On the Stability of Dynamic Diffusion Load Balancing*

Petra Berenbrink†
Simon Fraser University
School of Computing Science

Tom Friedetzky
University of Durham
Department of Computer Science

Russell Martin‡
University of Liverpool
Department of Computer Science

Abstract

We consider the problem of dynamic load balancing in arbitrary (connected) networks on n nodes. Our load generation model is such that during each round, n tasks are generated on arbitrary nodes, and then (possibly after some balancing) one task is deleted from every non-empty node. Notice that this model *fully saturates* the resources of the network in the sense that we generate just as many new tasks per round as the network is able to delete. We show that even in this situation 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. (It of course also works if we generate less than n new tasks per round, but the major contribution of this paper is the fully saturated case.) 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 in this setting.

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 utilization of resources. Of greatest importance here is an even distribution of the workload among the processors. In particular applications exposing some kind of "irregularity" require the use of a 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 ([11], [10]). The algorithm works in synchronized rounds. The basic idea is that in every round, every processor p balances load with all its neighbors (independently, i.e., pair-wise). Let ℓ_p be the load of p and ℓ_q the load of some of p's neighbor q, and let d_p denote the degree of the vertex p. One popular method in the discrete setting has p transferring $\max\{0, \lfloor(\ell_p - \ell_q)/(\max\{d_p, d_q\} + 1)\rfloor\}$ tasks to q in a given round. Some of many advantages of diffusion-type algorithms are the *locality* (no global knowledge regarding the overall

*A preliminary version of this paper entitled "Dynamic diffusion load balancing" was published in *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, *Lecture Notes in Computer Science 3580*, Springer-Verlag, pp. 1386–1398.

†Supported by NSERC Discovery Grant 250284-2002.

‡Supported by EPSRC grant "Discontinuous Behaviour in the Complexity of Randomized Algorithms."

load situation, or, in fact, anything except the strict neighborhood of any vertex is needed), its *simplicity*, and its *neighborhood preservation* (tasks tend to stay close to the processors where they are generated, which may help to maintain small communication overhead).

The diffusion load balancing algorithm has been thoroughly analyzed for *static* scenarios, where each processor has some initial number of tasks, and the objective is to distribute *this* load evenly among the processors as quickly as possible. Much work has been done under the assumption that every edge is only allowed to forward one task per round [15, 17, 21] or when a constant number of tasks can be passed by each processor [16]. We refer to these scenarios as *token distribution problems*. In addition, [12, 14] have studied the diffusion algorithm where tasks can be split arbitrarily, while [13] assumes that they are indivisible. Muthukrishnan, Ghosh, and Schultz [19] examined first- and second-order schemes for "coarse" load balancing (the goal being to reduce large discrepancies between the loads on vertices). Rabani, Sinclair, and Wanka [22] also considered the problem of coarse balancing. They model the load balancing process by a suitable Markov chain, and show this model is accurate until a certain threshold discrepancy is reached.

In contrast to the static case of load balancing and token distribution, in the *dynamic* setting during each round new tasks are generated (in some manner) on the set of processors, load is balanced amongst neighbors, then tasks are deleted from non-empty processors.

Much past work has studied the dynamic token distribution problem. Muthukrishnan and Rajaraman [20] studied a dynamic version where processors can forward a single task in each round. 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 [1]: A node sends a task to its neighbor if the load difference between them is at least $2\Delta + 1$, where Δ is the maximum degree of the underlying graph. 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. 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 [4] gave a positive result for token distribution when $\epsilon = 0$. They showed that under the above load generation model no processor has more than average load $\pm(2\Delta+1)\cdot n$. Anshelevich, et al. also showed how their result can be generalized for edges that can forward c tokens per time step. A node sends $\min\{c, \rho\}$ tasks to its neighbor if the load difference is at least $2\Delta c + \rho$. In this setting no processor has more than average load $\pm(2\Delta + 1)c \cdot n$ as long as the load change in every subset S of the nodes minus $a|S|$ is at most $c \cdot e(S)$. Additionally, they showed that a generalization of the algorithm is stable for two distinct types of jobs, and they extended their results to related flow problems.

In [6, 7] Awerbuch and Leighton use a variant of the token distribution model under the assumption that tokens can be split into arbitrarily sized parts. They use a "balancing" algorithm to approximate the multi-commodity flow problem with capacitated edges. Their method is an iterative approach where flow is queued at the vertices of the graph. In each step, the commodity which has the largest excess is shipped from one vertex to another, and then new flow is injected into the system. In this balancing process, edge capacities must always be respected. These edge capacities are analogous to the restrictions on the number of tasks that can be passed over any single edge in the token distribution problems. Furthermore, their model does not actually allow full use of those edge capacities, which is similar to the case in [20] where $\epsilon > 0$ was required to ensure stability. The work in [2] and [5] expands the results of Awerbuch and Leighton for packet routing, but again in these cases only a constant number of tasks can be moved across any edge in

a single time step.

Clearly the condition that processors can forward only a single task (or a constant number) per edge in each round significantly restricts the number and distribution of tasks that can be generated on (or deleted from) processors in each round and still obtain a stability result. Thus, in the results of [20] and [4] some dependence on the quantity $e(S)$ (or some measure of the "edge expansion") is to be expected.

Anagnostopoulos, Kirsch, and Upfal [3] consider the setting where there are no restrictions on the number of tasks balanced between processors in a time step, and they allow a broad range of injection models. Their protocol is similar to that studied in [16] for a static setting, but is not the typical diffusion load balancing procedure. In their setting, in each step nodes are matched randomly with adjacent neighbors and matched nodes equalize their load. Hence, every processor is only involved in a single load balancing action. They show that the system is stable as long as at most $wn\lambda$ tasks (in expectation) are generated in a time interval of length w, where $\lambda < 1$. Their proof method unfortunately cannot be generalized to the case of *full saturation* when $\lambda = 1$, which is the main focus of this paper.

