Theory Comput. Systems **34**, 77–98 (2001)
DOI: 10.1007/s002240010010

Theory of Computing Systems

Analyses of Load Stealing Models Based on Families of Differential Equations*

M. Mitzenmacher

Computer Science Department, Harvard University,
33 Oxford Street, Cambridge, MA 02138, USA
michaelm@eecs.harvard.edu

Abstract. In this paper we analyze the limiting behavior of several randomized work stealing algorithms in a dynamic setting. Our models represent the limiting behavior of systems as the number of processors grows to infinity using differential equations. The advantages of this approach include the ability to model a large variety of systems and to provide accurate numerical approximations of system behavior even when the number of processors is relatively small. We show how this approach can yield significant intuition about the behavior of work stealing algorithms in realistic settings.

1. Introduction

Work stealing is a natural paradigm for distributing workload in a parallel system in which underutilized processors seek out work from other processors. In contrast, in the *work sharing* paradigm overloaded processors attempt to pass on some of their work elsewhere in the system. In many cases work stealing can be a more effective means of balancing load than work sharing, especially in terms of communication efficiency: when all processors are busy, no attempts are made to migrate work across processors. Work stealing has therefore been a popular strategy for multithreaded computation. Several systems using the work stealing idea have been implemented (see p. 6 of [4]), including the Cilk system [6], [8], [9].

In this paper we analyze several simple randomized work stealing algorithms in a dynamic setting using simple Markovian models and the *fluid limit* approach that has

* Parts of this work were done while the author was at U.C. Berkeley and Compaq Systems Research Center.

similarly been used to study work sharing algorithms [32], [33], [35], [42], [43]. Primarily we study variations of the WS (Work Stealing) algorithm described by Blumofe and Leiserson [8], although our framework lacks their complexity as their model is for thread-based computations. At the thread level there are dependencies related to the order of completion that we do not consider. We instead focus on the situation where independent tasks enter the system over time according to a Poisson arrival process and require exponentially distributed service times. This model proves simplest for our analysis; however, as we explain, we can also use this technique to analyze other arrival and service distributions.

Our models capture the limiting behavior of work stealing systems as the number of processors grows to infinity by representing their behavior by differential equations. (This is often referred to as the fluid limit approach.) The advantages of this approach include the ability to model a large variety of systems and to provide accurate numerical approximations of system behavior even when the number of processors is relatively small.

The goals of this paper are to demonstrate the effectiveness of this modeling technique for work stealing algorithms and to develop insight into work stealing algorithms based upon these models. We therefore show how a number of variations of work stealing algorithms and different system parameters can be analyzed and compare the results of these models with simulation results for systems with a small number of processors.

Within this framework, we seek general rules of thumb that appear to apply to the work stealing paradigm. For example, we find that while stealing improves performance measures such as the expected time in the system, if we examine the fraction of processors with load at least j over all j, these numbers still have essentially the same behavior: they decrease geometrically (under Poisson arrivals and exponentially distributed service distributions). This contrasts, for example, with work sharing algorithms that make use of small amounts of random choice. Such models have been shown to have a completely different behavior, in that the tails of the loads fall doubly exponentially [33], [42]. We also find in our models that the time to transfer a task can have a significant impact on the effectiveness of load stealing; even small transfer delays can dramatically affect the system performance.

1.1. *Previous Work*

Work stealing has been treated extensively in a series of papers by Blumofe, Leiserson, and others [5]–[8], who apply work stealing in the Cilk system for parallel processing. Their models, which include not only computation time but also memory usage and communication costs, demonstrate work stealing algorithms that are optimal up to a constant factor in terms of execution time and within a constant factor of known lower bounds in terms of space and communication. Experiments on the Cilk system further show that their algorithms work well in practice. While our models do not capture this level of detail, we believe they provide insight into the behavior of work stealing. Other theoretical models for stealing algorithms have also been developed and analyzed by Rudolph et al. [39] and Karp and Zhang [18].

Work stealing has also been the subject of attention in the queueing theory literature, most notably in the early work by Eager et al. [11] and the later work by Mirchandaney et al. [28], [29]. Our work is similar to theirs, in that the underlying models and analysis are based on simple Markov chains, although both our mathematical approach and our focus are different.

The approach of using differential equations to study limiting versions of load balancing processes has been applied previously in several cases [2], [14], [26], [33], [42], [43]. Technically, the relationship between the limiting system consisting of a family of differential equations and systems with a finite number of processors can be derived using the theory of weak convergence; see, for instance, the body of work of Kurtz [12], [21]–[24], or a more recent treatment by Shwartz and Weiss [40]. The use of this approach in the study of algorithms dates back to work by Karp and Sipser [16], and has since been used to analyze several other algorithms, for example in [1], [14], [17], [25], [36], [37], and [44]. Note that here we focus on how to use the technique and what insight it gives us, in conjunction with simulations, about work stealing algorithms, rather than on the technical relationship between the limiting and finite systems.

The rest of the paper is organized as follows. In Section 2 we describe the basic model used in our analysis. We then derive a vector differential equation describing the asymptotic performance of a basic WS algorithm in this setting. We also demonstrate how to modify our analysis for simple variations of the work stealing algorithm. In Section 3 we consider how to extend our analysis to more complex and realistic models, including for example models where the service time is a fixed constant (instead of exponentially distributed) and where there is a transfer time associated with moving a task from one processor to another. Convergence issues are discussed in Section 4.

2. Simple Work Stealing Systems

In this section we consider variations of the WS algorithm described by Blumofe and Leiserson [8] in a dynamic setting. These variations share an interesting property: asymptotically, as the number of processors n goes to infinity, the fraction of processors with load at least i decreases geometrically for sufficiently large i.

2.1. *A Dynamic Model*

We describe our initial model of a *work stealing system*. The system has n processors that execute tasks dynamically generated at each processor as a Poisson process of rate $\lambda < 1$. Tasks complete after being serviced for an amount of time that is exponentially distributed with mean 1. All arrival and service times are independent of one another, and the random service times required by the tasks are not known to the processors. Tasks are served according to the First In First Out (FIFO) policy. The *load* of a processor is the number of tasks at that processor.

At certain times, a processor may attempt to steal a task from another processor. Following the terminology of [8], we call a processor attempting to steal a *thief*, and say that it attempts to steal from a *victim* processor. We assume that stealing is accomplished instantaneously, so that the stolen task joins the queue of the thief immediately. Tasks will be stolen from the end of the victim's queue.

We now provide a representation of the system useful for our analysis. We define $n_i(t)$ to be the number of processors with exactly i tasks at time t; $m_i(t)$ to be the number of processors with at least i tasks at time t; $p_i(t) = n_i(t)/n$ to be the fraction of processors of load i; and $s_i(t) = \sum_{k=i}^{\infty} p_i(t) = m_i(t)/n$ to be the tails of the $p_i(t)$. We drop the reference to t in the notation where the meaning is clear. For many systems, the s_i prove more convenient to work with than the p_i. Note that $s_0 = 1$ always, and that the s_i are nonincreasing as $s_{i-1} - s_i = p_i$. For the systems we consider, we also have that $\sum_i s_i < \infty$; this corresponds to the fact that the expected load is finite.

The state of the system at any given time can be represented by an infinite dimensional vector $\vec{s} = (s_0, s_1, s_2, \ldots)$. Note that our state only includes information regarding the number of processors of each load. Since steals are instantaneous, we are not concerned with distance between processors. The processors appear indistinguishable except for their load, and hence the vectors of the tails of the load is all the information we require. Also, under the assumption that service times are exponential, arrivals are Poisson, and all arrival and service times are independent, the entire system is *Markovian*: the future of the system depends only on its present state, and not on the past that brought it to that state.

