The Natural Work-Stealing Algorithm is Stable*

Petra Berenbrink and Tom Friedetzky and Leslie Ann Goldberg
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, United Kingdom
E-mail: (petra|tf|leslie)@dcs.warwick.ac.uk

April 26, 2001

Abstract

In this paper we analyse a very simple dynamic work-stealing algorithm. In the work-generation model, there are n *generators* which are *arbitrarily* distributed among a set of n *processors*. The distribution of generators is arbitrary — generators may even move at the beginning of each time step. During each time-step, each generator may generate a unit-time task which it inserts into the queue of its host processor. It generates such a task independently with probability λ. After the new tasks are generated, each processor removes one task from its queue and services it. Clearly, the work-generation model allows the load to grow more and more imbalanced, so, even when $\lambda < 1$, the system load can be unbounded. The natural work-stealing algorithm that we analyse is widely used in practical applications and works as follows. During each time step, each *empty* processor (with no work to do) sends a request to a randomly selected other processor. Any *non-empty* processor having received at least one such request in turn decides (again randomly) in favour of one of the requests. The number of tasks which are transferred from the non-empty processor to the empty one is determined by the so-called *work-stealing function* f. In particular, if a processor that accepts a request has ℓ tasks stored in its queue, then $f(\ell)$ tasks are transferred to the currently empty one. A popular work-stealing function is $f(\ell) = \lfloor \ell/2 \rfloor$, which transfers (roughly) half of the tasks. We analyse the *long-term behaviour* of the system as a function of λ and f. We show that the system is *stable* for any constant generation rate $\lambda < 1$ and for a wide class of functions f. Most intuitively sensible functions are included in this class (for example, every function $f(\ell)$ which is $\omega(1)$ as a function of ℓ is included). We give a quantitative description of the functions f which lead to stable systems. Furthermore, we give *upper bounds* on the average system load (as a function of f and n). Our proof techniques combine Lyapunov function arguments with domination arguments, which are needed to cope with dependency.

*This work was partially supported by EPSRC grant GR/M96940 "Sharper Analysis of Randomised Algorithms: a Computational Approach" and the IST Programme of the EU under contract numbers IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX).

1 Introduction

Load balancing is the process of distributing load among a set of processors. There are two main approaches to distributed load balancing, namely *sender-initiated* strategies, in which processors may decide to give away tasks, and *receiver-initiated* strategies (which are often referred to as *work-stealing*), in which processors may request extra work. In both cases, the decision to transfer tasks is typically *threshold based*. That is, it is based on having too many or too few tasks in one's own queue.

In recent years, there has been a lot of work devoted to rigorously analysing load balancing, most of which concentrates on sender-initiated approaches or related *allocation processes* such as *balls-into-bins games*. However, it appears that many practitioners prefer the receiver-initiated approach (work-stealing) because this approach appears to work better for their applications. The efficiency of work-stealing probably helps to explain the immense success of Leiserson et al.'s language `Cilk` [BJK+96], a language for multithreaded parallel programming which uses work-stealing in its kernel. There are numerous examples of practical applications of work-stealing. In [Fel94], Feldmann, Monien and Mysliwietz investigate the behaviour of parallel MIN/MAX-tree evaluation in the context of parallel game (chess) programs employing work-stealing strategies. In [Dec00], Decker introduces VDS (Virtual Data Space), a load-balancing system for irregular applications that makes use of work stealing and other strategies. In [MD98], Dutt and Mahapatra use work stealing for parallel branch&bound-type algorithms.

Despite the practical usefulness of work stealing, theoretical results for all but a few specialised applications are still missing. This paper attempts to close this gap.

Most existing theoretical work on load balancing assumes a more or less inherently well-behaved system. For instance, most work on sender-initiated load-balancing uses a work-generation model in which processors generate at most a constant number of new tasks per step. In balls-into-bins games the balls (tasks) choose their bin (processor) uniformly at random, which also yields a relatively balanced system.

In this paper we analyse a simple and fully distributed work-stealing algorithm. Our work-generation model allows for an arbitrary placement of n so-called *generators* among the set of n processors. Each generator generates a new task with a certain probability λ at each time step. In the extreme case, there can be one processor being host to n generators. In this case the one processor increases its load by expected $\lambda n - 1$ tasks per step, whereas all other processors do not generate tasks at all.

Our load-balancing algorithm follows a very simple and natural work-stealing approach. At each time step, each *empty* processor sends a request to one randomly chosen other processor. Each *non-empty* processor having received at least one such request selects one of them. Now each empty processor P whose request is accepted by a processor, Q, "steals" $f(\ell)$ tasks from Q, where ℓ denotes Q's load.

Our results are concerned mostly with the *stability* of the system. A system is said to be *unstable* if the system load (the sum of the load of all processors) grows unboundedly with time. We present both negative and positive results, depending on the choice of the work-stealing function f. First we show that if the work-stealing function is $\omega(1)$ as a function of the load (i.e., $f(\ell) = \omega(1)$ as a function of ℓ) then the system is stable. The system is also stable for other functions, e. g. if $f(\ell)$ is at least a sufficiently large polynomial in n, the number of processors, independently of ℓ. This is complemented by a straightforward lower bound: The system is unstable if the $f(\ell)$ is too small, for example, if $f(\ell) < \lambda n$. Finally, we provide upper bounds on the system load in a stationary system (again, depending on f).

1.1 New Results

Before we state our results, we will introduce the model we are using as well as the work-stealing algorithm itself.

The model. We are assuming a collection of n synchronised *processors*, connected by some network topology. During each time step, every processor may send a request to at most one other processor, and any processor receiving more than one such request will have to decide to accept at most one of them.

