Dynamic Diffusion Load Balancing *

Petra Berenbrink† and Tom Friedetzky‡
Simon Fraser University
School of Computing Science
Burnaby, B.C., V5A 1S6, Canada
E-mail: (petra,tkf)@cs.sfu.ca

Russell Martin§
Department of Computer Science
University of Warwick
Coventry, CV4 7AL, UK
URL: www.dcs.warwick.ac.uk/~martin

Abstract

We present the first analysis of a simple discrete diffusion scheme for dynamic load balancing. In each round $(1-\epsilon)n$ tasks are (arbitrarily) generated over a set of n processors, load is balanced amongst neighbours in some underlying graph, then a single task is deleted from each non-empty processor. We show that the system is stable, in that the total load remains bounded (as a function of n alone) over time. Our proof only requires that the underlying "communication" graph be connected, and holds even when $\epsilon = 0$. We further show that the upper bound we obtain is asymptotically tight (up to a moderate multiplicative constant) by demonstrating a corresponding lower bound on the system load for the particular example of a linear array (or path). We also show some simple negative results (i.e., instability) for work-stealing based diffusion-type algorithms.

1 Introduction

The use of parallel and distributed computing is established in many areas of science, technology, and business. One of the most crucial parameters of parallel machines is the efficient utilisation of resources. Of greatest importance here is an even distribution of the workload among the processors. In particular applications exposing some kind of "iregularity" require the use of load balancing mechanism.

A well known and much studied load balancing approach is the so-called *diffusion load balancing*, first introduced by Cybenko and Boillat ([9], [8]). The algorithm works in synchronised rounds. The basic idea is that in every round, every processor p balances load with all its neighbours (independently, i.e., pair-wise). Let ℓ_p be the load of p, ℓ_q the load of some of p's neighbour q, and let Δ denote the maximum degree of the underlying graph. Then p transfers $\max\{0, (\ell_p - \ell_q)/(\Delta+1)\}$ tasks to q. At the same time, q sends $\max\{0, (\ell_q - \ell_p)/(\Delta+1)\}$ tasks to p. Note that (i) transfer will occur in at most one direction (either $p \to q$ or $q \to p$), and (ii) in this model it is possible that a processor forwards (possibly infinitesimally) small fractions of a task to neighbours, which may be rectifiable in some setting, but clearly not in general. Since the idea of arbitrarily small task fragments doesn't really find an equivalent in the real world, the *discrete* counterpart of the algorithm has subsequently been studied, where $\max\{0, \lfloor(\ell_p - \ell_q)/(\Delta+1)\rfloor\}$ tasks are being transferred (notice the floor). This problem is commonly referred to as the *token distribution problem*.

Some of many advantages of diffusion-type algorithms are the *locality* (no global knowledge regarding the overall load situation, or, in fact, anything except the strict neighbourhood of any vertex is needed), its *simplicity*, and its *neighbourhood preservation* (tasks tend to stay close to the processors where they are generated, which may help to maintain small communication overhead).

*Research Report CS-RR-402, Department of Computer Science, University of Warwick, July 2004.
†Supported in part by NSERC, Canada.
‡Supported in part by MITACS (Mathematics of Information Technology and Complex Systems), Canada.
§A portion of this work was performed during a visit to the School of Computing Science at Simon Fraser University. Supported in part by the EPSRC grant "Discontinuous Behaviour in the Complexity of Randomized Algorithms".

The diffusion load balancing algorithm has been thoroughly analysed for *static* load balancing and token distribution problems (see Section 1.1), but a long time open problem was to analyse the approach in a dynamic setting. In a static problem, it is assumed that each processor has some initial number of tasks, and the objective is to distribute *this* load evenly among the processors, as quickly as possible. Clearly, the static approach is incapable of modelling a huge variety of "real world" problems (as well as theoretical ones) , thus the importance of and need for an analysis of the dynamic case. There, the only analytical results are for a related problem where neighbours do not balance load but are only allowed to exchange single tasks.

In this paper we present the first analysis of the simple diffusion scheme for the dynamic load balancing problem. We assume that $\lfloor(1-\epsilon)n\rfloor$ new tasks are generated per round and that every non-empty processor deletes one task per round. We assume an *arbitrary* $\epsilon \in [0, 1)$, i.e., in particular we allow for *full saturation* of the resources. The newly generated tasks may be arbitrarily distributed among the nodes of the network. To give some examples, the tasks may always be generated on the same processor, all tasks may be generated on one processor but the processor can change from round to round, or the tasks may be allocated at random. Note that, obviously, without load balancing the total number of tasks in the system may grow unboundedly with time (in the worst case, we generate n new tasks per step but delete only one).

We show that the system of processors is *stable* under the diffusion load balancing scheme and worst-case generation of tasks. By stable we mean that the total load in the system does not grow with time. In particular, we show that the total system load can be upper-bounded by $O(\Delta n^3)$, with Δ denoting the maximum degree of the network. Furthermore, we present a simple, asymptotically matching lower bound. Finally, we provide some results for systems in which only empty processors receive tasks due to balancing actions, i.e., a situation very similar to traditional work-stealing approaches. We show that in this model there are graphs for which the system cannot be stable for a significant class of generation parameters.

1.1 Known results

The work that has been done so far can be divided into two main classes: token distribution (at most one task can be moved across any edge during any time step) and load balancing (several tasks can be moved per edge and round). In the case of token distribution, there are results for balancing related algorithms, also assuming dynamic load generation model. Since every edge can only transport one task per step, the load generation is (not very surprisingly) very restricted compared to our model.

In the load balancing area, most diffusion-type algorithms assume that tasks can be split arbitrarily, and they assume static systems. Regarding dynamic load balancing, the algorithms analysed so far are generally very different from the simple diffusion approach.

Token distribution. Much work has been done on the static token distribution problem under the assumption that every edge is only allowed to forward one task per round; see [13], [14] and [16]. Awerbuch and Leighton [5, 6] consider a closely related algorithm to solve a routing problem. [1] and [4] expand their results for packet routing.