2.2. *A Simple WS Algorithm*

We initially study a variation of the WS algorithm described by Blumofe and Leiserson [8]. When a processor finds itself empty, it attempts to steal a task from a processor chosen uniformly at random. If a task is available—that is, the victim processor has more than one task—a task is stolen. For any n, it is easy to show that a system using this algorithm in our model is *stable*, in that its expected queue length is bounded as the time $t \to \infty$, since every queue behaves like an M/M/1 queue except when empty or when a task is stolen. Stability can be proven for instance by a coupling or stochastic domination argument.

We examine the limiting system corresponding to the behavior as the number of processors grows to infinity. In the interest of readability, we provide a more intuitive explication, and first consider a system without load stealing. Let δm_i represent the expected change in m_i over a small interval of time δt. We first consider arrivals; an arrival increases m_i if it occurs at a processor with load $i - 1$. Since we have a Poisson arrival process of rate λ at each processor, the probability of an arrival at a processor with i tasks is $\lambda\,\delta t + o(\delta t)$. Note that the probability of two or more arrivals at a queue is $o(\delta t)$. Hence the expected change in m_i due to arrivals is just $\lambda(m_{i-1} - m_i)\,\delta t + o(n\,\delta t)$. Similarly, the expected change in m_i due to departures is $(m_i - m_{i+1})\,\delta t + o(n\,\delta t)$. Hence, the expected behavior of the system over short intervals is given by

$$\delta m_i = \lambda(m_{i-1} - m_i)\,\delta t - (m_i - m_{i+1})\,\delta t + o(n\,\delta t).$$

We cancel the factor of n permeating the equations and let δt go to zero, in which case $\delta m_i/\delta t \to dm_i/dt$. This yields the equation

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}). \tag{1}$$

The equations (1) are easily derived more informally by assuming that over a short period dt that two events do not occur at a processor. Then by considering the rates at which m_i increases and decreases directly we have

$$dm_i = \lambda(m_{i-1} - m_i)\,dt - (m_i - m_{i+1})\,dt,$$

from which (1) follows.

Considerations similar in spirit to the law of large numbers suggest that as n goes to infinity, the behavior of the s_i become deterministic and are governed by the system of differential equations (1). Note that these equations depend only the *densities* of processors with a certain load.

Somewhat more formally, for each value of n, the behavior of the system is a Markov process, determined by the countable state space $\vec{s} = (s_0, s_1, s_2, \ldots)$. Each such Markov process has a corresponding generating operator, which roughly corresponds to the transition matrix on this state space. The sequence of generating operators may converge to a limit, which (under suitable technical conditions) corresponds to the limit of the appropriate sequence of Markov processes. Here, we are concerned with the limiting behavior as the number of processors n grows to infinity. The system of differential equations (1) is a convenient representation of the limit generating operator.

There exists a general theory regarding the convergence of the generating operator and the sequence of Markov processes in the setting we are concerned with here. When a family of Markov processes has transition rates independent of n, the system size, and dependent only on the densities, it is called a *density dependent jump Markov process*. Kurtz's work demonstrates that, as $n \to \infty$, the sequence of Markov process converges to the deterministic process given by the corresponding differential equations (subject to certain conditions). Given our intuitive derivation of the differential equations, this correspondence is hardly surprising; indeed, it is a functional law of large numbers for density dependent Markov processes. As the focus of this paper is not the mathematical theory behind this convergence, but rather how this methodology can be useful in studying the behavior of load stealing algorithms, we do not focus further on the technical details. The reader interested in the details of the theory behind this convergence is referred to other sources, including especially the work of Kurtz [12], [21]–[24] and more recent related treatments [42]–[44].

We now consider how to modify the above equations in the case of load stealing. Processors that complete their final task attempt to find a victim, thereby reducing the rate at which they actually empty. The probability of success is just s_2, the probability of choosing a victim processor that contains at least two tasks. Hence to the processors it appears as though they lose their final task at the rate $1 - s_2$, instead of at the rate 1. The corresponding modified equation is given by

$$\frac{ds_1}{dt} = \lambda(s_0 - s_1) - (s_1 - s_2)(1 - s_2). \tag{2}$$

For $i > 1$, s_i decreases whenever a processor with load i completes a task, or when a task is stolen. The rate at which thieves steal tasks is just $(s_1 - s_2)$, the rate at which processors complete their final task, yielding

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) - (s_i - s_{i+1})(s_1 - s_2), \qquad i \geq 2. \tag{3}$$

Of course for finite n the system again has the property that the transitions depend only on the density of processors with each load, so we have an appropriate set of differential equations that describe the limiting system. To understand the long term behavior of the system, we find a *fixed point* for the system of equations given by (2) and (3). A fixed point is a state at which $ds_i/dt = 0$ for all i; if the system reaches its fixed point, it remains there. Many processes are well behaved, in that they follow trajectories that converge to their fixed points. We emphasize, however, that not all systems have fixed points, not all systems that have fixed points converge to their fixed points, and systems can have multiple fixed points. Indeed, even systems that one might intuitively expect to be well behaved might have multiple fixed points, and correspondingly surprising behavior; see, for example, [13]. In the cases we study here, we can often prove the system has a unique fixed point and test for convergence experimentally, as we will show.

To determine a fixed point, we note the following facts:

- $s_0 = 1$ for all time.
- The rate at which tasks complete is $s_1 n$, the number of busy processors.
- The rate at which tasks are introduced is λn.
- At a fixed point, the rate at which tasks complete and the rate at which they are introduced must be equal.

We denote a fixed point by the vector $(\pi_0, \pi_1, \ldots)$. Then the above facts tell us that for any fixed point $\pi_0 = 1$ and $\pi_1 = \lambda$. From (2), and using the fact that $ds_1/dt = 0$ at the fixed point, we can solve for π_2; specifically, π_2 satisfies

$$\lambda(1-\lambda) - (\lambda - \pi_2)(1-\pi_2) = 0.$$

This is a quadratic in π_2 with two real roots, however, only one of these roots satisfies the constraint that $\pi_2 \le 1$, namely

$$\pi_2 = \frac{1+\lambda-\sqrt{1+2\lambda-3\lambda^2}}{2}.$$

Using induction, (3), and the fact that $ds_i/dt = 0$ at the fixed point, we find that for $i > 2$,

$$\pi_i = \pi_2 \left(\frac{\lambda}{1+\lambda-\pi_2}\right)^{i-2}.$$

Note that in this case the fixed point is clearly unique.

For $i \ge 2$, the π_i decrease geometrically. We contrast this result with the case of no stealing, where the fixed point is $\pi_i = \lambda^i$, as can be verified by (1). In both cases, the fraction of processors with load at least i decreases geometrically, but with load stealing the tails decrease faster. It is as though the service rate has increased due to stealing.

There is a useful interpretation for this phenomenon. Standard queueing theory yields that in a system with no stealing, arrival rate λ, and service rate μ, the tails of the loads decrease geometrically with ratio λ/μ between successive terms. (Recall that we have scaled so that $\mu = 1$ for (1), and thus the tails of the loads decrease geometrically as $\pi_i = \lambda^i$.) From the point of view of a processor with at least two tasks in a work stealing system, the *apparent* service rate μ' is the processor's own service rate 1 plus the rate at which a task is stolen from the processor, which is $\pi_1 - \pi_2 = \lambda - \pi_2$. Hence we expect the tails at the fixed point to decrease geometrically at rate $\lambda/\mu' = \lambda/(1+\lambda-\pi_2)$, and this intuition is verified by the derivation of the fixed point.

