SRC Technical Note
1998 - 002
February 8, 1998

How Useful Is Old Information?

Michael Mitzenmacher

digital
Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301
http://www.research.digital.com/SRC/

How Useful is Old Information?

EXTENDED ABSTRACT

Michael Mitzenmacher*

Abstract

We consider the problem of load balancing in dynamic distributed systems in cases where new incoming tasks can make use of old information. For example, consider a multi-processor system where incoming tasks with exponentially distributed service requirements arrive as a Poisson process, the tasks must choose a processor for service, and a task knows when making this choice the processor loads from T seconds ago. What is a good strategy for choosing a processor, in order for tasks to minimize their expected time in the system? Such models can also be used to describe settings where there is a transfer delay between the time a task enters a system and the time it reaches a processor for service.

Our models are based on considering the behavior of limiting systems where the number of processors goes to infinity. The limiting systems can be shown to accurately describe the behavior of sufficiently large systems, and simulations demonstrate that they are reasonably accurate even for systems with a small number of processors. Our studies of specific models demonstrate the importance of using randomness to break symmetry in these systems and yield important rules of thumb for system design. The most significant result is that only small amounts of load information can be extremely useful in these settings; for example, having incoming tasks choose the least loaded of two randomly chosen processors is extremely effective over a large range of possible system parameters. In contrast, using global information can actually degrade performance unless used correctly; for example, unlike most settings where the load information is current, having tasks go to the least loaded server can significantly hurt performance.

1 Introduction

Distributed computing systems, such as networks of workstations or mirrored sites on the World Wide Web, face the problem of using their resources effectively. If some hosts lie idle while others are heavily loaded, system performance can fall significantly. To prevent this, *load balancing* is used to distribute the workload, improving performance measures such as the expected time a task spends in the system. Although determining an effective load balancing strategy depends strongly on the details of the underlying system, general models from both queueing theory and computer science often provide valuable insight and general rules of thumb.

In this paper, we develop analytical models for the realistic setting where old load information is available. For example, suppose we have a system of n servers, and incoming tasks must choose a server and wait for service. If the incoming tasks know the current number of tasks already queued at each server, it is often best for the task to go to the server with the shortest queue [18]. In many actual systems, however, it is unrealistic to assume that tasks will have access to up to date load information; global load information may be updated only periodically, or the time delay for a task to move to a server may be long enough that the load information is out of date by the time the task arrives. In this case, it is not clear what the best load balancing strategy is.

Our models yield surprising results. Unlike similar systems in which up to date information is available, the strategy of going to the shortest queue can lead to extremely bad behavior when load information is out of date; however, the strategy of going to the shortest of two randomly chosen queues performs well under a large range of system parameters. This result suggests that systems which attempt to exploit global information to balance load too aggressively may suffer in performance, either by misusing it or by adding significant complexity.

***email:** michaelm@pa.dec.com This work originally appeared in the *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, 1997.

1.1 Previous Work

The problem of how to use old information is generally neglected in theoretical work, even though balancing workload from distributed clients based on incomplete or possibly out of date server load information may be an increasingly common system requirement. A recent work by Awerbuch, Azar, Fiat, and Leighton [2] covers a similar theme, although their models are substantially different from ours.

The idea of each task choosing from a small number of processors in order to balance the load has been studied before, both in theoretical and practical contexts. In many models, using just two choices per task can lead to an exponential improvement over one choice in the maximum load on a processor. In the static setting, this improvement appears to have first been noted by Karp, Luby, and Meyer auf der Heide [7]. A more complete analysis was given by Azar, Broder, Karlin, and Upfal [3]. In the dynamic setting, this work was extended in [12, 13]; similar results were independently reported in [22].

In the queuing theory community, similar previous work includes that of Towsley and Mirchandaney [17] and that of Mirchandaney, Towsley, and Stankovic [9, 10]. These authors examine how some simple load sharing policies are affected by communication delay, extending a similar study of load balancing policies by Eager, Lazowska, and Zahorjan [5]. Their analyses are based on Markov chains associated with the load sharing policies they propose and simulations.

Our works expands on this work in several directions. We apply a fluid-limit approach, in which we develop a deterministic model corresponding to the limiting system as $n \to \infty$. We call this system the *infinite system*, and also refer to the method as the infinite system approach. This approach has successfully been applied previously to study load balancing problems in [1, 12, 13, 14, 15, 22] (see also [1] for more references, or [21] for the use of this approach in a different setting), and it can be seen as a generalization of the previous Markov chain analysis. Using this technique, we examine several new models of load balancing in the presence of old information. In conjunction with simulations, our models demonstrate several basic but powerful rules of thumb for load balancing systems, most notably the effectiveness of using just two choices.

The remainder of this paper is organized as follows: in Section 2, we describe a general queueing model for the problems we consider. In Sections 3, 4, and 5, we consider different models of old information. For each such model, we present a corresponding infinite system, and using the infinite systems and simulations we determine important behavioral properties of these models. We conclude with a section on open problems and further directions for research.

2 The Bulletin Board Model

Our work will focus on the following natural dynamic model: tasks arrive as a Poisson stream of rate λn, where $\lambda < 1$, at a collection of n servers. Each task chooses one of the servers for service and joins that server's queue; we shall specify the policy used to make this choice subsequently. Tasks are served according to the First In First Out (FIFO) protocol, and the service time for a task is exponentially distributed with mean 1. We are interested in the expected time a task spends in the system in equilibrium, which is a natural measure of system performance, and more generally in the distribution of the time a customer spends in the queue. Note that the average arrival rate per queue is $\lambda < 1$, and that the average service rate is 1; hence, assuming the tasks choose servers according to a reasonable strategy, we expect the system to be *stable*, in the sense that the expected number of tasks per queue remains finite in equilibrium. In particular, if each task chooses a server independently and uniformly at random, then each server acts as an M/M/1 queue (Poisson arrivals, exponentially distributed service times) and is clearly stable. We will examine the behavior of this system under a variety of methods that tasks may use to choose their server.

