How Useful Is Old Information?

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 multiprocessor 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 queue lengths 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 queue length 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 carefully; for example, unlike most settings where the load information is current, having tasks go to the apparently least loaded server can significantly hurt performance.

Index Terms—Load balancing, stale information, old information, queuing theory, large deviations.

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 extremely busy, system performance can fall significantly. To prevent this, *load balancing* is often used to distribute the workload and improve 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 (such as, for instance, the time for a task to access various servers), general models from both queuing theory and computer science often provide valuable insight and general rules of thumb.

In this paper, we develop general analytical models for the realistic setting where old information about queue lengths is available. For convenience, we generally refer to the number of tasks queued at a server as its *load*. 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 queued at each server, it is often best for the task to go to the server with the shortest queue [25]. 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.

1.1 Related Previous Work

The problem of how to use old or inexact information is often 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. In the control theory community, some work has considered how to design optimal control policies in the face of delayed information, although currently these results appear to apply only to a single queue (see, e.g., [2], [3], [13]).

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 et al. [11]. A more complete analysis was given by Azar et al. [4]. In the dynamic setting, this work was extended to a queuing theoretic model in [18], [19]; similar results were independently reported in [29].

- *The author is with the Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138.*
E-mail: michaelm@eecs.harvard.edu.

Manuscript received 4 Nov. 1997; revised 10 Nov. 1998; accepted 10 June 1999.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 105884.

1045-9219/00/$10.00 © 2000 IEEE

Other similar previous work includes that of Towsley and Mirchandaney [24] and that of Mirchandaney et al. [15], [16]. These authors examine how some simple load sharing policies are affected by communication delay, extending a similar study of load balancing policies by Eager et al. [6], [7]. Their analyses are based on Markov chains associated with the load sharing policies they propose, as well as simulation results.

Our work is most related to the queuing models of the above work, although it 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 often call this system the *limiting system*. This approach has successfully been applied previously to study load balancing problems in [1], [18], [19], [20], [22], [29] (see also [1] for more references or [21], [28] for the use of this approach in different settings), 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, including the effectiveness of using just two choices.

The remainder of this paper is organized as follows: In Section 2, we describe a general queuing 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 limiting system and, using the limiting systems and simulations, we determine important behavioral properties of these models. In Section 6, we briefly consider the question of cheating tasks, a concept that ties our models to natural, but challenging, game theoretic questions. 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 task 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, hence, 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 task's 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 (possibly) their entry time. Note that, because of this lack of coordination among tasks, natural policies such as round-robin are not generally feasible—such a policy would require tasks to pass though a central coordinated server. 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. (Ties are broken randomly.)
- 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 queuing at the least loaded has been shown to perform significantly better in the case where the load information is up to date [6], [18], [19], [29]. It has also proven effective in other similar models [4], [11], [19]. Moreover, the strategy also appears to be practical and to 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 [25], [27]. Hence, it makes an excellent point of comparison in this setting. Other simple schemes that we do not examine here but can easily study with this model include threshold-based schemes [6], [20], where a second choice is made only if the first appears unsatisfactory.

We develop analytical results for the limiting case as $n \to \infty$, for which the system can be accurately modeled by a *limiting system*. The limiting system consists of a set of differential equations, which we shall describe below, that describe, in some sense, 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 [8], [14], [18], [19], [20], [22], [28], [29]; we emphasize that here we will not detour into a theoretical justification for this limiting approach and, instead, refer the reader to these sources for more information. (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 limiting system to estimate performance metrics such as the expected time in the system proves accurate, even for relatively small values of n [6], [18], [19], [20]. We shall verify this for the models we consider by comparing our analytical results with simulations.

3 Periodic Updates

The previous section described possible ways that the bulletin board can be used. We now turn our attention to how a bulletin 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 limiting 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, which we focus on in this section, 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 tasks on the bulletin board, but having j tasks: We say such a server is in state (i, j). Let $i, j \geq 1$. What is the rate at which a server leaves state (i, j)? A server leaves this state when a task departs, which happens at rate $\mu = 1$, or when a task arrives, which happens at rate $q_i(t)$. Similarly, we may ask at what rate tasks enter such a state. This can happen if a task arrives at a server with load i posted on the bulletin board, but having $j - 1$ tasks, or a task departs from a server with load i posted on the bulletin board, but having $j + 1$ tasks. This description naturally leads us to model the behavior of the system by the following set of differential equations:

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

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

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 queuing 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 [8], [14], [19], [20], [22], [28], [29]. 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 $b_i(t)$ to 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 λ the arrival rate and with load i, the total fraction of queues showing i on the board $b_i(t)$. 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 exist 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 limiting 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 $P = (P_{i,j}(k_0 T))$ 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 limiting 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 given by the differential equations 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 [5, pp. 60-64]); the probability of finishing with j tasks after T seconds, which we denote by $m_{i,j}$, is

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

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 determine a simple closed form for the state at the beginning of the phase 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 limiting system, important questions remain regarding convergence. One question stems from the approximation of a finite system with the corresponding limiting system: How good is this approximation? The second question is whether the trajectory of the limiting system given by the differential equations always converges to its fixed cycle and, if so, how quickly? For the first question, we note that the standard methods referred to previously (based on work by Kurtz [8], [14], [22]) provide only very weak bounds on the convergence rate between limiting and finite systems. By focusing on a specific problem, proving tighter bounds may be possible (see, for example, the discussion in [28]). In practice, however, as we shall see in Section 3.4, the limiting 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 see. (See also [19].) 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 limiting system approach does in fact yield a good approximation for the finite case; second, we wish to gain insight into the problem of load balancing using old information. We focus on the expected time in the system, as this appears the most interesting system measure. Because our limiting approach provides a full description of the system state, however, it can be used to predict other quantities of interest as well.

