SIAM J. COMPUT. © 2003 Society for Industrial and Applied Mathematics
Vol. 32, No. 5, pp. 1260–1279

THE NATURAL WORK-STEALING ALGORITHM IS STABLE*

PETRA BERENBRINK†, TOM FRIEDETZKY†, AND LESLIE ANN GOLDBERG‡

Abstract. In this paper we analyze a very simple dynamic work-stealing algorithm. In the work-generation model, there are n (work) *generators*. A *generator-allocation function* is simply a function from the n generators to the n processors. We consider a fixed, but arbitrary, distribution $\mathcal{D}$ over generator-allocation functions. During each time step of our process, a generator-allocation function h is chosen from $\mathcal{D}$, and the generators are allocated to the processors according to h. Each generator may then 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. For many choices of $\mathcal{D}$, the work-generation model allows the load to become arbitrarily imbalanced, even when $\lambda < 1$. For example, $\mathcal{D}$ could be the point distribution containing a single function h which allocates all of the generators to just one processor. For this choice of $\mathcal{D}$, the chosen processor receives around λn units of work at each step and services one. The natural work-stealing algorithm that we analyze 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 *nonempty* processor having received at least one such request in turn decides (again randomly) in favor of one of the requests. The number of tasks which are transferred from the nonempty 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 analyze the *long-term behavior* 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 monotonically nondecreasing function f which satisfies $0 \le f(\ell) \le \ell/2$ and $f(\ell) = \omega(1)$ as a function of ℓ is included). 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.

Key words. work-stealing, dynamic, stability

AMS subject classification. 60J20

DOI. 10.1137/S0097539701399551

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 analyzing load balancing, most of which concentrates on sender-initiated approaches or related

*Received by the editors December 12, 2001; accepted for publication (in revised form) February 13, 2003; published electronically August 6, 2003. This work was partially supported by EPSRC grant GR/M96940 "Sharper Analysis of Randomised Algorithms: A Computational Approach" and the Future and Emerging Technologies Programme of the EU under contracts IST-1999-14186 (ALCOM-FT) and IST-1999-14036 (RAND-APX). A preliminary version of this article was published in the *Proceedings of the* 42nd *Annual Symposium on Foundations of Computer Science* (FOCS'01), IEEE Computer Society Press, Los Alamitos, CA, 2001.

http://www.siam.org/journals/sicomp/32-5/39955.html

†School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (petra@cs.sfu.ca, tkf@cs.sfu.ca).

‡Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (leslie@dcs.warwick.ac.uk).

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 success of Leiserson et al.'s language `Cilk` [10], a language for multithreaded parallel programming which uses work-stealing in its kernel. There are numerous examples of practical applications of work-stealing. In [17], Feldmann, Mysliwietz, and Monien investigate the behavior of parallel MIN/MAX-tree evaluation in the context of parallel game (chess) programs employing work-stealing strategies. In [14], Decker introduces VDS (virtual data space), a load-balancing system for irregular applications that makes use of work-stealing and other strategies. In [23], Mahapatra and Dutt use work-stealing for parallel branch and bound algorithms.

Despite the practical usefulness of work-stealing, there are not many known theoretical results about its performance. Most existing theoretical work on load balancing assumes a rather well-behaved system. For instance, most work on sender-initiated load balancing uses a work-generation model in which each processor generates at most a constant number of new tasks per step. In balls-into-bins games, each ball (task) chooses its bin (processor) uniformly at random (u.a.r.), which also yields a relatively balanced system.

In this paper we analyze 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 has an expected increase of $\lambda n - 1$ 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 *nonempty* processor having received at least one such request selects one of them randomly. 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 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 (provided f is monotonically nondecreasing and satisfies $0 \leq f(\ell) \leq \ell/2$). This result still holds if we put an upper bound on the amount of work that can be "stolen" by a single request. That is, for an upper bound h_z which is independent of ℓ (but depends on n) and will be defined in Lemma 2, the work-stealing function defined by $f'(\ell) = \min(f(\ell), f(h_z))$ is also stable. The value h_z depends upon the function f, but it need not be very large. For the function $f(\ell) = \lfloor \ell/2 \rfloor$, the value h_z is bounded from above by a polynomial in n. See section 3 for details. Our stability results are complemented by a straightforward lower bound: The system is unstable if $f(\ell)$ is too small, for example, if $f(\ell) < \lambda n - 1$. 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 introduce our model and the work-stealing algorithm.

The model. We start with a collection of n synchronized *processors*, connected by some network topology. During each time step, every processor may send a request (requesting extra work) to at most one other processor, and any processor receiving more than one such request accepts at most one of them.

In our model, we have n *generators.* A *generator-allocation function* is a function from the n generators to the n processors. We consider a fixed, but arbitrary, distribution $\mathcal{D}$ over generator-allocation functions. During each time step of our process, a generator-allocation function h is chosen from $\mathcal{D}$, and the generators are allocated to the processors according to h. Each generator may then generate a unit-time task which it inserts into the queue of its host processor. It generates such a task independently with probability $\lambda \in [0, 1]$. After the new tasks are generated, each processor removes one task from its queue and services it. We assume constant service time for all tasks.

In the absence of a load-balancing mechanism, many choices of $\mathcal{D}$ allow the load to become arbitrarily imbalanced, even when $\lambda < 1$.

The algorithm. The work-stealing algorithm is very simple and natural. During each time step, each *empty* processor (with no work to do) sends a request to one other processor, which is chosen independently and u.a.r. Each *nonempty* processor that received at least one request selects one of these independently and u.a.r. Then each empty processor whose request was accepted "steals" tasks from the other processor.

The number of tasks which are transferred from the nonempty 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 one of the most desirable features of load-balancing algorithms.

In Theorem 10 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 work-stealing functions are suitable (for example, every monotonically nondecreasing function $f(\ell)$ which is $\omega(1)$ as a function of ℓ and satisfies $0 \le f(\ell) \le \ell/2$ is suitable). The rough requirement (for f to be suitable) is that for some finite value Φ_f (which may depend upon n) and some $z = O(\log n)$ and $T = \Theta(\log n)$, we may apply f to Φ_f z times and the resulting value is still at least $2T$. (That is, $f^z(\Phi_f) \ge 2T$.) Our stability result still holds if we put an appropriate upper bound on the amount of work that can be stolen by a single request. Details are given in section 3.