In [15], Muthukrishnan and Rajaraman study a dynamic version of the token distribution problem. They assume an adversarial load generation model. The adversary is allowed to generate and to delete tokens from the network in every round. The simple and elegant algorithm they consider is due to [2]: A node is sending a task to its neighbour if the load difference is at least $2\Delta + 1$. They show that the system is stable if the load change in every subset S of the nodes minus $a|S|$ is at most $(1-\epsilon)e(S)$ for $\epsilon > 0$. Here $e(S)$ is the number of outgoing edges of S and a is the change in the average load. Clearly, their notion of stability is different from ours: their system is said to be stable if the deviation of the load of any processor from the average load can be bounded. Muthukrishnan and Rajaraman left open the question whether the system is also stable for $\epsilon = 0$. Anshelevich, Kempe, and Kleinberg [3] gave a positive answer. Additionally, they showed that a generalisation of the algorithm is stable for two distinct types of jobs, and they extended their results to related flow problems.

Applied to our notion of stability (upper bound on system load), the analysis from [2] and [3] allows only for a load change that depends on the expansion factor of the underlying network, which may be very small (e.g., linear array), whereas we allow for n tasks to be generated on one processor.

Load balancing. A lot of work studied the diffusion algorithm for static problems under the assumption that tasks can be split arbitrarily; see [10, 11, 12].

In [17], Rabani, Sinclair, and Wanka develop a general technique for the quantitative analysis of static iterative load balancing schemes, like the diffusion scheme. Their main contribution is an effective way of characterising the deviation between the actual loads of a system where tasks can not be split, and the distribution generated by a related Markov chain for the fractional case.

Berenbrink, Friedetzky, and Goldberg [7] showed stability of a work stealing algorithm under a similar load generation model. They consider a flexible distribution of n generators among the nodes of the network, and allow each generator to generate a task with probability strictly smaller than one. They show that a very simple, parameterised work-stealing algorithm achieves stability (in our sense) for a wide range of parameters. Compared to the problem studied here, however, in their model processors are not restricted to exchanging load with their neighbours only, and the results only hold for the case where strictly less than n tasks are generated during any time step.

1.2 Our Model

Our parallel system is modelled by a connected graph $G = (V, E)$. The nodes V of the graph model our processors $\mathcal{P} = \{P_1, \ldots, P_n\}$, and the edges E model the underlying communication structure. If two nodes are connected with each other, this means that the processors modelled by the nodes can communicate directly. For us, this means that they are allowed to exchange tasks. Nodes not connected by an edge have to communicate via message passing. Furthermore, let Δ be the maximum degree of the graph. We assume that each processor maintains a queue in which yet-to-be-processed tasks are stored. One round looks as follows:

1. At most $(1 - \epsilon)n$ generators are arbitrarily distributed over the processors, where $0 \leq \epsilon < 1$. Each generator generates one task at the beginning of every time round. For $1 \leq i \leq (1 - \epsilon)n$, let $k_i^t = j$ if generator i is allocated to processor P_j in round t, and $k_i^t = 0$ if the generator is not allocated to any processor in that round.
2. Every generator generates one task.
3. Every processor balances its load with some or all its neighbours in the network (according to a well-defined scheme for doing this operation).
4. Every non-empty processor deletes one task.

Let $\hat{\ell}_i^t$ be the load of P_i directly after the load deletion phase in round t. A system is called *stable* if the number of tasks $L^t(\mathcal{P}) = \sum_{i=1}^n \hat{\ell}_i^t$ that are in the system at the end of round t does not grow with time, i.e. the total load $L^t(\mathcal{P})$ is bounded by a number that might depend on n, but not on the time t.

In this paper, we will mainly focus on one load balancing method called the *diffusion approach*. Every processor is allowed to balance its load with all its neighbours. We briefly consider a second approach in Section 4, only empty processors are allowed to "steal" load from their non-empty neighbours. This second method is called the *work stealing approach* in the following.

Diffusion approach. We begin with a detailed description of the first approach, an integral variant of the *First Order Diffusion scheme*. Let $\bar{\ell}_i^t$ be the load of processor P_i directly before the load balancing phase, and ℓ_i^t the load directly after the load balancing phase. Let $\alpha_{i,j}^t$ be the load that is to be sent from P_i to P_j in round t for $(i, j) \in E$ ($\alpha_{i,j}^t = 0$ otherwise). Then $\alpha_{i,j}$ and ℓ_i are calculated as follows:

$$\alpha_{i,j}^t := \max\left\{0, \left\lfloor \frac{\bar{\ell}_i^t - \bar{\ell}_j^t}{2\Delta} \right\rfloor\right\} \qquad \ell_i^t := \bar{\ell}_i^t - \sum_{(i,j)\in E} \alpha_{i,j}^t + \sum_{(j,i)\in E} \alpha_{j,i}^t$$

To compute $\hat{\ell}_i^t$, the load of processor P_i after load deletion, it remains to subtract one if $\ell_i^t > 0$, thus

$$\hat{\ell}_i^t := \max\{0, \ell_i^t - 1\}.$$

Note that the "standard" diffusion approach divides $\bar{\ell}_i^t - \bar{\ell}_j^t$ by $\Delta + 1$ instead of 2Δ. We need the 2Δ for our analysis.

1.3 New Results

We will now very briefly introduce our contributions. In Section 2, we prove Theorem 2.1, which states that we can upper-bound the total system load with $3\Delta n^3$. Notice that although the potential function that we use looks similar to the one used in [3], the proof technique is very different. Loosely speaking, our generation model allows for much "worse" things to happen (is, in some sense, a strict superset of the generation model of [3]), and thus the analysis is necessarily more involved. Also, due to the different nature of our diffusion approach, methods from (say) [3] can not be used in our case.