We have not shown that the trajectory of the system of differential equations always converges to its fixed point, and indeed we do not have a proof of convergence. (We prove

Table 1. Simulations versus estimates for the average time in the system in the simplest WS model. The relative error is between the simulations with 128 processors and the estimate based on the fixed point calculation.

λ	Sim(16)	Sim(32)	Sim(64)	Sim(128)	Estimate	Rel. Error (%)
0.50	1.631	1.626	1.622	1.620	1.618	0.15
0.70	2.153	2.133	2.119	2.114	2.107	0.30
0.80	2.678	2.617	2.586	2.576	2.562	0.56
0.90	3.905	3.711	3.624	3.586	3.541	1.24
0.95	5.936	5.368	5.138	5.000	4.887	2.25
0.99	17.863	14.368	12.183	11.306	10.462	7.46

something weaker in Section 4, where we discuss convergence issues further.) However, empirical testing by evaluating the differential equations numerically at different starting points suggests that the system is indeed well behaved in this regard. Even though the differential equations describe the limiting behavior as n goes to infinity, since the equations describe the expected behavior of the system for finite n, we would hope that the fixed point would provide accurate estimates for the behavior of finite systems.

We briefly demonstrate the accuracy of this approach comparing the predicted results for the expected time each task spends in the system based on the fixed point with simulations for this simple work stealing model in Table 1. All simulation results are based on the average of 10 simulations of 100,000 seconds each, with the first 10,000 seconds thrown out to mitigate the impact of starting with an empty system. The table demonstrates several important features:

- The prediction improves with the number of processors.
- The prediction improves as the arrival rate decreases.
- Even at only 128 processors, the predictions are extremely accurate, particularly at smaller arrival rates.

We now demonstrate the ease with which one can construct systems of differential equations describing variations of the basic model we have considered above.

2.3. *Threshold Stealing*

It is perhaps more realistic to suppose that thieves will steal only from processors whose load is at least some threshold T, in order to improve the chances that the cost of transferring the job is worthwhile. In this case, the probability that a steal fails to occur when a processor finishes all pending tasks is $1 - s_T$, and hence the limiting system behavior is described by the following set of differential equations:

$$\frac{ds_1}{dt} = \lambda(s_0 - s_1) - (s_1 - s_2)(1 - s_T), \tag{4}$$

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), \qquad 2 \le i \le T - 1, \tag{5}$$

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) - (s_i - s_{i+1})(s_1 - s_2), \qquad i \ge T. \tag{6}$$

Again, we seek a fixed point, beginning with $\pi_0 = 1$ and $\pi_1 = \lambda$. From (4) we obtain

$$\pi_2 = \frac{\lambda^2 - \lambda\pi_T}{1 - \pi_T}.$$

To find the value of π_T, we use a recurrence obtained from (5):

$$\pi_{i+1} = \pi_i - \lambda(\pi_{i-1} - \pi_i).$$

A simple induction establishes that

$$\pi_i = \frac{\lambda^i - \lambda\pi_T}{1 - \pi_T}.$$

From

$$\pi_T = \frac{\lambda^T - \lambda\pi_T}{1 - \pi_T},$$

we have a quadratic equation

$$\pi_T^2 - (1 + \lambda)\pi_T + \lambda^T = 0$$

that we may solve for π_T:

$$\pi_T = \frac{1 + \lambda \pm \sqrt{(1 + \lambda)^2 - 4\lambda^T}}{2}.$$

Note that only one of these roots satisfies the constraint that $\pi_T \leq 1$, namely

$$\pi_T = \frac{1 + \lambda - \sqrt{(1 + \lambda)^2 - 4\lambda^T}}{2}.$$

Alternatively, one could see that there is only one root π_T between 0 and 1 by noting the quadratic equation above takes on a positive value when we plug in $\pi_T = 0$ and a negative value when we plug in $\pi_T = 1$.

By (6) and the fact that the ds_i/dt are zero at the fixed point, we have for $i \geq T$ that

$$\pi_{i+1} = \pi_i - \frac{\lambda(\pi_{i-1} - \pi_i)}{1 + \pi_1 - \pi_2}. \tag{7}$$

We will show that $\pi_T = \lambda\pi_{T-1}/(1 + \pi_1 - \pi_2)$; then a simple induction using (7) yields

$$\pi_i = \pi_T \left(\frac{\lambda}{1 + \lambda - \pi_2} \right)^{i-T}.$$

Hence for $i > T$ the π_i again decrease geometrically at a faster rate than in a system without load stealing. This equation also matches the intuition developed in Section 2.2; for queues with load at least T, the apparent service rate μ' is again the service rate 1 plus the rate at a queue at which a task is stolen, which is $\pi_1 - \pi_2 = \lambda - \pi_2$.

To show $\pi_T = \lambda\pi_{T-1}/(1 + \pi_1 - \pi_2)$ we use the fact that $\sum_{i=1}^{T-1}(ds_i/dt) = 0$ at the fixed point. Most of the terms in this summation cancel (that is, we have telescoping sums), yielding

$$\lambda(\pi_0 - \pi_{T-1}) - (\pi_1 - \pi_T) + \pi_T(\pi_1 - \pi_2) = 0.$$

Using $\pi_0 = 1$ and $\pi_1 = \lambda$, the relation for π_T easily follows.

2.4. *Repeated Steal Attempts*

In the version of the WS algorithm as described in [8], if the thief fails to find a suitable victim on the first attempt, further attempts are made to find one. We can model this behavior by allowing empty processors to repeatedly make steal attempts at a certain rate, say r per unit time. To fit with our standard model, we assume that the time between steal attempts is exponentially distributed. This modification simply adds $r(s_0 - s_1)$ to the overall rate of steals. Hence, if there is a threshold T so that a victim must have at least T tasks, the equations describing the limiting system become

$$\begin{aligned}
\frac{ds_1}{dt} &= \lambda(s_0 - s_1) + r(s_0 - s_1)s_T - (s_1 - s_2)(1 - s_T),\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), \qquad 2 \le i \le T - 1,\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) - (s_1 - s_2)(s_i - s_{i+1})\\
&\quad - r(s_0 - s_1)(s_i - s_{i+1}), \qquad i \ge T.
\end{aligned}$$

This system has a unique fixed point, as can be shown by following the same approach as for the case of threshold stealing in Section 2.3. One must derive a recurrence that yields a quadratic equation for π_T, and checking this equation at the points 0 and 1 yields that there in only one possible value of π_T in the range [0, 1].

In this system at the fixed point $\vec{\pi}$ the π_i decrease geometrically for $i > T$, according to the formula

$$\pi_i = \pi_T \left(\frac{\lambda}{1 + r(1-\lambda) + \lambda - \pi_2}\right)^{i-T}.$$

This formula can be derived using the intuition of Section 2.2, or by an inductive argument from the equivalent of (7) for this model.

Note that, in the limit as r goes to infinity, π_T goes to 0. This stands to reason, since in the limiting system a processor with T tasks will have a task stolen immediately.

2.5. *Preemptive Stealing*

Instead of waiting until the task queue is empty, a thief processor may wish to begin attempting to steal work when the number of tasks it has left is sufficiently small. In such a system, a processor makes a steal attempt whenever it completes a task and is left with at most B tasks. It seems reasonable not to steal from a processor that would be left with only B or fewer tasks in such a model. Hence if there is a threshold $T \ge B + 2$ so that a victim must have at least T tasks, the equations describing the limiting system become

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1})(1 - s_T), \qquad 1 \le i \le B + 1, \tag{8}$$

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), \qquad B + 2 \le i \le T - 1, \tag{9}$$

$$\frac{ds_i}{dt} = \lambda(s_{i-1} - s_i) - (s_i - s_{i+1})(1 - s_{B+2}), \qquad i \ge T. \tag{10}$$

Note that the second set of equations above may not hold for any value of i, if $T = B + 2$.