Furthermore, we have a set of n so-called *generators*. At each time step, each generator generates a task with a probability of λ, for some fixed $\lambda \in [0, 1)$. The generators are arbitrarily distributed among the processors. A processor can host anything from zero to n generators. We do not even assume a fixed distribution but allow the generators to arbitrarily change hosts between any two time steps. This model allows for anything from "even distribution of generators" to "all generators on one processor". Whenever a generator generates a task, this task is instantly inserted into its host's queue. At each time step, each processor removes one task from its queue and services it. We assume constant service time for all tasks.

Obviously, without making use of some sort of load-balancing mechanism this model can lead to an arbitrarily high work load imbalance in the system. Suppose, for instance, all generators are hosted by just one processor. This processor generated expected λn tasks per step but services just one. Hence, after t time steps, it will have an expected load of $t\lambda n - t$, with the other processors being idle all the time.

The algorithm. The work-stealing algorithm is very simple and natural. Each each time step, each *empty* processor (with no work to do) sends a request to one other processor, which is chosen independently and uniformly at random. Each *non-empty* processor having received at least one such request selects one of the requests . Now each empty processor with its request being accepted "steals" tasks from the other processor.

The number of tasks which are transferred from the non-empty processor to the empty one is determined by the so-called *work-stealing function* f. In particular, if a processor that accepts a request has ℓ tasks stored in its queue, then $f(\ell)$ tasks are transferred to the currently empty one. A popular work-stealing function is $f(\ell) = \lfloor \ell/2 \rfloor$, which transfers (roughly) half of the tasks.

The results. Recall that a system is said to be stable if the system load (the sum of the load of all processors) does not grow unboundedly with time. Obviously, stability for large arrival rates is among the most desirable features of load-balancing algorithms.

In Theorem 1 we show that, given a suitable work-stealing function, our algorithm yields a stable system for *any* constant arrival rate $\lambda < 1$ and any distribution of the generators. Most intuitively sensible functions are included in this class (for example, every function $f(\ell)$ which is $\omega(1)$ as a function of ℓ is included). If the value of $f(\ell)$ is independent of ℓ then the function f may still be suitable, provided that $f(\ell)$ is big enough as a function of n. Details will be given later, but the rough requirement is that for some finite value Φ_f (which may depend upon n), and some $z = O(\log n)$ and $T = O(\log n)$, we may apply f to Φ_f z times and the resulting value is still at least $2T$. (That is, $f^z(\Phi_f) \geq 2T$.)

In Theorem 2 we provide upper bounds on the expected system load as well as corresponding tail bounds. The upper bounds are described in terms of Φ_f and n. For many natural work-stealing functions f, Φ_f is at most a polynomial in n, so the system-load bounds are polynomial in n. For example, Φ_f is at most a polynomial in n for the natural function $f(\ell) = \lfloor \ell/2 \rfloor$.

Finally, in Theorem 9 we classify some functions that do not result in a stable system. For example, the system is unstable if $f(\ell) < \lambda n$.

The proofs. Since the proofs are technical, we briefly introduce the underlying idea. We model our system by a discrete-time, countable Markov chain in which states are tuples giving the number of tasks currently allocated to each processor. The standard method for determining whether such a Markov chain is ergodic (i.e., whether it has a stationary distribution) is to find an appropriate Lyapounov function [Bor98, FMM95, MT93] (a potential function with an appropriate drift). Foster's theorem (see Theorem 2.2.3 of [FMM95]) shows that the chain is ergodic if and only if there is a positive Lyapounov function which is expected to decrease by a positive amount from every state except some finite subset of the state space. For many computer science applications, it is apparently prohibitively difficult to find such a one-step Lyapounov function, even when one is known to exist. Thus, multiple-step analysis is used [HLR96, GM99, AGM01]. We use the multiple-step extension of Foster's theorem due to Fayolle, Malyshev and Menshikov (see Lemma 10 below). The technical difficulty of our proof arises because of the lack of independence as the system evolves over multiple steps. To derive our results, we study the behaviour of a different and simpler Markov chain. The new Markov chain does not dominate the original chain forever, but we show that it does dominate the original chain for a sufficiently long period of time, and this enables us to prove that the original chain is ergodic. The proof of ergodicity, together with a martingale bound of [GBT01] gives us our bound on the stationary behaviour of the chain.

1.2 Known Results

Most known theoretical results on load-balancing are for unconditional algorithms (which perform load-balancing every few steps, regardless of the system state) or for sender-initiated approaches. First, there has been a lot of work on *static* problems, in which the number of jobs to be serviced is fixed and may even be known in advance. For these results, see [ABKU94, Vöc00, CS97, ACMR95, BCSV00].

In our paper, we work on *dynamic* load-balancing, in which tasks are generated over time. We will now describe previous work on this problem. In [ABS98], Adler, Berenbrink and Schröder consider a process in which m jobs arrive in each round to n servers and each server is allowed to remove one job per round. They introduce a simple algorithm in which each job chooses between 2 random servers. They show that, provided $m \leq n/6e$, no job is likely to wait more than $O(\log \log n)$ rounds. In [CFM$^+$98] the authors analyse several dynamic balls-into-bins games with deletion.

In [Mit96a], Mitzenmacher presents a new differential-equations approach for analysing both static and dynamic load-balancing strategies. He demonstrates the approach by providing an analysis of the supermarket model: jobs (customers) arrive as a Poisson stream of rate λn, $\lambda < 1$, at a collection of n servers. Each customer chooses d servers independently and uniformly at random, and waits for service at the one with the shortest queue. The service time of the customers is exponentially distributed with mean 1. Mitzenmacher achieves results on the expected time that a customer spends in the system. Furthermore, he shows that for any time interval of fixed length, the maximum system load is likely to be at most $\log \log n / \log d + O(1)$. In [VDK96] Vvedenskaya, Dobrushin and Karpelevich independently present similar results. For related results, see [Mit96b, MRS01, Mit97].