We do, however, note that our technique also captures the model where stability is defined in terms of deviation of any node's load from the average (for a thorough introduction to this model see, e.g., [15, 3]). Using the definitions from Section 1.2, we can show that if for every subset S of the nodes, the *load difference* induced in S by a combined generation/deletion process can be bounded such that

$$(L(S)^t - \bar{L}(S)^{t-1}) \leq (\text{avg}(t) - \text{avg}(t-1)) \cdot |S| + n$$

where $\text{avg}(t)$ denotes the average system load in step t, then total load of S can be bounded by $\text{avg}(t) + 3\Delta n^3$. We omit details due to space limitations; the interested reader is referred to the full version of this paper (which is in preparation).

Next, Theorem 3.2 in Section 3 provides an asymptotically matching lower bound. In Section 4 we discuss the problem of combining the diffusion-approach with the work-stealing approach and show that certain assumptions necessarily lead to instablily.

2 Analysis of the Dynamic Diffusion Algorithm

In this section we will show that the diffusion approach yields a stable system. Moreover, we are able to upper bound the maximum load that will be in the system by $O(\Delta n^3)$. Throughout, we assume that $n \geq 2$ and $\Delta \geq 2$.

In order to clarify the exposition, we first recall the notation we have already defined:

- $\bar{\ell}_i^t$ denotes the load of processor P_i after we have generated tasks at the start of round t, but before load is balanced,
- ℓ_i^t is the load of processor P_i immediately after the load balancing phase, and
- $\hat{\ell}_i^t$ is the load of processor P_i after the task deletion phase of round t (i.e. at the very end of round t).
- We will also use notation like $\bar{L}^t(S) = \sum_{i:P_i \in S} \bar{\ell}_i^t$ for a subset $S \subseteq \mathcal{P}$, with similar definitions for $L^t(S)$ and $\hat{L}^t(S)$.

With this notation, our main result about the diffusion approach to load balancing is

Theorem 2.1 *Let $n \geq 2$ denote the number of processors in the system, and an upper bound on the number of tasks that are generated during each time round. Let $\Delta \geq 2$ denote the maximum degree of the graph G that specifies the communication linkages in the network. Then, starting with an empty system, for all $t \geq 1$ we have*

$$\hat{L}^t(\mathcal{P}) = \sum_{i=1}^{n} \hat{\ell}_i^t \leq 3\Delta n^3.$$

We will prove this theorem by first giving a series of preliminary results. The proof of Theorem 2.1 uses a similar potential function as the one that was used in [3] (though what follows is very different). This idea is to prove an invariant that for all $t \geq 1$, every subset $S \subseteq \mathcal{P}$ satisfies the following inequality:

$$\hat{L}^t(S) \leq \sum_{i=n-|S|+1}^{n} i \cdot (4\Delta) \cdot n. \tag{1}$$

Then, Inequality (1) will immediately imply Theorem 2.1 (by taking $S = \mathcal{P}$).

We will often have occasion to refer to the right hand side of Inequality (1) for many sets, so to make our proofs that follow easier to read, we define the following function $f : \{1, \ldots, n\} \to \mathbb{Z}$ in this way

$$f(k) = \sum_{i=n-k+1}^{n} i \cdot (4\Delta) \cdot n. \tag{2}$$

Definition 2.2 *In what follows, we will refer to sets as being* bad *after load generation in round t, or after the load balancing phase of round t, etc., meaning that the load of the set* at that particular time *violates Inequality (1). For example, if we say that a set S is bad after load generation in round t, we mean that* $\bar{L}^t(S) > f(|S|)$.

Conversely, we will also refer to a set as being good *(after load generation, or load balancing, etc.) if it satisfies Inequality (1) (at the time in question).*

The first lemma we give is one that might, at first glance, be "obvious". This result states that if we consider any (non-empty) set S at the end of round t, there must have existed a set S' so that the load of S' *before* load balancing was at least as large as the load of S after load balancing, i.e. $\bar{L}^t(S') \geq L^t(S) \geq \hat{L}^t(S)$. The fact that might not be obvious is that we can assert that the two sets contain the same number of processors. We prove this in the following lemma.

Lemma 2.3 *Let $\emptyset \neq S \subseteq \mathcal{P}$ denote an arbitrary subset of processors. There exists a set $|S'|$ such that*

1. $|S'| = |S|$, *and*
2. $\bar{L}^t(S') \geq L^t(S)$.

Proof: The claim is clear if $S = \mathcal{P}$, since in this case we have $L^t(\mathcal{P}) \geq \hat{L}^t(\mathcal{P})$ and $\bar{L}^t(\mathcal{P}) = L^t(\mathcal{P})$. Taking $S' = \mathcal{P}$ then satisfies the conclusions of the theorem.

So we suppose that S is not the entire set of processors. In this case let $S_{in} = \{v : v \in S \text{ and } \exists w \notin S \text{ such that } \alpha_{wv}^t > 0)\}$. In other words, S_{in} is the subset of S consisting of processors that received tasks from *outside* of S during load balancing.

Case 1: $S_{in} = \emptyset$. This case is essentially the same as when $S = \mathcal{P}$. Since no processors in S received load from outside of S, the elements of S can only exchange load among themselves or send load to processors outside of S. Then it is clear that $\bar{L}^t(S) \geq L^t(S)$, so taking $S' = S$ again satisfies the desired conclusions.

Case 2: $S_{in} \neq \emptyset$. Let $R = \{w : w \notin S \text{ and } \exists v \in S_{in} \text{ such that } \alpha_{wv}^t > 0\}$. In other words, R is the set of nodes *not* in S that pushed tasks into S during load balancing. The main idea of what follows is that we are going to swap some elements of R for elements of S_{in} on a one-for-one basis to find the set S' we desire. More formally, let $L_{in} = \sum_{w \in R, v \in S_{in}} \alpha_{wv}^t$ denote the total flow from R to S during load balancing. We aim to find sets $R_1 \subseteq R$ and $S_1 \subseteq S_{in}$ with

1. $|R_1| = |S_1|$, and
2. $\bar{L}^t(R_1) \geq L^t(S_1) + L_{in} + (\text{flow from } S_1 \text{ to } S \backslash S_1)$.