Demonstrating that the fixed point is unique for this system is more complex. We note that we may derive the following recurrences for a fixed point from (8) and (9) above:

$$\begin{aligned} \pi_{i+1} &= \pi_i - \frac{\lambda(\pi_{i-1} - \pi_i)}{1 - \pi_T}, \qquad & 1 \le i \le B+1, \\ \pi_{i+1} &= \pi_i - \lambda(\pi_{i-1} - \pi_i), \qquad & B+2 \le i \le T-1. \end{aligned}$$

We begin with $\pi_0 = 1$ and $\pi_1 = \lambda$. Using the proper recurrence, we obtain

$$\pi_2 = \frac{\lambda^2 - \lambda\pi_T}{1 - \pi_T}.$$

Note that we have expressed π_2 as a function of π_T that is nonincreasing on the interval $[0, 1]$. Similarly, using the recurrences above we obtain equations of the form $\pi_3 = f_3(\pi_T)$, $\pi_4 = f_4(\pi_T)$, and so on. We may inductively show that on the interval $[0, 1]$

$$\frac{df_i}{d\pi_T} \ge \frac{df_{i+1}}{d\pi_T},$$

and hence the right-hand sides of all such derived equations are nonincreasing in π_T. However, applying these recurrences eventually yields $\pi_T = f_T(\pi_T)$ for some nonincreasing function π_T. As the left-hand side of the equation is increasing on $[0, 1]$ and the right-hand side is nonincreasing, there can be only one root in this interval, and hence only one valid fixed point.

At the fixed point, for $i > T$, the tails decrease geometrically according to

$$\pi_i = \pi_T \left(\frac{\lambda}{1 + \lambda - \pi_{B+2}} \right)^{i-(B+T)}.$$

Again, this formula can be derived using the intuition of Section 2.2, or by an inductive argument.

3. More Complex Variations

In this section we examine more complex extensions to the basic models we have described. In particular, many of our extensions are motivated by the goal of making the models more realistic. For convenience, we consider each modification separately, although it should be clear from the presentation that the extensions can be combined as desired, albeit by making the corresponding systems of differential equations more complicated and hence more difficult to solve.

In this section, unless otherwise noted, we have not attempted to prove the uniqueness of the fixed point, because the complexity of the variety of the system make such proofs algebraically unpleasant. Our experience suggests that the fixed points for these systems are unique, and we have proceeded under that assumption. Note that a practical approach for determining a fixed point numerically is simply to simulate the behavior of the family of differential equations, starting from an empty system where $s_i = 0$ for $i \geq 1$. For the systems we study, we have found that convergence to a fixed point occurs rapidly enough that this technique suffices. Also, since the family is infinite, we track values s_i only for $i \leq \ell$, where ℓ is a large enough value that the fraction of bins with load at least ℓ appears small enough to discount.

3.1. *Varying Service and Arrival Distributions*

In the framework we have established, we require exponential service times and Poisson arrivals, in order for the state to be properly represented by the simple state $\vec{s} = (s_0, s_1, s_2, \ldots)$. If instead, for example, the service time was not exponential, the state would need to encode the remaining service time at each server in some fashion. The memorylessness of the exponential distribution has therefore been imperative.

The need for exponential service times and Poisson arrivals appears to limit the usefulness of this approach. However, one can approximate other service times and arrival distributions using mixtures of these simple distributions. The approach, generally known as *Erlang's method of stages*, is explained more fully in Sections 4.2 and 4.3 of [20]. (For examples of this approach in similar load sharing models, see [33], [35], and [43].)

We demonstrate the method here by considering the case of constant service times. We replace the constant service time with a sequence of c stages of services; each stage of service is independent and exponentially distributed with mean $1/c$. A random variable that is a sum of independent exponential random variables of the same mean has a *gamma distribution*. As c goes to infinity, the expected time spent in these c stages of service remains 1 and the variance approaches 0; that is, the service time converges to a random variable that takes on the constant value 1 with probability 1. In practice, computing the fixed point requires limiting c to a reasonably small finite number, since the number of terms in the fixed point grows proportionally with c. Our simulations suggest that even for reasonably small c, however, the predictions become very accurate.

The state will again be represented by a vector $\vec{s} = (s_0, s_1, s_2, \ldots)$, but here s_i represents the fraction of processors *with at least i stages left to complete*. Note that, when a steal occurs, the values from s_1 up to s_c all change; similarly, if a steal occurs at a processor with i stages left to complete, then the values from s_{i-c+1} to s_i all change. Further, stages complete at a rate of c per unit time. Hence for the case where $T = 2$, that is, if we steal whenever possible, the resulting equations are

$$\frac{ds_1}{dt} = \lambda(s_0 - s_1) - c(s_1 - s_2)(1 - s_{c+1}),$$
$$\frac{ds_i}{dt} = \lambda(s_0 - s_i) + c(s_1 - s_2)s_{i+c} - c(s_i - s_{i+1}), \qquad 2 \leq i \leq c,$$
$$\frac{ds_i}{dt} = \lambda(s_{i-c} - s_i) - c(s_i - s_{i+1}) - c(s_i - s_{i+c})(s_1 - s_2), \qquad i \geq c + 1.$$

We do not currently know of any way to calculate the fixed point directly for this model. Suppose that we wish to simulate the behavior of the system of differential equations, in order to calculate the fixed point. In this case, the number of differential equations required to calculate loads up to some fixed ℓ is actually $\ell \cdot c$. The state

Table 2. Simulations versus estimates for the average time in the system in the constant time model ($T = 2$): simulations for 16, 32, 64, and 128 processors are compared with the estimates using 10 and 20 stage approximations of constant time.

λ	Sim(16)	Sim(32)	Sim(64)	Sim(128)	$c = 10$	$c = 20$
0.50	1.382	1.380	1.378	1.378	1.405	1.391
0.70	1.724	1.713	1.709	1.706	1.749	1.727
0.80	2.050	2.030	2.017	2.013	2.070	2.039
0.90	2.811	2.729	2.696	2.677	2.759	2.709
0.95	3.978	3.774	3.655	3.594	3.701	3.625
0.99	11.010	8.992	7.934	7.542	7.581	7.399

space is essentially c times larger. For straightforward implementations of simulating the differential equations, the complexity in terms of the number of operations required is proportional to the number of equations. Hence smaller values of c are easier to work with, although they give less accurate predictions.

In principle, this approach could be used to develop deterministic differential equations that approximate the limiting behavior of any service time distribution or arrival distribution to any desired accuracy. This is because the distribution function of any positive random variable can be approximated arbitrarily closely by a mixture of countably many gamma distributions. That is, we may approximate a random variable by choosing randomly with the appropriate distribution from a collection of gamma distributed random variables (see, for example, Lemma 3.9 of [19]). We can thus develop a suitable state space for a Markov process that approximates the underlying non-Markovian process. There is a tradeoff, however, in that the better the approximation we obtain, the larger the state space, and hence the more calculation required to evaluate the fixed point numerically.

The simulations presented in Table 2 demonstrate that for constant service times, taking $c = 20$ provides good approximations for actual systems. These results also show that systems with constant service times perform significantly better than systems with exponentially distributed service times, in terms of the average time spent in the system. This result is not surprising, given the fact that reducing variance in the service time improves the expected time in the system for single queues. We do not have a proof that constant service times yield smaller expected waiting times than exponential service times in this model; it would be interesting to prove such a result either using the fixed point (see Section 4.3 of [33]) or other techniques (see, for example, [15], [30], [31], [38], and [41]).