With regard to the first goal, we begin by noting that, for systems of 100 queues, the difference between the results from the simulations and the results obtained by calculating the expected time using the fixed cycle determined by the differential equations generally match to within 2 percent for the strategy of choosing from two or three servers (for the arrival rates presented here). In the case of choosing the shortest queue, the simulations are within about 10-20 percent of the limiting system. More details are given subsequently. However, because the results from simulations and the limiting system are

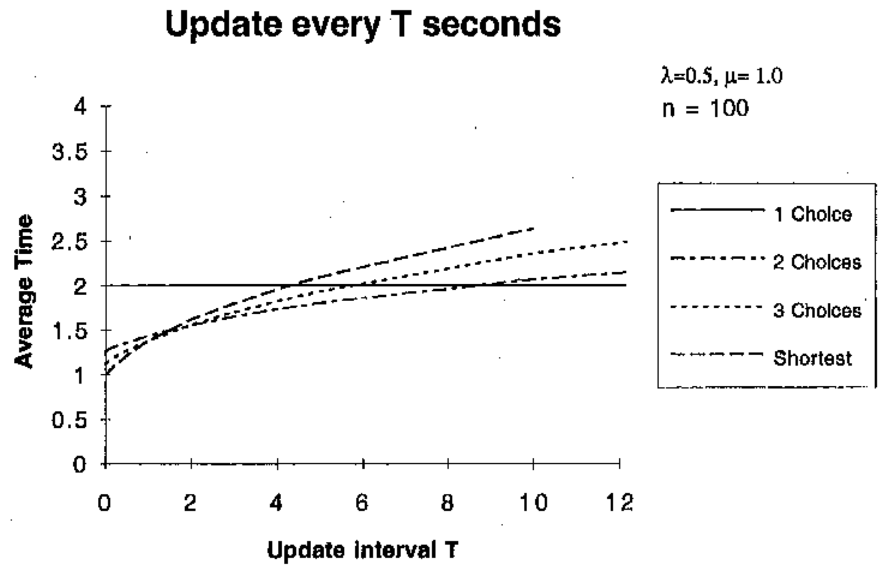

Fig. 1. Strategy comparison at $\lambda = 0.50$, 100 queues.

essentially indistinguishable, we choose to plot only the results from simulations to avoid excessive clutter (except in the case where one server is chosen at random; in this case, we simply apply standard formulae from queuing theory).

This setup allows us to emphasize the second goal, that of gaining insight into the behavior of these systems. Note that the intuition we derive from the plots would not change if we substituted the results from the equations, since they are essentially the same. This methodology may raise the question of why the limiting system models are useful at all. There are several reasons: First, simulating the differential equations is often faster than simulating the corresponding queuing system (we shall say more on this later). Second, the limiting systems provide a theoretical framework for examining these problems that can lead to formal theorems. Third, the limiting 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 Figs. 1 and 2, the results for various strategies are given for arrival rates $\lambda = 0.5$ and $\lambda = 0.9$ for $n = 100$ servers.[1] Simulations were performed for 50,000 time steps, with the first 5,000 steps ignored to allow the dependence on the initial state to not affect the results; the results presented are the average of three separate simulations. In all cases, the average time a task spends in the system for the simulations with $n = 100$ is higher than the expected time in the corresponding limiting system. When $\lambda = 0.5$, the deviation between the two results is smaller than 1 percent for all strategies. When $\lambda = 0.9$, for the strategy of choosing from two or three servers, the simulations are within 1-2 percent of the results obtained from the limiting system. In the case of choosing the shortest queue, the simulations are within 8-17 percent of the limiting 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 described in Section 3.1; however, this is suitably accurate to gauge system behavior. Again, we emphasize the accuracy of the limiting 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 in certain systems with current information [26], 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. This herd behavior has been noticed in real systems that use old information in load balancing; for example, in a discussion of the TranSend system, Fox et al. note that,

[1] In the simulations, queue choices were made without replacement. There is no difference in the limiting system, although, in practice, making choices without replacement yields small improvements. Also, the simulations were performed for specific values of T; specifically, they include $T = 0, 0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 15.0, 20.0, 25.0, 50.0$.

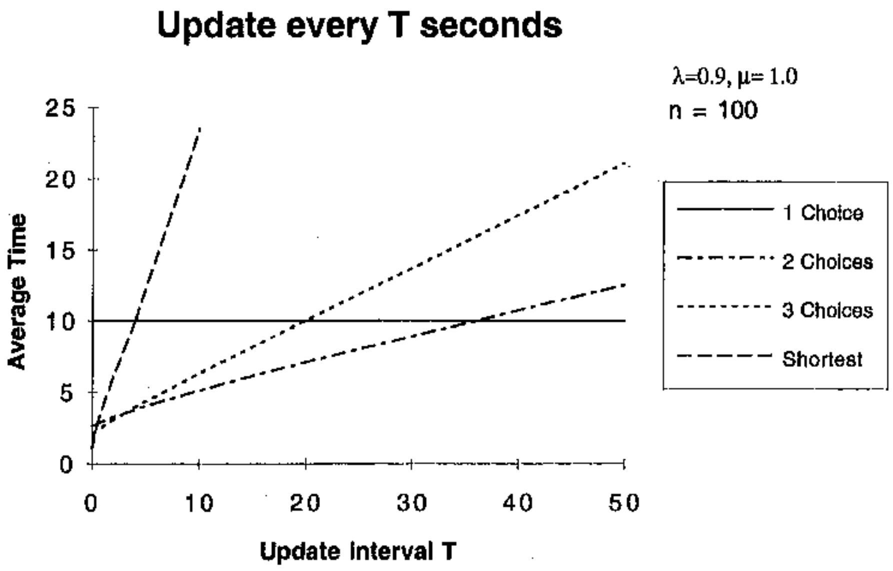

Fig. 2. Strategy comparison at $\lambda = 0.90$, 100 queues.

initially, they found "rapid oscillations in queue lengths" because their system updated load information periodically [10, Section 4.5].