Then we will take $S' = S \backslash S_1 \cup R_1$. Our choice of the set R_1 guarantees that S' will satisfy $\bar{L}^t(S') \geq L^t(S)$, since the elements of R_1 account for all flow that enters S during load balancing, plus all flow that passes from elements in S_1 to elements in $S \backslash S_1$ as well.

To do this, let $E_1 = \{(w, v) : w \in R, v \in S_{in}, \alpha^t_{vw} > 0\}$. Consider an edge $e_1 = (w_1, v_1) \in E_0$ where $\alpha^t_{e_1}$ is largest. Then, from the definition of α^t_{wv}, we see that $\bar{\ell}^t_{w_1} \geq 2\Delta\alpha^t_{w_1v_1} + \bar{\ell}^t_{v_1}$. The key observation is that by choosing the largest edge, the expression $\bar{\ell}^t_{w_1}$ accounts for all possible load that v_1 could have received during load balancing, *and* all tasks that w_1 pushes into the set S too (and any tasks that v_1 might happen to pass to other elements in S, since this is counted in the term $\bar{\ell}^t_{v_1}$). We set $R_1 := \{w_1\}$ and $S_1 := \{v_1\}$, and $E_2 = E_1 \backslash (\{(w_1, v') : v' \in S_{in}\} \cup \{(w', v_1) : w' \in R\})$.

Then, we iteratively apply this argument, namely take a largest edge $e_2 = (w_2, v_2) \in E_2$. (Note that $w_2 \neq w_1$ and $v_2 \neq v_1$.) The choice of largest edge then allows us to swap w_2 for v_2, again accounting for all tasks that w_2 pushes into S during load balancing, all tasks that v_2 receives, and any tasks that v_2 passes to other elements in S. Then, we add w_2 to R_1, i.e. $R_1 := R_1 \cup \{w_2\}$, add v_2 to S_1, so $S_1 := S_1 \cup \{v_2\}$, and delete the appropriate set of edges from E_1. Thus, $E_2 = E_1 \backslash (\{(w_2, v') : v' \in S_{in}\} \cup \{(w', v_2) : w' \in R\})$.

We continue to iterate this procedure, selecting an edge with largest α^t_{wv} value, and performing an exchange as before, until we finish step k with a set $E_k = \emptyset$. It is possible that this procedure terminates at a step when $R_1 = R$ or $S_1 = S_{in}$ (or both), or with one or both of R_1, S_1 being proper subsets of their respective sets. In any case, we have constructed sets R_1 and S_1 (each with $k \leq \min\{|S_{in}|, |R|\}$ elements), so that by taking $S' = (S \backslash S_1) \cup R_1$, this set S' satisfies the two conditions of the theorem. □

Rereading the previous proof, we see that we have proven an inequality about the load of the sets of highest loaded processors, before and after load balancing (which, of course, need not be equal to each other). Thus we can conclude the following result:

Corollary 2.4 *For $i \in [n]$, let $\bar{M}^t_i$ denote a set of i largest loaded processors before load balancing (in round t). Also let M^t_i denote a corresponding set of i largest loaded processors after load balancing. Then $\bar{L}^t(\bar{M}^t_i) \geq L^t(M^t_i)$.*

We also conclude another result from Lemma 2.3.

Corollary 2.5 *Fix $i \in \{1, \ldots, n\}$. Suppose that every subset with i processors is good after the load generation phase of round t. Then, after the load balancing phase (and thus after the task deletion phase), every subset with i processors is still good. (Of course, provided that $\bar{M}^t_i$ is good after load generation, we actually get the same conclusion from Corollary 2.4.)*

Our next result tells us that if a set is made bad by load generation, then the load balancing and deletion phases are sufficient to make that set good again.

Lemma 2.6 *Suppose that at the end of round t, every set $S \subseteq \mathcal{P}$ satisfies (1). Further, suppose that after the load generation phase in round $t+1$, there is some set $S \subseteq \mathcal{P}$ such that $\bar{L}^{t+1}(S) > f(|S|)$. Then, at the end of round $t+1$, S again satisfies Inequality (1).*

Proof: If there is more than one set S such that $\bar{L}^{t+1}(S) > f(|S|)$, we may apply the argument that follows to each, so we fix one of the possible sets S. Suppose that $x \in \{1, \ldots n\}$ denotes the number of tasks that were injected into this set during load generation in round $t+1$.

We first show that

$$\text{if } P_j \in S \text{ then } \bar{\ell}^{t+1}_j \geq (n - |S| + 1)(4\Delta)n - x. \tag{3}$$

In the case when $S = \{P_i\}$ for some i (that is, $|S| = 1$), this statement is clear, since we must have $\bar{\ell}^t_i > n(4\Delta)n$ to violate Inequality (1).

When $|S| \geq 2$ we can prove (3) by contradiction. So assume that some $P_j \in S$ satisfies $\bar{\ell}_j^{t+1} < (n - |S| + 1)(4\Delta)n - x$. Since S was good before load generation, but not after, we know that $\bar{L}^{t+1}(S) - f(|S|) > 0$. Then, using that $\bar{L}^{t+1}(S \backslash P_j) = \bar{L}^{t+1}(S) - \bar{\ell}_j^{t+1}$, and our assumption on $\bar{\ell}_j^{t+1}$, we conclude

$$\begin{aligned} \bar{L}^{t+1}(S \backslash P_j) &> \bar{L}^{t+1}(S) - (n - |S| + 1)(4\Delta)n + x \\ \bar{L}^{t+1}(S \backslash P_j) - f(|S \backslash P_j|) &> \bar{L}^{t+1}(S) - f(|S \backslash P_j|) - (n - |S| + 1)(4\Delta)n + x \\ \bar{L}^{t+1}(S \backslash P_j) - f(|S \backslash P_j|) &> \bar{L}^{t+1}(S) - f(|S|) + x > x. \end{aligned}$$

Since we injected x tasks into S during the load generation phase of round $t+1$, we know that $\bar{L}^{t+1}(S \backslash P_j) \leq \hat{L}^t(S \backslash P_j) + x$. Putting this together with our last inequality above, we see that

$$\begin{aligned} \hat{L}^t(S \backslash P_j) + x - f(|S \backslash P_j|) &\geq \bar{L}^{t+1}(S \backslash P_j) - f(|S \backslash P_j|) \;>\; x \\ \Longrightarrow \hat{L}^t(S \backslash P_j) - f(|S \backslash P_j|) &> 0. \end{aligned}$$

This is a contradiction to the assumption stated in the hypothesis that all sets satisfied (1) at the end of round t. Hence, we conclude what we wanted to show, namely Inequality (3).

