On the Power of Two Choices: Theory and Applications

Dan Ports, Sarah Lieberman, and Galen Pickard
{drkp,sarahl,gpickard}@mit.edu

May 19, 2005

Abstract

The power of two choices is a simple technique for randomized load balancing: to assign an item to a server, simply select two (or more) random servers, and assign it to the least heavily loaded. Nevertheless, it obtains a dramatic improvement over simply choosing one server at random: the maximum load drops by an exponential factor. We review this result, and provide a survey of variations on the technique that either improve its performance or generalize its applicability. In addition, we examine how it can be applied to several common problems.

Contents

1 Overview

Many computer applications require some form of load balancing. Such applications range from assigning tasks to processor queues to assigning data to nodes to peer to peer networks. While these numerous problems vary greatly in many respects, there is a simple technique, first studied in the early 1990s, which can introduce a surprisingly large improvement into many of them. This technique is known as *the power of two choices.*

The basic intuition of this concept can be best explained using the classic *balls and bins* problem, in which there are n balls which have to be placed in n bins. When a ball has to be assigned to a bin, two bins are chosen independently uniformly at random, and the ball is placed in the less full of the two. In the more general case, the ball is placed in the least full of d randomly chosen bins. Although it might seem counterintuitive that such a simplistic method for bin assignment should work well, the maximum load in any bin after n balls are placed is bounded by $\log\log n + O(1)$ with high probability. This is a substantial improvement over the $\Theta\left(\frac{\log n}{\log\log n}\right)$ result that can be obtained by simply choosing one bin uniformly at random. Moreover, due to the minimal amount of computation that goes into making a decision, this result can be achieved using only $O(d)$ computation time per item to be assigned.

This method is easy to implement and produces very satisfying results, so it has been the source for many research improvements since its introduction. There are, for example, many variations on this simple theme, where the basic idea is tweaked in some simple way to achieve improvements in the maximum load bound. In one approach, asymmetric tie-breaking is added by always choosing the leftmost bin. In another, the best options from the previous decision round are remembered, and these are among the options for the next time an item is to be assigned to a bin. Finally, the technique can be extended to more general geometries where each bin is not chosen uniformly, but rather proportional to the size of some area it is responsible for.

We will also examine some applications that have benefited form this technique. It can be used to analyze queue lengths in a server load-balancing problem known as the *supermarket model*, to distribute load more effectively in peer-to-peer networks based on consistent hashing, and to improve hashing in high-performance routers.

2 The Power of Two Choices

The general idea behind the power of two choices is a rather simple one. As originally introduced in a 1994 paper entitled "Balanced Allocations," by Azar, Broder, Karlin, and Upfal [1], and analyzed at great length in Mitzenmacher's 1996 PhD thesis entitled "The Power of Two Choices in Randomized Load Balancing" [2], the power of two choices is a paradigm for load balancing which is capable of an exponential improvement in asymptotic performance over the obvious load balancing strategy. All it entails, however, is that instead of assigning load uniformly at randomly, one should pick two independently and uniformly random possibilities as load arrives, and always choose the less loaded of the two.

While the general principle is simple, the improvement in asymptotic performance is remarkable. It is well known that the obvious load balancing strategy of assigning load uniformly at random produces a maximum load of $\frac{(1+o(1))\log n}{\log\log n}$ with high probability. However, when load is assigned sequentially, and the power of two choices is used, the maximum load drops to $\frac{\log\log n}{\log 2}+O(1)$ with high probability, which is effectively exponentially better than the obvious strategy. The power of two choices paradigm has the drawback that load needs to be assigned sequentially, rather than in parallel. In any system where this is possible, though, it offers a marked improvement in load balancing, while only increasing the computation needed each time load is assigned by at most a constant factor. Generalizing to d random choices instead of 2, we obtain the following result:

Theorem 1. *If n balls are assigned to n bins using the load-balancing technique where each ball chooses d bins, and is placed in the least-loaded of these d bins, the maximum load is $\frac{\log\log n}{\log d}+O(1)$ with high probability.*

Proof. The idea is simple, but a rigorous proof involves many technical details related to handling conditioning, which we shall omit; for the details, see [3] or [2].