In [RSAU91], Rudolph, Slivkin-Allalouf and Upfal present a simple distributed load-balancing strategy. They consider a work-generation model in which, at every time step, the load change of any processor due to local generation and service is bounded by some constant. They show that the expected load of any processor at any point of time is within a constant factor of the average load. In [LM93], Lüling and Monien use a similar work-generation model. They show that the expected load difference between any two processors is bounded by a constant factor. They also bound the corresponding variance. In [BFM98, BFS99] the authors introduce and investigate the performance of different randomised load-balancing algorithms for randomised and adversarial work-generation schemes. With high probability, the algorithms balance the system load up to an additive term of $O(\log \log n)$.

There is relatively little existing theoretical work on receiver-initialised approaches. The interesting thing is that this approach seems to be highly efficient in practice (much more than, say, "give-away-if-overloaded"), but there are no (or hardly any) rigorous theoretical results.

In [Mit98], Mitzenmacher applies his differential-equations approach in order to analyse several randomised work-stealing algorithms in a dynamic setting. In contrast to our work, he assumes that every processor has a Poisson generating process with rate $\lambda < 1$. Hence, in contrast to our generation model, the load is generated in a more-or-less balanced fashion and the system is stable even without any work-stealing. Each task has an exponentially distributed service time with mean 1. He models a number of work-stealing algorithms with differential equations and compares the equations with each other in order to predict which strategies will be most successful. For each set of equations, he shows that the queue-lengths decrease more quickly than for a set of equations which models the process with no work stealing.

In [BL94], Blumofe and Leiserson analyse a scheduling strategy for *strict multi-threaded computations*. A multi-threaded computation consists of a set of threads, each of which is a sequential ordering of unit-time tasks. During a computation, a thread can spawn other threads which are stored in a local queue. They present a work-stealing algorithm in which every idle processor tries to steal a thread from a randomly chosen other processor. The analysis shows that the expected time to execute such a multi-threaded computation on P processors is $O(T_1/P + T_\infty)$, where T_1 denotes the minimum sequential execution time of the multi-threaded computation, and T_∞ denotes the minimum execution time with an infinite number of processors. Furthermore, they estimate the probability that the execution time is increased by an additional factor. In [FS96], Fatourou and Spirakis develop an algorithm for k-strict multi-threaded computations. In this case, all data dependencies of a thread are directed to ancestors in at most k higher levels.

2 The Work-Stealing Process

Suppose that we have n processors and n "generators", which create work for the processors. Each processor keeps a queue of jobs which need to be done. The evolution of the system can be described by a Markov chain X. The state X_t after step t is a tuple $(X_t(1), \ldots, X_t(n))$ in which $X_t(i)$ represents the length of the i'th processor's queue after step t. Initially, all of the queues are empty, so the start state is $(0, \ldots, 0)$. Figure 1 describes the transition from state X_t to state X_{t+1}. In the figure, N denotes $\{1, \ldots, n\}$. The parameter λ governs the rate at which jobs are generated – each generator creates a job during each step independently with probability λ and adds the job to the queue of its current host.

1. Each generator generates a new job independently with probability λ. It adds the job to the queue of its current host. Formally, let $h_t : N \to N$ be the function in which $h_t(i)$ is the host of the i'th generator. The i'th processor updates the size of its queue from $X_t(i)$ to $X'_t(i)$, where $X'_t(i)$ is defined to be $X_t(i)$ plus the sum of $|h_t^{-1}(i)|$ independent Bernoulli random variables with mean λ.

2. Each processor with an empty queue chooses a request destination uniformly at random (u.a.r.) from N. Formally, $r_t(i)$ is defined to be 0 if $X'_t(i) > 0$. Otherwise, $r_t(i)$ is chosen u.a.r. from N.

3. Each processor which receives a request chooses a recipient and allocates some of its load to give away to the recipient. Formally, we start by setting $j_t^+(i) = j_t^-(i) = 0$ for all $i \in N$. Then every $k \in N$ for which $r_t^{-1}(k)$ is non-empty chooses ℓ u.a.r. from $r_t^{-1}(k)$ and sets $j_t^+(\ell) = j_t^-(k) = f(X'_t(k))$.

4. Finally, the work is shared, and each queue processes one job. Formally, for all i, $X_{t+1}(i)$ is set to $\mathrm{Max}(0, X'_t(i) + j_t^+(i) - j_t^-(i) - 1)$.

Figure 1: The Markov chain X. The transition from X_t to X_{t+1}.

3 Upper bounds

In this section we prove that the system is stable for every suitable work-stealing function. Our analysis does not depend upon the assignment of generators to hosts. In the analysis, we assume that this assignment is made by an adversary at each step.

As already outlined in Section 1, the basic idea is the following. We define a Markov Chain X modelling our system. Since this chain is difficult to analyse directly, we introduce a second chain Z and investigate properties of Z instead. Then, using a coupling, we relate the results to chain X itself.

To put it *very* informally and non-rigorously, the core idea is to show that during an interval of length $T = O(\log n)$ not too many requests are sent. Since in our model not sending a request means servicing a task, we can show that in this case the system load decreases. Obviously, the crux is bounding the number of requests during the interval. Informally, this is done by assuming (for contradiction) that there are many requests during the interval, say at least R. Since the system load is initially high, there is at least one processor, processor P, which initially has a high load. This implies that after around $R' < R$ requests, we can view most of the requests that have been accepted in a tree with P at the root, and the leaves being processors that either directly or indirectly received a portion of P's initial load. By showing that (i) there are many leaves, and (ii) the tree does not grow very deep, we can conclude that after R' requests, there are many processors having a large load (at least T), and none of them will send a request during the next T steps. Hence, we can contradict the assumption that R requests get sent during the interval. Of course, this kind of proof-by-contradiction is invalid if we want to avoid conditioning the random variables during the T steps, so we have to do things more carefully.