Interestingly, as the update interval $T \to \infty$, the utility of the bulletin board becomes negligible (and, in fact, it can actually be misleading!). The limit as $T \to \infty$ corresponds to a setting with no information; in this case, in the distributed setting, the best strategy is to choose 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. Note that this holds even as the number of queues grows arbitrarily large; these plots accurately reflect the trends of the limiting system as the number of queues grows to infinity!

The same behavior is also apparent even with a much smaller number of servers. In Fig. 3, we examine simulations of the same strategies with only eight servers, which is a realistic number for a current multiprocessor machine. In this case, the approximations given by the limiting system are less accurate, although, for $T > 1$, they are still within 20 percent of the simulations. Other simulations of small systems demonstrate similar behavior and, as the number of servers n grows, the limiting system grows more accurate. Hence, even for small systems, the limiting 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 limiting system rapidly reaches the fixed cycle suggested in Section 3.2.

3.5 On Simulating the Limiting System

Although the limiting system approach provides a useful technique for studying load balancing models, it becomes difficult to use in the periodic update model (and other models for old information) at high arrival rates or for large values of T because the number of variables to track grows large. For example, suppose we simulate the differential equations, truncating the system at sufficiently large values of i and j that we denote by I and J. Then, we must keep track of $I \cdot J$ variables $P_{i,j}$. At high arrival rates (say, $\lambda = 0.99$) and/or high values of T, we will need to make I and J both extremely large to obtain accurate calculations and, hence, simulating the differential equations over a period of time becomes very slow, comparable to or worse than the time required to simulated the underlying queuing system.

In practice, however, we expect such high arrival rates and extremely large values of T are unlikely to be of interest. In the normal case, then, we expect I and J to be relatively small, in which case simulating the differential equations is generally quicker than simulating the underlying queuing model.

An actual time comparison depends on such factors as the time granularity used for simulating steps of the differential equations, the length of time (or number of times) one simulates the actual queuing process, as well as the quality of the code. As an example, we have found that simulating the differential equations for values $\lambda = 0.9$, $I = J = 50$, $T = 5.0$, and the granularity of successive time steps $dt = 0.01$ for 500 seconds takes approximately the same time as a single simulation for the queuing network with 100 queues over 5,000 seconds.

Examining this comparison more closely, we see that simulating the differential equations as described above requires updating 2,500 entries $P_{i,j}$ each 50,000 times, where each update requires a small constant number of floating point operations. Simulating a system of 100 servers as described above requires handling approximately 450,000 arrival events and departure events. Ignoring cache issues and assuming one cycle per instruction, we find that, for the times to be equal, the cost for handling a task would be approximately in the small thousands of

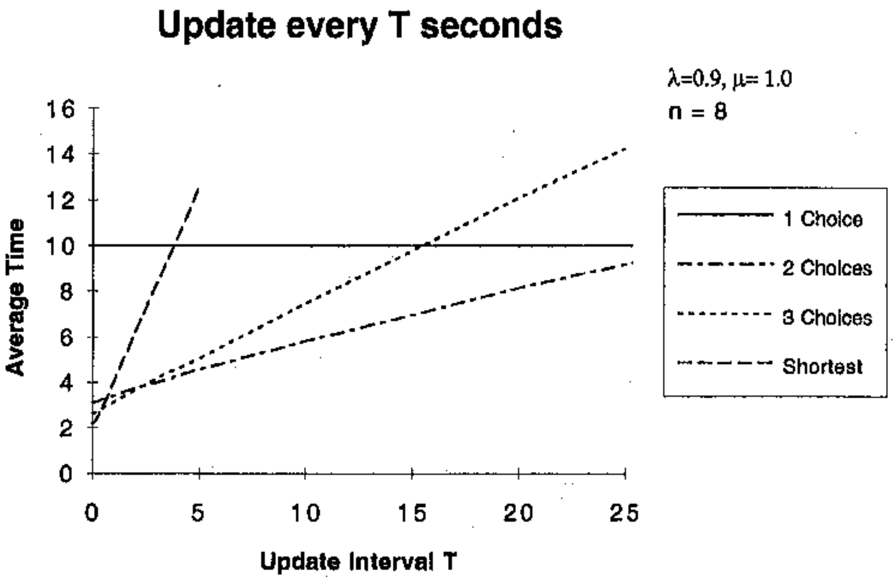

Fig. 3. Strategy comparison at $\lambda = 0.90$, eight queues.

instructions. Since each entering task requires generating a random arrival and service time, placing a new entry in a priority queue, choosing and comparing a random selection of servers, updating the recorded statistics, etc., this appears reasonable.

Repeated simulations, however, appear necessary if one is concerned with the variance introduced by the simulation. Moreover, using the differential equations, one can stop as soon as the appropriate fixed cycle is reached; with simulations, one must rely on repeated simulations to determine when the variance appears sufficiently small.