We consider the number of bins containing at least i balls. If β_i is a bound on the number of bins with at least i balls, then the probability that any ball becomes the $i+1$st in its bin is bounded by $\left(\frac{\beta_i}{n}\right)^d$ since this requires that each of the d bins chosen is one of the β_i with at least i balls. Using Bernoulli processes to bound the number of such balls, we obtain a relation

$$\beta_{i+1} = O\left(\left(\frac{\beta_i}{n}\right)^d\right) \tag{1}$$

Using (1) and induction (which requires considerable care with conditioning), we can see that the sequence β_i/n decreases exponentially with i, and so after $\log_d \log n + O(1)$ steps, it decreases below 1. Thus, with high probability, there is no bin with load greater than $\log_d \log n + O(1) = \frac{\log\log n}{\log d} + O(1)$ ∎

Remark: Notice that using two random choices instead of one gives a very dramatic improvement from $O\left(\frac{\log n}{\log\log n}\right)$ to $O(\log\log n)$, but adding additional choices only gives a smaller improvement by changing the base of the outer logarithm.

2.1 The Fluid-Limit Approach

We can also analyze the behavior of the system of balls-and-bins using two or more choices in a quite different way. This method is referred to as the *fluid-limit* approach because it replaces the discrete balls-and-bins system with a simpler approximation similar to those that occur in fluid systems [4]. The essential idea is that we will express some property of a system of balls and bins with a Markov chain, then consider the limit of the system as the number of balls and bins approaches infinity. We can model the limiting behavior of the Markov process as a differential equation. Even though the actual number of bins is finite, the solution to the differential equation is a close approximation to the behavior in the finite-size case with high probability.

We give an example to better understand this approach. We will begin by proving a result about the balls-and-bins problem with d choices.

Theorem 2. *Suppose we throw $m = cn$ balls into n bins, where each ball is placed in the least-loaded of d bins selected independently and uniformly at random. Define $y(c)$ to be the value satisfying*

$$c = \sum_{i=0}^{\infty} \frac{y(c)^{id+1}}{id+1}$$

(or 1, whichever is smaller). Then the fraction of non-empty bins is within $O\left(\sqrt{\frac{\log n}{n}}\right)$ of $y(c)$.

Proof. Suppose T balls have been thrown. Let $Y(T)$ be the number of non-empty bins at this time. Unless all of the bins selected for the next ball are among the $Y(T)$ ones that area already non-empty, a new bin will be made non-empty. This gives the recurrence for our Markov chain:

$$\mathbf{E}\left[Y(T-1) - Y(T)\right] = 1 - \left(\frac{Y(T)}{N}\right)^d \tag{2}$$

or, if we let $t = T/n$ and $y(t) = \frac{Y(t)}{n}$,

$$\frac{\mathbf{E}\left[y(t + 1/n) - y(t)\right]}{1/n} = 1 - (y(t))^d \tag{3}$$

The form of (3) suggests that in the limit as $n \to \infty$, the solution approaches that of the differential equation

$$\frac{dy}{dt} = 1 - y^d \tag{4}$$

which we can express instead as

$$\frac{dt}{dy} = \frac{1}{1 - y^d} = \sum_{i=0}^{\infty} y^{id} \tag{5}$$

and easily solve by integration to see that

$$t = \sum_{i=0}^{\infty} \frac{y(c)^{id+1}}{id+1} \tag{6}$$

This solution is the result we wish to prove, if we take $t = c$. The missing ingredient in this proof is the proof that the solution to the differential equation in (4) is actually close to the solution of the discrete process modeled by (3), with the high-probability error bound stated in the theorem. The proof of this claim is quite technical and so will not be presented here. It relies upon Kurtz's theorem [5] which — very informally stated — shows that for scaled Markov chains like (3), taking the limit as $n \to \infty$ does in fact describe the limiting behavior of the system. It also gives a Chernoff-like bound: the probability that the deviation is at least ϵ is $O\left(e^{-nC(\epsilon)}\right)$ for some function $C(\epsilon) = \Theta\left(\frac{1}{\epsilon^2}\right)$. This can be used to obtain the desired error bound. $\square$

This method is quite flexible, and can be used for much more complex analyses. For example, one can create a *family* of differential equations representing the fraction of bins that contain at least i balls, for all i. The resulting differential equations cannot readily be solved, but they can be analyzed to see that as i increases, the fraction of bins containing at least i decreases doubly-exponentially. This leads to an alternate proof of the now-familiar result that the heaviest-loaded bin contains $O(\log \log n)$ balls, with high probability.

We will return to this technique later, when we use it in Sections 3.3 and 4.1 to prove other results.

3 Variations

The basic power-of-two-choices result has since been modified in several ways, both to generalize it to variations on the same problem, and to improve its performance further. In this section, we consider a few of these changes.