In order to define suitable work-stealing functions f and to state our theorems precisely, we need some definitions. We start with some brief definitions regarding Markov chains. For more details, see [GS92]. The Markov chains that we will consider are *irreducible* (every state is reachable from every other) and *aperiodic* (the gcd of the lengths of valid paths from state i to itself is 1). An irreducible aperiodic Markov chain (Υ_t) is said to be *recurrent* if, with probability 1, it returns to its start state. That is, it is recurrent if

$$\Pr(\Upsilon_t = \Upsilon_0 \text{ for some } t \geq 1) = 1.$$

Otherwise, it is said to be *transient.* It is said to be *positive recurrent* or *ergodic* if the expected time that it takes to return to the start state is finite. In particular, let

$$T_{\text{ret}} = \min\{t \geq 1 \mid \Upsilon_t = \Upsilon_0\}.$$

The chain is said to be positive recurrent if $E[T_{\text{ret}}] < \infty$. A positive recurrent chain has a unique stationary distribution π. When we analyse the Markov Chain X we will use Lemma 10 from the appendix. The Markov chain X satisfies the initial conditions of the lemma. That is, it is time-homogeneous, irreducible, and aperiodic and has a countable state space.

Definitions. We will assume that for a positive constant δ, the arrival rate λ is at most $1-\delta$. Let c be a constant which is sufficiently large with respect to δ^{-1} (see the proof of Lemma 4) and let $\alpha = 4e(c+1)$. Let ϵ be $(1-\lambda/(1-\nu))/4$. Let $\nu = n^{-2} + n^{-\alpha}$. Let $z = \lceil \alpha \lg n \rceil$ and let $T = \lceil \frac{2c}{1-\lambda/(1-\nu)} \lg n \rceil$. Let g be the function given by $g(x) = f(x-T)$. Let Φ_f be the smallest integer such that $g^z(\Phi_f/n) \geq 2T$. If no such integer exists, say $\Phi_f = \infty$. Then our only restriction on the work-stealing function f is that Φ_f has to be finite.We can assume without loss of generality that $0 \leq f(x) \leq x/2$.[1] We also assume that $f(x)$ is monotonically non-decreasing in x. $\Phi(X_t)$ is the system load after step t. That is, $\Phi(X_t) = \sum_{i=1}^{n} X_t(i)$.

[1]This assumption is without loss of generality because the names of the processors are irrelevant. If the kth processor gives the ℓth processor more than half of its jobs, we can simply swap the names of the two processors, referring to them as the ℓth and the kth, respectively.

We will prove the following theorems.

Theorem 1 *Let δ be a positive constant and λ an arrival rate which is at most $1-\delta$. Let f be a suitable work-stealing function satisfying the conditions above. Then for every n which is sufficiently large with respect to δ^{-1}, the Markov chain X is positive recurrent.*

Theorem 1 guarantees that the Markov chain X has a stationary distribution π. The next theorem is concerned with the value of the total system load in the stationary distribution.

Theorem 2 *Let δ be a positive constant and λ an arrival rate which is at most $1-\delta$. Let f be a work-stealing function satisfying the conditions above. Then for every n which is sufficiently large with respect to δ^{-1},*

$$E_\pi[\Phi(X_t)] \leq \Phi_f + 2n^2T/\epsilon + nT,$$

and for any non-negative integer m,

$$\Pr_\pi[\Phi(X_t) > \Phi_f + 2nTm + nT] \leq \exp(-(m+1)\epsilon/(n+\epsilon)).$$

3.1 A Simpler Markov chain

Let C be the set of states x with $\Phi(x) < \Phi_f$. In this section we will define a simper Markov chain Z that will be used in order to analyse the Markov chain X with a start state $x \notin C$.

The state space of Z is more complicated than the state space of X, but in some sense the information contained in a state of Z is less precise than the information contained in a state of X. In particular, a state Z_t consists of a tuple $(L_t(1), \ldots, L_t(n), Y_t(1), \ldots, Y_t(n))$. The variable $L_t(i)$ is an indicator of how "loaded" the i'th processor is after step t. If $L_t(i) = 0$ (signalling "not loaded") then $Y_t(i)$ denotes the length of the i'th processor's queue after step t. Otherwise, $Y_t(i) = 0$ and $L_t(i)$ gives a crude measure of the extent to which the i'th processor is loaded. (More detail about the meaning of $L_t(i)$ will be given below.)

We will be observing the evolution of the Markov chain X starting at a state $X_0 = x$ with $\Phi(x) \geq \Phi_f$. This condition guarantees that for some $i \in N$, $X_0(i) \geq \Phi_f/n$. Let J_x be the smallest such i. In the following, we will be paying special attention to load which originates at processor J_x. Thus, in the Markov chain Z, the state Z_0 which corresponds to X_0 is defined follows. $L_0(J_x) = 2^z$ and $Y_0(J_x) = 0$. For all $i \neq J_x$, $L_0(i) = 0$ and $Y_0(i) = X_0(i)$. For convenience, we will say that a processor i is "heavily loaded" in state Z_t if and only if $L_t(i) > 0$. Thus, J_x is the only processor which is deemed to be "heavily loaded" in Z_0. Note that the state Z_0 is strictly a function of x. We will refer to this state as $Z(x)$. The transition from Z_t to Z_{t+1} is described in Figure 2. It may look surprising at first that the "heavy load" parameter $L_t(k)$ is halved every time a heavily loaded processor transfers load. This halving allows us to study the dissemination of load from J_x without considering the many dependent events.

Let R'_t be the set of requests made during the transition from Z_t to Z_{t+1}. (This transition is referred to as "step $t+1$".) That is, $R'_t = |\{i \mid r_t(i) > 0\}|$. Let τ' be the smallest integer such that $R'_0 + \cdots + R'_{\tau'-1} \geq cn \lg n$. Let Ψ be the smallest integer such that, for some i, $L_\Psi(i) = 1$. Intuitively, $L_\Psi(i) = 1$ means that i has received load (directly or indirectly) from J_x (so it is "heavily loaded") but this load has been split many times (it has been split z times, in fact). The following lemma shows that, with high probability, there are no such i and Ψ if at most $cn \log n$ requests are sent.

