Analyses of Load Stealing Models Based on Differential Equations

Michael Mitzenmacher
Digital Systems Research Center
130 Lytton Ave.
Palo Alto, CA 94301
michaelm@pa.dec.com

Abstract

In this paper we develop models for and analyze 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 [4, p.6]), including the Cilk system [6, 8].

In this paper we analyze several simple randomized work stealing algorithms in a dynamic setting using simple Markovian models and an approach that has similarly been used to study work sharing algorithms [28, 29, 30, 37, 38]. Primarily we study variations of the WS algorithm described by Blumofe and Leiserson [8]. We focus on the dynamic model where 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, representing their behavior by 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.

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

1.1 Previous Work

Work stealing has been treated extensively in a series of papers by Blumofe, Leiserson, and others [5, 6, 7, 8], who use work stealing in their Cilk system for parallel processing. Their models, which include not only computation time but also memory usage and communication costs, demonstrate that their work-stealing algorithms are optimal up to a constant factor in terms of execution time, and existentially optimal up to constant factors in terms of space and communication. Their experiments further show that their algorithms work well in practice. Other work stealing algorithms have also been developed and analyzed by Rudolph, Slivkin-Allalouf, and Upfal [34] and Karp and Zhang [15].

Work stealing has also been the subject of attention in the queueing theory literature, most notably in the early work by Eager, Lazwowska, and Zahorjan [9] and the later work by Mirchandaney, Towsley, and Stankovic [24, 25]. Our work is similar to theirs, although both our 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, 11, 22, 29, 37, 38]. 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 large deviations; see, for instance, the body of work of Kurtz [10, 17, 18, 19, 20], or a more modern treatment by Shwartz and Weiss [35]. The use of this approach in the study of algorithms dates back to work by Karp and Sipser [13], and has since been used to analyze several other algorithms, for example in [1, 11, 14, 21, 31, 32, 39]. Note that here we focus on how to use the technique and what insight it, in conjunction with simulations, gives us 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 family of equations describing the performance of a basic work stealing 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 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: in the limiting model, 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 dynamically generated tasks, generated at each processor as a Poisson process of rate $\lambda < 1$. Tasks require an amount of service that is exponentially distributed with mean 1 before completing. The service times required by the tasks are not known to the processors. Tasks are served according to a 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 size 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. As we shall see, the s_i prove much more convenient to work with than the p_i. Note that $s_0 = 1$ always, and that the s_i are non-increasing as $s_{i-1} - s_i = p_i$. For the well-behaved systems we will be considering, we also have that $\lim_{i\to\infty} s_i = 0$.

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 size. Since we are not making use of locality, the processors are indistinguishable, and this is all the information we require. Also, under the assumption that service times are exponential and arrivals are Poisson, 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 $t \to \infty$, by a straightforward comparison with a system without stealing.

To gain insight into how to set up the appropriate limiting system, let us first consider a system without load stealing. Let dm_i represent the expected change in m_i over a small interval of time dt. We think of dt as being a small enough interval of time so that only one event (an arrival or departure) can happen at each processor in the interval. Let us 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 an arrival occurs at each processor with i tasks is λdt. Hence the expected change in m_i due to arrivals is just $\lambda(m_{i-1} - m_i)dt$. Similarly, the expected change in m_i due to departures is $(m_i - m_{i+1})dt$. Hence, the expected behavior of the system over short intervals is given by

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

Canceling the factor of n permeating the equations we find:

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

Note that the system of differential equations (1) are independent of n, the number of processors in the system. Instead, the are determined by *densities* of processors with a certain size. Further, the differential equations no longer describe a random process, but a deterministic one: given an initial condition, the solution of the system can be shown to be unique. Again, these differential equations describe the *expected* behavior of the system over small periods of time.

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 behavior of the Markov process converges to that of the deterministic process given by the corresponding differential equations (subject to certain conditions). That the system behaves according to its expectation in the limit is hardly surprising; essentially, it is a variation of the law of large numbers for density dependent Markov processes. Rather than focus on the technical details of this convergence, we explore how to use this methodology. The details of the theory behind this convergence can be found in many sources, including [10, 29, 37, 38, 39].