We will allow the tasks choice of server to be determined by load information from the servers. It will be convenient if we picture the load information as being located at a *bulletin board*. We strongly emphasize that the bulletin board is a purely *theoretical* construct used to help us describe various possible load balancing strategies and need not exist in reality. The load information contained in the bulletin board need *not* correspond exactly to the actual current loads; the information may be erroneous or approximate. Here, we focus on the problem of what to do when the bulletin board contains old information (where what we mean by old information will be specified in future sections).

We shall focus on *distributed* systems, by which we mean that the tasks cannot directly communicate in order to coordinate where they go for service. The decisions made by the tasks are thus based only on whatever load information they obtain and their entry time. Although our modeling technique can be used for a large class of strategies, in this paper we shall concentrate on the following natural, intuitive strategies:

- Choose a server independently and uniformly at random.
- Choose d servers independently and uniformly at random, check their load information from the bulletin board, and go to the one with the smallest load.[1]
- Check all load information from the bulletin board, and go to the server with the smallest load.

The strategy of choosing a random server has several advantages: it is easy to implement, it has low overhead, it works naturally in a distributed setting, and it is known that the expected lengths of the queues remain finite over time. However, the strategy of choosing a small number of servers and queueing at the least loaded has been shown to perform significantly better in the case where the load information is up to date [5, 12, 13, 22]. It has also proved effective in other similar models [3, 7, 13]. Moreover, the strategy also appears to be practical and have a low overhead in distributed settings, where global information may not be available, but polling a small number of processors may be possible. Going to the server with the smallest load appears natural in more centralized systems where global information is maintained. Indeed, going to the shortest queue has been shown to be optimal in a variety of situations in a series of papers, starting for example with [18, 20]. Hence it makes an excellent point of comparison in this setting.

We develop analytical results for the limiting case as $n \to \infty$, for which the system can be accurately modeled by an *infinite system*. The infinite system consists of a set of differential equations, which we shall describe below, that describe the expected behavior of the system. This corresponds to the exact behavior of the system as $n \to \infty$. More information on this approach can be found in [6, 8, 12, 13, 14, 15, 22]. (We note, however, that this approach works only because the systems for finite n have an appropriate form as a Markov chain; indeed, we initially require exponential service times and Poisson arrivals to ensure this form.) Previous experience suggests that using the infinite system to estimate performance metrics such as the expected time in the system proves accurate, even for relatively small values of n [5, 12, 13, 14]. We shall verify this for the models we consider by comparing our analytical results with simulations.

3 Periodic Updates

The previous section has described possible ways that the bulletin board can be used. We now turn our attention to how a bulletin a board can be updated. Perhaps the most obvious model is one where the information is updated at periodic intervals. In a client-server model, this could correspond to an occasional broadcast of load information from all the servers to all the clients. Because such a broadcast is likely to be expensive (for example, in terms of communication resources), it may only be practical to do such a broadcast at infrequent intervals. Alternatively, in a system without such centralization, servers may occasionally store load information in a readable location, in which case tasks may be able to obtain old load information from a small set of servers quickly with low overhead.

We therefore suggest the *periodic update* model, in which the bulletin board is updated with accurate information every T seconds. Without loss of generality, we shall take the update times to be $0, T, 2T, \ldots$. The time between updates shall be called a *phase*, and phase i will be the phase that ends at time iT. The time that the last phase began will be denoted by T_t, where t is the current time.

The infinite system we consider will utilize a two-dimensional family of variables to represent the state space. We let $P_{i,j}(t)$ be the fraction of queues at time t that have true load j but have load i posted on the bulletin board. We let $q_i(t)$ be the rate of arrivals at a queue of size i at time t; note that, for time-independent strategies, the rates $q_i(t)$ depend only on the load information at the bulletin boards and the strategy used by the tasks, and hence is the same as $q_i(T_t)$. In this case, the rates q_i change whenever the bulletin board is updated.

We first consider the behavior of the system during a phase, or at all times $t \neq kT$ for integers $k \geq 0$. Consider a server showing i customers on the bulletin board, but having j customers: we say such a server is in state (i, j). Let $i, j > 1$. What is the rate at which a server leaves state (i, j)? A server leaves this state when customer departs, which happens at rate $\mu = 1$, or a customer arrives, which happens at rate $q_i(t)$. Similarly, we may ask the rate at which customers enter such a state. This can happen if a customer arrives at a server with load i posted on the bulletin

[1] In this and other strategies, we assume that ties are broken randomly. Also, the d choices are made without replacement in our simulations; in the infinite system setting, the difference between choosing with and without replacement is negligible.

board but having $j-1$ customers, or a customer departs from a server with load i posted on the bulletin board but having $j+1$ customers. This description naturally leads us to model the behavior of the system by the following set of differential equations:

$$\frac{dP_{i,0}(t)}{dt} = P_{i,1}(t) - P_{i,0}(t)q_i(t)\,; \tag{1}$$

$$\begin{aligned}\frac{dP_{i,j}(t)}{dt} &= (P_{i,j-1}(t)q_i(t) + P_{i,j+1}(t)) \\ &\quad - (P_{i,j}(t)q_i(t) + P_{i,j}(t))\ ,\ \ j \geq 1.\end{aligned} \tag{2}$$

These equations simply measure the rate at which servers enter and leave each state. (Note that the case $j = 0$ is a special case.) While the queueing process is random, however, these differential equations are deterministic, yielding a fixed trajectory once the initial conditions are given. In fact, these equations describe the limiting behavior of the process as $n \to \infty$, as can be proven with standard (albeit complex) methods [6, 8, 13, 14, 15, 21, 22]. Here we take these equations as the appropriate limiting system and focus on using the differential equations to study load balancing strategies.