Lemma 3 *Suppose $x \notin C$. Run Markov chain Z starting at $Z_0 = Z(x)$. Then*

$$\Pr(\Psi \leq \tau') \leq n^{-\alpha}.$$

1. Let $h_t : N \to N$ be the function in which $h_t(i)$ is the host of the i'th generator. If $L_t(i) > 0$ then $Y'_t(i)$ is defined to be 0 (just like $Y_t(i)$). Otherwise, $Y'_t(i)$ is defined to be $Y_t(i)$ plus the sum of $|h_t^{-1}(i)|$ Bernoulli random variables with mean λ.
2. $r_t(i)$ is defined to be 0 *except* when $L_t(i) = 0$ and $Y'_t(i) = 0$. In this case, $r_t(i)$ is chosen u.a.r. from N.
3. Start by setting
$$j_t^+(i) = j_t^-(i) = l_t^-(i) = l_t^+(i) = 0,$$
for each $i \in N$. Then every $k \in N$ for which $r_t^{-1}(k)$ is non-empty chooses ℓ u.a.r. from $r_t^{-1}(k)$ and sets $l_t^+(\ell) = l_t^-(k) = L_t(k)/2$ and $j_t^+(\ell) = j_t^-(k) = f(Y'_t(k))$.
4. For all $i \in N$, $L_{t+1}(i)$ is set to $L_t(i) + l_t^+(i) - l_t^-(i)$. If $L_{t+1}(i) > 0$ then $Y_{t+1}(i) = 0$. Otherwise, $Y_{t+1}(i)$ is set to $\mathrm{Max}(0, Y'_t(i) + j_t^+(i) - j_t^-(i) - 1)$.

Figure 2: The transition from Z_t to Z_{t+1}.

Proof: Since at most n requests are sent in a single step, the total number of requests sent during steps $1, \ldots, \tau'$ is at most $(c+1)n \lg n$.

Imagine that the value $L_0(J_x) = 2^z$ corresponds to 2^z tokens which initially sit at processor J_x. If $L_t(k) > 0$ then the instruction $l_t^+(\ell) = l_t^-(k) = L_t(k)/2$ in Step 3 of the transition from Z_t to Z_{t+1} transfers half of processor k's tokens to processor ℓ. (Eventually, tokens are sub-divided rather than transferred, but this will not concern us here.) The event $\Psi \le \tau'$ occurs iff some token is transferred z times during steps $1, \ldots, \tau'$.

What is the probability that a given token is transferred z times? This is at most
$$\binom{(c+1)n \lg n}{z} n^{-z} \le \left(\frac{e(c+1)\lg n}{z}\right)^z.$$
The probability that there exists a token which is transferred z times is thus at most
$$\left(\frac{2e(c+1)\lg n}{z}\right)^z \le \left(\frac{2e(c+1)}{\alpha}\right)^{\alpha \lg n} = n^{-\alpha}.$$
$\square$

The next lemma shows that, with high probability, the number of requests sent during the observed T time steps is less than $cn \log n$. This means that we have very little idle time during this period, which in turn implies the decrease of the system load (as we will see later).

Lemma 4 *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$.*
$$\Pr(\tau' \le T) \le n^{-2}.$$

Proof: Recall that R'_t is the number of requests during the transition from Z_t to Z_{t+1}. In particular, $R'_t = |\{i \mid L_t(i) = 0 \wedge Y'_t(i) = 0\}|$. R'_t is a random variable which depends only upon the state Z_t and upon the host-distribution function h_t.

To make the conditioning clear, we will let $R'(s,h)$ be the random variable whose distribution is the same as that of R'_t, conditioned on $Z_t = s$ and $h_t = h$.

By a Chernoff bound (see [McD89]),

$$\Pr(|R'(s,h) - E(R'(s,h)| \geq \frac{cn\lg n}{8T}) \leq 2\exp\left(-2\left(\frac{cn\lg n}{8T}\right)^2/n\right).$$

Let σ denote $2\exp\left(-2\left(\frac{cn\lg n}{8T}\right)^2/n\right)$. Note that σ is exponentially small in n. (This follows from the definition of T.)

Say that (s,h) is "dangerous" if $E(R'(s,h)) \geq (cn\lg n)/(4T)$. Note that if (Z_t, h_t) is not dangerous, then, with probability at least $1-\sigma$, the number of requests during the transition from Z_t to Z_{t+1} is at most $(cn\lg n)/(4T) + (cn\lg n)/(8T) \leq (cn\lg n)/(2T)$.

Now suppose that (s,h) is dangerous and let k be any processor. Then

$$\begin{aligned}\Pr(r_t^{-1}(k) = \emptyset \mid Z_t = s \wedge h_t = h) &\leq \sigma + \left(1 - \frac{1}{n}\right)^{\frac{cn\lg n}{8T}} \\ &\leq \sigma + \left(1 - \frac{1}{n}\right)^{\frac{\delta n}{18}} \\ &\leq 1 - \gamma,\end{aligned}$$

for a small positive constant γ which depends upon δ (but not upon c or n). Let M_t denote the number of heavily loaded processors,

$$M_t = |\{i \mid L_t(i) > 0\}|.$$

If (s,h) is dangerous then

$$E[\,|\{k \mid L_t(k) > 0 \wedge r_t^{-1}(k) = \emptyset\}| \mid Z_t = s \wedge h_t = h] \leq (1-\gamma)M_t.$$

Thus by Markov's inequality,

$$\Pr(\,|\{k \mid L_t(k) > 0 \wedge r_t^{-1}(k) = \emptyset\}| \geq (1-\gamma/2)M_t \mid Z_t = s \wedge h_t = h) \leq 1 - \frac{\gamma}{2-\gamma}.$$