3.1 Geometric Generalization

One notable limitation of the seminal works on the power of two choices is the reliance on the supposition that bins are chosen uniformly at random. In practice, however, not all bins are created equal. When dealing with a distributed system, for example, some servers might be a priori more likely to be chosen to handle an incoming item. In particular, this is a situation which arises in many distributed hash tables, in which each server is assigned to a random point on a circle, and handles all items which fall between itself and the nearest server to its left. In this particular situation, each server is responsible for incoming items falling into an average of $\frac{1}{n}$ of the circle, but, because their loci are random, there is potentially a large variance in the fractions of incoming items assigned to each server. We discuss this application further in Section 4.2.

In a 2004 paper entitled "Geometric Generalizations of the Power of Two Choices" [6], Byers, Considine, and Mitzenmacher address this issue. In this paper, an inductive argument which directly parallels that made in the original paper by Azar et al. [3] is presented, proving that the maximum load of any bin when using the power of d choices and circular locus assignment is $\frac{\log \log n}{\log d} + O(1)$ with high probability, which differs from the classical result on a uniform distribution only in the constant factor. Furthermore, they generalize to a more abstract case in which the sizes of the n bins are determined through an unspecified random process which has the properties that the maximum bin size is at most $\delta_1 \frac{\ln n}{n}$ with high probability and the number of bins of size at least $\frac{c}{n}$ is bounded by $\delta_2 n e^{-\frac{c}{\delta_3}}$ with probability $1 - o(\frac{1}{n^2})$ for all c in the range $2 \leq c \leq \delta_4 \ln n$. These properties are held by a number of distributions, and, most notably, are held by schemes involving placing servers on both circles and two-dimensional tori and assigning to them the regions to which they are the closest server.

Furthermore, in this paper, experimental results are shown which demonstrate the practical usefulness of the power of two choices in a distributed hashing system using the circular locus paradigm. In the results shown, placement based on a single choice results in maximum loads which seem to match the $O(\log n)$ prediction nicely, topping out at a maximum reported load of 32 when running 1000 trials of placing 2^{24} balls into 2^{24} bins. By contrast, using two choices reduced the maximum load to at most 6 for over 4000 trials, ranging from $n = 2^{12}$ to $n = 2^{24}$. As predicted, increasing the number of choices to 3 or 4 had only a small impact on the observed maximum load.

3.2 Asymmetric Tie-breaking

The result that tie-breaking strategies which favor bins which are a priori less likely to receive future load out-perform (at least in practice, if not asymptotically) purely random strategies is one which is generally rather intuitive. Much less intuitive, however, is the result that it is useful to introduce artificial asymmetries, even in cases where it would be feasible to choose bins independently and uniformly at random. In a 1999 paper entitled "How Asymmetry Helps Load Balancing" [7], Vöcking presents a variation on the general power of two choices scheme which intentionally produces asymmetries, and shows how it out-performs purely symmetric schemes using independent uniform distributions.

Vöcking's algorithm, called Always-Go-Left, performs significantly better than the standard power-of-two-choices algorithm, producing a maximum load of at most $\frac{\ln \ln n}{d \ln \phi_d} + O(1)$ with high probability, where $\phi_d < 2$ is an extension of the golden ratio. Notably, in this scheme, the maximum load decreases linearly with the number of choices, whereas in the classic scheme it only decreased as as the logarithm of the number of choices. This algorithm works by partitioning the bins into d groups of size $\Theta\frac{n}{d}$, then choosing one bin from each group for each incoming ball, with the choice of a bin within a group being uniform and independent for each ball. If two bins are equally loaded, an asymmetric tie-breaking strategy is used, in which the bin with the lowest index is chosen preferentially. Using a witness tree argument, Vöcking proves that the loading of an already heavy bin is significantly less likely in this scheme than in the classic one, to an extent which produces a better asymptotic runtime.

This result is surprising mainly because, on the surface, it is quite counter-intuitive. Since all bins are equally likely to be chosen, the natural assumption would be that an unbiased tie-breaking strategy would perform best, as the end goal is to create as close to an even distribution of balls in bins as possible. However, when the bin choices are being made dependently, this intuition does not stand. Since the smaller indexed bin is chosen, there will be a tendency for the groups of bins with smaller numbers to fill faster. This is a good thing, however, when one bin from each group is chosen for each ball, as it means that the more full bins are more concentrated in certain groups, so the probability of highly filled bins being chosen from every group is low.