For integers $k \geq 0$, at $t = kT$ there is a state jump as the bulletin board is updated. At such t, necessarily $P_{i,j}(t) = 0$ for all $i \neq j$, as the load of all servers is correctly portrayed by the bulletin board. If we let $P_{i,j}(t^-) = \lim_{z\to t^-} P_{i,j}(z)$, so that the $P_{i,j}(t^-)$ represent the state just before an update, then

$$P_{i,i}(t) = \sum_j P_{j,i}(t^-).$$

3.1 Specific Strategies

We consider what the proper form of the rates q_i are for the strategies we examine. It will be convenient to define the load variables $b_i(t)$ be the fraction of servers with load i posted on the bulletin board; that is, $b_i(t) = \sum_{j=0}^{\infty} P_{i,j}(t)$.

In the case where a task chooses d servers randomly, and goes to the one with the smallest load on the bulletin board, we have the arrival rate

$$q_i(t) = \lambda \frac{\left(\sum_{j\geq i} b_j(t)\right)^d - \left(\sum_{j>i} b_j(t)\right)^d}{b_i(t)}.$$

The numerator is just the probability that the shortest posted queue length of the d choices on the bulletin board is size i. To get the arrival rate per queue, we scale by λ, the arrival rate per queue, and $b_i(t)$, the total fraction of queues showing i on the board. In the case where $d = 1$, the above expression reduces to $q_i(t) = \lambda$, and all servers have the same arrival rate, as one would expect.

To model when tasks choose the shortest queue on the bulletin board, we develop an interesting approximation. We assume that there always exists servers posting load 0 on the bulletin board, and we use a model where tasks go to a random server with posted load 0. As long as we start with some servers showing 0 on the bulletin board in the infinite system (for instance, if we start with an empty system), then we will always have servers showing load 0, and hence this strategy is valid. In the case where the number of queues is finite, of course, at some time all servers will show load at least one on the billboard; however, for a large enough number of servers the time between such events is large, and hence this model will be a good approximation. So for the shortest queue policy, we set the rate

$$q_0(t) = \frac{\lambda}{b_0(t)},$$

and all other rates $q_i(t)$ are 0.

3.2 The Fixed Cycle

In a standard deterministic dynamical system, a natural hope is that the system converges to a *fixed point*, which is a state at which the system remains forever once it gets there; that is, a fixed point would correspond to a point $P = (P_{i,j})$ such that $\frac{dP_{i,j}}{dt} = 0$. The above system clearly cannot reach a fixed point, since the updating of the bulletin board at time $t = kT$ causes a jump in the state; specifically, all $P_{i,j}$ with $i \neq j$ become 0. It is, however, possible to find a *fixed cycle* for the system. We find a point P such that if $P = (P_{i,j}(k_0T))$ for some integer $k_0 \geq 0$, then $P = (P_{i,j}(kT))$ for all $k \geq k_0$. In other words, we find a state such that if the infinite system begins a phase in that state, then it ends the phase in the same state, and hence repeats the same cycle for every subsequent phase. (Note that it also may be possible for the process to cycle only after multiple phases, instead of just a single phase. We have not seen this happen in practice, and we conjecture that it is not possible for this system.)

To find a fixed cycle, we note that this is equivalent to finding a vector $\vec{\pi} = (\pi_i)$ such that if π_i is the fraction of queues with load i at the beginning of the phase, the same distribution occurs at the end of a phase. Given an initial $\vec{\pi}$, the arrival rate at a queue with i tasks from time 0 to T can be determined. By our assumptions of Poisson arrivals and exponential service times, during each phase each server acts as an independent M/M/1 queue that runs for T seconds, with some initial number of tasks awaiting service. We use this fact to find the π_i.

Formulae for the distribution of the number of tasks at time T for an M/M/1 queue with arrival rate λ and i tasks initially have long been known (for example, see [4, pp. 60-64]); the probability of finishing with j tasks after T seconds, which we denote by $m_{i,j}$, is

$$\begin{aligned} m_{i,j}(T) &= \lambda^{\frac{1}{2}(j-i)} \mathrm{e}^{-(1+\lambda)T} [B_{j-i}(2T\sqrt{\lambda}) + \lambda^{-\frac{1}{2}} B_{i+j+1}(2T\sqrt{\lambda}) \\ &\quad + (1-\lambda) \sum_{k=1}^{\infty} \lambda^{-\frac{1}{2}(1+k)} B_{i+j+k+1}(2T\sqrt{\lambda})], \end{aligned}$$

where here $B_z(x)$ is the modified Bessel function of the first kind. If $\vec{\pi}$ gives the distribution at the beginning and end of a phase, then the π_i must satisfy $\pi_i = \sum_j \pi_j m_{j,i}(T)$, and this can be used to determine the π_i.

It seems unlikely that we can use the above characterization to find a closed form for the state at the beginning of the phase of for the fixed cycle in terms of T. In practice we find the fixed cycle easily by running a truncated version of the system of differential equations (bounding the maximum values of i and j) above until reaching a point where the change in the state between two consecutive updates is sufficiently small. This procedure works under the assumption that the trajectory always converges to the fixed cycle rapidly. (We discuss this more in the next section.) Alternatively, from a starting state we can apply the above formulae for $m_{i,j}$ to successively find the states at the beginning of each phase, until we find two consecutive states in which the difference is sufficiently small. Simulating the differential equations has the advantage of allowing us to see the behavior of the system over time, as well as to compute system measurements such as the expected time a task spends in the system.

3.3 Convergence Issues

Given that we have found a fixed cycle for the relevant infinite system, important questions remain regarding convergence. One question stems from the approximation of a finite system with the corresponding infinite system: how good is this approximation? The second question is whether the trajectory of the infinite system always converges to its fixed cycle, and if so, how quickly?