Let us now consider how to modify the above equations in the case of load stealing. A processor that completes its final task attempts to find a victim, thereby reducing the rate at which it actually empties. The probability of success is just s_2, the probability of choosing a victim processor that contains at least two tasks. Hence, to the processor, it appears as though it loses its final task only 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

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

Notice that we maintain the property that our system of equations is density dependent (independent of n). 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. Most well-behaved processes follow trajectories that converge to their fixed points.

To determine the 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 the fixed point, the rate at which tasks complete and the rate at which they are introduced must be equal.

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

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

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

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

Hence, for $i \geq 2$, the π_i decrease geometrically. Let us contrast this result with the case of no stealing, where the fixed point is $\pi_i = \lambda^i$, as can be verified by equation (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 the 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 load decrease geometrically with a ratio λ/μ between successive terms. (Recall that we have scaled so that $\mu = 1$ for the equation (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 customers in the work stealing system, the *apparent* service rate μ' is the standard service rate 1 plus the rate at which a task is stolen, which is $\pi_1 - \pi_2 = \lambda - \pi_2$. Hence we expect the tails 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 briefly demonstrate the accuracy of the limiting approach by comparing the predicted results for the expected time each task spends in the system with simulations for this simple WS 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:

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

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

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

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

Table 1: Simulations vs. estimates for the simplest WS model. The relative error is between the simulatins with 128 processors and the estimate based on the fixed point calculation.

Again, we seek a fixed point, beginning with $\pi_0 = 1$ and $\pi_1 = \lambda$. From equation (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 equation (5):

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

This yields

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

which allows us to solve for π_T:

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

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

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

We will show that $\pi_T = \frac{\lambda\pi_{T-1}}{1+\pi_1-\pi_2}$; then a simple induction using equation (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 a system without load stealing. This equation also matches the intuition develop in Section 2.2; for queues with load at least T, the apparent service rate μ' is again the standard service rate 1 plus the rate at which a task is stolen, which is $\pi_1 - \pi_2 = \lambda - \pi_2$.

To show $\pi_T = \frac{\lambda\pi_{T-1}}{1+\pi_1-\pi_2}$ we use the fact that $\sum_{i=1}^{T-1} \frac{ds_i}{dt} = 0$ 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 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 we have two parameters: B, the number of tasks at which a processor begins steal attempts, and T, a threshold such that a processor with i tasks will only steal from a processor with at least $i+T$ tasks. The limiting system has the following form:

$$\begin{aligned}
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1})(1 - s_{i+T-1}), 1 \leq i \leq B+1 \\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}), B+2 \leq i \leq T-1 \\
\frac{ds_i}{dt} &= \lambda(s_{i-1} - s_i) - (s_i - s_{i+1}) \\
&\quad - (s_i - s_{i+1})(s_1 - s_{\min(B+2,i-T+2)}) \ , \ i \geq T
\end{aligned}$$

In this case, for $i > B+T$, the tails decrease geometrically according to

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

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

2.5 Repeated Steal Attempts

In 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 a suitable victim. 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. If there is a threshold T so that a victim must have at least T tasks, the equations describing the limiting system become:

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

Again, in this system at the fixed point $\vec{\pi}$ the π_i decrease

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

Table 2: Simulations vs. Estimates for 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.

geometrically for $i > T$, according to the formula

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

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 load T tasks will have a task stolen immediately.

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).

3.1 Varying service and arrival distributions

The need for exponential service 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 [16][Sections 4.2 and 4.3]. We demonstrate the method here by considering the case of constant service times. We replace the constant service time with a collection of c stages of services; each stage of service is exponentially distributed with mean $1/c$. A service distribution of this type is a *gamma distribution.* As c goes to infinity, the expected time spent in these c stages of service remains 1 and the variance approaches zero; that is, the service appears like a constant random variable. 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. 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. For the case where $T = 2$, that is, if we steal whenever possible, the resulting equations are:

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

In principle, this approach could be used to develop deterministic differential equations that approximate the 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. 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 numerically evaluate the fixed point.

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. We do not have a proof that this holds in general; it would be interesting to prove such a result either using the fixed point (see [29, Section 4.3]) or other techniques (see, for example, [12, 26, 27, 33, 36]).