If $S = \mathcal{P}$, then our lemma follows immediately. In this case, the lower bound in (3) is also a lower bound on the load of each processor after the load balancing phase, i.e. $\ell_i^t \geq (4\Delta)n - n > 0$ for all P_i (since $x = n$ when $S = \mathcal{P}$). Thus, each processor will delete one task during the deletion phase. Since we injected at most n tasks into the system and deleted n tasks, the set $S = \mathcal{P}$ again satisfies (1), and we are done.

So, we now assume that $S \neq \mathcal{P}$. Then, in a similar manner as before, we can show

$$\text{if } P_j \notin S, \text{ then } \bar{\ell}_j^{t+1} \leq (n - |S|)(4\Delta)n + n. \tag{4}$$

To see this, again assume the contrary, so that some $P_j \notin S$ satisfies $\bar{\ell}_j^{t+1} > (n - |S|)(4\Delta)n + n$. Then we have the following inequalities

$$\hat{L}^t(S \cup P_j) + n \;\geq\; \bar{L}^{t+1}(S \cup P_j) \tag{5}$$

$$\bar{L}^{t+1}(S \cup P_j) - f(|S \cup P_j|) \;>\; \bar{L}^{t+1}(S) - f(|S|) + n. \tag{6}$$

Inequality (5) holds simply because we insert n tasks into the system, and Inequality (6) follows by breaking up the difference on the left hand side into constituent parts, and using our assumption about $\bar{\ell}_j^{t+1}$. These inequalities together imply

$$\hat{L}^t(S \cup P_j) - f(|S \cup P_j|) + n \;\geq\; \bar{L}^{t+1}(S) - f(|S|) + n \tag{7}$$

$$\hat{L}^t(S \cup P_j) - f(|S \cup P_j|) \;\geq\; \bar{L}^{t+1}(S) - f(|S|) \;>\; 0. \tag{8}$$

The final inequality in (8) comes from our assumption that $\bar{L}^{t+1}(S) > f(|S|)$. Of course, (8) violates the hypothesis of the theorem stating that all sets satisfied Inequality (1) at the end of round t. Hence, we obtain the upper bound on the load of elements not in S, as expressed in (4).

The rest of this lemma is a simple calculation. We first note that no load will be passed from $\mathcal{P} \backslash S$ into S during the load balancing phase because of the load differences in the processors. Then, since our network G is connected, there must be an edge (i, j) with $P_i \in S$ and $P_j \notin S$. Using our bounds (3) and (4) for $\bar{\ell}_i^t$ and $\bar{\ell}_j^t$, respectively, we find that

$$\alpha_{ij}^{t+1} \geq \frac{\bar{\ell}_i^{t+1} - \bar{\ell}_j^{t+1}}{2\Delta} - 1 \geq \frac{4\Delta n - n - x}{2\Delta} - 1 \geq 2n - \frac{n}{\Delta} - 1 \geq \frac{3}{2}n - 1.$$

The last two inequalities use the facts that $x \le n$ and $\Delta \ge 2$. We see this final ratio is at least n (with our assumption that $n \ge 2$). Hence, during round $t+1$, at most n tasks were injected into the set S during load generation, and at least n tasks were removed from S during the load balancing phase (and none were inserted into S during this phase). Therefore, after load balancing (and thus also after the task deletion phase) S again satisfies Inequality (1). $\square$

Lemma 2.6 tells us that if a set is made bad by the load generation phase, then the load balancing and deletion phases are sufficient to make this set good. The essential task that remains to be shown is that load balancing cannot, in some way, change a good set into a bad one. Corollary 2.5 tells us half the story. We need a little more to cover all possible sets.

Lemma 2.7 *Suppose that at all sets are good at the end of round t, but that after load generation in round $t+1$, there exists a bad set S with $|S| = i$. Then after load balancing and deletion, there exists no bad set with i processors.*

Proof: Without loss of generality, we can assume that $S = \bar{M}_i^t$, the largest i processors. Lemma 2.6 tells us that S is not bad at the end of round $t+1$. We therefore have to show that we do not somehow change a good set (of i processors) into a bad set during the load balancing phase. This proof is similar in flavor to that of Lemma 2.3, except that the argument is somewhat more delicate in this case.

Since we injected at most n tasks into the set S to change S from a good set into a bad set, we know that $\bar{L}^{t+1}(S) - n \le f(|S|)$. Our goal now is to show that any set S' of i processors will satisfy $L^{t+1}(S') \le \bar{L}^{t+1}(S) - n$, meaning that S' is good after load balancing.

So with this mind, fix some set S' where $|S'| = i$. We assume that $S' \neq S$, otherwise by Lemma 2.6 there is nothing to prove. Define the following sets:

$$S_{common} = S \cap S' \quad S_{old} = S \backslash S_{common} \quad S_{new} = S' \backslash S_{common}.$$

We note that $|S_{new}| = |S_{old}| \ge 1$. From our previous argument in Lemma 2.6, we know that the load difference (after generation, but before balancing) of any pair of processors, one from S and one from $\mathcal{P} \backslash S$, is at least $4\Delta n - 2n$.