For the first question, we note that the standard methods referred to previously provide only very weak bounds on the convergence rate between infinite and finite systems. By focusing on a specific problem, proving tighter bounds may be possible (see, for example, the discussion in [21]). In practice, however, as we shall see in Section 3.4, the infinite system approach proves extremely accurate even for small systems, and hence it is a useful technique for gauging system behavior.

For the second question, we have found in our experiments that the system does always converge to its fixed cycle, although we have no proof of this. The situation is generally easier when the trajectory converges to a fixed point, instead of a fixed cycle, as we shall mention in subsequent sections. (See also [13].) Proving this convergence hence remains an interesting open theoretical question.

3.4 Simulations

We present some simulation results, with two main purposes in mind: first, we wish to show that the infinite system approach does in fact yield a good approximation for the finite case; second, we wish to gain insight into the problem load balancing using old information. We choose to emphasize the second goal. As such, we plot data from simulations of the actual queueing process (except in the case where one server is chosen at random; in this case we apply standard formulae from queueing theory). We shall note the deviation of the values obtained from the infinite system and these simulations where appropriate.

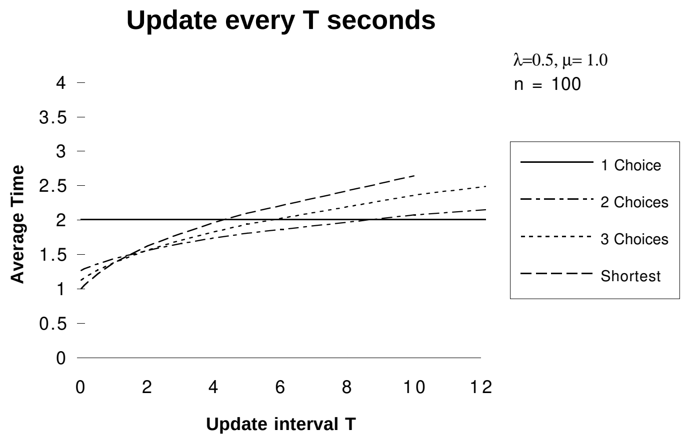

Figure 1: Strategy comparison at $\lambda = 0.50$.

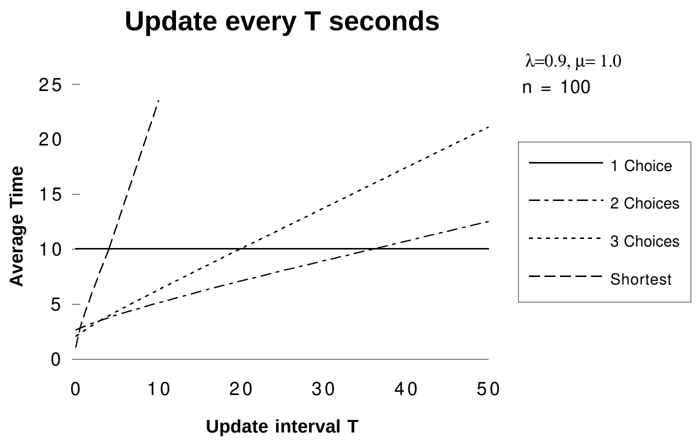

Figure 2: Strategy comparison at $\lambda = 0.90$.

This methodology may raise the question of why the infinite system models are useful at all. There are several reasons: first, simulating the differential equations is often much faster than simulating the corresponding queueing system; this issue will be explored further in the final version of the paper. Second, the infinite systems provide a theoretical framework for examining these problems that can lead to formal theorems. Third, the infinite system provides good insight into and accurate approximations of how the system behaves, independent of the number of servers. This information should prove extremely useful in practice.

In Figures 1 and 2, the results for various strategies are given for arrival rates $\lambda = 0.5$ and $\lambda = 0.9$ for $n = 100$ servers. In all cases, the average time a task spends in the system for the simulations with $n = 100$ are higher than the expected time in the corresponding infinite system. When $\lambda = 0.5$, the deviation between the two results are smaller than 1% for all strategies. When $\lambda = 0.9$, for the strategy of choosing from two or three servers, the simulations are within 1-2% of the results obtained from the infinite system. In the case of choosing the shortest queue, the simulations are within 8-17% of the infinite system, again with the average time from simulations being larger. We expect that this larger discrepancy is due to the inaccuracy of our model for the shortest queue system, as mentioned in Section 3.1; however, this is suitably accurate to gauge system behavior. These results demonstrate the accuracy of the infinite system approach.

Several surprising behaviors manifest in the figures. First, although choosing the shortest queue is best when information is current ($T = 0$), for even very small values of T the strategy performs worse than randomly selecting a queue, especially under high loads (that is, large λ). Although choosing the shortest queue is known to be suboptimal

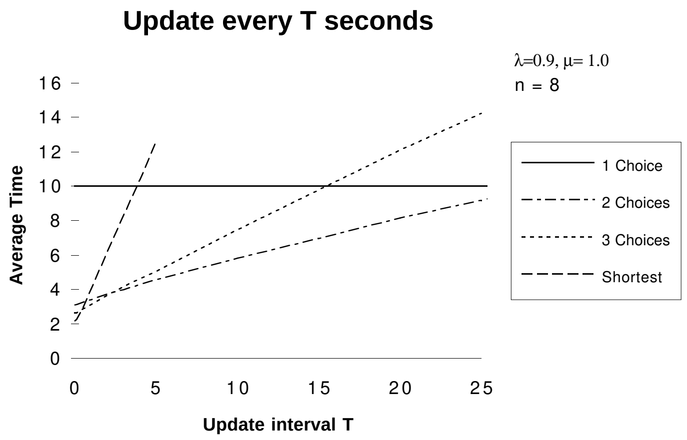

Figure 3: Strategy comparison at $\lambda = 0.90$.

in certain systems with current information [19], its failure in the presence of old information is dramatic. Also, choosing from just two servers is the best of our proposed strategies over a wide range of T, although for sufficiently large T making a single random choice performs better.