3.6 More Complex Centralized Strategies

In this section, we briefly consider centralized strategies under this delay model. Although we focus on distributed strategies throughout the rest of the paper, we detour slightly here in order to demonstrate the use of limiting systems for centralized strategies and to gain insight into the potential gains from centralization.

One would expect that a more sophisticated strategy for dealing with the old load information might yield better performance. For instance, if the system uses the load information on the bulletin board to develop an estimate for the current load, this estimate may be more accurate than the information on the board itself. Therefore, in this section, we consider more complex strategies that attempt to estimate the current queue length and gauge their performance. These strategies require significant centralization in that all incoming tasks must have access to the complete bulletin board and more detailed information about the entire system. We believe these strategies are practical for systems of reasonable size (hundreds of processors) and, hence, are worth examining.

Our model is still that the bulletin board is updated every T seconds. Our first proposed strategy requires that the arrival rate to the system and the entire composition of the bulletin board be known to the incoming tasks; also, tasks need to know the time since the last update. This situation could arise if the bulletin board is periodically broadcast to the servers generating the tasks. This strategy still assumes that tasks do not coordinate actions and, thus, the centralization required for this strategy is minimal. The idea of the strategy is to use our knowledge of the arrival rate to calculate the expected number of tasks at the servers and, then, choose a server with the smallest expected load uniformly at random. We describe a strategy that approximates this one closely and has the advantage that the underlying calculations are quite simple.

In this proposed strategy, which we call the *time-based strategy*, we split each phase of T seconds into smaller subintervals; in a subinterval $[t_k, t_{k+1})$, a task will choose a server randomly from all servers with load at most k. The division of the phase is inductively determined by the loads at the beginning of the phase, which is information available on the bulletin board. At time 0, tasks choose from all servers with load 0 posted on the board (if any exist). Hence, $t_0 = 0$. Tasks begin also choosing from servers with load 1, when the expected number of arrivals to servers of load 0 has been 1, so that

$$t_1 = t_0 + b_0/\lambda.$$

Similarly, tasks begin choosing from servers with load at most k when the expected number of arrivals to servers of load $k-1$ is 1, or at

$$t_k = t_{k-1} + \frac{\sum_{i<k} b_i}{\lambda}.$$

Intuitively, this strategy attempts to equalize the load at the servers in the natural way.

A limiting system, given by a series of differential equations, can be used to model this system. The equations are entirely similar to (1) and (2), except that the expression

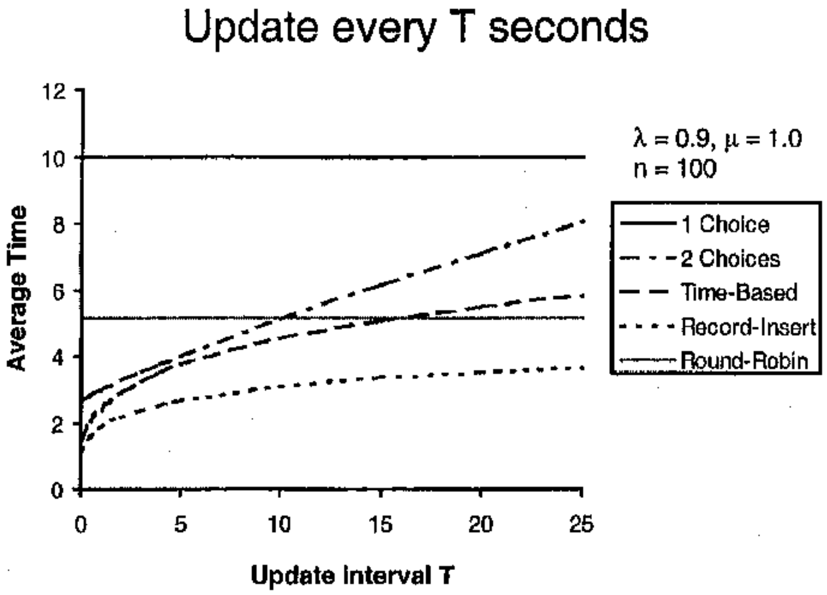

Fig. 4. Comparing centralized strategies vs. distributed strategies: Centralized strategies can perform better.

for $q_i(t)$ changes depending on the subinterval t_k. (We leave the remaining work of the derivation to the reader.)

Our second proposed strategy, which we call the *record-insert strategy*, requires more centralization in that we allow the tasks to update the global bulletin board when they choose a server. That is, every time a new task enters, the system load for the server on the bulletin board is incremented, but deletions are not recorded until the board is updated (every T seconds). Tasks choose a queue uniformly at random from those with the smallest load on the board.[2] This strategy may be feasible when the tasks use a centralized system for placement, but there is a large delay for servers to update load information. This strategy is essentially the one used to solve the problem of herding behavior in the TranSend system mentioned previously [9], [10].

Again, a limiting system given by a family of differential equations can model this system. We still use $P_{i,j}$ to represent the fraction of queues with load i on the bulletin and j at the queue; however, the load at the board is now incremented on arrival. The resulting equations are again similar to (1) and (2) with this difference:

$$\frac{dP_{i,0}(t)}{dt} = P_{i,1}(t) - P_{i,0}(t)q_i(t)\ ,\ i \geq 0; \tag{3}$$

$$\frac{dP_{0,j}(t)}{dt} = P_{0,j+1}(t) - P_{0,j}(t)\ ,\ j \geq 1; \tag{4}$$

$$\begin{aligned}\frac{dP_{i,j}(t)}{dt} &= (P_{i-1,j-1}(t)q_{i-1}(t) + P_{i,j+1}(t)) \\ &\quad - (P_{i,j}(t)q_i(t) + P_{i,j}(t))\ ,\ i,j \geq 1.\end{aligned} \tag{5}$$

Unfortunately, this system proves more complicated because the expression for $q_i(t)$ becomes more complicated. Now, $q_i(t)$ is zero unless i is the smallest load apparent in the system. Because the smallest load changes over time, the system will have discontinuous behavior; this makes the differential equations slightly harder to simulate.