In order to show our result, we will consider the load balancing actions of round $t+1$ in three stages. We first compute (and fix) the values of $\alpha_{i,j}^{t+1}$. Then we proceed this way:

Stage 1. Internal load balancing actions among processors of S, and among processors of $\mathcal{P} \backslash S$. After this stage, the load difference between a pair of processors, one from S and one from $\mathcal{P} \backslash S$ is still at least $4\Delta n - 2n$.

Stage 2. Processors in S_{old} balance with those in S_{new}. This can only move load from S_{old} to S_{new} because of the high load difference between processors of these two sets.

Stage 3. All remaining load balancing actions are performed. Which ones remain? Because there are no balancing actions from $S_{new} \subseteq \mathcal{P} \backslash S$ into $S_{common} \subseteq S$, the only remaining ones are

(a) S_{common} to S_{new},

(b) S_{old} to $\mathcal{P} \backslash (S' \cup S_{old})$, and

(c) S_{common} to $\mathcal{P} \backslash (S' \cup S_{old})$.

The balancing actions of (a) and (b) do not change the load of $S' = S_{common} \cup S_{new}$, and those of (c) can only decrease the load of S'. Hence, if we can show the load of S' after Stage 2 is at most $\bar{L}^{t+1}(S) - n$, then we get the conclusion we want.

To this end, let $L_1(S_{new})$ denote the load of S_{new} after Stage 1, and $L_2(S_{new})$ the load after Stage 2 (and similarly for other sets S_{old}, S, etc.). Let $A = \sum_{j \in S_{old}, k \in S_{new}} \alpha_{j,k}^{t+1}$ denote the total load transferred during

Stage 2 from S_{old} to S_{new}, and let B denote the load that remains in S_{old} after Stage 2. We note the following equations hold:

$$\begin{aligned}
L_2(S') &= L_2(S) + L_2(S_{new}) - L_2(S_{old}) \\
L_1(S_{old}) &= A + B \\
L_2(S_{old}) &= B \\
L_2(S_{new}) &= L_1(S_{new}) + A \\
L_2(S) &= L_1(S) - A.
\end{aligned}$$

All of these equations together imply that

$$\begin{aligned}
L_2(S') &= L_1(S) - A + L_1(S_{new}) + A - B \\
&= L_1(S) + L_1(S_{new}) - B \\
&= L_1(S) + L_1(S_{new}) + A - L_1(S_{old}).
\end{aligned}$$

Since Stage 1 did not change the total load of S (so $L_1(S) = \bar{L}^{t+1}(S)$), if we can show that

$$L_1(S_{new}) + A - L_1(S_{old}) \leq -n \tag{9}$$

we obtain our desired result. Having arrived at the crux of the problem, we now demonstrate Inequality (9).

First note that if, in fact, there are no edges from S_{old} to S_{new}, then $A = 0$. In this case, if we pair the vertices from S_{old} with those from S_{new}, then Inequality (9) follows immediately using the fact that the load difference of processors in S_{old} and S_{new} is at least $4\Delta n - 2n$.

Suppose there is at least one edge from S_{old} to S_{new}. Because of the load difference of processors in S_{old} and S_{new}, we see that any edge for which $\alpha_{j,k}^{t+1}$ is positive, we in fact have that $\alpha_{j,k}^{t+1} \geq n$.

Consider the subgraph G' that consists of processors in S_{old} and S_{new} and edges which were used to pass load from S_{old} to S_{new} during Stage 2. Choose an edge from G' such that the value of $\alpha_{j,k}^{t+1}$ is maximised. Assume (for simplicity) that $j = 1$ and $k = 2$. As in Lemma 2.3, we conclude that $\bar{\ell}_1^{t+1} \geq 2\Delta\alpha_{1,2}^{t+1} + \bar{\ell}_2^{t+1}$. Define $A_{1,2} = \sum_{k \in S_{new}} \alpha_{1,k}^{t+1} + \sum_{j \in S_{old}} \alpha_{j,2}^{t+1}$, the total flow out of P_1 (into S_{new}) and into P_2 (from S_{old}). Since $\alpha_{1,2}^{t+1}$ has maximum value over edges, we see that $\bar{\ell}_1^{t+1} \geq 2\Delta\alpha_{1,2}^{t+1} + \bar{\ell}_2^{t+1} \geq A_{1,2} + \bar{\ell}_2^{t+1}$. Hence, we see that $\bar{\ell}_2^{t+1} + A_{1,2} - \bar{\ell}_1^{t+1} \leq 0$. Indeed, if at least one of P_1 and P_2 has degree strictly smaller than Δ in G', this difference is smaller than or equal to $-n$, which is what we want on the right hand side of Inequality (9)!

In either case, consider the subgraph G'' obtained from G' by deleting the processors P_1, P_2, and all edges adjacent to them. As before, if there are no edges, we can pair the remaining processors however we like, and then we get the desired inequality. Otherwise, if we can show that $L_1(S_{new} \backslash P_2) + (A - A_{1,2}) - L_1(S_{old} \backslash P_1) \leq -n$ we again have shown Inequality (9).

The point is that we can proceed in an inductive manner as before, until we either find a pair $P_j \in S_{old}, P_k \in S_{new}$ where P_j sent load to P_k during Stage 2 *and* one of P_j and P_k has degree (in the remaining subgraph of G') that is strictly less than Δ (in which case $\bar{\ell}_k^{t+1} + A_{j,k} - \bar{\ell}_j^{t+1} \leq -n$), or we obtain a subgraph that has processors remaining, but no edges (and in this case we pair up the remaining processors however we like, and the large load difference between processors in the two sets gives us Inequality (9)). Whatever occurs, we can pair up processors in a one-to-one fashion to prove Inequality (9), and thus, our lemma. □

Now we are prepared to prove our main result.

Proof: [Theorem 2.1]

We prove this theorem by induction on t. Inequality (1) holds when $t = 1$, for however we inject the first n tasks into the system, all sets are good at the end of the first round.

So assume that at the end of round t, all sets are good. Fix $i \in \{1, \ldots, n\}$. If all sets of i processors are good after the load generation phase, then from Corollary 2.5 they are all good at the end of round $t + 1$.