We suggest some helpful intuition for these behaviors. If the update interval T is sufficiently small, so that only a few new tasks arrive every T seconds, then choosing a shortest queue performs very well, as tasks tend to wait at servers with short queues. As T grows larger, however, a problem arises; all the tasks that arrive over those T seconds will go only to the small set of servers that appear lightly loaded on the board, overloading them while other servers empty. The system demonstrates what we call *herd behavior*: herds of tasks all move together to the same locations. (As a real-life example of this phenomenon, consider what happens at a supermarket when it is announced that "Aisle 7 is now open." Very often Aisle 7 quickly becomes the longest queue.) As the update interval $T \to \infty$, the utility of the bulletin board becomes negligible (and, in fact, it can actually be misleading!), and the best strategy approaches choosing a server at random. Although this intuition is helpful, it remains surprising that making just two choices performs substantially better than even three choices over a large interval of values of T that seem likely to arise in practice.

The same behavior is also apparent even with a much smaller number of servers. In Figure 3 we examine simulations of the same strategies with only eight servers, which is a realistic number for a current multi-processor machine. In this case the approximations given by the infinite system are less accurate, although for $T > 1$ they are still within 20% of the simulations. Other simulations of small systems demonstrate similar behavior, and as the number of servers n grows the infinite system grows more accurate. Hence, even for small systems, the infinite system approach provides reasonable estimates of system behavior and demonstrates the trends as the update interval T grows.

Finally, we note again that in all of our simulations of the differential equations, the infinite system rapidly reaches the fixed cycle suggested in Section 3.2.

4 Continuous Update

The periodic update system is just one possible model for old information; we now consider another natural model for distributed environments. In a *continuous update* system, the bulletin board is updated continuously, but the board remains T seconds behind the true state at all times. Hence every incoming task may use load information from T seconds ago in making their destination decision. This model corresponds to a situation where there is a transfer delay between the time incoming jobs determine which processor to join and the time they join.

We will begin by modeling a similar scenario. Suppose that each task, upon entry, sees a billboard with information with some time X ago, where X is an exponentially distributed random variable with mean T, and these random variables are independent for each task. We examine this model, and later consider what changes are necessary to replace the random variable X by a constant T.

Modeling this system appears difficult, because it seems that we have to keep track of the past. Instead, we shall think of the system as working as follows: tasks first enter a waiting room, where they obtain current load information about queue lengths, and immediately decide upon their destination according to the appropriate strategy. They then wait for a time X that is exponentially distributed with mean T and independent among tasks. Note that tasks have no information about other tasks in the waiting room, including how many there are and their destinations. After their wait period is finished, they proceed to their chosen destination; their time in the waiting room is not counted as time in the system. We claim that this system is equivalent to a system where tasks arrive at the servers and choose a server based on information from a time X ago as described. The key to this observation is to note that if the arrival process to the waiting room is Poisson, then the exit process from the waiting room is also Poisson, as is easily shown by standard arguments. Interestingly, another interpretation of the waiting room is as a communication delay, corresponding to the time it takes a task from a client to move to a server. This model is thus related to similar models in [9].

The state of the system will again be represented by a collection of numbers for a set of ordered pairs. In this case, $P_{i,j}$ will be the fraction of servers with j current tasks and i tasks sitting in the waiting room; similarly, we shall say that a server is in state (i, j) if it has j tasks enqueued and i tasks in the waiting room. In this model we let $q_j(t)$ be the arrival rate of tasks into the waiting room that choose servers with current load j as their destination. The expression for q_j will depend on the strategy for choosing a queue, and can easily be determined, as in Section 3.1.

To formulate the differential equations, consider first a server with in state (i, j), where $i, j \geq 1$. The queue can leave this state in one of three ways: a task can complete service, which occurs at rate $\mu = 1$; a new task can enter the waiting room, which occurs at rate $q_j(t)$; or a message can move from the waiting room to the server, which (because of our assumption of exponentially distributed waiting times) occurs at rate $\frac{i}{T}$. Similarly one can determine three ways in which a server can enter (i, j). The following equations include the boundary cases:

$$\begin{aligned}
\frac{dP_{0,0}(t)}{dt} &= P_{0,1}(t) - q_0(t)P_{0,0}(t)\,; \\
\frac{dP_{0,j}(t)}{dt} &= P_{0,j+1}(t) + \frac{P_{1,j-1}(t)}{T} - q_j(t)P_{0,j}(t) \\
&\quad - P_{0,j}(t)\,,\ j \geq 1; \\
\frac{dP_{i,0}(t)}{dt} &= q_0(t)P_{i-1,0}(t) + P_{i,1}(t) - q_0(t)P_{i,0}(t) \\
&\quad - \frac{iP_{i,0}(t)}{T}\,,\ i \geq 1; \\
\frac{dP_{i,j}(t)}{dt} &= P_{i,j+1}(t) + \frac{(i+1)P_{i+1,j-1(t)}}{T} + q_j(t)P_{i-1,j}(t) \\
&\quad - q_j(t)P_{i,j}(t) - P_{i,j}(t) - \frac{iP_{i,j}(t)}{T}\,,\ i, j \geq 1.
\end{aligned}$$

4.1 The Fixed Point

Just as in the periodic update model the system converges to a fixed cycle, simulations demonstrate that the continuous update model quickly converges to a fixed point, where $\frac{dP_{i,j}(t)}{dt} = 0$ for all i, j. We therefore expect that in a suitably large finite system, in equilibrium the distribution of server states is concentrated near the distribution given by the fixed point. Hence, by solving for the fixed point, one can the estimate system metrics such as the expected time in the queue (using, for example, Little's Law). The fixed point can be approximated numerically by simulating the differential equations, or it can be solved for using the family of equations $\frac{dP_{i,j}(t)}{dt} = 0$. In fact, this approach leads to predictions of system behavior that match simulations quite accurately, as we will detail in Section 4.3.