Simulations of systems of 100 queues demonstrate that these strategies can perform substantially better than choosing two when n is reasonably large and T grows large, as shown in Fig. 4. Again, we emphasize that the results from the limiting systems provide results very close to that of the simulations of 100 queues; for the expected time in the system, our simulation results are within 4 percent for the time-based strategy and within 2 percent for the record-insert strategy. We present simulation results only for convenience; the same intuitions can be derived from the differential equations alone.

As one might expect, record-insert does better than time-based, demonstrating the power of the tasks being able to update an actual centralized bulletin board directly. However, choosing the shortest of two random servers still performs reasonably well in comparison, demonstrating that, in distributed settings where global information may be difficult to maintain or the arrival rate is not known in advance, it remains a strong choice. We also compare these strategies with a simple round-robin strategy, which is the natural choice for load balancing in a centralized system where no load information is available. Indeed, as previously mentioned, the record-insert strategy becomes a round-robin strategy in the limit. The expected time a job spends in a round-robin system with n servers in equilibrium can be calculated using standard queuing theory, as each queue behaves like a G/M/1 queue (see, e.g., [12, chapter 6]). Again, as one might expect, for suitably small delays, making use of the available load information even in limited ways yields better performance.

[2] We note that performance improves slightly if the tasks break ties in some fixed order, such as by machine index; in this case, for sufficiently long updates T, the strategy becomes a round-robin scheme. However, this model cannot be easily described by a limiting system.

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 its 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 [15].

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 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 then 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 described in Section 4.3.

Using techniques discussed in [19], [20], 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 proof here. Our simulations suggest that, in fact, the limiting 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 [19], [20].) Although we can prove this for some special cases, proving exponential convergence for these systems in general remains an open question.

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 task'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

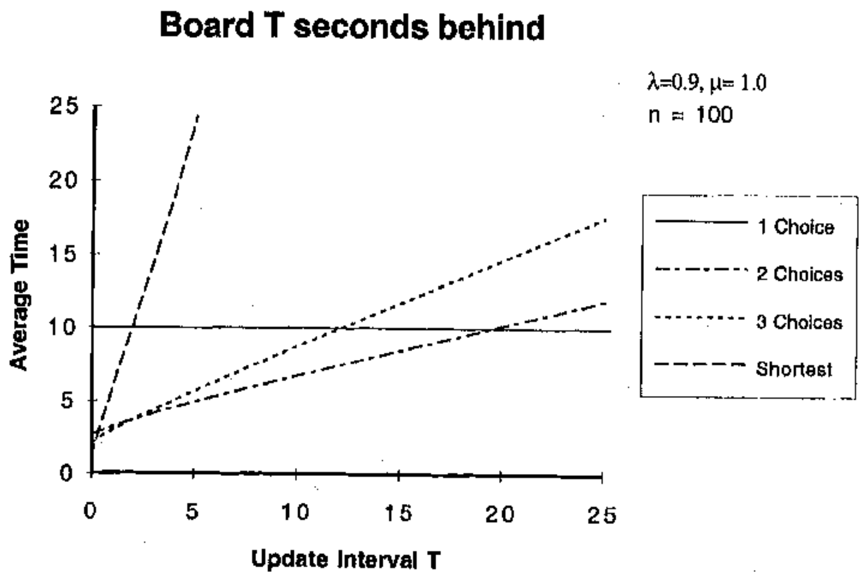

Fig. 5. Each task sees the loads from T seconds ago.

though the waiting time is constant. Taking a reasonably sized r can lead to a good approximation for constant time. Other distributions can be handled similarly. (See, e.g., [20].)

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 tasks 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 queuing systems. We emphasize that, for the continuous update case, we only develop a useful limiting system for the case where X is distributed exponentially; in other cases, we rely solely on simulations of the actual queuing system. 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 limiting 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 similarly to the periodic update model, as shown in Fig. 5. 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 (Fig. 6). All three of the strategies we consider do extremely well, much better than when X is the fixed constant T. One might think from these results that there is some error in our simulation of this case. The limiting system, however, verifies the simulation results; we found that the results from the simulations and the limiting system match within 1-2 percent when two or three choices are used and 5-20 percent 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 tasks 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 tasks become aware that the system loads have changed. In the case where X is randomly distributed, however, tasks 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.

We justify this interpretation by considering other distributions for X; results from simulations in the cases where X is uniformly distributed on $[T/2, 3T/2]$ and on $[0, 2T]$ are given in Fig. 7 and Fig. 8. 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.

