\mathrm{OPT}$ *w.h.p.* for

$n = m$, and is $(n/m) + \Theta\left(\sqrt{(n/m)\log m}\right) = \text{OPT} \cdot \left(1 + \Theta\left(\sqrt{(m/n)\log m}\right)\right)$ *w.h.p.* for large $n$, using standard balls-in-bins results (e.g. [11]). Reducing the number of available machines effectively replaces $m$ with $|S|$ in the latter expression.

If the jobs are not identical, we can still argue that the makespan is $O(\log m/\log\log m) \cdot \text{OPT}$, using a generalization to weighted balls of the usual balls-in-bins results. Here we assume that we have a bound on both the weight of the largest ball and the expected weight of the balls assigned to any one bin. These quantities are both bounded by the optimal makespan.

LEMMA 3.1. *Let $n$ balls with non-negative weights $w_1, w_2, \ldots w_n$ be distributed independently and uniformly at random into $b \leq m$ bins. Let $W = \max\left(\max(w_i), \frac{1}{b}\sum_{i=1}^{n} w_i\right)$. Let $X$ be the maximum over all bins of the total weight of the balls in that bin. Then for any fixed $c$, there exists a constant $k$ such that $X \leq kW \lg m/\lg\lg m$ with probability $1 - o(m^{-c})$.*

*Proof.* The proof is a straightforward generalization of the usual argument using characteristic functions for for $n = m$ identical balls.

Let $p = 1/b$, $q = 1 - p$, and $\alpha > 0$. Let $X_i$ be a random variable representing the weight that the $i$-th ball contributes to some particular bin. Let $S = \sum_i X_i$. Then

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha S}\right] &= \mathrm{E}\left[\prod_i e^{\alpha X_i}\right] \\
&= \prod_i \left(qe^0 + pe^{\alpha w_i}\right) \\
&\leq \prod_i \left(1 + pe^{\alpha w_i}\right) \\
&\leq \prod_i \exp\left(pe^{\alpha w_i}\right) \\
&= \exp\left(p\sum_i e^{\alpha w_i}\right).
\end{aligned}$$

Now apply Markov's inequality to get

$$\begin{aligned}
\Pr\left[S > t\right] &= \Pr\left[e^{\alpha S} > e^{\alpha t}\right] \\
&< \exp\left(p\sum_i e^{\alpha w_i}\right) - \exp(\alpha t) \\
&= \exp\left(p\sum_i e^{\alpha w_i} - \alpha t\right).
\end{aligned}$$

We now choose the $w_i$ to maximize this quantity subject to the given constraints $w_i \leq W$ and $\sum_i w_i \leq bW$. Observe that this is equivalent to maximizing $\sum_i e^{\alpha w_i}$. Since $e^{\alpha w_i}$ is convex, this is maximized subject to the sum constraint by setting $w_i = W$ for $i = 1 \ldots b$ and $w_i = 0$ elsewhere. We thus have

$$\begin{aligned}
\Pr\left[S > t\right] &< \exp\left(p\sum_i e^{\alpha w_i} - \alpha t\right) \\
&\leq \exp\left(pbe^{\alpha W} - \alpha t\right) \\
&= \exp\left(e^{\alpha W} - \alpha t\right).
\end{aligned}$$

It is not hard to show that the best choice for $\alpha$ is $\ln(t/W)/W$, giving

$$\begin{aligned}
\Pr\left[S > t\right] &< \exp\left(e^{\ln(t/W)} - (t/W)\ln(t/W)\right) \\
&= \exp\left((t/W) - (t/W)\ln(t/W)\right). \\
&= \exp\left((t/W)(1 - \ln(t/W)\right).
\end{aligned}$$

Finally, set $t/W = k\ln m/\ln\ln m$ to get

$$\begin{aligned}
&\Pr\left[S > t\right] \\
&< \exp\left(\frac{k\ln m}{\ln\ln m}\left(1 - \ln k - \ln\ln m + \ln\ln\ln m\right)\right) \\
&= m^{\left\{k\left(\frac{1-\ln k}{\ln\ln m} - 1 + \frac{\ln\ln\ln m}{\ln\ln m}\right)\right\}} \\
&= m^{-k(1-o(1))}.
\end{aligned}$$

So the probability that any of the $b$ bins exceeds $kW\ln m/\ln\ln m$ is at most $bm^{-k(1-o(1))} \leq m^{1-k(1-o(1))}$, which is $o\left(m^{-c}\right)$ for sufficiently large $k$.