Using techniques discussed in [13, 14], one can prove that, for all the strategies we consider here, the fixed point is *stable*, which informally means that the trajectory remains close to its fixed point (once it gets close). We omit the straightforward argument in this extended abstract. Our simulations suggest that in fact the infinite system *converges exponentially* to its fixed point; that is, that the distance between the fixed point and the trajectory decreases geometrically quickly over time. (See [13, 14].) Although we can prove this for some special cases, proving exponential convergence for these systems in general remains an open question.

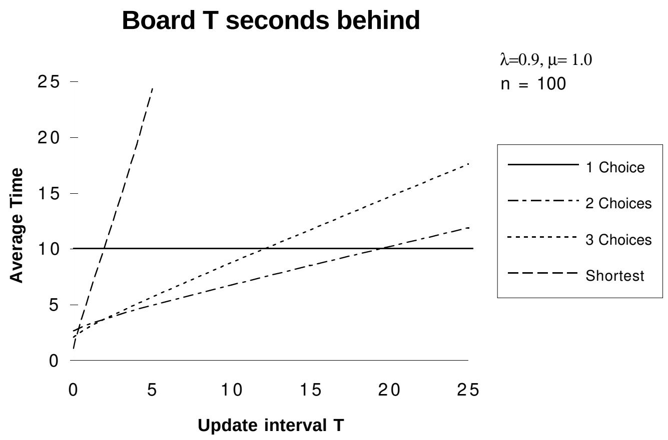

Figure 4: Each task sees the loads from T seconds ago.

4.2 Continuous Update, Constant Time

In theory, it is possible to extend the continuous update model to approximate the behavior of a system where the bulletin board shows load information from T seconds ago; that is, where X is a constant random variable of value T. The customer's time in the waiting room must be made (approximately) constant; this can be done effectively using Erlang's *method of stages*. The essential idea is that we replace our single waiting room with a series of r consecutive waiting rooms, such that the time a task spends in each waiting room is exponentially distributed with mean T/r. The expected time waiting is then T, and the variance decreases with r; in the limit as $r \to \infty$, it is as though the waiting time is constant. Taking a reasonable sized r can lead to a good approximation for constant time. Other distributions can be handled similarly. (See, e.g., [14].)

In practice, this model is difficult to use, as the state of a server must now be represented by an $r + 1$-dimensional vector that keeps track of the queue length and number of customers at each of the r waiting rooms. Hence the number of states to keep track of grows exponentially in r. It may still be possible to use this approach in some cases, by truncating the state space appropriately; however, for the remainder, we will consider this model only in simulations.

4.3 Simulations

As in Section 3.4, we present results from simulating the actual queueing systems. We have chosen the case of $n = 100$ queues and $\lambda = 0.9$ as a representative case for illustrative purposes. As one might expect, the infinite system proves more accurate as n increases, and the differences among the strategies grow more pronounced with the arrival rate.

We first examine the behavior of the system when X, the waiting room time, is a fixed constant T. In this case the system demonstrates behavior remarkably similar to the periodic update model, as shown in Figure 4. For example, choosing the shortest server performs poorly even for small values of T, while two choices performs well over a broad range for T.

When we consider the case when X is an exponentially distributed random variable with mean T, however, the system behaves radically differently (Figure 5). All three of the strategies we consider do extremely well, much better than when X is the fixed constant T. We found that the deviation between the results from the simulations and the infinite system are very small; they are within 1-2% when two or three choices are used, and 5-20% when tasks choose the shortest queue, just as in the case of periodic updates (Section 3.4).

We suggest an interpretation of this surprising behavior, beginning by considering when customers choose the shortest queue. In the periodic update model, we saw that this strategy led to "herd behavior", with all tasks going to the same small set of servers. The same behavior is evident in this model, when X is a fixed constant; it takes some time before entering customers become aware that the system loads have changed. In the case where X is randomly distributed, however, customers that enter at almost the same time may have different views of the system, and thus make different choices. Hence the "herd behavior" is mitigated, improving the load balancing. Similarly, performance improves with the other strategies as well.

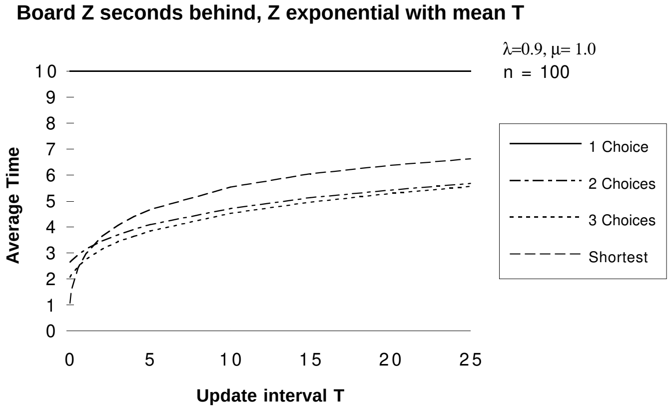

Figure 5: Each task sees the loads from X seconds ago, where the X are independent exponential random variables with mean T.