3.2. *Transfer Delay*

Up to this point we have assumed that a job can be transferred instantaneously to another processor. More realistically, moving a task from the victim to the thief will require some time for transfer. This clearly can have an effect on the efficacy of stealing; indeed, similar analysis of load stealing shown the importance of delay [10], [34], [28]. For convenience, here we model the transfer delay as an exponentially distributed variable with mean $1/r$ (that is, transfers occur at rate r), although it can also be modeled as a fixed constant, or some other distribution, using the technique of Section 3.1.

In this model we allow a thief processor to only steal one task at a time; that is, as long as there is a task on the way from another processor, it will not attempt to steal again. We expand our state space to distinguish explicitly thief processors who are awaiting a stolen task from other processors. Our state space will hence consist of two infinite dimensional vectors: $(s_0, s_1, \ldots)$ and $(w_0, w_1, \ldots)$. Here s_i refers to the fraction of all processors not awaiting a stolen task with at least i tasks, and w_i refers to the fraction of all processors awaiting a stolen task with at least i tasks.

This change in state space calls for changes in the fixed point conditions. For example, we now have that $s_0 + w_0 = 1$ for all time. Also, $s_1 + w_1 = \lambda$ at the fixed point, as the rate at which tasks are served must equal the rate at which tasks enter the system. Finally, the expected number of tasks per queue in the system is

$$\sum_{i\ge 1} i(s_i - s_{i+1}) + \sum_{i\ge 1} i(w_i - w_{i+1}) + w_0 = \sum_{i\ge 1} s_i + \sum_{i\ge 0} w_i.$$

The w_0 term accounts for the extra tasks in transit between processors.

The equations below describe this process under the conditions that a steal is attempted only when a processor empties and a steal occurs only if the victim processor has at least T tasks. Note the relationship between the w_i and the s_i. For example, whenever a steal occurs, some processor contributing to the s vector changes to contributing to the w vector; similarly, when a stolen task arrives, some processor contributing to the w vector instead contributes to the s vector.

$$\begin{aligned}
\frac{ds_0}{dt} &= rw_0 - (s_1 - s_2)(s_T + w_T),\\
\frac{ds_1}{dt} &= \lambda(s_0 - s_1) + rw_0 - (s_1 - s_2),\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) + rw_{i-1} - (s_i - s_{i+1}), \qquad 2 \le i \le T-1,\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) + rw_{i-1} - (s_i - s_{i+1})\\
&\quad - (s_i - s_{i+1})(s_1 - s_2), \qquad i \ge T,\\
\frac{dw_0}{dt} &= -rw_0 + (s_1 - s_2)(s_T + w_T),\\
\frac{dw_i}{dt} &= \lambda(w_{i-1} - w_i) - rw_i - (w_i - w_{i+1}), \qquad 1 \le i \le T-1,\\
\frac{dw_i}{dt} &= \lambda(w_{i-1} - w_i) - rw_i - (w_i - w_{i+1})\\
&\quad - (w_i - w_{i+1})(s_1 - s_2), \qquad i \ge T.
\end{aligned}$$

Note that we allow tasks to be stolen from a processor that is waiting for a task. We might expect that a thief should not attempt to steal a task unless in so doing it reduces the expected time that task will remain in the system. If the stolen task is put at the end of the queue it arrives at, such a rule would suggest that the best threshold T must satisfy $T \approx (1/r) + (\lambda/r)$; the term λ/r accounts for arrivals during transfer. To minimize the expected time for all tasks, however, this simple rule is only a rough approximation. As seen in the example for $r = 0.25$ presented in Table 3, the fixed point solutions to

Table 3. The expected time in the system with transfer delays, where $r = 0.25$, according to simulations and estimates from the fixed point of the differential equations. The best threshold is $T = 4 = 1/r$ for small arrival rates, but is larger at higher arrival rates.

	$T=3$		$T=4$		$T=5$		$T=6$	
λ	Sim(128)	Est.	Sim(128)	Est.	Sim(128)	Est.	Sim(128)	Est.
0.50	1.986	1.985	1.950	1.950	1.955	1.954	1.967	1.967
0.70	2.973	2.971	2.939	2.938	2.963	2.961	3.011	3.008
0.80	4.038	4.030	4.003	3.996	4.025	4.020	4.082	4.079
0.90	7.099	7.076	7.056	7.015	7.025	7.001	7.045	7.026
0.95	13.162	13.106	13.089	13.016	13.048	12.956	13.067	12.925

the differential equations can be used to determine correctly the best threshold value for various arrival rates.

Given the numerical accuracy of this approach, we examine more closely the effect of transfer delay on the expected time a task spends in the system. We focus on the specific case of $\lambda = 0.9$ and $\lambda = 0.95$. Although these are reasonably high loads, in systems with low load the overall effect of stealing is much less substantial, and hence the case of high load is more interesting. Figure 1 shows the effect of various transfer delays as calculated by numerically determining the fixed point from the differential equations with delays $1/r = 0.1, 0.5, 1, 2, 4, 5, 10$, and 20. In each case, the result from the best choice of the threshold T is used, although, as we have seen in Table 3, there is little deviation between the best and near-best thresholds. While delay naturally increases the expected time spent in the system, it is surprising how large an effect even a small delay can have on this measure of performance. (This behavior is easily verified by simulations.)

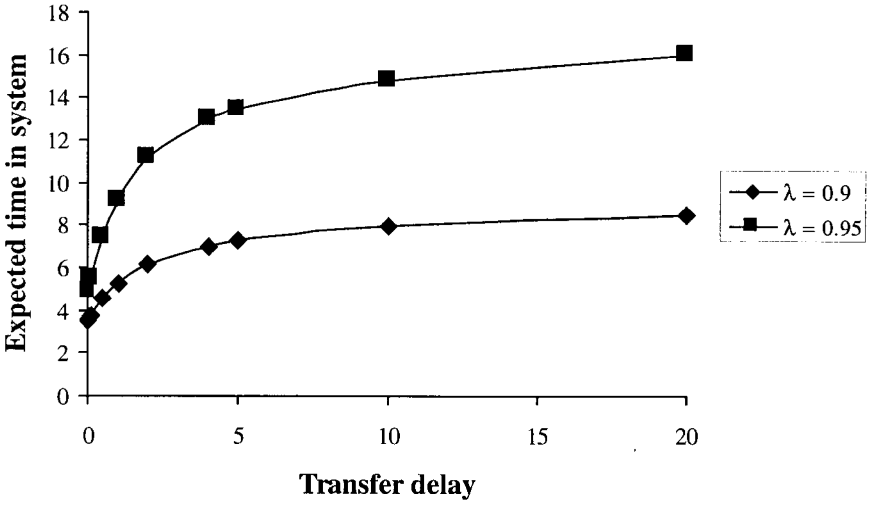

Fig. 1. The effects of transfer delay.

A possible explanation is, under these high arrival rates, an empty processor is unlikely to stay empty for long. Hence, in many cases, a system where there is stealing and delay must make many mistakes, stealing a task when it would have been better to leave it. Interestingly, this effect is significant even under apparently small delays.

3.3. *Multiple Choices*

In load sharing algorithms, systems that have some choice of where to place new jobs have proven to have different performance characteristics than systems where jobs are placed randomly [3], [33], [42]. For example, suppose that, upon entry, a task chooses two servers uniformly at random, and queues at the one with the smaller load. There is an exponential improvement in average time spent in the system as a function of the arrival rate over a system where each task queues at a random server. Specifically, with n queues, a Poisson arrival process of λn customers per unit time, and exponentially distributed service times of mean 1, the average time spent in the system when customers are assigned randomly is $1/(1-\lambda)$. In the limit as $\lambda \to 1$, the expected time when a task chooses the best of two servers is $\log_2(1/(1-\lambda))$.