In Theorem 12 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 work-stealing function $f(\ell) = \lfloor \ell/2 \rfloor$.

Finally, in Theorem 22, we classify some work-stealing functions that do not result in a stable system. For example, the system is unstable if $f(\ell) < \lambda n - 1$.

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 Lyapunov function [11, 16, 25] (a potential function with an appropriate drift). Foster's theorem (see Theorem 2.2.3 of [16]) shows that the chain is ergodic if and only if there is a positive Lyapunov 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 Lyapunov function, even when one is known to exist. Thus, multiple-step analysis is used [21, 18, 3]. We use the multiple-step extension of Foster's theorem due to Fayolle, Malyshev, and Menshikov (see Lemma 7). 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 behavior 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 [8], gives us our bound on the stationary behavior 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 [4, 34, 13, 2, 5].

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 [1], 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 [12] the authors analyze several dynamic balls-into-bins games with deletion.

In [26], Mitzenmacher presents a new differential-equations approach for analyzing 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 u.a.r. 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 [35] Vvedenskaya, Dobrushin, and Karpelevich independently present similar results. For related results, see [27, 30, 28].

In [33], 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. The balancing strategy works as follows. Each processor stores its tasks in a local queue. Whenever a processor accesses its local queue, the processor performs a balancing operation with a probability inversely proportional to the size of its queue. The balancing operation examines the queue size of a randomly chosen processor and then equalizes their load. They show that the expected load of any processor at any point of time is within a constant factor of the average load.

In [22], Lüling and Monien use a similar work-generation model. A processor initiates a load-balancing action if its load has changed by a constant factor since its last balancing action. They show that the expected load difference between any two processors is bounded by a constant factor. They also bound the corresponding variance.

In [6, 7] the authors introduce and investigate the performance of certain randomized load-balancing algorithms under stochastic and adversarial work-generation models. They consider two different work-generation models. In the first model, in each step, each processor generates a task with some probability $p < 1$, and then each nonempty processor services a task with probability $p(1+\epsilon)$ for $\epsilon > 0$. In the second model, each processor is allowed to change its load in each step, provided that the load is only increased or decreased by at most a fixed constant amount. With high probability, the algorithms balance the system load up to additive terms of $O(\log\log n)$ and $O((\log\log n)^2)$, respectively. In particular, in the first model, the maximum load of any processor can be upper bounded by one of these terms (depending on the algorithm), whereas in the second model, the maximum load of any processor can be upper bounded by the average load plus $O(\log\log n)$. The algorithms and analysis of [6, 7] are fundamentally different from the one considered here. In particular, their algorithms are sender-initiated, i.e., overloaded processors seek to distribute their load. Moreover, their algorithms are considerably more complicated than ours.

There is relatively little existing theoretical work on receiver-initiated 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 [29], Mitzenmacher applies his differential-equations approach in order to analyze several randomized 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 [9], Blumofe and Leiserson analyze a scheduling strategy for *strict multithreaded computations*. A multithreaded 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 multithreaded computation on P processors is $O(T_1/P + T_\infty)$, where T_1 denotes the minimum sequential execution time of the multithreaded 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 [15], Fatourou and Spirakis develop an algorithm for k-strict multithreaded 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 $(X_t(1), \dots, X_t(n))$ in which $X_t(i)$ represents the length of the ith processor's queue after step t. Initially, all of the queues are empty, so the start state is $(0, \dots, 0)$.

Let $N = \{1, \dots, n\}$. Let $\mathcal{P} = \{h \mid N \to N\}$ be the set of all generator-allocation functions. When generators are allocated according a particular function $h \in \mathcal{P}$, $h(i)$ is designated as the host of the ith generator. The Markov chain X has three parameters. $\mathcal{D}$ is an arbitrary distribution over $\mathcal{P}$. A new generator-allocation function is selected from $\mathcal{D}$ during each step of the chain. 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. Finally, the function f is the work-stealing function. In section 3, we will state the properties that f must satisfy for our analysis. Figure 1 describes the transition from state X_t to state X_{t+1}.

1. Choose the generator-allocation function h_t from $\mathcal{D}$.
2. Each generator generates a new job independently with probability λ. It adds the job to the queue of its current host. In particular, the ith 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.
3. Each processor with an empty queue chooses a request destination 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.
4. 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 nonempty chooses ℓ u.a.r. from $r_t^{-1}(k)$ and sets $j_t^+(\ell) = j_t^-(k) = f(X'_t(k))$.
5. The work is shared and then each queue processes one job. Formally, for all i, $X_{t+1}(i)$ is set to $\max(0, X'_t(i) + j_t^+(i) - j_t^-(i) - 1)$.

FIG. 1. *The Markov chain X. The transition from X_t to X_{t+1}.*

3. Work-stealing functions.

In this section, we state the properties that the work-stealing function, f, must satisfy for our analysis.

DEFINITION 1. *We 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 14) and let $\alpha = 4e(c+1)$. Let $z = \lceil \alpha \lg n \rceil$. Let $\nu = n^{-2} + n^{-\alpha}$ and let $T = \lceil \frac{2c}{1-\lambda/(1-\nu)} \lg n \rceil$. Note that T is positive as long as $\nu < \delta$, and we will consider values of n for which this is true. Let g be the function given by $g(\ell) = f(\ell - T)$. We will use the function g in our analysis of the work-stealing process. Suppose that a processor has ℓ units of work in its queue. If no units of work are generated or stolen during T steps, it will then have $\ell - T$ units. Finally, it may give away $f(\ell - T) = g(\ell)$ units to another processor which requests work. Let Φ_f be the smallest integer such that, for all $j \in \{0, \dots, z\}$, $g^j(\Phi_f/n) \geq 2T$, where $g^j(y)$ denotes the j-fold application of function g to argument y. That is, $g^0(y) = 1$, $g^1(y) = g(y)$, $g^2(y) = g(g(y))$, and so on. If no such integer Φ_f exists, say $\Phi_f = \infty$. Informally, Φ_f is a quantity that is so large that if we start with Φ_f units of work and focus on the (at least Φ_f/n) units of work in some particular queue and allow this work to be stolen up to z times, the quantity of work remaining at every processor involved is at least $2T$. This idea will be made more precise later.*

We require the work-stealing function f to satisfy the following properties.