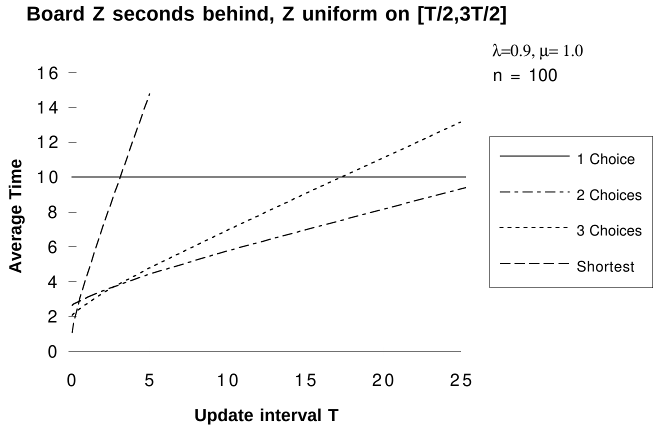

Figure 6: Each task sees the loads from X seconds ago, where the X are independent uniform random variables from $[T/2, 3T/2]$.

We justify this interpretation by considering other distributions for X; the cases where X is uniformly distributed on $[T/2, 3T/2]$ and on $[0, 2T]$ are given in Figures 6 and Figures 7. Both perform noticeably better than the case where X is fixed at T. That the larger interval performs dramatically better suggests that it is useful to have some tasks that get very accurate load information (i.e, where X is close to 0); this also explains the behavior when X is exponentially distributed.

This setting demonstrates how randomness can be used for symmetry breaking. In the periodic update case, by having each task choose from just two servers, one introduces asymmetry. In the continuous update case, one can also introduce asymmetry by randomizing the age of the load information.

This setting also demonstrates the danger of assuming that a model's behavior does not vary strongly if one changes underlying distributions. For example, in many cases in queueing theory, results are proven for models where service times are exponentially distributed (as these results are often easier to obtain), and it is assumed that the behavior when service times are constant (with the same mean) is similar. In some cases there are even provable relationships between the two models (see, for example, [11, 16]). In this case, however, changing the distribution of the random variable X causes a dramatic change in behavior.

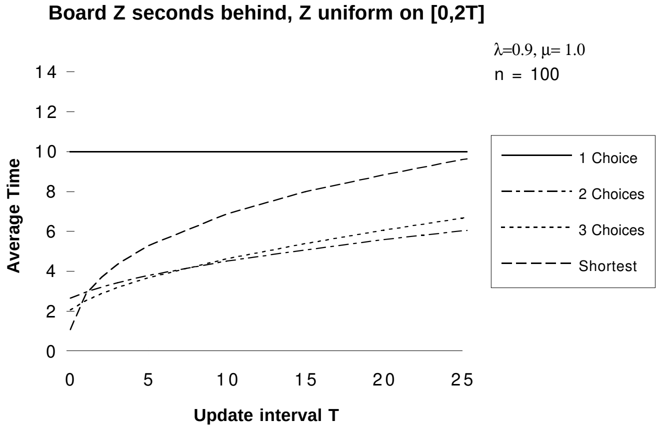

Figure 7: Each task sees the loads from X seconds ago, where the X are independent uniform random variables from $[0, 2T]$.

5 Individual updates

In the models we have considered thus far, the bulletin board contains load information from the same time t for all the servers. It is natural to ask what happens when servers update their load information at different times, as may be the case in systems where servers individually broadcast load information to clients. In an *individual update* system, the servers update the load information at the bulletin board individually. For convenience we shall assume the time between each update for every server is independent and exponentially distributed with mean T. Note that, in this model, the bulletin board contains only the load information and does not keep track of when the updates have occurred.

The state of the system will again be represented by a collection of ordered pairs. In this case, $P_{i,j}$ will be the fraction of servers with true load j but load i posted on the bulletin board. We let $q_i(t)$ be the arrival rate of tasks to servers with load i posted on the bulletin board; the expression for q_i will depend on the strategy for choosing a queue. We let $S_i(t)$ be the total fraction of servers with true load i at time t, regardless of the load displayed on the bulletin board; note $S_i(t) = \sum_j P_{j,i}(t)$.

The true load of a server and its displayed load on the bulletin board match when an update occurs. Hence when considering how $P_{i,i}$ changes, there will a term corresponding to when one of the fraction S_i of servers with load i generates an update. The following equations are readily derived in a similar fashion as in previous sections.

$$\begin{aligned}
\frac{dP_{i,0}(t)}{dt} &= P_{i,1}(t) - P_{i,0}(t)q_i(t) - P_{i,0}(t)/T\ ; \\
\frac{dP_{i,j}(t)}{dt} &= P_{i,j-1}(t)q_i(t) + P_{i,j+1}(t) - P_{i,j}(t)q_i(t) \\
&\quad - P_{i,j}(t) - P_{i,j}(t)/T,\ \ j \geq 1\ , i \neq j; \\
\frac{dP_{0,0}(t)}{dt} &= P_{i,1}(t) - P_{i,0}(t)q_i(t) - P_{0,0}(t)/T + S_0(t)/T\ ; \\
\frac{dP_{i,i}(t)}{dt} &= P_{i,i-1}(t)q_i(t) + P_{i,i+1}(t) - P_{i,i}(t)q_i(t) \\
&\quad - P_{i,i}(t) - P_{i,i}(t)/T + S_i(t)/T\ \ , i \geq 1.
\end{aligned}$$

As with the continuous update model, in simulations this model converges to a fixed point, and one can prove that this fixed point is stable. Qualitatively, the behavior appears similar to the periodic update model, as can be seen in Figure 8.

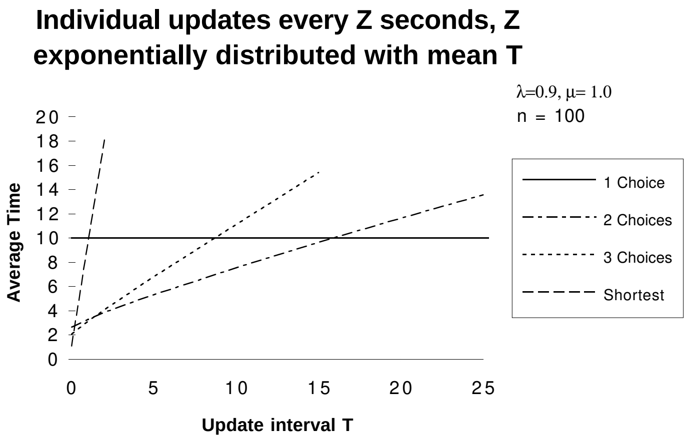

Figure 8: Each server updates the board every X seconds, where X is exponentially distributed with mean T.