3.3 Memory

Another modification to the power of two choices technique is to keep state, using memory from the previous decision in order to make the next one. In 2002, Mitzenmacher, Prabhakar, and Shah proposed what they termed the $(d, m)-$*memory policy* [8]. In this process, d bins are chosen uniformly at random, just as in the standard power-of-two-choices method. The difference here is that there are also m bins already selected and stored in memory. The least full of all $m + d$ bins is chosen, and the ball is placed there. Then, the m emptiest bins out of all $m + d$ are stored in memory for the next time a ball has to be assigned to a bin.

For the purpose of comparison, the simple power-of-two-choices method in which no memory is kept will be referred to as the *d-random* policy. The asymmetric case in which the leftmost bin is chosen out of a tied pair will be termed the *d-left* policy. The $(1,1)$-memory policy compares favorably with both the 2-left and 2-random policies. While 2-random has a bound of $\frac{\log\log n}{\log 2} + O(1)$, (1,1)-memory achieves a bound of $\frac{\log\log n}{2\log\phi} + O(1)$, where $\phi = \frac{1+\sqrt{5}}{2}$. This means that replacing one of the random choices with a choice from memory improves the 2-random technique. For $d > 2$, the improvement is even more marked. The maximum load under the $(d,1)$-memory policy is reduced to $\frac{\log\log n}{\log(2d-1)} + O(1)$. This means that having one bin saved in memory from the previous round is nearly equivalent to having a choice between $2d$ random bins. So, for the price of one bin being kept as state, we have an improvement equivalent to the one that would result from doubling the number of choices. Intuitively, this is because the one bin saved in memory is the best of d randomly chosen bins, so choosing between it and d others is like choosing the best of $2d$total. The fact that the improvement is similar to multiplying d by a constant, rather than adding a constant, can easily justify the extra resources that would have to be devoted to save a small amount of state.

The analysis performed by the authors deals almost exclusively with cases where $m = 1$, as the scheme is feasible but the analysis becomes much more complex for $m > 1$. The $(d,1)$ case has significant enough results on its own, though, that the more complex analysis is unnecessary. We will prove the bound that results from using the $(1,1)$-memory policy to assign balls.

Theorem 3. *If n balls are assigned to n bins using the $(1,1)$-memory policy, the maximum load of any bin is $\frac{\log\log n}{\log(2d-1)} + O(1)$ with high probability.*

Proof. We begin by defining some notation. Time t will refer to the time before the tth ball is dropped in a bin. The fraction of bins with a load of at least i at time t will be denoted by the term $s_i(t)$, and $p_i(t)$ will be the probability that at time t the bin in memory has a load of at least i. Given these definitions, we can determine the following relation:

$$\mathbf{E}\left[s_i(t+1) - s_i(t) \mid s(t)\right] = \frac{s_{i-1}(t)\cdot p_{i-1}(t) - s_i(t)\cdot p_i(t)}{n} \tag{7}$$

This is because s_i increases by $1/n$ whenever the number of bins with at least i balls increases by 1, and s_i will only increase if some bin that had $i-1$ balls before gains an ith ball. This will only happen in the case where both the randomly selected bin and the one in memory had $i-1$ balls but not i in them.

To make analysis simpler, this equation will be converted into a differential equation using the fluid-limit method of Section 2.1. First, t will be scaled so that time runs from 0 to 1, and t changes by increments of $\Delta t = 1/n$ for the n items that need to be assigned. This gives us

$$\frac{\mathbf{E}\left[s_i(t+\Delta t) - s_i(t) \mid s(t)\right]}{t} = s_{i-1}(t)\cdot p_{i-1}(t) - s_i(t)\cdot p_i(t) \tag{8}$$

Taking the limit as $\Delta t \to 0$, this then becomes the following differential equation:

$$\frac{ds_i}{dt} = s_{i-1}\cdot p_{i-1} - s_i\cdot p_i \tag{9}$$

We also need an equation to govern the $p_i(t)$ term. There are three cases in which the next bin placed in memory will have a load of size i or more. First, it could be that the current bin in memory has a load of $i-1$, but all of the randomly chosen bins have loads of i or more. Second, the bin in memory could have a load of i or more, but one of the randomly selected ones has a load of $i-1$. Finally, if all of the randomly chosen bins and the one in memory have loads of at least i items already, then the one that gets placed in memory will also. In order to get an equation for $p_i(t)$ we need to sum the probabilities of each of these three scenarios. This will give us the equation for $p_i(t)$ shown below:

$$p_i(t+1) = \left(p_{i-1}(t) - p_i(t)\right) s_i(t) + p_i(t)\left(s_{i-1}(t) - s_i(t)\right) + p_i(t) \cdot s_i(t) \tag{10}$$