Property 1. $0 \leq f(\ell) \leq \ell/2$.

Property 2. $f(\ell)$ is monotonically nondecreasing in ℓ.

Property 3. Φ_f is finite.

Properties 1 and 2 are natural and easy to understand. We conclude this section by showing that many natural work-stealing functions which satisfy Properties 1 and 2 also satisfy Property 3. We start with a general lemma and then conclude with particular examples.

LEMMA 2. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Let $h_0 = 2T$. Suppose that there are positive integers $h_1, \ldots, h_z$ satisfying $f(h_i - T) \geq h_{i-1}$. Then $\Phi_f \leq nh_z$.*

Proof. We wish to show that for all $j \in \{0, \ldots, z\}$, $g^j(h_z) \geq 2T$. Since f satisfies Property 1, the condition $f(h_i - T) \geq h_{i-1}$ implies that $h_{i-1} \leq h_i$. Therefore, for any $j \in \{0, \ldots, z\}$, $h_{z-j} \geq h_0 \geq 2T$. Thus, it suffices to prove $g^j(h_z) \geq h_{z-j}$, which we will do by induction on j with base case $j = 0$. For the inductive step, note that

$$g^{j+1}(h_z) = f(g^j(h_z) - T) \geq f(h_{z-j} - T) \geq h_{z-(j+1)},$$

where the first inequality uses the monotonicity of f (Property 2) and the inductive hypothesis. $\square$

COROLLARY 3. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Suppose that $f(\ell) = \omega(1)$ as a function of ℓ. Then f satisfies Property* 3.

Proof. Since $f(\ell) = \omega(1)$, the function gets arbitrarily big and the values $h_1, \ldots, h_z$ in Lemma 2 exist. $\square$

Corollary 3 demonstrates that having $f(\ell) = \omega(1)$ is *sufficient* in the sense that this, together with Properties 1 and 2, implies Property 3. Having $f(\ell) = \omega(1)$ is not *necessary* though, as the following observation shows.

OBSERVATION 4. *Suppose that the work-stealing function f satisfies Properties* 1 *and* 2. *Let $h_0 = 2T$. Suppose (as in Lemma* 2*) that there are positive integers $h_1, \ldots, h_z$ satisfying $f(h_i - T) \geq h_{i-1}$. Let f' be the work-stealing function given by $f'(\ell) = \min(f(\ell), f(h_z))$. Then f' satisfies Properties* 1–3 *and has $\Phi_{f'} \leq nh_z$.*

We end the section by giving an upper bound for Φ_f when f is a member of a popular class of work-stealing functions.

LEMMA 5. *Let $f(\ell) = \lfloor \ell/r \rfloor$ for some $r \geq 2$. This function satisfies Properties* 1–3 *and satisfies*

$$\Phi_f \leq n(2T + 2r)(2r)^z.$$

Proof. We use Lemma 2. Let $h_i = (2T + 2r)(2r)^i$ for $i \in \{1, \ldots, z\}$. Then for $i \in \{1, \ldots, z\}$,

$$\begin{aligned} f(h_i - T) &= f((2T + 2r)(2r)^i - T) \\ &\geq \frac{(2T + 2r)(2r)^i}{r} - \frac{T}{r} - 1 \\ &= \frac{(2T + 2r)(2r)^i}{2r} + \frac{(2T + 2r)(2r)^i}{2r} - \frac{2T}{2r} - \frac{2r}{2r} \\ &\geq (2T + 2r)(2r)^{i-1} \\ &\geq h_{i-1}. \qquad \square \end{aligned}$$

Remark 6. The value Φ_f corresponding to the function f in Lemma 5 is bounded from above by a polynomial in n. To see this, note that the multiplier in the definition of T is

$$\frac{1}{1-\frac{\lambda}{1-\nu}} \le \frac{1}{1-\frac{1-\delta}{1-\nu}} = \frac{1-\nu}{\delta-\nu} \le \frac{1}{\delta-\nu} \le \frac{2}{\delta},$$

where the last inequality assumes $\nu \le \delta/2$, which is true if n is sufficiently large with respect to the constant δ^{-1}.

4. Upper bounds.

In this section we prove that the system is stable for every work-stealing function satisfying Properties 1–3 in section 3. Our analysis does not depend upon the particular distribution $\mathcal{D}$ which governs the allocation of generators—the analysis works for an arbitrary distribution.

As already outlined in section 1, the basic idea is the following. The Markov chain X models our system. Since this chain is difficult to analyze 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 nonrigorously, 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.

4.1. Background.

We start with some brief definitions regarding Markov chains. For more details, see [19]. The Markov chains that we consider are *time-homogeneous* (transition probabilities do not change over time) and *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

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

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

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

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 analyze the Markov chain X we will use the following generalization of Foster's theorem, due to Fayolle, Malyshev, and Menshikov (Theorem 2.2.4 of [16]).

Lemma 7 (Foster; Fayolle, Malyshev, Menshikov [16]). *A time-homogeneous irreducible aperiodic Markov chain ζ with a countable state space Ω is positive recurrent if and only if 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:*

$$E[\Phi(\zeta_{t+\beta(\zeta_t)}) - \Phi(\zeta_t) \mid \zeta_t = x] \leq -\xi\beta(x), \qquad x \notin C', \tag{1}$$

$$E[\Phi(\zeta_{t+\beta(\zeta_t)}) \mid \zeta_t = x)] < \infty, \qquad x \in C'. \tag{2}$$

We also use the following Chernov–Hoeffding inequalities. The first of these is a special case of Theorem 4.2 of [31] and the second is taken from Theorem 5.7 of [24].