6 Open Questions and Conclusions

We have considered the question of how useful old information is in the context of load balancing. In examining various models, we have found a surprising rule of thumb: choosing the least loaded of two random choices according to the old load information performs well over a large range of system parameters and is generally better than similar strategies, in terms of the expected time a task spends in the system. We have also seen the importance of using some randomness in order to prevent customers from adopting the same behavior, as demonstrated by the poor performance of the strategy of choosing the least loaded server in this setting.

We believe that there is a great deal more to be done in this area. Generally, we would like to see these models extended and applied to more realistic situations. For example, it would be interesting to consider this question with regard to other load balancing scenarios, such as in virtual circuit routing, or with regard to metrics other than the expected time in the system, such as in a system where tasks have deadlines. A different theoretical framework for these problems, other than the infinite system approach, might be of use as well. In particular, it would be convenient to have a method that yields tighter bounds in the case where n, the number of servers, is small. Finally, the problem of handling more realistic arrival and service patterns appears quite difficult. In particular, it is well known that when service distributions are heavy-tailed, the behavior of a load balancing system can be quite different than when service distribution are exponential; however, we expect our rule of thumb performs well in this scenario as well.

7 Acknowledgments

The author thanks the many people at Digital Systems Research Center who offered input on this work while it was in progress. Special thanks go to Andrei Broder, Ed Lee, and Chandu Thekkath for their many helpful suggestions.

References

[1] M. Alanyali and B. Hajek, "Analysis of Simple Algorithms for Dynamic Load Balancing", *INFOCOM '95*.

[2] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton, "Making Commitments in the Face of Uncertainty: How to Pick a Winner Almost Every Time", *Proceedings of the 28th ACM Symposium on the Theory of Computing*, 1996, pp. 519–530.

[3] Y. Azar, A. Broder, A. Karlin, and E. Upfal, "Balanced Allocations", *Proceedings of the 26th ACM Symposium on the Theory of Computing*, 1994, pp. 593–602.

[4] D. R. Cox, W. L. Smith, **Queues**, Wiley, 1961.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, “Adaptive load sharing in homogeneous distributed systems”, *IEEE Transactions on Software Engineering*, Vol. 12, 1986, pp. 662–675.

[6] S. N. Ethier and T. G. Kurtz, **Markov Processes: Characterization and Convergence**, 1986, John Wiley and Sons.

[7] R. M. Karp, M. Luby, and F. Meyer auf der Heide, “Efficient PRAM Simulation on a Distributed Memory Machine”, *Proceedings of the 24th ACM Symposium on the Theory of Computing*, 1992, pp. 318–326.

[8] T. G. Kurtz, **Approximation of Population Processes**, SIAM, 1981.

[9] R. Mirchandaney, D. Towsley, and J. A. Stankovic, “Analysis of the Effects of Delays on Load Sharing”, *IEEE Transactions on Computers*, Vol. 38, 1989, pp. 1513–1525.

[10] R. Mirchandaney, D. Towsley, and J. A. Stankovic, “Adaptive Load Sharing in Heterogeneous Distributed Systems”, *Journal of Parallel and Distributed Computing*, Vol. 9, 1990, pp. 331–346.

[11] M. Mitzenmacher, “Constant Time per Edge is Optimal on Rooted Tree Networks”, *Proc. of the* 8^{th} *ACM Symp. on Parallel Algorithms and Architectures*, 1996, pp. 162–169.

[12] M. Mitzenmacher, “Load Balancing and Density Dependent Jump Markov Processes”, *Proc. of the* 37^{th} *IEEE Symp. on Foundations of Computer Science*, 1996, pp. 213–222.

[13] M. Mitzenmacher, “The Power of Two Choices in Randomized Load Balancing”, Ph.D. thesis, University of California, Berkeley, September 1996.

[14] M. Mitzenmacher, “On the Analysis of Randomized Load Balancing Schemes”, to appear in *SPAA ’97*.

[15] A. Shwartz and A. Weiss, **Large Deviations for Performance Analysis**, 1995, Chapman & Hall.

[16] G. D. Stamoulis and J. N. Tsitsiklis, “The Efficiency of Greedy Routing in Hypercubes and Butterflies”, *IEEE Transactions on Communications*, Vol. 42(11), 1994, pp. 3051–3061.

[17] D. Towsley and R. Mirchandaney, “The Effect of Communication Delays on the Performance of Load Balancing Policies in Distributed Systems”, *Proceedings of the Second International MCPR Workshop*, 1988, pp. 213–226.

[18] R. Weber, “On the Optimal Assignment of Customers to Parallel Servers”, *J. of Appl. Prob.*, Vol 15, 1978, pp. 406–413.

[19] W. Whitt, “Deciding Which Queue to Join: Some Counterexamples”, *Operations Research*, Vol 34, 1986, pp. 55–62.

[20] W. Winston, “Optimality of the Shortest Line Discipline”, *J. of Appl. Prob.*, Vol 14, 1977, pp. 181–189.

[21] N. C. Wormald, “Differential Equations for Random Processes and Random Graphs”, *Annals of Appl. Prob.*, Vol 5, 1995, pp. 1217–1235.

[22] N.D. Vvedenskaya, R.L. Dobrushin, and F.I. Karpelevich. “Queueing System with Selection of the Shortest of Two Queues: an Asymptotic Approach”, *Problems of Information Transmission*, Vol 32, 1996, pp. 15–27.