Using large deviations theory, the authors demonstrate that the Markov chain for the value of $p_i(t)$ changes state at a much faster rate than the one for the $s_i(t)$ values. Therefore, in the equation for p_i we can instead find the equilibrium values for p_i given that the s_i values are fixed. This gives us

$$p_i = p_{i-1} \cdot s_i + p_i\left(s_{i-1} - s_i\right), \tag{11}$$

which can also be stated as

$$p_i = \frac{p_{i-1} \cdot s_i}{1 - s_{i-1} + s_i}. \tag{12}$$

This p_i equation can then be used to substitute for p_i in (9). Solving the resulting system of differential equations shows that maximum load of any bin is $\frac{\log\log n}{\log(2d-1)}$, and applying Kurtz's theorem to return to the discrete, probabilistic case gives us the same result with high probability with only a $O(1)$ additive factor. $\square$

Remark: To bound the $(d,1)$-memory case, we can use following equations instead:

$$\frac{ds_i}{dt} = s_{i-1}^d \cdot p_{i-1} - s_i^d p_i \tag{13}$$

$$p_i = \frac{p_{i-1} \cdot s_i^d}{1 - d\left(s_{i-1} - s_i\right) s_{i-1}^{d-1}} \tag{14}$$

Solving these equations in the same manner as above will give a high-probability maximum load for the system of

$$\frac{\log\log n}{\log f(d)} + O(1),$$

where $f(d) = \frac{1}{2}\left(2d+1+\sqrt{4d^2+1}\right)$, which falls bnetween $2d$ and $2d+1$. Therefore, this bound is slightly better than the bound for a $2d$-random system.

4 Applications

These results have direct applications to a number of load-balancing problems. In this section, we will consider what needs to be done to apply it to three problems, and what results are obtained. The technique proves to be a useful load-balancing technique not just asymptotically in theory, but also empirically in practice and simulation.

4.1 Supermarket Load Balancing

The power-of-two-choices technique can be applied to a load-balancing problem known as the *supermarket system*. The model is as follows: there are n servers, each maintaining its own FIFO queue of waiting customers. The arrival of customers is modeled as a Poisson process with mean λn ($\lambda < 1$), and the service time for each customer as an exponential distribution with mean 1.

If each customer chooses a server at random, then this is simply a set of n servers, each of which receives a new customer as a Poisson process with mean λ, and exponentially-distributed service time. This is a standard model referred to as the $\mathsf{M/M/1}$ model, and has been extensively studied in the realm of queuing theory. In particular, it is well known that the system reaches a steady state, and once it does, the average time a customer spends in the system (including both waiting in line and being serviced) is $\frac{1}{1-\lambda}$ [9].

This seems quite reminiscent of the balls-and-bins problem — though now customers are sometimes removed from queues — which suggests a power-of-two-choices approach. Indeed, we can improve on this result if instead each customer chooses d servers at random, and selects the least-loaded one to wait in the queue for. Mitzenmacher showed that this system also converges to a steady state, and the average time in the system is $T_D(\lambda) = \sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}}$ in the steady state [10]. Note that the value of the summands decreases doubly-exponentially with i, since i is in the exponent of the exponent. Taking limits shows that this complicated summation is indeed the dramatic improvement we have come to expect from applying the power of two choices: it is $O\left(\log \frac{1}{1-\lambda}\right)$, and in fact as λ approaches 1, it converges to $\log_d \frac{1}{1-\lambda}$.

To prove this result, we define $s_i(t)$ to be the fraction of servers with queues containing i or more customers. We can view this as th estate space of a Markov chain, as in Section 2.1; unlike the one-dimensional example in Section 2.1, however, this state space is *infinite*-dimensional. Nevertheless, we can still use the same technique of finding a differential equation that captures the evolution of the system for infinite n.

Suppose there are n queues, and consider the change in m_i, the number of queues with at least i customers, over some infinitesimal time segment. The probability that a new customer arrives is λn. m_i increases only if this new customer selects d queues that have at least $i-1$ customers, and not all of them have at least i customers. So the increase in m_i is $\lambda n \left(s_{i-1}^d - s_i^d\right)$. Since the service time for each customer is exponentially-distributed, a customer will be removed from the queue with constant probability. m_i decreases only when a customer is removed from a queue of size exactly i, so the expected decrease is $n\left(s_i - s_{i+1}\right)$. This gives us the differential equation

$$\frac{dm_i}{dt} = \lambda n \left(s_{i-1}^d - s_i^d\right) - n\left(s_i - s_{i+1}\right) \tag{15}$$

or, noting that $m_i = ns_i$ and scaling,

$$\frac{ds_i}{dt} = \lambda \left(s_{i-1}^d - s_i^d\right) - \left(s_i - s_{i+1}\right) \tag{16}$$

These equations apply for $i > 0$; the base case is $s_0 = 1$, i.e. all queues always contain at least 0 elements.