Applying Lemma 3.1 to the random-preference algorithm yields:

THEOREM 3.2. *The random-preference algorithm achieves a makespan of $O(\log m/\log\log m) \cdot OPT$ with high probability for any fixed set $S$ of available machines.*

*Proof.* Apply Lemma 3.1 with $b = |S|$ and $W = \text{OPT}$.

Note that the probability that Theorem 3.2 fails is polynomial in $m$, while the number of possible subsets of available machines is exponential. So it is possible that the makespan for some particular subset $S$ is much worse. This is not a problem if we assume an oblivious adversary, but may become one if a more powerful adversary can choose $S$ after determining the algorithm's preference lists—which it can do by observing the algorithm's behavior.

## 4 Algorithm based on binning and hashing

In this section, we introduce an algorithm called *BinHash* which achieves constant approximation ratios for both makespan and reassignment costs. Unlike the random-preference algorithm of Section 3.2, the makespan bound is deterministic and holds in all states. (The reassignment cost bound is still probabilistic.)

The algorithm is based on the observation that if the number of jobs $n \leq \alpha|S|$ for some constant load factor $\alpha \in (0, 1]$, we can get low reassignment costs and (trivially) optimal makespan by assigning each job to the first empty machine on its preference list. This process—the **hashing** step—is structurally equivalent to hashing with open addressing, and the number of reassignments caused by adding or removing a job is bounded by the length of chains in the corresponding hashing algorithm, which is a constant for fixed $\alpha$.

However, since we cannot assume $n \leq \alpha|S|$ in general, we add an initial **binning** step where jobs are assigned to $\max(1, \lfloor\alpha|S|\rfloor)$ bins, which are then hashed to machines. The binning step sorts jobs by size, and then assigns each job to the bin whose index, expressed in binary, is the longest available suffix of the index of the job. This is a form of round-robin assignment that, by spreading the jobs roughly uniformly among the bins, guarantees a constant-factor approximation of the optimal makespan. At the same time, the number of jobs that move when the number of bins changes is small, since in addition to guaranteeing an even spread by size the binning procedure also guarantees an even spread by job count, and adding or deleting a bin only splits some existing bin or combines two previous bins.

We now give a formal definition of the algorithm. Given $n$ jobs with sizes $p_0 \geq p_1 \geq \cdots \geq p_{n-1}$, and a set $S \in 2^{[m]}/\{\emptyset\}$ of available machines, *BinHash* computes an assignment of $n$ jobs to the machines in $S$ in two stages:

1. Binning stage: Let $b = \max\{\lfloor\alpha|S|\rfloor, 1\}$, where $\alpha \in (0, 1)$ is the load factor parameter of the algorithm. Assign the $n$ jobs to $b$ bins by calling the function $\mathcal{B}(\{p_0, p_1, \ldots, p_{n-1}\}, b) \mapsto \langle B_0^{(b)}, B_1^{(b)}, \ldots, B_{b-1}^{(b)}\rangle$, where $B_i^{(b)}$ is computed by

$$A_i \leftarrow \left\{ j = i + k \cdot 2^{\lceil \log (i+1) \rceil} \;\middle|\; k \geq 0 \wedge j < n \right\}$$

$$B_i^{(b)} \leftarrow A_i - \bigcup_{l=i+1}^{b-1} B_l^{(b)}$$

   for $i = b-1, b-2, \ldots, 0$.

   In other words, each bin $B_i^{(b)}$ gets precisely those jobs whose binary expansions include the binary expansion of $i$ as a suffix, provided there are no higher-numbered bins that capture them first. It is immediate from the definition of $B_i^{(b)}$ that every jobs is assigned to exactly one bin.

2. Hashing stage: The bins are now assigned to the machines in $S$ in order by a *uniform hashing* function $\mathcal{H}(i, S)$, where for each $i$ from 0 to $\lfloor\alpha m\rfloor$, bin $i$ is hashed to the first machine in a fixed random permutation of all $m$ machines that is in $S$ and not occupied by a lower-numbered bin.

We now show that the *BinHash* algorithm achieves low makespan. This follows from the even distribution of the sizes of the bins and the fact that at most one bin is assigned to each machine.