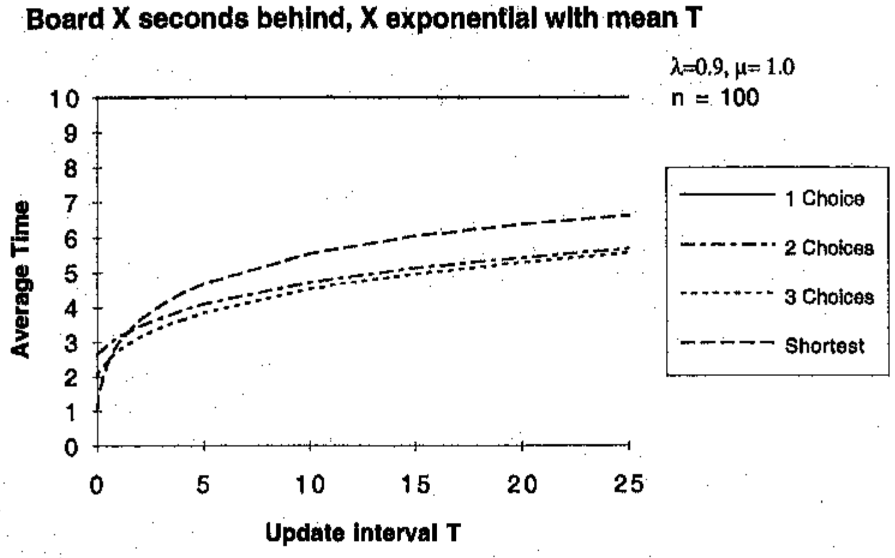

Fig. 6. Each task sees the loads from X seconds ago, where the X are independent exponential random variables with mean T.

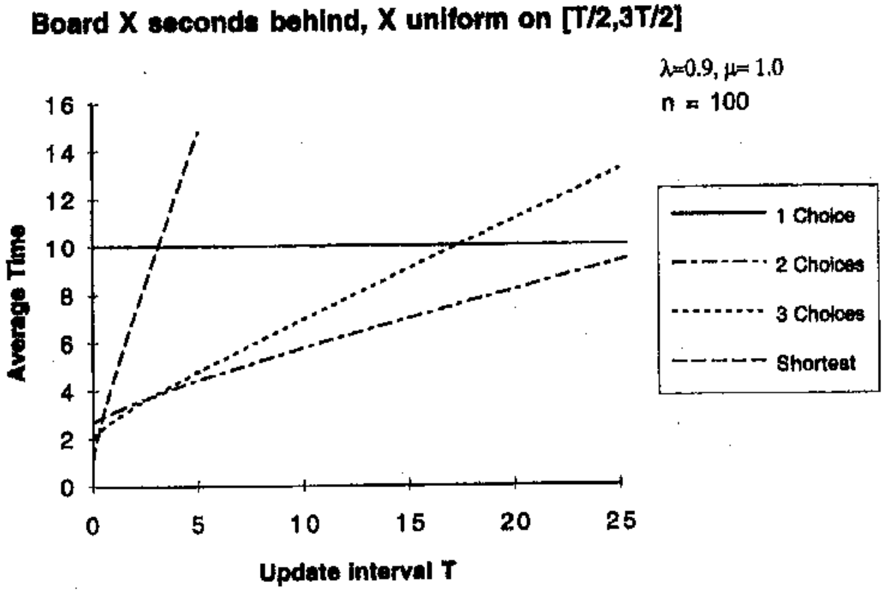

Fig. 7. Each task sees the loads from X seconds ago, where the X are independent uniform random variables from $[T/2, 3T/2]$.

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 queuing 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, [17], [23]). In this case, however, changing the distribution of the random variable X causes a dramatic change in behavior.

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

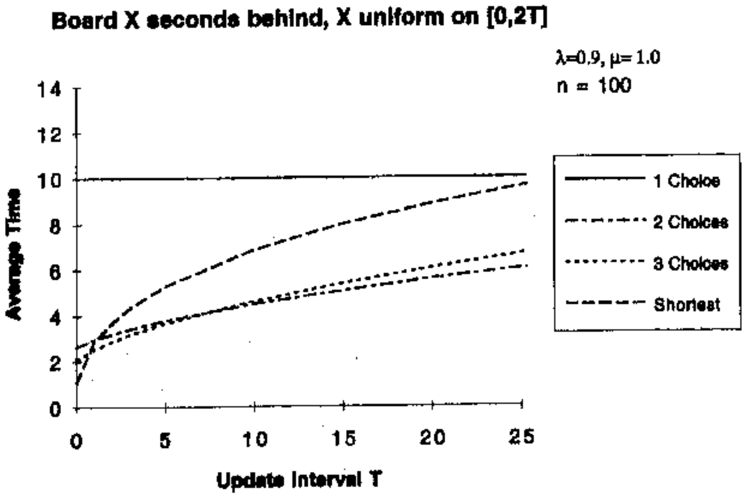

Fig. 8. Each task sees the loads from X seconds ago, where the X are independent uniform random variables from $[0, 2T]$.

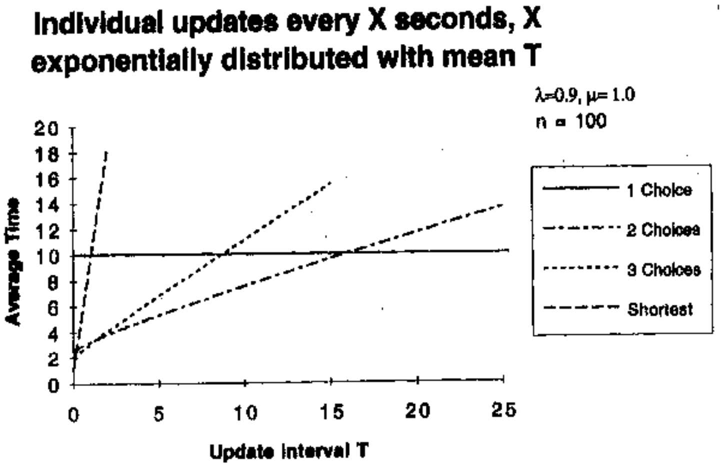

Fig. 9. Each server updates the board every X seconds, where X is exponentially distributed with mean T.