We can now identify the steady-state behavior of the system by finding a fixed point of the differential equation system in (16):

Theorem 4. *For $d \geq 2$, the system in* (16) *has a finite fixed point $s_i = \lambda^{\frac{d^i-1}{d-1}}$.*

Proof. At the fixed point, $\frac{ds_i}{dt} = 0$ for all $i \geq 1$. Summing over $i \geq 1$, we obtain a telescoping sum

$$\begin{aligned} \lambda \sum_{i=1}^{\infty} \left(s_{i-1}^d - s_i^d \right) - \sum_{i=1}^{\infty} \left(s_i - s_{i+1} \right) = 0 \\ \lambda \left(s_0 - s_\infty \right) - \left(s_1 - s_\infty \right) = 0 \end{aligned}$$

But $s_0 = 1$ by definition, and s_∞ goes to zero, since we know that queue lengths stabilize at some finite value — this is a well-known result in queueing theory for the basic $\mathsf{M/M/1}$ case, and applying the power-of-two-choices technique only improves things. So $s_1 = \lambda$.

With this base case for s_1, a standard inductive argument using (16) now proves the claim that $s_i = \lambda^{\frac{d^i-1}{d-1}}$. $\square$

It is also necessary to show that the system actually converges to its fixed point. This proof of the following theorem is quite technical, so we shall omit the details:

Theorem 5. *Suppose the system begins in a finite state ($s_j(0) = 0$ for some j). Then for all time $t \geq 0$, the $s_i(t)$ always decrease doubly exponentially with respect to i. If the system begins in the empty state, then as $t \to \infty$, $s_i(t)$ approaches $\lambda^{\frac{d^i-1}{d-1}}$ from below.*

Proof. See [10]. $\square$

We can now use this fixed point of the infinite-limit system to show properties about our actual systems of finite size. This is done in the same way as Section 2.1, though there are some more technical details (which we omit) relating to the fact that we now have an infinite set of differential equations.

Theorem 6. *Suppose we have a supermarket system with n servers that begins with all queues empty. Then at steady-state, the expected time a customer spends in the system is*

$$\sum_{i=1}^{\infty} \lambda^{\frac{d^i-d}{d-1}} + o(1)$$

Proof. Suppose a customer arrives in the system at time t, and consider the probability that he becomes the ith customer in some queue. This happens only if he selects d queues that have at least $i-1$ customers, but not all have at least i customers, i.e. $s_{i-1}(t) - s_i(t)$. We take the expected value by multiplying by i and summing over all i:

$$\begin{aligned} \mathbf{E}\left[\text{queue time}\right] &= \sum_{i=1}^{\infty} i \left(s_{i-1}(t)^d - s_i(t)^d \right) \\ &= \sum_{i=0}^{\infty} s_i(t)^d \end{aligned}$$

Theorems 4 and 5 tell us that as t grows and the system approaches its steady state,

$$\lim_{t\to\infty} \mathbf{E}\,[\text{queue time}] = \sum_{i=0}^{\infty} \lambda^{\frac{d^i-d}{d-1}}$$

Note that since the system starts with all queues empty, the expected queue time converges from below, by Theorem 5; i.e. it is always less than the limiting value.

Now we use Kurtz's theorem to relate the result of the infinite system to the finite systems of which it is the limiting case. This introduces a $o\,(1)$ term (with respect to n) that depends on the values of T and λ, and proves the theorem. $\square$

The above result applies for $d \geq 2$; if $d = 1$, we have the standard $\mathsf{M/M/1}$ model and the expected time in the system is $\frac{1}{1-\lambda}$, which we can write as $\sum_{i=0}^{\infty} \lambda^i$. Notice that applying the power-of-two-choices technique causes the summed terms to decrease doubly-exponentially rather than exponentially (the i is now in the exponent of the exponent). Hence, the expected waiting time is reduced to $O\left(\log_d \frac{1}{1-\lambda}\right)$. A dramatic improvement comes from allowing even two choices.