LEMMA 4.1. *Assuming that the optimal makespan of assigning $n$ jobs to $|S|$ machines is OPT, then*

$$\max_{0 \leq i \leq b-1} |B_i^{(b)}| \leq \frac{4n}{\alpha|S|}$$

*and*

$$\max_{0 \leq i \leq b-1} \sum_{j \in B_i^{(b)}} p_j \leq (1 + 2/\alpha) \cdot OPT$$

*for $b = \max\{\lfloor\alpha|S|\rfloor, 1\}$.*

*Proof.* By noting that $B_i^{(b)}$ contains all such $j$ whose longest available suffix in binary expansion is $i$, we can verify that,

$$B_i^{(b)} \subseteq \left\{ j = i + k \cdot 2^{\lfloor \log b \rfloor} \;\middle|\; k \geq 0 \wedge j < n \right\}. \tag{4.2}$$

Therefore, for any $0 \leq i \leq b-1$

$$|B_i^{(b)}| \leq \lfloor \frac{n}{2^{\lfloor \log b \rfloor}} \rfloor \leq \frac{2n}{b} \leq \frac{4n}{\alpha|S|}.$$

As for the loads of bins, note that $OPT \geq \max\{p_0, \frac{1}{|S|}\sum_{j=0}^{n-1} p_j\}$. According to (4.2) and the non-increasing order of $p_j$, we have that, for each $0 \leq i \leq b-1$,

$$\begin{aligned}
\sum_{j \in B_i^{(b)}} p_j &\leq \sum_{k \geq 0} p_{i + k \cdot 2^{\lfloor \log b \rfloor}} \\
&= p_i + \sum_{k \geq 1} \frac{1}{2^{\lfloor \log b \rfloor}} \cdot 2^{\lfloor \log b \rfloor} \cdot p_{i + k \cdot 2^{\lfloor \log b \rfloor}} \\
&\leq p_0 + \sum_{k \geq 1} \frac{1}{2^{\lfloor \log b \rfloor}} \sum_{t=0}^{-1+2^{\lfloor \log b \rfloor}} p_{i + k \cdot 2^{\lfloor \log b \rfloor} - t} \\
&\leq p_0 + \frac{1}{2^{\lfloor \log b \rfloor}} \sum_{j=0}^{n-1} p_j
\end{aligned}$$

$$
\begin{aligned}
&\leq p_0 + \frac{2}{b}\sum_{j=0}^{n-1} p_j \\
&\leq p_0 + \frac{2}{\alpha|S|}\sum_{j=0}^{n-1} p_j \\
&\leq (1+2/\alpha)\cdot OPT.
\end{aligned}
$$

To show low reassignment costs, we must take into account both reassignments caused by moving bins (when a machine leaves or becomes available) and reassignments caused by moving jobs between bins. The former is bounded by the fact that the number of jobs in each bin is roughly equal. The latter requires an analysis of the hashing step. To simplify the argument, we consider first only the case where a single machine becomes unavailable. The case of a new machine becoming available has the same cost by symmetry, and we will show later in Theorem 4.1 that the cost of larger transformations can be expressed in terms of these two cases.

LEMMA 4.2. *For $T \subset S$, and $|T| = |S| - 1$, the algorithm* BinHash *reassigns at most an expected $\frac{4(2-\alpha)n}{\alpha(1-\alpha)|S|}$ jobs when moving from state $S$ to state $T$.*

*Proof.* The total reassignments can be upper bounded by the sum of reassignments caused by binning and the reassignments caused by bin displacements due to hashing.

According to the definition of the binning stage in *BinHash*, we can verify that for any $b \geq 1$,

$$
\begin{aligned}
B^{(b)}_{b-2^{\lfloor \log b \rfloor}} &= B^{(b+1)}_{b-2^{\lfloor \log b \rfloor}} \cup B^{(b+1)}_{b} \\
B^{(b)}_{i} &= B^{(b+1)}_{i} \qquad \text{for } i \neq b - 2^{\lfloor \log b \rfloor}.
\end{aligned}
$$

Thus the reduction of the last bin is the only case for the reassignments caused by the binning process.

For the single machine in $S \backslash T$, there might be a bin $i$ assigned to it in state $S$. When the state changes from $S$ to $T$, bin $i$ will be assigned to some other machine in $T$ by the hashing step. This may lead to a recursive displacement of further bins with larger index than $i$. Since the placement of bin $i$ in $T$ is uncorrelated with the preferences of later bins, the number of such displacements can be bounded by the maximum number of bin displacements caused by inserting an additional bin with a random index into state $T$'s assignment.