This motivates examining the following work stealing strategy: a thief chooses d random potential victims simultaneously, and then (if possible) steals load from the most heavily loaded victim. If the victim must have load at least T, the probability that a steal fails to occur equals the probability that all d victims have load less than T; this happens with probability $(1-s_T)^d$. Similarly, the probability that a victim processor with load i is found is $(1-s_{i+1})^d - (1-s_i)^d$. Hence, if we constrain d to be a fixed constant, independent of the number of processors n, then we can write a corresponding limiting system with the following form:

$$\begin{aligned}
\frac{ds_1}{dt} &= \lambda(s_0 - s_1) - (s_1 - s_2)(1-s_T)^d, \\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), \qquad 2 \le i \le T-1, \\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) \\
&\quad - ((1-s_{i+1})^d - (1-s_i)^d)(s_1 - s_2), \qquad i \ge T.
\end{aligned}$$

We note that for this system, it is easy to show using previously discussed techniques that the fixed point is in fact unique.

Table 4 compares a system where two potential victims are chosen to a system where just one choice is made. Choosing more victims does improve performance, especially at higher arrival rates, but just choosing a single victim generally yields most of the gain possible. The intuition of Section 2 that suggests why tails fall geometrically offers helpful insight: using d choices makes steals occur at most d times the usual rate for even the most heavily loaded queues, and hence the best we could hope is that the tails fall geometrically at rate $\lambda/(1 + d(\lambda - \pi_2))$. Since systems where multiple choices are made would require additional complexity, it is by no means clear that the gain would be worthwhile in a real system.

Table 4 also shows again that the estimate derived from the fixed point yields accurate

Table 4. Simulation results comparing the average time in the system with one choice and two (with $T = 2$, 128 processors) and the corresponding estimate from the fixed point.

	Sim(128)		Estimate
λ	1 choice	2 choices	2 choices
0.50	1.620	1.436	1.433
0.70	2.114	1.680	1.673
0.80	2.576	1.879	1.864
0.90	3.586	2.260	2.220
0.95	5.000	2.742	2.640
0.99	11.306	4.597	4.011

predictions for actual systems of 128 processors at reasonable arrival rates. The error is less than 1% for $\lambda \leq 0.8$ and only about 5% at $\lambda = 0.95$.

3.4. *Multiple Steals*

In certain situations, stealing more than one task may be appropriate. For example, if the threshold T for stealing is high, then stealing more than one process should improve the expected time a task spends in the system. We consider the WS algorithm where when a steal is made $k \leq T/2$ tasks are taken. Note that when a steal occurs, not only s_1 increases, but $s_2, s_3, \ldots, s_k$ do as well. Similarly, when a steal occurs, many s_i values decrease. Taking this into consideration yields the following family of differential equations:

$$\begin{aligned}
\frac{ds_1}{dt} &= \lambda(s_0 - s_1) - (s_1 - s_2)(1 - s_T),\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) + (s_1 - s_2)s_T, \qquad 2 \leq i \leq k,\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), \qquad k + 1 \leq i \leq T - k,\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) - (s_1 - s_2)(s_T - s_{i+k}),\\
&\quad T - k + 1 \leq i \leq T,\\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) - (s_1 - s_2)(s_i - s_{i+k}),\\
&\quad T + 1 \leq i.
\end{aligned}$$

Other variations for stealing multiple jobs in the WS algorithm can be modeled similarly. As one might expect, in this model (where the time for a transfer is zero) increasing the number of jobs stolen so as to equalize the processor loads improves performance.

More complicated algorithms that steal multiple items at a time are also possible. For example, we consider a variation of a load balancing algorithm suggested by Rudolph et al. [39], in which a processor at certain randomly determined steps chooses another processor uniformly at random and the two machines balance the load between them. Here, we can model a rebalancing event at a processor as a process that occurs at an exponential rate $r(i)$, perhaps depending on the number of items i at the processor. When a rebalancing event occurs, the tasks are balanced between this processor and another processor chosen uniformly at random. At the end, the two processors will either have the same number of tasks or the number of tasks will differ by at most one. For convenience, we assume in the second case that the processor with the larger initial load will also have the larger final load. Surprisingly, this system can be represented in a quite straightforward manner. It suffices to note that a rebalancing will increase s_i whenever one processor has load k with $k < i$ and the other processor has load at least $2i - k$. Similarly, a re-balancing will decrease s_i whenever one processor has load j with $i \leq j \leq 2i - 2$ and the other processor has load at most $2i - j - 2$. Hence, for $i \geq 1$,

$$\begin{aligned}\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1})\\ &\quad - \sum_{j=i}^{2i-2} \sum_{k=0}^{2i-2-j} (r(j) + r(k))(s_k - s_{k+1})(s_j - s_{j+1})\\ &\quad + \sum_{k=0}^{i-1} \sum_{j=2i-k}^{\infty} (r(j) + r(k))(s_k - s_{k+1})(s_j - s_{j+1}).\end{aligned}$$

3.5. *Varying Processor Speeds, Varying Arrival Rates, and Static Systems*

Thus far the systems studied have been homogeneous, in that all processors run at the same rate and tasks arrive at the system at the same rate. We note that this is not necessary; we can model different processor types by keeping a separate state vector for each type of processor. For example, if there are two types of processors, fast and slow, then we can represent slow processors by a vector $\vec{s} = (s_0, s_1, s_2, \ldots)$ and fast processors by a vector $\vec{w} = (w_0, w_1, w_2, \ldots)$. Here s_i would be the fraction of all processors that are fast and have load at least i, and similarly for the w_i. In our limiting model, each processor type must correspond to a fixed fraction of the total number of processors.

We can also enhance the model by introducing the concept of internal and external arrival rates. That is, we can replace the arrival rate λ by $\lambda_{\text{ext}} + \lambda_{\text{int}}$, where λ_{ext} corresponds to the rate of new tasks arriving into the system and λ_{int} corresponds to the rate of new tasks being spawned by tasks already at the processor.

Note λ_{int} can be made to depend on the number of tasks at the processor if desired. In particular, by setting $\lambda_{\text{ext}} = 0$ and $\lambda_{\text{int}} = 0$ when there are no tasks in the queue, we can model a static system that starts in some initial state and runs until all queues are empty. For sufficiently large systems, this approach can give a good approximation for the amount of time until all jobs complete.

4. Convergence and Stability

Thus far, when considering families of differential equations, we have focused on finding a fixed point and when possible proving it is the only fixed point, with the intuition that the system converges to that fixed point. Our experience with simulations indeed suggests that this is the case. To justify this intuition formally one would hope to prove that, regardless of the starting point, the trajectory given by the solution of differential equations does in fact approach the fixed point quickly over time under some metric regardless of the initial starting point. That is, we would like to show *convergence* of the system to its fixed point. Such convergence results have been shown previously for similar systems in [33], [42], and [43].

In some cases where we cannot prove convergence, we can prove a weaker result, namely the *stability of the fixed point*. Techniques for proving stability similar to those used here are also described in Section 4.6 of [33]. For our purposes, we say that a fixed point is stable if the L_1 distance to the fixed point is nonincreasing over time. That is, given any initial starting point $s_i(0)$ for our family of differential equations and a corresponding fixed point π_i, we say the fixed point is stable if the distance $D(t) = |s_i(t) - \pi_i|$ is nonincreasing. This is stronger than the standard definition.[1] Although stability only shows that the trajectory does not ever head away from the fixed point, it provides some reason to believe that the trajectory given by the differential equations converges rapidly to its fixed point from any suitable starting point (namely where the expected load is finite, or $\sum_i s_i < \infty$)) under the L_1 metric. In the work stealing setting, both stability and convergence results prove difficult. Even for the simple system given by (2) and (3), we can currently only prove the stability of the fixed point for sufficiently small arrival rates λ, as shown in the theorem below.