Lemma 8 (Chernov). *Let $Z_1, \dots, Z_s$ be independent Bernoulli trials with $\Pr(Z_i = 1) = p$. Let $\widehat{Z} = \sum_{i=1}^{s} Z_i$. Then for any ρ in $(0, 1]$, $\Pr(\widehat{Z} < (1-\rho)sp) \leq \exp(-sp\rho^2/2))$.*

Lemma 9 (Hoeffding). *Let $Z_1, \dots, Z_s$ be independent random variables with $a_i \leq Z_i \leq b_i$ for suitable constants a_i, b_i and all $1 \leq i \leq s$. Also let $\widehat{Z} = \sum_{i=1}^{s} Z_i$. Then for any $t > 0$,*

$$\Pr\Big(|\widehat{Z} - E(\widehat{Z})| \geq t\Big) \leq \exp\Big(-2t^2 \Big/ \sum_{i=1}^{s} (b_i - a_i)^2\Big).$$

4.2. Results. Our Markov chain X is time-homogeneous, irreducible, and aperiodic. Its state space is countable. Therefore, it satisfies the initial conditions of Lemma 7. We will prove the following theorem.

Theorem 10. *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 Properties 1–3 in section 3. Then for every n which is sufficiently large with respect to δ^{-1}, the Markov chain X is positive recurrent.*

Theorem 10 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. Recall from Definition 1 that $\nu = n^{-2} + n^{-\alpha}$. Our next theorem uses the following additional definitions.

Definition 11. *Let ϵ be $(1 - \lambda/(1 - \nu))/4$. Let $\Phi(X_t)$ be the system load after step t. That is, $\Phi(X_t) = \sum_{i=1}^{n} X_t(i)$.*

Theorem 12. *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 Properties 1–3 in section 3. Then for every n which is sufficiently large with respect to δ^{-1},*

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

and for any nonnegative integer m,

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

4.3. A simpler Markov chain. Let C be the set of states x with $\Phi(x) < \Phi_f$. In this section we define a simpler Markov chain Z that will be used in order to analyze 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

$$(L_t(1), \dots, L_t(n), Y_t(1), \dots, Y_t(n)).$$

The variable $L_t(i)$ gives a crude indication of the load at processor i after step t. In any initial state that we will consider, *exactly one* processor (which we will call J_x) will have $L_0(J_x)$ large. All other processors will have $L_0(i) = 0$. Informally, these variables will have the following role for a processor i. Let t be the first time step during which processor i steals some of the work that originally sat at processor J_x. For $t' \leq t$, the variable $Y_{t'}(i)$ denotes the load of processor i and $L_{t'}(i) = 0$. For $t' > t$, the variable $L_{t'}(i)$ is positive and $Y_{t'}(i) = 0$. The exact value of $L_{t'}(i)$ gives an indication of how many times the work that processor i acquired at step t has been split (and, therefore, of how long it will last).

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 a load which originates at processor J_x. Thus, in the Markov chain Z, the state Z_0 which corresponds to X_0 is defined as 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

1. Choose the generator-allocation function h_t from $\mathcal{D}$.
2. 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.
3. $r'_t(i)$ 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.
4. 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 nonempty chooses ℓ u.a.r. from ${r'_t}^{-1}(k)$ and sets $l_t^+(\ell) = l_t^-(k) = L_t(k)/2$ and $j_t^+(\ell) = j_t^-(k) = f(Y'_t(k))$.
5. 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
$$\max(0, Y'_t(i) + j_t^+(i) - j_t^-(i) - 1).$$

FIG. 2. *The transition from Z_t to Z_{t+1}.*

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

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

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 $(c+1)n \lg n$.

Recall the construction of $Z(x)$ from the beginning of section 4.3. In particular, there is one "heavily loaded" processor, J_x, with $L_0(J_x) = 2^z$. Every other processor i has $L_0(i) = 0$.

Imagine that the value $L_0(J_x) = 2^z$ corresponds to a collection of 2^z tokens which initially sit at processor J_x. The value $L_t(k)$ gives the number of tokens which sit at processor k following step t. This is always a power of 2. If $L_t(k) > 1$, 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} splits the collection of tokens sitting at processor k and transfers half of these tokens to processor ℓ. The event $\Psi \leq \tau'$ occurs if and only if some token has its group split z times during steps $1, \ldots, \tau'$.

What is the probability that a given token has its group split z times? This is at most

$$\binom{(c+1)n \lg n}{z} n^{-z} \leq \left(\frac{e(c+1)\lg n}{z}\right)^z.$$

The probability that there exists a token which has its group split z times is thus at most

$$\left(\frac{2e(c+1)\lg n}{z}\right)^z \leq \left(\frac{2e(c+1)}{\alpha}\right)^{\alpha \lg n} = n^{-\alpha}. \qquad \square$$

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

$$\Pr(\tau' \leq T) \leq n^{-2}.$$

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. In particular, every processor i with $L_t(i) = 0$ and $Y_t(i) = 0$ contributes 1 to R'_t independently with probability $(1-\lambda)^{|h_t^{-1}(i)|}$ and contributes 0 to R'_t otherwise.

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 Lemma 9,

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

Let σ denote $\exp(-2(\frac{cn \lg n}{8T})^2/n)$. 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 during step t, i.e.,

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

Let ξ_t denote the number of heavily loaded processors during step t that don't get requested, i.e.,

$$\xi_t = |\{k \mid L_t(k) > 0 \wedge {r'_t}^{-1}(k) = \emptyset\}|.$$

If (s,h) is dangerous, then

$$E[\xi_t \mid Z_t = s \wedge h_t = h] \leq (1-\gamma)M_t.$$

Thus by Markov's inequality,

$$\Pr\bigl(\xi_t \geq (1-\gamma/2)M_t \mid Z_t = s \wedge h_t = h\bigr) \leq 1 - \frac{\gamma}{2-\gamma}.$$

If $\xi_t < (1-\gamma/2)M_t$, then at least $(\gamma/2)M_t$ of the M_t heavily loaded processors give away work, so $M_{t+1} \geq (1+\gamma/2)M_t$. We say that the step following on from a dangerous state is "useful" if this occurs. We have just seen that for every dangerous state (s,h), the probability that the next step is useful is at least $\frac{\gamma}{2-\gamma}$.

Thus, if we have D dangerous states during some time interval, the number of useful steps following them dominates (from above) the sum of D independent Bernoulli random variables with probability $p = \frac{\gamma}{2-\gamma}$. Applying Lemma 8 with $D = (2/p)\log_{1+\gamma/2}(n)$ and $\rho = 1/2$, we find that the probability that this sum is less than $\log_{1+\gamma/2}(n)$ is at most $\exp(-\log_{1+\gamma/2}(n)/4)$. This means that if we have D dangerous states, then the probability that there are at least $\log_{1+\gamma/2}(n)$ useful steps following them is at least $1 - \exp(-\log_{1+\gamma/2}(n)/4)$.