3.2 Transfer time

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. For convenience we model the transfer time 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 explicitly distinguish thief processors who are awaiting a stolen task from other processors. Our state space will hence consist of two infinite dimensional vectors: $(s_0, s_1, \ldots)$ will record the fraction of processors not awaiting a stolen task, where as usual s_i refers to the fraction of such processors with at least i tasks. A second vector, $(w_0, w_1, \ldots)$ will similarly record the fraction

λ	$T=3$ Sim(128)	$T=3$ Est.	$T=4$ Sim(128)	$T=4$ Est.	$T=5$ Sim(128)	$T=5$ Est.	$T=6$ Sim(128)	$T=6$ 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

Table 3: The expected time with transfer times, 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.

of processors awaiting a stolen task. The variable w_i refers to the fraction of such processors currently with at least i tasks.

We note 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 $\sum_{i\geq 1} s_i + \sum_{i\geq 0} w_i$; this summation accounts for the extra tasks in transit between processors.

The equations below describe this process when a steal is attempted only when a processor empties and a steal happens only when the victim processor has at least T tasks:

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

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. Such a rule would suggest that the best threshold T must satisfy $T \approx (1/r) + 1$. 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 the differential equations can be used correctly determine the best threshold value for various arrival rates.

3.3 Multiple choices

In load sharing algorithms, systems that have some choice of where to place new jobs has proven to have different performance characteristics than systems where jobs are placed randomly [3, 29, 37]. 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 such performance measures as the average time in the system and the expected heaviest load in the system over a system where each task queues at a random server. 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:

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

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 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 that again the estimate derived fixed point is an accurate prediction for actual systems of 128 processors at reasonable arrival rates.

λ	Sim(128) 1 choice	Sim(128) 2 choices	Estimate 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

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

3.4 Multiple steals

In certain situation 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. Let us 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 is made, many s_i values decrease. Taking this into consideration yields the following family of differential equations:

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

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, Slivkin-Allalouf, and Upfal [34], 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 re-balancing 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 re-balancing event occurs, the tasks are balanced between this processor and another processor chose uniformly at random. (For convenience, the processor with the larger initial load with also have the larger final load.) Surprisingly, this system can be represented in a quite straightforward manner: for $i \geq i$,

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

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 $\vec{w} = (w_0, w_1, w_2, \ldots)$. 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 $\lambda_{ext} + \lambda_{int}$, 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 $\lambda_{ext} = 0$ and $\lambda_{int} = 0$ 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, the limiting system can give a good approximation for the amount of time until the system completes all jobs.

4 Convergence and Stability

Thus far, when considering families of differential equations, we have focused on finding a fixed point, with the intuition that the system converges to that fixed point. To justify this intuition one would hope to prove that, regardless of the starting point, the trajectory taken by the path given by the solution of differential equations does in fact approach the fixed point quickly over time under some metric. 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 [29, 37, 38].

In some cases where we cannot prove convergence, we can prove a weaker result, namely the *stability of the fixed point.* For our purposes, we shall say that a fixed point is stable if the L_1 distance to the fixed point is non-decreasing over time. (This is stronger than the standard definition.) Although this only shows that the trajectory does not ever head away from the fixed point, it provides some reason to believe that the system converges rapidly to its fixed point. Techniques for proving stability are also described in [29, Section 4.6]. In the work stealing setting, both stability and convergence results prove difficult. Even for the simple system given by equations (2) and (3), we can only prove the stability of the fixed point for sufficiently small arrival rates λ, as we shall show in the theorem below.

We note 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, remains an important open question.

Theorem 1 *The system given by equations (2) and (3) are stable for λ such that $\pi_2 \le 1/2$.*

Proof: Define $\epsilon_i(t) = s_i(t) - \pi_i$. (Note $\epsilon_0(t)$ is identically 0.) We drop the explicit dependence on t when the meaning is clear. The L_1 distance $D(t)$ is then $\sum_{i\ge1} |\epsilon_i(t)|$.

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

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

$$\frac{d\epsilon_1}{dt} = -\lambda\epsilon_1 - (\epsilon_1 - \epsilon_2)(1 - s_2) + \epsilon_2(\pi_1 - \pi_2); \quad (8)$$

$$\frac{d\epsilon_i}{dt} = \lambda(\epsilon_{i-1} - \epsilon_i) - (\epsilon_i - \epsilon_{i+1})(1 + \pi_1 - \pi_2) - (\epsilon_1 - \epsilon_2)(s_i - s_{i+1}),\ i > 1; \quad (9)$$

Note that

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

Using the above, we examine the terms sum of containing ϵ_i in $\frac{dD}{dt}$, and show that the resulting expression is non-positive for each i.