Suppose that $n < m$, and let $\Delta(n, m, i)$ denote the expected displacements led by inserting an object with index $i \geq 0$ to a hash table with $m$ slots and $n$ objects by uniform hashing. It is obvious that all objects with priority $h < i$ will not move, therefore it is equivalent to assume that all such objects and their slots are not available to our analysis, thus with probability $\frac{n-i}{m-i}$ object $i$ hits a slot which is occupied by object $i'$ where $i' > i$, and then triggers an expected $\Delta(n, m, i')$ displacements led by inserting object $i'$.

$$
\Delta(n, m, i) = \begin{cases} 1 & i = n \\ 1 + \frac{n-i}{m-i}\Delta(n, m, i') & \\ \qquad \text{for some } i' > i & o.w.. \end{cases}
$$

By induction, we can show that $\Delta(n, m, i) \leq 1/(1 - \frac{n}{m})$ for all $i \geq 0$.

Applying this to the bin assignment, we have at most $\Delta(b-1, |S|-1, i)$ displacements of bins, where $b = \max\{\lfloor \alpha|S| \rfloor, 1\}$, therefore,

$$
\Delta(b-1, |S|-1, i) \leq \frac{1}{1-\alpha}.
$$

The total reassignment costs in terms of jobs is thus bounded from above by:

$$
(1 + 1/(1-\alpha)) \max_{0 \leq i \leq b-1} |B^{(b)}_i| \leq \frac{4(2-\alpha)n}{\alpha(1-\alpha)|S|}.
$$

We now combine the results in Lemmas 4.1 and 4.2 to obtain the full result:

THEOREM 4.1. *The following claims hold for any constant $0 < \alpha < 1$:*

*For any $n > 0$ and state $S$, assuming that the optimal makespan of assigning $n$ jobs to $|S|$ machines is OPT, then the makespan of the assignment obtained by running* BinHash *with the $n$ jobs on $S$ is within $(1 + 2/\alpha) \cdot OPT$.*

*For any states $S$ and $T$, the expected number of reassignments performed by* BinHash *going from $S$ to $T$,*

$$
\mathrm{E}[r_B(S, T)] \leq 2\left(1 + \frac{4(2-\alpha)}{\alpha(1-\alpha)}\right) \cdot r^*(S, T),
$$

*where the expectation is taken over the randomness of uniform hashing, and $r^*$ is defined as in Theorem 2.2.*

*Proof.* Since only one bin is assigned to each machine, the upper bound on makespan follows directly from Lemma 4.1.

For the state transition from $S$ to $T$, where $T \subset S$, according to Lemma 4.2, we have that

$$
\begin{aligned}
\mathrm{E}[r_B(S, T)] &\leq \min\left\{ n, \frac{4(2-\alpha)n}{\alpha(1-\alpha)} \sum_{k=|T|+1}^{|S|} \frac{1}{k} \right\} \\
&\leq \min\left\{ n, \frac{4(2-\alpha)n}{\alpha(1-\alpha)} \ln \frac{|S|}{|T|} \right\} \\
&\leq \left(1 + \frac{4(2-\alpha)}{\alpha(1-\alpha)}\right)(1 - |T|/|S|)n \\
&= \left(1 + \frac{4(2-\alpha)}{\alpha(1-\alpha)}\right) \cdot r^*(S, T).
\end{aligned}
$$

For a general $S$ and $T$, we have that

$$\begin{aligned}
\mathrm{E}[r_B(S,T)] &\leq \mathrm{E}[r_B(S,S\cap T)] + \mathrm{E}[r_B(T,S\cap T)] \\
&\leq \left(1+\frac{4(2-\alpha)}{\alpha(1-\alpha)}\right)\cdot r^*(S,S\cap T) \\
&\quad + \left(1+\frac{4(2-\alpha)}{\alpha(1-\alpha)}\right)\cdot r^*(T,S\cap T)) \\
&\leq 2\left(1+\frac{4(2-\alpha)}{\alpha(1-\alpha)}\right)\cdot r^*(S,T).
\end{aligned}$$

It is not hard to show that the coefficient on reassignment costs is minimized at $\alpha = 2-\sqrt{2} \approx 0.59\ldots$. Here the reassignment costs are bounded by Theorem 2.2 at approximately $48.6 \cdot r^*$ and the makespan at $(3+\sqrt{2})\cdot \mathrm{OPT} \approx 4.142 \cdot \mathrm{OPT}$. There are a number of loose inequalities in the proof of Theorem 2.2, and we believe that a more careful analysis would show that the correct minimum coefficient on reassignment costs is closer to 12 in most cases.