Now, if there are actually at least $\log_{1+\gamma/2}(n)$ useful steps during steps 1–t, $M_{t+1} = n$, so there can be no further dangerous states. We conclude that with probability at least $1 - \exp(-\log_{1+\gamma/2}(n)/4)$ there are at most D dangerous steps *ever*.

If we make c sufficiently large with respect to γ, then $D < c \lg n/2$.

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 nondangerous steps contribute at most $cn \lg n/2$ requests during the first T steps. Thus, the probability that $\tau' \leq T$ is at most

$$\exp(-\log_{1+\gamma/2}(n)/4) + \sigma T \leq n^{-2}. \qquad \square$$

Lemmas 13 and 14 imply the following.

COROLLARY 15. *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$.*

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

LEMMA 16. *Suppose $x \notin C$. Run Markov chain Z starting at $Z(x)$. For any $t \leq \Psi$ and any $i \in N$, either $L_t(i) = 0$ or, for some $j \in \{0, \ldots, z\}$, $L_t(i) = 2^j$.*

Proof. The lemma is proved by induction on t with the base case $t = 0$. Consider the assignment

$$L_{t+1}(i) = (L_t(i) - l_t^-(i)) + l_t^+(i)$$

in the transition from Z_t to Z_{t+1} in Figure 1. If the second term in the expression, $l_t^+(i)$, is greater than zero, then it is equal to $L_t(k)/2$ for some k with $r'_t(i) = k$, so $L_t(i) = 0$. The first term in the expression, $L_t(i) - l_t^-(i)$, is either $L_t(i)$ or $L_t(i)/2$. Thus, either $L_{t+1}(i)$ is $L_t(i)$ or it is $L_t(k)/2$ for some k. Using the terminology from the proof of Lemma 13, $L_{t+1}(i) = 2^{z-m}$ means that the tokens that sit at processor i after step $t+1$ have had their group split m times. Since $t \leq \Psi$, $m \leq z$. $\square$

4.4. Proof of Theorem 10. Our first task is to relate the Markov chain X to the simpler Markov chain Z. Recall the definitions of τ', R'_t, and Ψ from section 4.3. 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 $R_0 + \cdots + R_{\tau-1} \geq cn \lg n$.

LEMMA 17. *If $x \notin C$, then*

$$\Pr(\tau \leq T \mid X_0 = x) \leq \nu.$$

Proof. A (Markovian) coupling[1] of the Markov chains X and Z is a stochastic process (X_t, Z_t) such that (X_t), considered marginally, is a faithful copy of X, and (Z_t), considered marginally, is a faithful copy of Z. We will describe a coupling starting from state $(x, Z(x))$. That is, in our coupling, $X_0 = x$ and $Z_0 = Z(x)$. The coupling will have the property that for all $t \leq \min(T, \tau', \Psi)$ and all i,

$$r'_t(i) = r_t(i). \tag{3}$$

From (3), we can conclude that whenever the Z chain satisfies $T < \tau' < \Psi$, the coupled X chain satisfies $T < \tau$. Thus,

$$\Pr(T < \tau \mid X_0 = x) \geq \Pr(T < \tau' < \Psi \mid Z_0 = Z(x)),$$

so the lemma follows from Corollary 15.

To give the details of the coupling, we will use the notation in Figures 1 and 2. Recall from Definition 1 that g is the function given by $g(y) = f(y - T)$, where f is

[1]The word "coupling" is normally used in reference to combining two copies of the same Markov chain, so we are using the word in a slightly nonstandard way.

the work-stealing function which is guaranteed to satisfy $g^j(\Phi_f/n) \geq 2T$ for a finite Φ_f and for all $j \in \{0, \ldots, z\}$.

Our coupling will satisfy the following invariants for any $i \in N$ and any $t \leq \min(T, \tau', \Psi)$:

(1) $r'_t(i) = r_t(i)$,
(2) $L_t(i) = 0$ implies $X_t(i) = Y_t(i)$, and
(3) $L_t(i) = 2^j$ implies $X_t(i) \geq g^{z-j}(\Phi_f/n) - t$.

As we observed above, our objective is to describe a coupling that satisfies invariant (1). The other invariants will help us to show that our constructed coupling is indeed a coupling in the sense that the marginal distributions are correct. The purpose of the third invariant is to ensure that, in the chain X, a node will not become empty soon if the corresponding node in chain Z is heavily loaded.

The coupling is as follows. We start with $X_0 = x$ and $Z_0 = Z(x)$. Recall the construction of $Z(x)$ from section 4.3. In particular, $L_0(J_x) = 2^z$ and $Y_0(J_x) = 0$. For every other i, $L_0(i) = 0$ and $Y_0(i) = x(i)$. Invariants (2) and (3) are satisfied for $t = 0$ since $X_0(J_x) \geq \Phi_f/n$.

Now the transition from (X_t, Z_t) to (X_{t+1}, Z_{t+1}) is given as follows. In part 1 of the transition, the same generator-allocation function h_t is chosen for both chains. The $X'_t(i)$ variables are defined in part 2 of the transition from X_t to X_{t+1}. In part 2 of the coupled transition from Z_t to Z_{t+1}, we set $Y'_t(i) = 0$ if $L_t(i) > 0$. Otherwise, we set $Y'_t(i) = X'_t(i)$. Note that, since invariant (2) held after step t, the $Y'_t(i)$ variables are set according to the correct marginal distribution. The $r_t(i)$ variables are defined in part 3 of the transition from X_t to X_{t+1}. In part 3 of the coupled transition from Z_t to Z_{t+1}, we set $r'_t(i) = r_t(i)$. To show that the marginal distribution is correct, we observe that if $L_t(i) = 0$, then we defined $Y'_t(i)$ to be $X'_t(i)$. Thus, the $r'_t(i)$ variables are assigned correctly. On the other hand, if $L_t(i) > 0$, then, by Lemma 16, $L_t(i) = 2^j$ for some $j \in \{0, \ldots, z\}$ so by invariant (3), $X_t(i) \geq g^{z-j}(\Phi_f/n) - t \geq 2T - t > 0$. Thus, $X'_t(i) > 0$, and $r'_t(i)$ is defined correctly. In part 4 of the transition from X_t to X_{t+1}, we do the following. For every $k \in N$ for which $r_t^{-1}(k)$ is nonempty, we choose ℓ u.a.r. from $r_t^{-1}(k)$. Since ${r'_t}^{-1}(k) = {r_t}^{-1}(k)$, we can make the same choice for k in part 4 of the transition from Z_t to Z_{t+1}.