If $|\{k \mid L_t(k) > 0 \wedge r_t^{-1}(k) = \emptyset\}| < (1-\gamma/2)M_t$ then at least $(\gamma/2)M_t$ of the M_t heavily loaded processors give away work, so $M_{t+1} \geq (1+\gamma/2)M_t$. Thus, if $Z_t = s$ and $h_t = h$ and (s,h) is dangerous then

$$\Pr(M_{t+1} \geq (1+\gamma/2)M_t) \geq \frac{\gamma}{2-\gamma}.$$

Also, if $M_{t+1} \geq (1+\gamma/2)M_t$ occurs at least $\log_{1+\gamma/2} n$ times during the process, then we have $M_t = n$, so there are no more dangerous steps. Using a Chernoff bound (and domination) the probability that there are more than $2\left(\frac{2-\gamma}{\gamma}\right)\log_{1+\gamma/2} n$ dangerous steps (ever) is at most $\exp(-\log_{1+\gamma/2}(n)/4)$. If we make c sufficiently large with respect to γ then

$$2\left(\frac{2-\gamma}{\gamma}\right)\log_{1+\gamma/2} n < c\lg n/2.$$

Now we have that, except for probability $\exp(-\log_{1+\gamma/2} n/4)$, the dangerous steps contribute fewer than $cn\lg n/2$ requests (ever). Furthermore, except for probability at most σT, the un-dangerous steps contribute at most $cn\lg n/2$ requests during the first T steps. Thus, the probability that τ' occurs is at most $\exp(-\log_{1+\gamma/2} n/4) + \sigma T \leq n^{-2}$. □

Lemmas 3 and 4 imply the following.

Corollary 5 *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$.*

$$\Pr(T < \tau' < \Psi) \geq 1 - n^{-\alpha} - n^{-2}.$$

Let R_t be the set of requests made during the transition from X_t to X_{t+1}. That is, $R_t = |\{i \mid r_t(i) > 0\}|$. Let τ be the smallest integer such that $R_0 + \cdots + R_{\tau-1} \geq cn \lg n$. Now suppose $x \in C$ and consider running Z_t from $Z_0 = Z(x)$ for T steps. As long as τ' and Ψ do not occur, we can couple this with a run of (X_t) from $X_0 = x$ in such a way that exactly the same requests occur in both copies. The coupling is the straightforward one. A processor which is not heavily loaded makes all of the same choices in both copies. At some point a processor may become heavily loaded in the Z process, so it will never send more requests. In the X process, it will instead receive a certain amount of work which (since J_x's queue is initially large) is enough to sustain it (and the processors that it later passes work to) for up to z work transfers in all. Since Ψ does not occur, fewer than z transfers occur, so the processor makes no further requests (even in the X process). We conclude that the same requests occur in both copies. In particular, since τ' does not occur, neither does τ. This implies Lemma 6.

3.2 Proof of Theorem 1

Lemma 6 *If $x \notin C$ then* $\Pr(\tau \leq T \mid X_0 = x) \leq \nu$.

The next lemma shows that the load has an appropriate drift when $\tau > T$.

Lemma 7 *If $x \notin C$ then* $E[\Phi(X_T) \mid (X_0 = x) \wedge \tau > T] \leq \Phi(x) - 2\epsilon T$.

Proof: Let $Y = R_0 + \cdots + R_{T-1}$. Then using Lemma 6,

$$\begin{aligned} & E[Y \mid (X_0 = x) \wedge \tau > T] \\ = \; & \frac{E[Y \mid X_0 = x] - \Pr(\tau \leq T \mid X_0 = x) E[Y \mid (X_0 = x) \wedge \tau \leq T]}{\Pr(\tau > T \mid X_0 = x)} \\ \leq \; & \frac{\lambda n T}{1 - \nu}. \end{aligned}$$

If $\tau > T$ then the number of jobs serviced during steps 1–T is at least $nT - cn \lg n$. Thus,

$$\begin{aligned} E[\Phi(X_T) \mid (X_0 = x) \wedge \tau > T] &\leq \Phi(x) - (1 - \frac{\lambda}{1-\nu}) nT + cn \lg n \\ &\leq \Phi(x) - \frac{1 - \frac{\lambda}{1-\nu}}{2} nT \\ &= \Phi(x) - 2\epsilon nT. \end{aligned}$$

□

Lemma 8 *If $x \notin C$ then* $E[\Phi(X_T) \mid X_0 = x] \leq \Phi(x) - \epsilon T$.

Proof:

$$\begin{aligned} E[\Phi(X_T) \mid X_0 = x] &= \Pr(\tau > T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau > T] \\ &+ \Pr(\tau \leq T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau \leq T]. \end{aligned}$$

Now we use Lemma 7 to show that the first term on the right-hand side is at most $\Phi(x) - 2\epsilon nT$. Use Lemma 6 to show that the second term on the right-hand side is at most νnT. □

Combining Lemma 8 and Lemma 10, we get a proof of Theorem 1.

3.3 Proof of Theorem 2

Let $W_i = \Phi(X_{iT})$ for $i \in \{0, 1, 2, \ldots\}$. Lemma 8 shows that the process $W_0, W_1, \ldots$ behaves like a supermartingale above Φ_f. That is, it satisfies Equation 6 with $\xi = \epsilon T$ and $\beta = \Phi_f$. In itself, this does not imply that $E[W_t]$ is unbounded (see Pemantle and Rosenthal's paper [PR99] for counter-examples). However, we also have

$$|W_{t+1} - W_t| \leq nT \tag{1}$$

for any t. That is, Equation 7 is satisfied with $\nu_{\max} = nT$. This implies (for example, by Theorem 1 of [PR99] or by Theorem 2.3 of [Haj82]) that $E_\pi[W_t]$ is finite (so Equation 9 is satisfied). Lemma 12 can now be applied with $p_{\max} = 1$ to get

$$E_\pi[W_t] \leq \Phi_f + 2n^2T/\epsilon, \tag{2}$$

and for any non-negative integer m,

$$\Pr_\pi[W_t > \Phi_f + 2nTm] \leq \left(\frac{nT}{nT + \epsilon T}\right)^{(m+1)}. \tag{3}$$

The theorem now follows from the observation that $n/(n + \epsilon) \leq \exp(-\epsilon/(n + \epsilon))$ and the observation that for $0 \leq j < T$, $\Phi(X_{iT+j}) \leq \Phi(X_{iT}) + nj$.