If there is some bad set of i processors after load generation, then Lemmas 2.6 and 2.7 show that all sets of size i are still good at the end of round $t+1$.

Finally, it is not possible that during load balancing a (good or bad) set of i processors will lead to the creation of a bad set of $j(\neq i)$ processors. For suppose there is some bad set of $j(\neq i)$ processors at the end of round $t+1$. Lemma 2.3 tells us that there must exist a set of j processors that was bad before the load balancing phase, but then Lemmas 2.6 and 2.7 again tell us that there is no bad set of j processors at the end of round $t+1$, a contradiction to our assumption that there was a bad set of j processors at the end of the round. □

3 A Matching Lower Bound

In this section we provide a simple example that asymptotically matches the upper bound from Section 2. Consider the linear array $G = (V, E)$ with $V = \{0, \dots, n-1\}$ and $E = \{(i, i+1) | 0 \leq i < n-1\}$. Furthermore, suppose that during every time step, n new tasks are generated on node $n-1$. The idea of the proof essentially follows from a few simple observations, which we state without formal proof.

Observation 3.1

1. *Clearly, the system must be periodic since it is stable and there is a finite number of possible configurations it can be in, i.e., there is a "run-in" phase during which load is being built up (essentially, load is being distributed from node $n-1$ to all other nodes), followed by periodical behaviour.*
2. *Suppose the period length is T. Then we see that once the system is periodic, during any T steps, node i ($i > 0$) must send exactly $T \cdot i$ many tasks to nodes $i-1, \dots, 0$, because that is just the number of tasks that those nodes delete, i.e., node i sends i many tasks on average during any of those time steps (it does, in fact, send* exactly *i tasks to node $i-1$, thus $T = 1$; more about that later).*
3. *Another obvious fact is that once the system has finished the initial run-in phase, every processor must delete one task in very time step. If that were not the case, the system could not possibly be stable (we would delete strictly fewer tasks that are generated per period, i.e., the system load would increase by at least one during every period).*
4. *In our setting, load will never be sent towards processor $n-1$.*

Note that Theorem 3.2 implies that the preceeding analysis of our algorithm is tight up to a multiplicative constant, because the line graph has maximum degree $\Delta = 2$, and thus we have an upper bound of $O(n^3)$ on the system load.

Theorem 3.2 *The system described above on the linear array is stable with a total steady-state system load of $\Theta(n^3)$.*

Proof: We begin by showing that node i will never send more than i tasks to node $i-1$; the proof is by induction on time. The claim is trivially true in time step 1. Let α_i^t denote the number of tasks that processor i sends to processor $i-1$ in time step t. (We may extend the definition to $\alpha_n^t = n$ and $\alpha_0^t = 0$ for all t.) Suppose the claim holds for some $t-1 > 1$, i.e., $\alpha_i^{t-1} \leq i$ for $n-1 \geq i > 0$. Let ℓ_i^t denote node ℓ_i's load before the balancing in step t, $0 \leq i < n$. Clearly, for large enough values of t we have $\ell_i^t = \ell_i^{t-1} + \alpha_{i+1}^{t-1} - \alpha_i^{t-1} - 1$ and $\ell_{i-1}^t = \ell_{i-1}^{t-1} + \alpha_i^{t-1} - \alpha_{i-1}^{t-1} - 1$ (see Observation 3.1 (3)). Using the facts that

$$\alpha_i^{t-1} = \left\lfloor \frac{\ell_i^{t-1} - \ell_{i-1}^{t-1}}{4} \right\rfloor \qquad \text{and} \qquad \frac{\ell_i^{t-1} - \ell_{i-1}^{t-1}}{4} \leq \left\lfloor \frac{\ell_i^{t-1} - \ell_{i-1}^{t-1}}{4} \right\rfloor + \frac{3}{4},$$

we can conclude that

$$\begin{aligned}
\alpha_i^t &= \left\lfloor \frac{\ell_i^t - \ell_{i-1}^t}{2\Delta} \right\rfloor \le \frac{\ell_i^t - \ell_{i-1}^t}{2\Delta} = \frac{\ell_i^t - \ell_{i-1}^t}{4} \\
&= \frac{(\ell_i^{t-1} + \alpha_{i+1}^{t-1} - \alpha_i^{t-1} - 1) - (\ell_{i-1}^{t-1} + \alpha_i^{t-1} - \alpha_{i-1}^{t-1} - 1)}{4} \\
&= \frac{\ell_i^{t-1} - \ell_{i-1}^{t-1}}{4} + \frac{\alpha_{i+1}^{t-1} - 2\alpha_i^{t-1} + \alpha_{i-1}^{t-1}}{4} \le \left\lfloor \frac{\ell_i^{t-1} - \ell_{i-1}^{t-1}}{4} \right\rfloor + \frac{3}{4} + \frac{\alpha_{i+1}^{t-1} - 2\alpha_i^{t-1} + \alpha_{i-1}^{t-1}}{4} \\
&= \alpha_i^{t-1} + \frac{3}{4} + \frac{\alpha_{i+1}^{t-1} - 2\alpha_i^{t-1} + \alpha_{i-1}^{t-1}}{4} = \frac{2\alpha_i^{t-1} + \alpha_{i+1}^{t-1} + \alpha_{i-1}^{t-1}}{4} + \frac{3}{4} \\
&\le \frac{2i + (i+1) + (i-1)}{4} + \frac{3}{4} = i + \frac{3}{4}
\end{aligned}$$

From the above we know that node i will never send more than i tasks to node $i-1$ per time step (fractional tasks do not exist in our model). However, in order to obtain stability, at least i tasks on average are necessary. Thus, we can conclude that once the system is run in, node i will always send i tasks to node $i-1$, i.e., the system is in fact periodic with period length $T = 1$. Clearly, there are many possible fixed points with this property. However, since we are interested in a lower bound, we pick the one with smallest total load, i.e., the one in which node 0 is empty at the end of a step, receives one tasks from node 1, deletes it, and so on. Since a load difference of $2\Delta i = 4i$ implies i tasks being sent, this means that, directly before balancing, node i has a load of $\sum_{j=0}^{i} 4j = 2i(i+1)$, and thus the total system load is $\sum_{i=0}^{n-1} 2i(i+1) = (2n^3 - 2n)/3$. Together with the upper bound of $3\Delta n^3 = 6n^3$ we get the statement of the theorem. □