We need to prove that the coupling maintains invariants (1), (2), and (3). Invariant (1) (the one that we actually want) is by construction. Invariant (2) is not too difficult. Lemma 16 shows that all of the variables $L_t(i)$ are nonnegative. Furthermore, the analysis in the proof of Lemma 16 reveals that $L_{t+1}(i) = 0$ implies $L_t(i) = 0$. (To see this, recall that $L_{t+1}(i) = (L_t(i) - l_t^-(i)) + l_t^+(i)$. The second of these terms is nonnegative, and the first is either $L_t(i)$ or $L_t(i)/2$.) Thus, whenever we have $L_{t+1}(i) = 0$ we have $Y_t(i) = X_t(i)$, and in the coupling we get $Y'_t(i) = X'_t(i)$. In part 5 of the transition from X_t to X_{t+1} we set $X_{t+1}(i) = \max(0, X'_t(i) + j_t^+(i) - j_t^-(i) - 1)$, and in part 5 of the transition from Z_t to Z_{t+1}, we set $Y_{t+1}(i) = \max(0, Y'_t(i) + j_t^+(i) - j_t^-(i) - 1)$. Thus we need only argue that the $j_t^+(i)$ and $j_t^-(i)$ variables get the same values in both copies. The $j_t^-(i)$ variable is the same since $Y'_t(i) = X'_t(i)$. The $j_t^+(i)$ variables are positive only if processor i made a request, namely, $r_t(i) = r'_t(i) = k$, for some k. Since $L_{t+1}(i) = 0$, we know $L_t(k) = 0$. Hence, $Y'_t(k) = X'_t(k)$, and the $j_t^+(i)$ values are indeed the same.

Finally, we need to prove that the coupling maintains invariant (3). Suppose that $L_{t+1}(i) = 2^j$. We wish to show that $X_{t+1}(i) \geq g^{z-j}(\Phi_f/n) - (t+1)$. First, suppose

$L_t(i) = 0$. In this case, $L_{t+1}(i) = L_t(k)/2$, where $r_t(i) = k$. Then

$$\begin{aligned} X_{t+1}(i) &\geq f(X_t(k)) - 1 \\ &\geq f(g^{z-j-1}(\Phi_f/n) - t) - 1 \\ &\geq f(g^{z-j-1}(\Phi_f/n) - T) - 1 \\ &= g^{z-j}(\Phi_f/n) - 1 \\ &\geq g^{z-j}(\Phi_f/n) - (t+1), \end{aligned}$$

where the first inequality follows from the transition in Figure 1 and the second inequality follows from the facts that invariant (3) held after step t and that f is monotonically nondecreasing (Property 2 in section 3). The third inequality also follows from the fact that f is monotonically nondecreasing.

Second, suppose $L_t(i) = 2^j$. In this case $r_t^{-1}(i)$ is empty, so

$$X_{t+1}(i) \geq X_t(i) - 1 \geq g^{z-j}(\Phi_f/n) - t - 1.$$

Finally, suppose $L_t(i) = 2^{j+1}$. (To see that these are the only cases, namely, that $L_t(i) \in \{0, 2^j, 2^{j+1}\}$, see the proof of Lemma 16.) In this case r_t^{-1} is nonempty, so

$$X_{t+1}(i) \geq X_t(i) - f(X_t(i)) - 1.$$

Since f satisfies $f(\ell) \leq \ell/2$ (Property 1 in section 3), we have

$$X_{t+1}(i) \geq f(X_t(i)) - 1,$$

which is the same as the first case. $\square$

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

LEMMA 18. *If $x \notin C$, then*

$$E[\Phi(X_T) \mid (X_0 = x) \wedge (\tau > T)] \leq \Phi(x) - 2\epsilon n T.$$

Proof. Let A_t be the number of new jobs that arrive in the system during the transition from X_t to X_{t+1}. Namely,

$$A_t = \sum_{i \in N} (X'_t(i) - X_t(i)).$$

Let $Y = A_0 + \cdots + A_{T-1}$. Splitting $E[Y \mid X_0 = x]$ into two conditional expectations, conditioned on whether or not $\tau > T$, we find

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

By Lemma 17, the denominator is at least $1 - \nu$. The numerator is at most $E[Y \mid X_0 = x]$, which is $\lambda n T$, since during each of the T steps each of the n generators generates a new job independently with probability λ. Thus,

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

If $\tau > T$, then the number of jobs serviced during steps 1–T is at least $nT - cn\lg n$. (If a processor does not make a request, then it certainly services a job.) Thus, the quantity

$$E[\Phi(X_T) \mid (X_0 = x) \wedge (\tau > T)]$$

is at most the initial load, $\Phi(x)$, plus the expected number of arrivals, which we have seen above is at most $\frac{\lambda nT}{1-\nu}$, minus the expected number of services, which is at least $nT - cn\lg n$. Putting all of this together, we get

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

where the second inequality uses the definition of T in Definition 1 and the equality uses the definition of ϵ in Definition 11. ∎

LEMMA 19. *Suppose that n is sufficiently large with respect to δ^{-1}. If $x \notin C$, then*

$$E[\Phi(X_T) \mid X_0 = x] \le \Phi(x) - \epsilon nT.$$

Proof.