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 $P_{i,j}$ 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 Fig. 9. We note again that the simulations and the results from the differential equations are very close. For two or three choices, the results are within 1 percent for small T and within 5 percent for larger T. For the strategy of going to the shortest queue, the deviation is slightly larger.

TABLE 1
Comparing Simulation Results for Antisocial Tasks (which Choose the Shortest) against those that Choose Two, for $\lambda = 0.9$ and $n = 100$

T	p	Avg. Time All Tasks	Avg. Time 2 Choices	Avg. Time Shortest	Variance All Tasks	Variance 2 Choices	Variance Shortest
1	0.00	3.23286					
1	0.01	3.21072	3.22093	2.19877	5.73117	5.74186	3.63718
1	0.05	3.17061	3.21389	2.34814	5.62948	5.67621	4.02956
1	0.10	3.14132	3.20978	2.52474	5.58554	5.65450	4.54205
1	0.25	3.20098	3.25693	3.03311	6.05849	5.94553	6.35980
5	0.00	4.94051					
5	0.01	4.95386	4.95677	4.66575	13.8029	13.8821	11.8131
5	0.05	5.05692	5.05668	5.06154	14.4591	14.5105	13.4837
5	0.10	5.21456	5.17956	5.52974	15.6083	15.6597	15.7552
5	0.25	6.06968	5.70758	7.15609	23.6380	22.0182	26.9240
10	0.00	6.74313					
10	0.01	6.80669	6.80588	6.88703	26.4946	26.5391	22.0827
10	0.05	7.00344	6.97692	7.50776	28.4836	28.6189	25.6448
10	0.10	7.36957	7.26152	8.34185	32.7326	32.7395	31.6201
10	0.25	8.91193	8.23577	10.9422	54.8097	52.0265	57.6721

6 Competitive Scenarios

We have assumed thus far in our models that all tasks adopt the same underlying strategy and the goal has been to reduce the expected time for all tasks. In a more competitive environment, tasks may instead independently act in their own best interests and it is necessary to consider the effects of antisocial, competitive tasks which may not follow the proposed universal strategy.

We consider briefly a specific example. Suppose we have a system where each task is supposed to choose from the shortest of two randomly chosen servers. In this case, an antisocial task may attempt to improve its own situation by obtaining the entire bulletin board and proceeding to a server with the smallest posted load. Do such tasks do better than other tasks? If so, in a competitive environment, tasks have little motivation to follow the suggested strategy.

We study the problem by examining the situation where each task adopts the antisocial strategy with probability p. With such a model, it is possible to set up a corresponding limiting system since each task's strategy can be expressed as a probabilistic mixture of two strategies; for example, in this case,

$$q_0(t) = \frac{p\lambda}{b_0(t)} + \frac{(1-p)\lambda\left(\sum_{j\geq 0} b_i(t)\right)^2 - \left(\sum_{j>0} b_i(t)\right)^2}{b_0(t)}$$
$$= \frac{\lambda\left[p + (1-p)(1 - \left(\sum_{j>0} b_i(t)\right)^2)\right]}{b_0(t)}.$$

For $i > 0$,

$$q_i(t) = (1-p)\lambda\frac{\left(\sum_{j\geq i} b_i(t)\right)^d - \left(\sum_{j>i} b_i(t)\right)^d}{b_i(t)}.$$

We consider the case where all tasks see load information from exactly T seconds ago. In this case, as discussed in Section 4.3, we do not use a limiting system, as the state space grows rather complex; instead, we use simulations.[3] The results demonstrate some interesting behaviors. Table 1 provides numerical results based on simulations for $\lambda = 0.9$ and $n = 100$ servers. When T is small or the fraction p of competitive tasks is sufficiently small, competitive tasks reduce their average time by acting against the standard strategy. In cases where choosing two servers performs poorly, introducing competitive tasks can actually reduce the average time for everyone, although, more often, antisocial tasks do better at the expense of other tasks. For larger values of T or p, system performance degrades for all tasks and the average time antisocial tasks spend in the system can grow much larger than that of other tasks. In this sense, tasks are motivated not to choose the shortest for, if too many do so, their average time in the system will be larger than those that do not.

The situation becomes even more interesting, however, if the measure of performance is not the average time in the system, but a more complicated measure. For example, it may be important for some tasks to finish by a certain deadline and, in this case, the goal is to maximize the probability that it finishes by its deadline. Our simulations have also shown that, in the model described above, even when p and T are such that choosing the server with the shortest posted queue increases the average time for a task,

[3] We could instead have presented results for the periodic update setting, where the board is updated every T seconds. In this case, the limiting system again matches simulations of 100 servers quite well. One finds similar behaviors in such systems.

the *variance* in the time in the system of tasks which adopt this strategy can be lower than other tasks (Table 1). Intuitively, this is probably because some tasks that make only two choices will be quite unlucky and choose two very long queues. Hence, tasks with deadlines may be motivated to try another strategy, even though it appears worse in terms of the average time in the system.

We believe there are many open questions to consider in this area, and we discuss them further in the conclusion.

7 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 tasks 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 limiting 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 distributions are exponential; however, we expect our rule of thumb to perform well in this scenario as well.