4.2 Distributed Hash Tables

The power-of-two-choices technique can also be used to improve load balancing in peer-to-peer networks based on *distributed hash tables*. A distributed hash table is a system for storing and locating data in a decentralized peer-to-peer network. The basis for many distributed hash tables is the technique of *consistent hashing* [11], where both data objects and servers are mapped to locations in a circular identifier space using a common hash function. Then the server that is the first successor of the key of some data object in the identifier ring is the one responsible for storing it. Layered atop this is a *distributed lookup service* such as Chord [12], which builds a routing topology for the network that makes it possible to efficiently determine the successor of a given key even as nodes join and leave the system, and a *distributed hash table* such as DHash [13] that stores the values associated with keys.

These systems scale well in general, but direct application of consistent hashing introduces a problem with load balancing. The problem is that the servers are assigned essentially randomly to locations in the ring (assuming a suitable hash function), so the intervals that each server is responsible for can vary in length. For m servers, the expected size of the interval for each server is $\frac{1}{m}$ of the ring, indicating that load is distributed equally. However, this only holds in expectation; we can only guarantee with high probability that each server receives $\frac{\log m}{m}$ of the total load. Indeed, with high probability there is at least one server with $\Theta\left(\frac{\log m}{m}\right)$ load, and at least one with $\Theta\left(\frac{1}{m^2}\right)$ load [6].

The standard solution to this problem is to have each node operate $\Theta\,(\log m)$ "virtual nodes," which guarantees with high probability that no node will be responsible for more than a $O\left(\frac{1}{m}\right)$ fraction of the ring in total. However, when applying this technique to a distributed hash table, this increases the complexity of the routing layer. Without virtual nodes, each Chord node maintains $\Theta\,(\log m)$ pointers to other nodes in the ring in order to achieve its $O\,(\log m)$-message lookup operations [12]. If each node is now operating

$\Theta(\log m)$ virtual nodes, however, it will need to maintain a routing table for each one, thereby requiring a total of $\Theta\left(\log^2 m\right)$ pointers to other nodes. This is an important increase, since increasing the routing table size means increasing the amount of maintenance traffic required to ensure that all pointers are up to date even as nodes join and leave.

Byers et. al. propose using the power of two choices instead of virtual nodes in order to achieve load balancing in distributed hash tables based on consistent hashing [14]. Their solution is simple, straightforward, and achieves similar performance to the virtual nodes approach, without the additional routing overhead. They propose simply allowing each data object to map to d keys rather than 1, then choosing the least loaded of the d to store the data on. This does increase the cost to perform a query because now d nodes must be contacted in order to find the data. The authors propose addressing this problem by storing redirection pointers at each of the $d-1$ nodes that an item hashes on which it is not stored; the redirection pointers are small compared to the data, so there is not much storage cost, though a maintenance cost is incurred in keeping these up to date in the presence of churn.

Theorem 7. *The d-choices approach for load balancing in Chord ensures that, with high probability, the maximum load on any node is no more than* $\log_d \log m + O(1)$ *times greater than the* $\frac{1}{m}$ *expected value.*

Proof. By application of the geometric generalization in Section 3.1. □

Applying Vöcking's ALWAYS-GO-LEFT asymmetric tie-breaking scheme, described in Section 3.2, improves this bound to $\frac{\log\log m}{d}+O(1)$. In practice, the authors suggest breaking ties by choosing the server responsible for the shortest arc of the identifier space in the ring, since it is the least likely to be selected by hashing when future items are added. There exists no theoretical analysis of this tie-breaking strategy, but simulation results have shown that it leads to the best performance in practice. Indeed, using only two choices and this tie-breaking rule provides better load-balancing performance in simulation than the standard approach of using $\lfloor \log_2 m \rfloor$ virtual nodes.

4.3 Router IP Lookups

The power-of-two-choices technique can be applied directly to the problem of performing routing table lookups in high-performance network routers. These devices store a table mapping IP address prefixes to the next routing hop, and must be able to quickly find the next hop associated with the longest prefix matching a IP packet's destination. This is typically accomplished by constructing a set of hash tables, each containing all prefixes of a certain length, and performing a binary search in order to find the longest matching prefix [15]. The goal is to minimize the number of memory transfers required to perform a lookup by ensuring that the contents of hash buckets fit in a single cache line. In the static, off-line case, this can be done by building a slightly relaxed version of a perfect hash table that is allowed to have a small number of entries in the same bucket.