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

By Lemma 18, this is at most

$$\Pr(\tau > T \mid X_0 = x)(\Phi(x) - 2\epsilon nT) + \Pr(\tau \le T \mid X_0 = x) E[\Phi(X_T) \mid (X_0 = x) \wedge \tau \le T].$$

Since at most n messages arrive per step, this is at most

$$\Pr(\tau > T \mid X_0 = x)(\Phi(x) - 2\epsilon nT) + \Pr(\tau \le T \mid X_0 = x)(\Phi(x) + nT).$$

This can be rearranged as

$$\begin{aligned} &\Phi(x) - (1 - \Pr(\tau \le T \mid X_0 = x))(2\epsilon nT) + \Pr(\tau \le T \mid X_0 = x)(nT) \\ = \; &\Phi(x) - 2\epsilon nT + \Pr(\tau \le T \mid X_0 = x)(2\epsilon nT + nT). \end{aligned}$$

By Lemma 17, this is at most

$$\Phi(x) - 2\epsilon nT + \nu(2\epsilon nT + nT) = \Phi(x) - \epsilon nT - (\epsilon nT - \nu 2\epsilon nT - \nu nT).$$

The lemma follows from the fact that

$$\nu \le \frac{\epsilon}{2\epsilon + 1}, \tag{4}$$

which is true, provided that n is sufficiently large with respect to δ^{-1}. To establish (4), refer to Definitions 1 and 11. If n is sufficiently large, then $\nu \le \delta/2$, so

$$4\epsilon = 1 - \frac{\lambda}{1-\nu} \ge 1 - \frac{1-\delta}{1-\nu} = \frac{\delta - \nu}{1-\nu} \ge \frac{\delta}{2}.$$

Also,

$$\nu \le \frac{\delta/8}{2(\delta/8) + 1} \le \frac{\epsilon}{2\epsilon + 1}. \qquad ∎$$

Combining Lemmas 19 and 7, we get a proof of Theorem 10.

4.5. Proof of Theorem 12. The proof of Theorem 12 uses the following theorem, which is Theorem 1 of [8].

LEMMA 20 (Bertsimas, Gamarnik, and Tsitsiklis [8]). *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*

$$E[\Phi(\zeta_{t+1}) - \Phi(\zeta_t) \mid \zeta_t = x] \leq -\xi, \qquad \Phi(x) > \beta, \tag{5}$$

and

$$|\Phi(\zeta_{t+1}) - \Phi(\zeta_t)| \leq \nu_{\max}, \tag{6}$$

and, for any x,

$$\Pr[\Phi(\zeta_{t+1}) > \Phi(\zeta_t) \mid \zeta_t = x] \leq p_{\max} \tag{7}$$

and

$$E_{\pi'}[\Phi(\zeta_t)] < \infty, \tag{8}$$

then for any nonnegative 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

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

Let $W_i = \Phi(X_{iT})$ for $i \in \{0, 1, 2, \dots\}$. Lemma 19 shows that the process $W_0, W_1, \dots$ behaves like a supermartingale above Φ_f. That is, it satisfies (5) with $\xi = \epsilon nT$ and $\beta = \Phi_f$. In itself, this does not imply that $E[W_t]$ is bounded (see Pemantle and Rosenthal's paper [32] for counterexamples). However, we also have

$$|W_{t+1} - W_t| \leq nT \tag{9}$$

for any t. That is, (6) is satisfied with $\nu_{\max} = nT$. This implies (for example, by Theorem 1 of [32] or by Theorem 2.3 of [20]) that $E_\pi[W_t]$ is finite (so (8) is satisfied). Lemma 20 can now be applied with $p_{\max} = 1$ to get

$$E_\pi[W_t] \leq \Phi_f + 2nT/\epsilon, \tag{10}$$

and for any nonnegative integer m,

$$\Pr_\pi[W_t > \Phi_f + 2nTm] \leq \left(\frac{nT}{nT + \epsilon nT}\right)^{(m+1)}. \tag{11}$$

The theorem now follows from the observation that for $0 \leq j < T$,

$$\Phi(X_{iT+j}) \leq \Phi(X_{iT}) + nj.$$

5. 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 put some restrictions on $\mathcal{D}$ in order to obtain instability. For example, if $\mathcal{D}$ is the point distribution containing a single function h which allocates one generator to each processor, then the system will be stable even without any work-stealing.

The proof of our lower bound uses the following lemma, which is Theorem 2.2.7 of [16].

LEMMA 21 (Fayolle, Malyshev, and Menshikov [16]). *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 $\Phi(x) \le C$,*
2. *$E[\Phi(\zeta_1) - \Phi(\zeta_0) \mid \zeta_0 = x] \ge \xi$ 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.*

If we use $k = 1$ in the statement of the following theorem, we find that the system is unstable if $f(\ell) < \lambda n - 1$.

THEOREM 22. *Let δ be a positive constant and λ an arrival rate which is at most $1 - \delta$. Suppose that $\mathcal{D}$ contains a single generator-allocation function h which distributes the n generators equally among some set of k processors. Suppose that for all ℓ, $f(\ell) \le j(n)$. Then the Markov chain X is transient if*

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

Proof. This theorem can be proven easily using Lemma 21. Recall that the start state X_0 is $(0, \dots, 0)$ (all queues are initially empty). 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 gives work to only one other processor per round, and there are at most k processors with generators, 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 21 with Φ as the system load, it is easy to see that the system is transient if $k(j(n) + 1) < \lambda n$. □

6. Conclusions.

We have analyzed a very simple work-stealing algorithm, which is successfully being used in practical applications. In this paper we have analyzed 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 shown that for every $\lambda > 0$ there is a class of unsuitable work-stealing functions, for which it is not stable. Finally, we have derived upper bounds on the system load when the system is stable.

It would be interesting to know whether there is a nice characterization of the class of functions that lead to stability. It would also be interesting to know how far our upper bounds on system load are from the truth. We suspect that the system load is actually much smaller than our upper bounds indicate, but it would be useful to have rigorous experimental results.

Acknowledgments. We thank Hesham Al-Ammal for useful discussions. We also thank the referees, both of whom provided many helpful comments.

REFERENCES

[1] M. Adler, P. Berenbrink, and K. Schröder, *Analyzing an infinite parallel job allocation process*, in Proceedings of the 6th European Symposium on Algorithms (ESA'98), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1998, pp. 417–428.

[2] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen, *Parallel randomized load balancing*, in Proceedings of the 27th Symposium on Theory of Computing (STOC'95), ACM Press, New York, 1995, pp. 234–247.