4 Lower Bounds

In this section we give the straightforward lower bound which shows that the system is not stable for unsuitable work-stealing functions. Of course we have to restrict the distributions of the generators in order to obtain instability. For example, if every processor has exactly one generator, then the system will be stable even without any work stealing. If we use $k = 1$ in the statement of the following theorem, we find that the system is unstable if $f(\ell) < \lambda n$.

Theorem 9 *Let δ be a positive constant and λ an arrival rate which is at most $1 - \delta$. Suppose the n generators are equally distributed among some set of k processors. Suppose that for all ℓ, $f(\ell) \leq j(n)$. Then the Markov chain X is transient if*

$$k \cdot (j(n) + 1) < \lambda n.$$

Proof: This theorem can be proven easily using Lemma 11. First, we bound the amount of work that can be done during any given step. When a processor steals work, it only gets enough work for at most $j(n)$ rounds. Since each processor only gives work to one other processor per round, at most $j(n)k$ processors without generators have work to do during any given step. Thus, at most $(j(n) + 1)k$ tasks can be done during any step. The expected load increase of the system during a step is λn. Using Lemma 11 it is easy to see that the system is transient if $k(j(n) + 1) < \lambda n$. (Let Φ be the system load.) $\square$

5 Conclusions

We have analysed a very simple work-stealing algorithm, which is successfully being used in practical applications. However, so far a rigorous analysis was still missing. In this paper we have analysed its performance for a wide range of parameters. We have shown that it is stable for any constant generation rate $\lambda < 1$ and a wide class of work-stealing functions f. On the other hand, we have also shown that it cannot be stable for any $\lambda > 0$ however small, given a certain (relatively small) class of "unsuitable" functions. Finally, we have derived upper bounds on the complete system load in the case of a stable system.

There are a few open questions related to this algorithm and its analysis, respectively. Firstly, can one close the gap between "stable" work-stealing functions and "unstable" ones? Secondly, experimental results seem to indicate that the complete system load is considerably smaller than our upper bound (although in simulations, one cannot always easily tell a constant from, say, a $\log(n)$). We believe it worthwhile to further investigate this matter.

Acknowledgements

We thank Hesham Al-Ammal for useful discussions.

References

[ABKU94] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations (extended abstract). In *Proceedings of the 26th Symposium on Theory of Computing (STOC'94)*, pages 593–602, 1994.

[ABS98] Micah Adler, Petra Berenbrink, and Klaus Schröder. Analyzing an infinite parallel job allocation process. In *Proceedings of the 6th European Symposium on Algorithms (ESA'98)*, pages 417–428, 1998.

[ACMR95] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th Symposium on Theory of Computing (STOC'95)*, pages 234–247, 1995.

[AGM01] H. Al-Ammal, L.A. Goldberg, and P. MacKenzie. An improved stability bound for binary exponential backoff. *Theory of Computing Systems*, 2001. To appear.

[BCSV00] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. In *Proceedings of the 32th ACM Symposium on Theory of Computing (STOC'00)*, 2000.

[BFM98] Petra Berenbrink, Tom Friedetzky, and Ernst W. Mayr. Parallel continuous randomized load balancing. In *Proceedings of the 10th Symposium on Parallel Algorithms and Architectures (SPAA '98)*, pages 192–201. ACM, 1998.

[BFS99] Petra Berenbrink, Tom Friedetzky, and Angelika Steger. Randomized and adversarial load balancing. In *Proceedings of the 11th Symposium on Parallel Algorithms and Architectures (SPAA'99)*, pages 175–184, 1999.

[BJK$^+$96] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, 1996.

[BL94] Robert Blumofe and Charles Leiserson. Scheduling multithreaded computations by work stealing. In *Proceedings of the 37th Symphosium on Foundations of Computer Science (FOCS'94)*, pages 362–371, 1994.

[Bor98] A.A. Borovkov. *Ergodicity and Stability of Stochastic Processes*. John Wiley and Sons, 1998.

[CFM$^+$98] Richard Cole, Alan Frieze, Bruce Maggs, Michael Mitzenmacher, Andrea Richa, Ramesh Sitaraman, and Eli Upfal. On balls and bins with deletions. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98)*, pages 145–158, 1998.

[CS97] Artur Czumaj and Volker Stemann. Randomized allocation processes. In *Proceedings of the 38th Symposium on Foundations on Computer Science (FOCS'97)*, pages 194–203, 1997.

[Dec00] T. Decker. Virtual Data Space – Load balancing for irregular applications. *Parallel Computing*, 26(13-14):1825–1860, Nov. 2000.

[Fel94] B. Monien Feldmann, P. Mysliwietz. Studying overheads in massively parallel min/max-tree evaluation. In *Proceedings of the 5th Symposium on Parallel Algorithms and Architectures*, pages 94–103, 1994.

[FMM95] G. Fayolle, V.A. Malyshev, and M.V. Menshikov. *Topics in the Constructive Theory of Countable Markov Chains*. Cambridge University Press, 1995.

[FS96] Panagiota Fatourou and Paul Spirakis. Scheduling algorithms for strict multithreaded computations. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'96)*, pages 407–416, 1996.

[GBT01] D. Gamarnik, D. Bertsimas, and J. N. Tsitsiklis. Performance of multiclass markovian queueing networks via piecewise linear Lyapunov functions. In *Annals of Applied Probability*, 2001. To appear.

[GM99] L.A. Goldberg and P.D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. *Journal of Computer and Systems Sciences*, 58:232–258, 1999.