We emphasize that, in practice, one can check for convergence to the fixed point numerically using various starting points to convince oneself that the system is well behaved. Devising proofs for the convergence or better proofs for the stability of the work stealing systems described here, however, remain important open questions.

Theorem 1. *The system given by* (2) *and* (3) *is stable for* λ *such that* $\pi_2 \le \frac{1}{2}$.

Proof. Define $\varepsilon_i(t) = s_i(t) - \pi_i$. (Note $\varepsilon_0(t)$ is identically 0.) We drop the explicit dependence on t when the meaning is clear. The L_1 distance $D(t)$ is then $\sum_{i\ge1} |\varepsilon_i(t)|$.

As $D(t) = \sum_{i=1}^{\infty} |\varepsilon_i(t)|$, the derivative of D with respect to t, or dD/dt, is not well defined if $\varepsilon_i(t) = 0$ for some i. We explain how to cope with this problem at the end of the proof, and we suggest the reader proceed by temporarily assuming $\varepsilon_i(t) \ne 0$.

As $d\varepsilon_i/dt = ds_i/dt$, we may obtain equations for $d\varepsilon_i/dt$ using (2). It is convenient to write the derivatives $d\varepsilon_i/dt$ in the following form:

$$\frac{d\varepsilon_1}{dt} = -\lambda\varepsilon_1 - (\varepsilon_1 - \varepsilon_2)(1 - s_2) + \varepsilon_2(\pi_1 - \pi_2), \tag{11}$$

$$\begin{aligned}\frac{d\varepsilon_i}{dt} &= \lambda(\varepsilon_{i-1} - \varepsilon_i) - (\varepsilon_i - \varepsilon_{i+1})(1 + \pi_1 - \pi_2) \\ &\quad - (\varepsilon_1 - \varepsilon_2)(s_i - s_{i+1}), \qquad i > 1.\end{aligned} \tag{12}$$

[1] There are many variations of the basic notion of stability, including asymptotic stability, uniformly asymptotic stability, etc. Definitions are covered in many texts on the subject, and there are more specialized treatises, such as [45]. We note also that here we consider the L_1 metric. We point out that as far as we know it is certainly possible that one could prove convergence or stability for larger ranges using this or another metric, such as for example the supremum metric.

Note that

$$\frac{dD}{dt} = \sum_{i=1}^{\infty} \frac{d|\varepsilon_i|}{dt}.$$

Using the above, we examine the sum of terms containing ε_i in dD/dt, and show that the resulting expression is nonpositive for each i.

We first consider the case where $i \geq 3$, as the cases $i = 1, 2$ are more difficult. There are several subcases, depending on whether ε_{i-1}, ε_i, and ε_{i+1} are positive or negative. We first consider the case where they are all positive. Then the sum of terms containing ε_i in dD/dt are

$$\varepsilon_i[(-\lambda - 1 - \pi_1 + \pi_2) + \lambda + (1 + \pi_1 - \pi_2)] = 0.$$

More generally, let $\operatorname{sgn}(x)$ be the sign of x; that is, $\operatorname{sgn}(x)$ is 1 if x is positive, -1 is x is negative, and 0 if x is positive. Under the assumption that $\varepsilon_i(t) \neq 0$ for all i, we have that the sum of terms containing ε_i in dD/dt for $i \geq 3$ are

$$\varepsilon_i[(-\lambda - 1 - \pi_1 + \pi_2) \operatorname{sgn}(\varepsilon_i) + \lambda \operatorname{sgn}(\varepsilon_{i+1}) + (1 + \pi_1 - \pi_2) \operatorname{sgn}(\varepsilon_{i+1})].$$

It is easily checked that this sum is always nonpositive, by noting that, regardless of the sign of ε_i, the first term in the brackets dominates the others, and its sign is the same as that of ε_i.

For the case $i = 2$, there are ε_2 terms in all other $d|\varepsilon_j|/dt$ terms. However, the terms from $d|\varepsilon_2|/dt$ dominate all others. For example, if ε_2 is positive, the corresponding terms in $d|\varepsilon_2|/dt$ are

$$(-\lambda - (1 - s_2) - (\pi_1 - \pi_2) - s_3)\varepsilon_2.$$

Even if all other ε_j are set so that the coefficients of ε_2 in $d|\varepsilon_j|/dt$ are positive, the sum of the other ε_2 terms is just

$$(\lambda + (1 - s_2) + (\pi_1 - \pi_2) + s_3)\varepsilon_2,$$

and hence the sum of terms containing ε_2 in dD/dt is always nonpositive.

The only difficulty lies in the case where $i = 1$. In this case, if (for example) ε_1 and ε_2 are both positive, then the sum of terms containing ε_1 from $d|\varepsilon_1|/dt$ and $d|\varepsilon_2|/dt$ is $-(1 - s_3)\varepsilon_1$. If $\varepsilon_j < 0$ for $j \geq 3$, however, then the sum of ε_1 terms from all other $d|\varepsilon_j|/dt$ is $s_3\varepsilon_1$. Hence the total sum of all terms could be as much as $-(1 - 2s_3)\varepsilon_1$, which is positive when $s_3 > \frac{1}{2}$. This case, however, requires that $s_3 \leq \pi_3$, since $\varepsilon_3 < 0$. Hence if $\pi_3 \leq \frac{1}{2}$, the coefficient of ε_1 is negative as desired.

Explicitly checking the remaining cases reveals that the worst case in this instance is when ε_1 and ε_2 are both negative and $\varepsilon_j > 0$ for $j \geq 3$. Then the corresponding sum of terms is $(1 - 2s_3)\varepsilon_1$, with the limitation that $s_3 \leq s_2 \leq \pi_2$. Hence, if we restrict λ so that $\pi_2 < \frac{1}{2}$, then the ε_1 terms are always nonpositive, so we have stability.

We now consider the technical problem of defining dD/dt when $\varepsilon_i(t) = 0$ for some i. Since we are interested in the forward progress of the system, it is sufficient to consider the upper right-hand derivatives of ε_i. (See, for instance, p. 16 of [27].) That is, we may define

$$\left.\frac{d|\varepsilon_i|}{dt}\right|_{t=t_0} \equiv \lim_{t\to t_0^+} \frac{|\varepsilon_i(t)|}{t - t_0},$$

and similarly for $d\Phi/dt$. Note that this choice has the following property: if $\varepsilon_i(t) = 0$, then $(d|\varepsilon_i|/dt)|_{t=t_0} \geq 0$, as it intuitively should be. The above proof applies unchanged with this definition of dD/dt, with the understanding that the case $\varepsilon_i > 0$ includes the case where $\varepsilon_i = 0$ and $d\varepsilon_i/dt > 0$, and similarly for the case $\varepsilon_i < 0$. □

In the case of threshold stealing, we have essentially the same result.

Theorem 2. *The system developing according to* (4), (5), *and* (6) *is stable for* λ *such that* $\pi_2 \leq \frac{1}{2}$.