It is also worth noting that the makespan coefficient $1 + 2/\alpha$ can be reduced somewhat by increasing $\alpha$. However, this dramatically increases the reassignment costs as $\alpha$ approaches 1, and the makespan bound never drops below $3 \cdot \mathrm{OPT}$, which is not much better than the bound for $\alpha = 2-\sqrt{2}$. However, it is not out of the question that a more sophisticated algorithm could achieve higher utilization of the available machines without blowing up the reassignment costs.

## 5 Conclusions and future work

We have described a new problem of path-independent load balancing for unreliable machines, where the goal is to minimize makespan while simultaneously minimizing the cost of reassigning jobs from one machine to another subject to the constraint that assignments cannot depend on the previous history. We have also obtained some initial results showing that it is possible to achieve constant approximation ratios to both the optimal makespan and optimal reassignment costs.

However, much work still needs to be done. The proven constant approximation ratios for the *BinHash* algorithm—particularly for makespan—are still quite high, and it would be useful to have an algorithm with better constants.

The assumption of identical machines is a strong one. It is not clear whether our results can be generalized to the case of uniform machines (where different machines have different capacities) or to the even more general case of nonuniform machines (where different jobs may have different effective sizes on different machines). This last case may be particularly important in interdomain routing, as particular flows may be forbidden from traveling over certain pipes by contractual requirements or security concerns.

Finally, we have made very generous assumptions about the nature of the jobs and the nature of the adversary. It would be interesting to determine whether it is possible to solve path-independent load balancing with jobs that vary over time or with a more powerful adversary that can observe and respond to the algorithm's actions.

## References


[1] G. Aggarwal, R. Motwani, and A. Zhu. The load rebalancing problem. *J. Algorithms*, 60(1):42–59, 2006.

[2] S. Albers. On randomized online scheduling. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, pages 134–143, 2002.

[3] M. Caesar and J. Rexford. BGP routing policies in ISP networks. *IEEE Network Magazine, special issue on Interdomain Routing*, Nov/Dev 2006.

[4] R. Graham. Bounds for certain multi-processing anomalies. *Bell Systems Technical Journal*, 45:1563–1581, 1966.

[5] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[6] T. Griffin, F. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Transactions on Networking*, 10:232–243, 2002.

[7] B. Kalyanasundaram and K. Pruhs. Fault-tolerant real-time scheduling. *Algorithmica*, 28(1):125–144, 2000.

[8] B. Kalyanasundaram and K. Pruhs. Fault-tolerant scheduling. *SIAM J. Comput.*, 34(3):697–719, 2005.

[9] J. Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[10] M. Mitzenmacher, A. W. Richa, and R. Sitaraman. The power of two random choices: A survey of techniques and results. In S. Rajasekaran, P. Pardalos, J. Reif, and J. Rolim, editors, *Handbook of Randomized Computing*, volume I, pages 255–305. Springer, 2001.

[11] M. Raab and A. Steger. "Balls into bins"—a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science, Second International Workshop, RANDOM'98, Barcelona, Spain, October 8-10, 1998, Proceedings*, volume 1518 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 1998.

[12] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4), RFC 1771*, Mar. 1995.

[13] J. F. Sibeyn. The sum of weighted balls. Technical Report RUU-CS-91-37, Department of Computer Science, Utrecht University, 1991.

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

[15] J. L. Sobrinho. Network routing with path vector protocols: Theory and applications. In *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[16] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A next-generation interdomain routing protocol. In *Proceedings of ACM SIGCOMM '05*, Philadelphia, PA, Aug. 2005.

[17] H. Wang, H. Xie, Y. R. Yang, L. E. Li, Y. Liu, and A. Silberschatz. Stable egress route selection for interdomain traffic engineering: Model and analysis. In *Proceedings of the 13th International Conference on Network Protocols (ICNP) '05*, Boston, MA, Nov. 2005.