Broder and Mitzenmacher applied the power-of-two-choices technique to solve this problem dynamically using multiple hash functions [16]. The idea is straightforward: use d different hash functions, and always store new entries in whichever hash bucket is the least full. Assuming ideal random hash functions, the hashed values will be uniformly distributed, and so the fundamental power-of-two-choices result of Theorem 1 tells us that with high probability no bucket will be larger than $\log_d \log n + O(1)$. This can generally be made small enough to fit in a single cache line. Moreover, the need to use d different hash functions and fetch the resulting buckets is not particularly troublesome, since the d hash functions can be computed in parallel and then the appropriate locations fetched from memory in parallel or pipelined.

A truly random hash function is impractical to use, however, so typically a simpler hash function such as a 2-universal hash family is used. A natural question is whether this suffices to give the results described above. A theoretical analysis has not yet been given, but simulation results show that simple hash functions work well on examples of real IP routing data sets (which are non-adversarial). The simulation showed that the maximum bucket size is very small; for example, when hashing 32,000 entries to 32,000 buckets using only two choices, no bucket contained more than 4 items, and only a small fraction contained more than 2. This is an impressive result for a technique far simpler than the offline generation of semi-perfect hash functions.

5 Conclusion

The power-of-two-choices technique was a breakthrough in the world of load balancing algorithms. There are numerous applications that have benefited greatly from the insight of the approach. Among these, of course, are the application discussed here, such as supermarket load balancing, hash tables, and peer-to-peer network load distribution. These are, however, just a subset of the total group of algorithms and applications that benefited.

In addition to its being a very useful technique on its own, the power-of-two-choices technique has also spawned many variations on the original theme. The use of asymmetry or a stored state have also introduced further improvements in the load balancing properties, and the technique has been generalized to certain geometric problems where the random selection is not uniform.

Much of the beauty of these techniques is their simplicity. While the proofs of their bounds might sometimes be complicated, the techniques themselves are very straightforward and easy to implement. The all run in constant time per item to be assigned, so there is very little overhead, and yet they provide a dramatic improvement in terms of bounds on their maximum load.

References

[1] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," in *Proc. Twenty-Sixth Annual ACM Symposium on Theory of Computing*, (Montréal), pp. 593–602, May 1994.

[2] M. D. Mitzenmacher, *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California at Berkeley, 1996.

[3] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM Journal on Computing*, vol. 29, no. 1, pp. 180–200, 2000.

[4] M. Mitzenmacher, "Studying balanced allocations with differential equations," *Combinatorics, Probability and Computing*, vol. 8, pp. 473–482, 1999.

[5] T. G. Kurtz, "Solutions of ordinary differential equations as limits of jump Markov processes," *J. Applied Probability*, vol. 7, pp. 49–58, 1970.

[6] J. W. Byers, J. Considine, and M. Mitzenmacher, "Geometric generalizations of the power of two choices," in *Proc. ACM Symposium on Parallel Algorithms and Architectures*, June 2004.

[7] B. Vöcking, "How asymmetry helps load balancing," in *IEEE Symposium on Foundations of Computer Science*, pp. 131–141, 1999.

[8] M. Mitzenmacher, B. Prabhakar, and D. Shah, "Balls and bins with memory," in *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 799–808, 2002.

[9] A. Ferrier, "Queuing theory — a straightforward introduction." Available at `http://www.new-destiny.co.uk/andrew/past_work/queueing_theory/Andy/`, June 1999.

[10] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[11] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *ACM Symposium on Theory of Computing*, pp. 654–663, May 1997.

[12] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[13] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proc. 1st USENIX Symposium on Network Systems Design and Implementation '04*, Mar. 2004.

[14] J. W. Byers, J. Considine, and M. Mitzenmacher, "Simple load balancing for distributed hash tables," in *Proc. 2nd International Workshop on Peer-to-Peer Systems*, (Berkeley), pp. 80–88, Feb. 2003.

[15] V. Srinivasan and G. Varghese, "Fast address lookups using controlled prefix expansion," *ACM Transactions on Computer Systems*, vol. 17, no. 1, pp. 1–40, 1999.

[16] A. Broder and M. Mitzenmacher, "Using multiple hash functions to improve ip lookups," in *Proc. IEEE INFOCOM 2001*, pp. 1454–1463, 2001.

[17] M. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing, Volume 1* (P. Pardalos, S. Rajasekaran, J. H. Reif, and J. D. Rolim, eds.), vol. 9 of *Combinatorial Optimization*, pp. 255–312, Berlin: Springer-Verlag, draft ed., 2001.