An entirely different flavor of problems arises from considering the problem of old information in the context of game theory. We have generally assumed in our models that all tasks adopt the same underlying strategy and the goal has been to reduce the expected time for all tasks. In a more competitive environment, tasks may instead independently act in their own best interests and, hence, in Section 6, we considered the effects of antisocial tasks which may not follow the proposed strategy. More generally, we may think of these systems as multiplayer games, which leads to several interesting questions: If each task is an individual player, what is the optimal strategy for a self-interested player (i.e., a task whose only goal is to minimize its own expected time in the system, say)? How easily can this strategy be computed on-line? Is this strategy different than the optimal strategy to minimize the average expected time and, if so, how? Are there simple stable strategies in which no task is motivated to deviate from the strategy for its own gain?

Acknowledgments

Much of this work was undertaken while the author was at Digital Systems Research Center, Palo Alto, California, and the author thanks the many people there 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," *Math. Operations Research*, vol. 22, no. 4, pp. 840-871, 1997.

[2] E. Altman and P. Nain, "Closed Loop Control with Delayed Information," *Proc. ACM Sigmetrics Conf. Measurement and Modeling of Computer Systems*, pp. 193-204, Newport, R.I., June 1992.

[3] E. Altman and S. Stidham, "Optimality of Monotonic Policies for Two-Action Markovian Decision Processes, with Applications to Control of Queues with Delayed Information," *Queueing Systems: Theory and Applications*, vol. 21, nos. III-IV, pp. 267-291, 1995.

[4] Y. Azar, A. Broder, A. Karlin, and E. Upfal, "Balanced Allocations," *Proc. 26th ACM Symp. Theory of Computing*, pp. 593-602, 1994.

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

[6] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Eng.*, vol. 12, pp. 662-675, 1986.

[7] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation Review*, vol. 16, pp. 53-68, 1986.

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

[9] A. Fox, Private communication.

[10] A. Fox, S.D. Gribble, Y. Chawathe, E.A. Brewer, and P. Gauthier, "Cluster-Based Scalable Network Services," *Proc. 16th ACM Symp. Operating Systems Principles (SOSP-16)*, vol. 31, pp. 78-91, Oct. 1997.

[11] R.M. Karp, M. Luby, and F. Meyer auf der Heide, "Efficient PRAM Simulation on a Distributed Memory Machine," *Proc. 24th ACM Symp. Theory of Computing*, pp. 318-326, 1992.

[12] L. Kleinrock, *Queueing System: Volume 1: Theory*. New York: John Wiley & Sons, 1975.

[13] J. Kuri and A. Kumar, "Optimal Control of Arrivals to Queues with Delayed Queue Length Information," *IEEE Trans. Automatic Control*, vol. 40, pp. 1,444-1,450, 1995.

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

[15] R. Mirchandaney, D. Towsley, and J.A. Stankovic, "Analysis of the Effects of Delays on Load Sharing," *IEEE Trans. Computers*, vol. 38, pp. 1,513-1,525, 1989.

[16] R. Mirchandaney, D. Towsley, and J.A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," *J. Parallel and Distributed Computing*, vol. 9, pp. 331-346, 1990.

[17] M. Mitzenmacher, "Constant Time per Edge Is Optimal on Rooted Tree Networks," *Proc. Eighth ACM Symp. Parallel Algorithms and Architectures*, pp. 162-169, 1996.

[18] M. Mitzenmacher, "Load Balancing and Density Dependent Jump Markov Processes," *Proc. 37th IEEE Symp. Foundations of Computer Science*, pp. 213-222, 1996.

[19] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," PhD thesis, Univ. of California, Berkeley, Sept. 1996.

[20] M. Mitzenmacher, "On the Analysis of Randomized Load Balancing Schemes," *Theory of Computing Systems*, vol. 32, pp. 361-386, 1999.

[21] M. Mitzenmacher, "Tight Thresholds for the Pure Literal Rule," Technical Report 1997-011, DEC/SRC, 1997.

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

[23] G.D. Stamoulis and J.N. Tsitsiklis, "The Efficiency of Greedy Routing in Hypercubes and Butterflies," *IEEE Tran. Comm.*, vol. 42, no. 11, pp. 3,051-3,061, 1994.

[24] D. Towsley and R. Mirchandaney, "The Effect of Communication Delays on the Performance of Load Balancing Policies in Distributed Systems," *Proc. Second Int'l MCPR Workshop*, pp. 213-226, 1988.

[25] R. Weber, "On the Optimal Assignment of Customers to Parallel Servers," *J. Applied Probability*, vol 15, pp. 406-413, 1978.

[26] W. Whitt, "Deciding Which Queue to Join: Some Counterexamples," *Operations Research*, vol 34, pp. 55-62, 1986.

[27] W. Winston, "Optimality of the Shortest Line Discipline," *J. Applied Probability*, vol 14, pp. 181-189, 1977.

[28] N.C. Wormald, "Differential Equations for Random Processes and Random Graphs," *Annals Applied Probability*, vol 5, pp. 1,217-1,235, 1995.

[29] 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, pp. 15-27, 1996.

Michael Mitzenmacher received his BA from Harvard University in 1991, studied mathematics in England on a Churchill Fellowship, and then completed his PhD in computer science at the University of California Berkeley in 1996. He is an assistant professor in computer science at Harvard University. He worked for Digital Systems Research Center in Palo Alto, California, before moving to Harvard in 1999. He currently works on error-correcting codes, stochastic bin-packing, and algorithms for the World Wide Web. His general interests include dynamic processes and randomized algorithms.