Proof. The proof follows the same case by case analysis pattern as Theorem 1, where the only problem is in bounding the coefficient of ε_1. □

5. Conclusions

We have suggested an approach for analyzing load stealing systems based on limiting models of such systems that can be represented by families of differential equations. The advantages of this modeling technique include simplicity, generality, and the ability to predict performance accurately. Our limiting models yield results that provide insight into why simple, decentralized work stealing schemes prove effective in practice. In particular, in an idealized dynamic setting where steals occur instantaneously, the tails of the task queues at the processors decrease geometrically at a faster rate than without load stealing. We have also examined more complex models, such as a model where there is a delay associated with the time to transfer a task. We have seen that even a small amount of delay can impact performance, a point which we believe merits further study in actual systems.

At a more theoretical level, it would be useful to develop a more general framework for handling the convergence issues for the families of differential equations that arise from stealing systems. Although the work of [33] and [43] suggests a framework that can be used to prove convergence for several load sharing systems, it does not appear that they are sufficient to handle the complexity of many stealing problems.

Acknowledgments

The author thanks the anonymous reviewers for several helpful comments regarding the structure and content of the paper.

References

[1] D. Achlioptas and M. Molloy. The analysis of a list-coloring algorithm on a random graph. In *Proceedings of the* 38*th IEEE Symposium on Foundations of Computer Science*, pages 204–212, 1997.

[2] M. Alanyali and B. Hajek. Analysis of simple algorithms for dynamic load balancing. In *INFOCOM* 95, vol. 1, pages 230–238, 1995.

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

[4] R. Blumofe. Executing Multithreaded Program Efficiently. Ph.D. thesis, Massachusetts Institute of Technology, September 1995.

[5] R. Blumofe, M. Frigo, C. Joerg, C. Leiserson, and K. Randall. An analysis of dag-consistent distributed shared-memory algorithms. In *Proceedings of the* 8*th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 297–308, 1996.

[6] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, and Y. Zhou. Cilk: an efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, 1996.

[7] R. Blumofe and C. Leiserson. Space-efficient scheduling of multithreaded computations. In *Proceedings of the* 25*th Annual ACM Symposium on Theory of Computing*, pages 362–371, 1993.

[8] R. Blumofe and C. Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the* 35*th Annual IEEE Conference on Foundations of Computer Science*, pages 356–368, 1994.

[9] The Cilk Project. Available at `http://theory.lcs.mit.edu/∼cilk/home/intro.html`.

[10] M. Dahlin. Interpreting stale load information. In *Proceedings of the* 19*th IEEE International Conference on Distributed Computing Systems* (*ICDCS*), pages 285–296, May 1999. Also available as TR98-20, Department of Computer Sciences, University of Texas at Austin.

[11] D. L. Eager, E. D. Lazowska, and J. Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation Review*, 16:53–68, March 1986.

[12] S. N. Ethier and T. G. Kurtz. *Markov Processes*: *Characterization and Convergence*. Wiley, New York 1986.

[13] R. J. Gibbens, P. J. Hunt, and F. P. Kelly. Bistability in communication networks. In *Disorder in Physical Systems* (G. R. Grimmett and D. J. A. Welsh, eds.), pages 113–128. Oxford University Press, Oxford, 1990.

[14] B. Hajek. Asymptotic analysis of an assignment problem arising in a distributed communications protocol. In *Proceedings of the* 27*th Conference on Decision and Control*, pages 1455–1459, 1988.

[15] M. Harchol-Balter and D. Wolfe. Bounding delays in packet-routing networks. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 248–257, 1995.

[16] R. M. Karp and M. Sipser. Maximum matchings in sparse random graphs. In *Proceedings of the* 22*nd IEEE Symposium on Foundations of Computer Science*, pages 364–375, 1981.

[17] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the* 22*nd ACM Symposium on the Theory of Computing*, pages 352–358, 1990.

[18] R. M. Karp and Y. Zhang. A randomized parallel branch-and-bound procedure. In *Proceedings of the* 20*th ACM Symposium on the Theory of Computing*, pages 290–300, 1988.

[19] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, New York, 1979.

[20] L. Kleinrock. *Queueing Systems*, Volume I. Wiley, New York, 1976.

[21] T. G. Kurtz. Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7:49–58, 1970.

[22] T. G. Kurtz. Limit theorems for sequences of jump Markov processes approximating ordinary differential processes. *Journal of Applied Probability*, 8:344–356, 1971.

[23] T. G. Kurtz. Strong approximation theorems for density dependent Markov chains. *Stochastic Processes and Applications*, 6:223–240, 1978.

[24] T. G. Kurtz. *Approximation of Population Processes*. CBMS–NSF Regional Conference Series in Applied Mathematics SIAM, Philadelphia, PA, 1981.

[25] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the* 29*th ACM Symposium on the Theory of Computing*, pages 150–159, 1997.

[26] J. Martin and Y. M. Suhov. Fast Jackson networks. *Annals of Applied Probability*, 9(3):854–870, 1999.

[27] A. N. Michel and R. K. Miller. *Qualitative Analysis of Large Scale Dynamical Systems*. Academic Press, New York, 1977.

[28] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Analysis of the effects of delays on load sharing. *IEEE Transactions on Computing*, 38:1513–1525, 1989.

[29] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Adaptive load sharing in heterogeneous systems. *Journal of Parallel and Distributed Systems*, 9:331–346, 1990.

[30] M. Mitzenmacher. Bounds on the greedy routing algorithm for array networks. In *Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 248–259, 1994. To appear in the *Journal of Computer Systems and Science*.

[31] M. Mitzenmacher. Constant time per edge is optimal on rooted tree networks. In *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 162–169, 1996.

[32] M. Mitzenmacher. Load balancing and density dependent jump Markov processes. In *Proceedings of the* 37*th IEEE Symposium on Foundations of Computer Science*, pages 213–222, 1996.

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

[34] M. Mitzenmacher. How useful is old information? In *Proceedings of the* 16*th ACM Symposium on Principles of Distributed Computing*, 1997, pages 83–91. Journal version to appear in *IEEE Transactions on Parallel and Distributed Systems*.

[35] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the* 9*th ACM Symposium on Parallel Algorithms and Architectures*, pages 292–301, 1997. Journal version in *Theory of Computing Systems*, 32:361–386, 1999.

[36] M. Mitzenmacher. Tight Thresholds for the Pure Literal Rule. Technical Note 1997-011, Digital Systems Research Center, Palo Alto, CA, June 1997.

[37] B. Pittel, J. Spencer, and N. Wormald. Sudden emergence of a giant k-core in a random graph. *Journal of Combinatorial Theory, Series B*, 67:111–151, 1996.

[38] R. Righter and J. Shanthikumar. Extremal properties of the FIFO discipline in queueing networks. *Journal of Applied Probability*, 29:967–978, November 1992.

[39] L. Rudolph, M. Slivkin-Allalouf, and E. Upfal. A simple load balancing scheme for task allocation in parallel machines. In *Proceedings of the* 3*rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 237–245, 1991.

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

[41] G. D. Stamoulis and J. N. Tsitsiklis. The efficiency of greedy routing in hypercubes and butterflies. *IEEE Transactions on Communications*, 42(11):3051–3061, November 1994. An early version appeared in the *Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 248–259, 1991.

[42] 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*, 32:15–27, 1996.

[43] N. D. Vvedenskaya and Y. M. Suhov. Dobrushin's mean-field approximation for a queue with dynamic routing. *Markov Processes and Related Fields*, 3(4):493–527, 1997. Also available as Technical Report 3328, INRIA, December 1997.

[44] N. C. Wormald. Differential equations for random processes and random graphs. *Annals of Applied Probability*, 5:1217–1235, 1995.

[45] T. Yoskizawa. *Stability Theory and the Existence of Periodic Solutions and Almost Periodic Solutions*. Springer-Verlag, New York, 1975.

Received August 20, 1999, *and in revised form March* 15, 2000, *and in final form July* 17, 2000.
Online publication October 25, 2000.