4 Some Instability Results for Work Stealing

In this section we will consider a variation of our load balancing process where we may transfer tasks to empty processors only. This approach is similar to the diffusion approach, only the computation of the $\alpha_{i,j}^t$ is different. The value of $\alpha_{i,j}^t$, the load that is sent from P_i to P_j, is larger than zero iff P_j is empty (and P_i non-empty). This method is referred to as *work stealing*.

$$\alpha_{i,j}^t = \begin{cases} \lfloor \frac{\bar{\ell}_i^t}{\Delta+1} \rfloor & : \quad \bar{\ell}_j^t = 0 \\ 0 & : \quad \text{otherwise} \end{cases}$$

Note that the bounds below also hold when we divide by 2Δ instead of $\Delta + 1$. We use the above definition as worst case assumption. In [7] the authors showed that simple work stealing yields a stable system. They assumed that there are at most $(1-\epsilon)n$ new tasks generated per round, for some $\epsilon \in (0, 1]$. The important point to note is that in [7], the processor communication links correspond to a complete graph on n vertices. Here we will see that the work stealing method can fail (in the sense that the total load is unbounded over time) if the graph is no longer the complete graph.

We consider the line network. In a line, we have an edge between node P_i and P_{i+1} for $1 \le i \le n-1$. Hence, the maximum degree is 2.

Observation 4.1 *Assume we have n processors connected as a line and n generators are all on processor 1. Then the diffusion work stealing system is not stable.*

Proof: Let us assume the system is in a state where P_2 is empty and P_1 has k tasks directly before the balancing. Then it will transfer $k/3$ tasks to P_2 during the load balancing step. It is easy to see that it will take at least

$$t = \frac{k}{3(n-1)} + \sum_{i=1}^{n-2} i = \frac{k}{3(n-1)} + \frac{n^2-4n-3}{2}$$

time steps until P_2 is empty again. To see that, assume that all other processors are empty. Then it takes $n-2$ steps until load will reach P_n, it takes $n-3$ time steps until load will reach P_{n-1}, and so on. In the meantime, the load of P_1 increases by $t(n-1)$ tasks. Thus, the load of P_1 after t steps is at least

$$k - \frac{k}{3} + \left(\frac{k}{3(n-1)} + \frac{n^2-4n-3}{2}\right)(n-1) \geq k.$$

This shows that the load of P_1 increases between any two consecutive balancing actions. □

The next observation shows that already very small networks are not stable under adversarial injections.

Observation 4.2 *Assume we have a network with a pair of nodes u and v that are not connected by an edge. Let assume that the degree of u is not larger that the degree of v, and let δ be degree of u. Then the work stealing system is not stable under an adversarial load generation scheme that generates $\delta + 2$ tasks per round.*

Proof: Simply allocate 2 generators on node u and one generator on every of the δ neighbours of u. Then none of the neighbours will ever balance with u and the load of u will increase by one per round. □

Similar to the observation above it is easy to show that the system is not stable under a wide class of randomised injection patterns. Define the process in a way that the expected load of u increases between two load balancing actions.

References

[1] W. Aiello, E. Kushilevitz, R. Ostrovsky, and A. Rosen. Adaptive packet routing for bursty adversarial traffic. *J. Computer and Systems Sciences* **60** (2000), pp. 482–509.

[2] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC 1993)*, pp. 632–641.

[3] E. Anshelevich, D. Kempe, and J. Kleinberg. Stability of load balancing algorithms in dynamic adversarial systems. *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002)*, pp. 399–406.

[4] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2001)*, pp. 158–167.

[5] B. Awerbuch and T. Leighton. A simple local control algorithm for multi-commodity flow. *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science (FOCS 1993)*, pp. 459–468.

[6] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. *Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC 1994)*, pp. 487–496.

[7] P. Berenbrink, T. Friedetzky, and L.A. Goldberg. The natural work-stealing algorithm is stable. *SIAM Journal of Computing, SICOMP 32(5) 1260-1279 (2003)*

[8] J.E. Boillat. Load balancing and Poisson equation in a graph. *Concurrency: Practice and Experiences* **2** (1990), pp. 289–313.

[9] G. Cybenko. Load balancing for distributed memory multiprocessors. *J. Parallel and Distributed Computing* **7** (1989), pp. 279–301.

[10] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *J. Parallel Computing* **25** (1999), pp. 789–812.

[11] R. Elsässer and B. Monien. Load balancing of unit size tokens and expansion properties of graphs. *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2003)*, pp. 266–273.

[12] R. Elsässer, B. Monien, and R. Preis. Diffusion schemes for load balancing on heterogeneous networks. *Theory of Computing Systems* **35** (2002), pp. 305–320.

[13] B. Ghosh, F.T. Leighton, B.M. Maggs, S. Muthukrishnan, C.G. Plaxton, R. Rajaraman, A.W. Richa, R.E. Tarjan, and D. Zuckerman. Tight analyses of two local load balancing algorithms. *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC 1995)*, pp. 548–558.

[14] F.M. auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. *Algorithmica* **15** (1996), pp. 413–427.

[15] S. Muthukrishnan and R. Rajaraman. An adversarial model for distributed load balancing. *Proceedings of the 10th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 1998)*, pp. 47–54.

[16] D. Peleg and E. Upfal. The token distribution problem. *SIAM J. Computing* **18** (1989), pp. 229–243.

[17] Y. Rabani, A. Sinclair, and R. Wanka. Local divergence of Markov chains and the analysis of iterative load-balancing schemes. *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pp. 694–703.