[3] H. Al-Ammal, L. A. Goldberg, and P. MacKenzie, *An improved stability bound for binary exponential backoff*, Theory Comput. Systems, 34, (2001), pp. 229–244.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, *Balanced allocations (extended abstract)*, in Proceedings of the 26th Symposium on Theory of Computing (STOC'94), ACM Press, New York, 1994, pp. 593–602.

[5] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking, *Balanced allocations: The heavily loaded case*, in Proceedings of the 32th ACM Symposium on Theory of Computing (STOC'00), ACM Press, New York, 2000, pp. 745–754.

[6] P. Berenbrink, T. Friedetzky, and E. W. Mayr, *Parallel continuous randomized load balancing*, in Proceedings of the 10th Symposium on Parallel Algorithms and Architectures (SPAA '98), ACM Press, New York, 1998, pp. 192–201.

[7] P. Berenbrink, T. Friedetzky, and A. Steger, *Randomized and adversarial load balancing*, in Proceedings of the 11th Symposium on Parallel Algorithms and Architectures (SPAA'99), ACM Press, New York, 1999, pp. 175–184.

[8] D. Bertsimas, D. Gamarnik, and J. N. Tsitsiklis, *Performance of multiclass Markovian queueing networks via piecewise linear Lyapunov functions*, Ann. Appl. Probab., 11 (2001), pp. 1384–1428.

[9] R. Blumofe and C. Leiserson, *Scheduling multithreaded computations by work stealing*, in Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS'94), IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 356–368.

[10] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, *Cilk: An efficient multithreaded runtime system*, J. Parallel Distrib. Comput., 37 (1996), pp. 55–69.

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

[12] R. Cole, A. Frieze, B. Maggs, M. Mitzenmacher, A. Richa, R. Sitaraman, and E. Upfal, *On balls and bins with deletions*, in Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM '98), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1998, pp. 145–158.

[13] A. Czumaj and V. Stemann, *Randomized allocation processes*, in Proceedings of the 38th Symposium on Foundations on Computer Science (FOCS'97), IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 194–203.

[14] T. Decker, *Virtual Data Space—Load balancing for irregular applications*, Parallel Comput., 26 (2000), pp. 1825–1860.

[15] P. Fatourou and P. Spirakis, *Scheduling algorithms for strict multithreaded computations*, in Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'96), Lecture Notes in Comput. Sci., Springer-Verlag, New York, 1996, pp. 407–416.

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

[17] B. M. Feldmann, P. Mysliwietz, and B. Monien, *Studying overheads in massively parallel min/max-tree evaluation*, in Proceedings of the 5th Symposium on Parallel Algorithms and Architectures, ACM Press, New York, 1994, pp. 94–103.

[18] L. Goldberg and P. MacKenzie, *Analysis of practical backoff protocols for contention resolution with multiple servers*, J. Comput. Systems Sci., 58 (1999), pp. 232–258.

[19] G. Grimmet and D. Stirzaker, *Probability and Random Processes*, 2nd ed., Oxford University Press, Oxford, 1992.

[20] B. Hajek, *Hitting time and occupation time bounds implied by drift analysis with applications*, Adv. Appl. Probab., 14 (1982), pp. 502–525.

[21] J. Håstad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, SIAM J. Comput., 25 (1996), pp. 740–774.

[22] R. Lüling and B. Monien, *A dynamic distributed load balancing algorithm with provable good performance*, in Proceedings of the 5th Symposium on Parallel Algorithms and Architectures (SPAA'93), ACM Press, New York, 1993, pp. 164–172.

[23] N. R. Mahapatra and S. Dutt, *Adaptive quality equalizing: High-performance load balancing*

for parallel branch and bound across applications and computing systems, in Proceedings of the Joint IEEE Parallel Processing Symposium/Symposium on Parallel and Distributes Processing, IEEE Computer Society Press, Los Alamitos, CA, 1998, pp. 796–800.

[24] C. McDiarmid, *On the method of bounded differences*, in Surveys in Combinatorics, London Math. Soc. Lecture Note Ser. 141, Cambridge University Press, Cambridge, UK, 1989, pp. 148–188.

[25] S. Meyn and R. Tweedie, *Markov Chains and Stochastic Stability*, Springer-Verlag, London, 1993.

[26] M. Mitzenmacher, *Density dependent jump Markov processes and applications to load balancing*, in Proceedings of the 37th Symposium on Foundations of Computer Science (FOCS'96), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 213–222.

[27] M. Mitzenmacher, *The Power of Two Random Choices in Randomized Load Balancing*, Ph.D. thesis, Graduate Division, University of California at Berkeley, 1996.

[28] M. Mitzenmacher, *On the analysis of randomized load balancing schemes*, in Proceedings of the 9th Symposium on Parallel Algorithms and Architectures (SPAA'97), ACM Press, New York, 1997, pp. 292–301.

[29] M. Mitzenmacher, *Analysis of load stealing models based on differential equations*, in Proceedings of the 10th Symposium on Parallel Algorithms and Architectures (SPAA'98), ACM Press, New York, 1998, pp. 212–221.

[30] M. Mitzenmacher, A. Richa, and R. Sitaraman, *The power of two randomized choices: A survey of techniques and results*, in Handbook of Randomized Computing, Kluwer Academic, Dordrecht, The Netherlands, 2001, pp. 255–305.

[31] R. Motwani and P. Raghavan, *Randomized Algorithms,* Cambridge University Press, Cambridge, UK, 1995.

[32] R. Pemantle and J. S. Rosenthal, *Moment conditions for a sequence with negative drift to be uniformly bounded in l^r*, Stochastic Process. Appl., 82 (1999), pp. 143–155.

[33] L. Rudolph, M. Slivkin-Allalouf, and E. 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), ACM Press, New York, 1991, pp. 237–245.

[34] B. Vöcking, *How asymmetry helps load balancing*, in Proceedings of the 40th Symposium on Foundations of Computer Science (FOCS'00), IEEE Computer Society Press, Los Alamitos, CA, 2000, pp. 131–140.

[35] N. D. Vvedenskaya, R. L. Dobrushin, and F. I. Karpelevich, *Queueing system with selection of the shortest of two queues: An asymptotic approach*, Problems Inform. Transmission, 32 (1996), pp. 15–27.