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

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

Similarly, in all subcases, the sum telescopes to something linear in ϵ_i with a coefficient that is zero or the opposite sign from ϵ_i. Hence the sum of terms containing ϵ_i in dD/dt is always non-positive for $i \ge 3$.

For the case $i = 2$, there are ϵ_2 terms in all other $\frac{d|\epsilon_j|}{dt}$ terms. However, the terms from $\frac{d|\epsilon_2|}{dt}$ dominate all others. For example, if ϵ_2 is positive, the corresponding terms in $\frac{d|\epsilon_2|}{dt}$ are

$$(-\lambda - (1 - s_2) - (\pi_1 - \pi_2) - s_3)\epsilon_2.$$

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

$$(\lambda + (1 - s_2) + (\pi_1 - \pi_2) + s_3)\epsilon_2,$$

and hence the sum of terms containing ϵ_2 in dD/dt is always non-positive.

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 $\frac{d|\epsilon_1|}{dt}$ and $\frac{d|\epsilon_2|}{dt}$ is $-(1 - s_3)\epsilon_1$. If $\epsilon_j < 0$ for $j \ge 3$, however, then the sum of ϵ_1 terms from all other $\frac{d|\epsilon_j|}{dt}$ is $s_3\epsilon_1$. Hence the total sum of all terms could be as much as $-(1 - 2s_3)\epsilon_1$, which is positive when $s_3 > 1/2$. This case, however, requires that $s_3 \le \pi_3$, since $\epsilon_3 < 0$. Hence if $\pi_3 \le 1/2$, the coefficient of ϵ_1 is negative as desired.

The worst case in this instance is when ϵ_1 and ϵ_2 are both negative and $\epsilon_j > 0$ for $j \ge 3$. Then the corresponding sum of terms is $(1-2s_3)\epsilon_1$, with the limitation that $s_3 \le s_2 \le \pi_2$. Hence, if we restrict λ so that $\pi_2 < 1/2$, then the ϵ_1 terms are always non-positive, so we have stability.

We now consider the technical problem of defining dD/dt when $\epsilon_i(t) = 0$ 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, [23, p. 16].) That is, we may define

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

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

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

Theorem 2 *The system given by equations (4),(5), and (6) are stable for λ such that $\pi_2 \le 1/2$.*

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

We leave the reader to consider the other systems we have described.

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 accurately predict performance. Our results provide some 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 expect that these models will prove useful in designing future systems that use load stealing methods to balance load.

References

[1] D. Achlioptas and M. Molloy. The analysis of a list-coloring algorithm on a random graph. In *Proceedings of the 38th 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*, 1995.

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

[4] R. Blumofe. *Executing Multithreaded Program Efficiently*. PhD 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 8th Annual ACM Syposium on Parallel Algorithms and Architectures*, 1996.

[6] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. In *Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 1995.

[7] R. Blumofe and C. Leiserson. Space-efficient scheduling of multithreaded computations. In *Proceedings of the 25th 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 35th Annual IEEE Conference on Foundations of Computer Science*, 1994.

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

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

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

[12] 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.

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

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

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

[16] L. Kleinrock. *Queueing Systems, Volume I: Theory*. John Wiley and Sons, 1976.

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

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

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

[20] T. G. Kurtz. *Approximation of Population Processes. CBMS-NSF Regional Conf. Series in Applied Math.* SIAM, 1981.

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

[22] J. Martin and Y. M. Suhov. Fast jackson networks. available at `www.statslab.cam.ac.uk/~jmb`, January 1998.

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

[24] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Analysis of the effects of delays on load sharing. *Journal of Parallel and Distributed Systems*, 1513-1525:331–346, November 1989.

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

[26] 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*.

[27] 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.

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

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

[30] M. Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures*, pages 292–301, 1997.

[31] M. Mitzenmacher. Tight thresholds for the pure literal rule. Technical Report Technical Note 1997-011, Digital Systems Research Center, June 1997.

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

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

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

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

[36] 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*, p. 248-259, 1991.

[37] 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.

[38] N. D. Vvedenskaya and Y. M. Suhov. Dobrushin's mean-field approximation for a queue with dynamic routing. Technical Report 3328, INRIA, December 1997.

[39] N. C. Wormald. Differential equations for random processes and random graphs. *Annals of Appl. Prob.*, 5:1217–1235, 1995.