[GS92] G.R. Grimmet and D.R. Stirzaker. *Probability and Random Processes, Second Edition*. Oxford University Press, 1992.

[Haj82] B. Hajek. Hitting time and occupation time bounds implied by drift analysis with applications. In *Adv. Appl. Probab.*, pages 14:502–525, 1982.

[HLR96] J. Håstad, T. Leighton, and B. Rogoff. Estimating the multiplicities of conflicts to speed their resolution in multiple access channels. *SIAM J. on Computing*, 25(4):740–774, 1996.

[LM93] Reinhard Lüling and Burkhard Monien. A dynamic distributed load balancing algorithm with provable good performance. In *Proceedings of the 5th Symposium on Parallel Algorithms and Architectures (SPAA'93)*, pages 164–172, 1993.

[McD89] C. McDiarmid. On the method of bounded differences. In *Surveys in Combinatorics, London Math. Soc. Lecture Notes Series 141*, pages 148–188. Cambridge University Press, 1989.

[MD98] N. R. Mahapatra and S. Dutt. Adaptive Quality Equalizing: High-performance load balancing for parallel branch and bound across applications and computing systems. In *Proc. 12th IPPS*, 1998. To appear.

[Mit96a] Michael Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96)*, pages 213–222, 1996.

[Mit96b] Michael Mitzenmacher. *The Power of Two Random Choices in Randomized Load Balancing*. PhD thesis, Graduate Division of the University of California at Berkley, 1996.

[Mit97] Michael Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pages 292–301, 1997.

[Mit98] Michael Mitzenmacher. Analysis of load stealing models based on differential equations. In *Proceedings of the 10th Symphosium on Parallel Algorithms and Architectures (SPAA'98)*, pages 212–221, 1998.

[MRS01] Michael Mitzenmacher, Andrea Richa, and Ramesh Sitaraman. The power of two randomized choices: A survey of techniques and results. In *Handbook of Randomized Computing*. Kluwer Press, 2001. (Submitted for inclusion.).

[MT93] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993.

[PR99] Robin Pemantle and Jeffrey S. Rosenthal. Moment conditions for a sequence with negative drift to be uniformly bounded in l^r. In *Stoch. Proc. Appl.*, pages 82:143–155, 1999.

[RSAU91] Larry Rudolph, Miriam Slivkin-Allalouf, and Eli Upfal. A simple load balancing scheme for task allocation in parallel machines. In *Proceedings of the 3rd Symposium on Parallel Algorithms and Architectures (SPAA '91)*, pages 237–245, 1991.

[VDK96] 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(1):15–27, 1996.

[Vöc00] Berthold Vöcking. How asymmetry helps load balancing. In *Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS'00)*, pages 131–140, 2000.

Appendix

A Useful Lemmas

The proof of Theorem 1 uses the following generalisation of Foster's theorem, due to Fayolle, Malyshev and Menshikov (Theorem 2.2.4 of [FMM95]).

Lemma 10 (Foster; Fayolle, Malyshev, Menshikov) *A time-homogeneous irreducible aperiodic Markov chain ζ with a countable state space Ω is positive recurrent iff there exists a positive function $\Phi(x)$, $x \in \Omega$, a number $\xi > 0$, a positive integer-valued function $\beta(x)$, $x \in \Omega$, and a finite set $C' \subseteq \Omega$, such that the following inequalities hold.*

$$\begin{aligned} E[\Phi(\zeta_{t+\beta(\zeta_t)}) - \Phi(\zeta_t) \mid \zeta_t = x] &\leq -\xi\beta(x), x \notin C' & (4) \\ E[\Phi(\zeta_{t+\beta(\zeta_t)}) \mid \zeta_t = x)] &< \infty, x \in C'. & (5) \end{aligned}$$

The proof of Theorem 9 uses the following theorem, which is Theorem 2.2.7 of [FMM95].

Lemma 11 (Fayolle, Malyshev, Menshikov) *An irreducible aperiodic time-homogeneous Markov chain ζ with countable state space Ω is transient if there is a positive function Φ with domain Ω and there are positive constants C, d and ξ such that*

1. *there is a state x with $\Phi(x) > C$, and a state x with $\Phi(x) \leq C$, and*
2. *$E[\Phi(\zeta_1) - \Phi(\zeta_0) \mid \zeta_0 = x] \geq \xi$ for all x with $\Phi(x) > C$, and*
3. *if $|\Phi(x) - \Phi(y)| > d$ then the probability of moving from x to y in a single move is* 0.

The proof of Theorem 2 uses the following theorem, which is Theorem 1 of [GBT01].

Lemma 12 *[Bertsimas, Gamarnik, Tsitsiklis] Consider a time-homogeneous Markov chain ζ with a countable state space Ω and stationary distribution π'. If there is a positive function $\Phi(x)$, $x \in \Omega$, a number $\xi > 0$, and a number $\beta \geq 0$ such that*

$$E[\Phi(\zeta_{t+1}) - \Phi(\zeta_t) \mid \zeta_t = x] \leq -\xi, \Phi(x) > \beta, \tag{6}$$

and

$$|\Phi(\zeta_{t+1}) - \Phi(\zeta_t)| \leq \nu_{\max}, \tag{7}$$

and, for any x,

$$\Pr[\Phi(\zeta_{t+1}) > \Phi(\zeta_t) \mid \zeta_t = x] \leq p_{\max}, \tag{8}$$

and

$$E_{\pi'}[\Phi(\zeta_t)] < \infty, \tag{9}$$

then for any non-negative integer m,

$$\Pr_{\pi'}[\Phi(\zeta_t) > \beta + 2\nu_{\max} m] \leq \left(\frac{p_{\max}\nu_{\max}}{p_{\max}\nu_{\max} + \xi}\right)^{m+1},$$

and

$$E_{\pi'}[\Phi(\zeta_t)] \leq \beta + \frac{2p_{\max}(\nu_{\max})^2}{\xi}.$$