In a different approach in the dynamic setting, Berenbrink, Friedetzky, and Mayr [9] consider a load balancing scheme that uses a "collision protocol" (see also [18]) to resolve load balancing requests amongst lightly- and heavily-loaded processors. They show stability with this balancing protocol and a variety of randomized task generation models where each processor receives, in expectation, strictly less than one newly generated task per round.

1.1 Our Results

In this paper we present the first analysis of the simple diffusion scheme for the dynamic load balancing problem that allows full saturation of the resources. We assume that n new tasks are generated per round and, after load balancing, every non-empty processor deletes one task each round. (With small modifications our proofs will carry through to the case when we generate *at most* n tasks per round.) In contrast to [4] and [20], the newly generated tasks may be arbitrarily distributed among the nodes of the network, regardless of any "edge expansion" type of condition as in those models. For example, the tasks may always be generated on the same processor, or all tasks may be generated on one processor but the processor can change from round to round, or alternatively, the tasks may be allocated at random each round. 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 the generation model described above. 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 $O(\Delta n^3)$, where Δ denotes the maximum degree of the network. Furthermore, we present a simple, asymptotically matching lower bound when the network is a path.

Our technique also captures a different scenario, similar to that in [4, 20], where stability is defined in terms of deviation of any processor's load from the average. In this scenario we have two separate phases, the first where tasks are generated on and/or deleted from nodes, and the second where tasks are then balanced amongst nodes. Let $\bar{L}^t(S)$ denote the total load of the nodes in the set S after the task generation/deletion phase, and $L^t(S)$ denote the total load of S after the balancing step at time t. Assume that the generation/deletion phase satisfies the following condition:

$$\bar{L}^t(S) - L^{t-1}(S) \leq (\mathrm{avg}(t) - \mathrm{avg}(t-1)) \cdot |S| + \rho \quad \text{for every subset } S,$$

where $\text{avg}(t)$ denotes the average system load in step t. Then the total load of S can be bounded by $|S| \cdot \text{avg}(t) + 5\Delta n\rho$.

For both proofs of our results we use a potential function. Although the potential function we use looks similar to the one used in [4], the proof technique is very different. The proof method in [4] very much relies upon the restriction of their generation/deletion model, where the number of tasks inserted into/deleted from a set S is bounded by a function of $e(S)$, the number of edges that join the set S to its complement $\bar{S}$. This, together with the bounded capacities on the edges of the graph, allows for a direct analysis of how the loads of sets might change in a single step of their process. The arbitrary distribution of tasks in our generation model and the unrestricted capacity of the edges in our network (i.e. unknown bounds on load transferred into a set S in a single step) does not allow us to directly obtain similar results, so we need a different proof to show stability under our model.

In the final part of our paper we discuss a different method of load balancing, one which is commonly referred to as *work stealing*. In this framework, processors that are empty after task generation will balance with processors that are not empty, but no other balancing actions are permitted. We show that for this work-stealing protocol there are graphs for which the system cannot be stable for a significant class of generation parameters. These results show that restricting balancing actions to empty processors is not sufficient in general.

In contrast, Berenbrink, Friedetzky, and Goldberg [8] showed stability of a work stealing algorithm under a load generation model that is similar to many of those already mentioned. They consider a flexible distribution of n generators among the nodes of the network, where each generator is allowed to generate a task with probability *strictly smaller* than one. In this setting a very simple, parameterized work-stealing algorithm achieves stability (in our sense) for a wide range of parameters. The important point to note is that their model applies only when the set of processors (and their communication linkages) forms a complete graph, and their results only hold for the case where strictly less than n tasks (in expectation) are generated during any time step.

Our model is defined in the next section, and the formal definition of the diffusion approach to load balancing is given following that.

1.2 Our Model

Our parallel system is modeled 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 modeled 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. n generators are arbitrarily distributed over the processors, and each generator generates one task at the beginning of every time round. For $1 \leq i \leq n$, 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 processor balances its load with some or all its neighbors in the network (according to a well-defined scheme for doing this operation).
3. 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 $\hat{L}^t(\mathcal{P}) = \sum_{i=1}^n \hat{\ell}_i^t$ that are in the system at the end of round t does not grow with time, i.e. the total load $\hat{L}^t(\mathcal{P})$ is bounded by a number that might depend on n, but not on the time t.

We will mainly focus on one load balancing method called the *diffusion approach*. Every processor is allowed to balance its load with all its neighbors. As mentioned previously, we briefly consider a second approach in Section 4 where only empty processors are allowed to take load from their non-empty neighbors. We call this second method the *work stealing approach*.

Diffusion approach. We begin with a detailed description of the first approach, an integral variant of the *First-Order Diffusion scheme* from [19]. 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). Recall that d_i denotes the degree of vertex i. Then $\alpha_{i,j}$ and ℓ_i are calculated as follows:

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

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

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

One of the "standard" diffusion approaches divides $\bar{\ell}_i^t - \bar{\ell}_j^t$ by $\max\{d_i, d_j\}+1$ instead of $2\max\{d_i, d_j\}$. We need the change for our analysis.

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 by $O(\Delta n^3)$. This generalizes the results of [4] to the case of unbounded edge capacities and, hence, analyzes the standard diffusion approach. Theorem 3.2 in Section 3 provides an asymptotically matching lower bound, showing that our upper bound is tight, up to a multiplicative constant. 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 instability.

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 connected graph G that specifies the communication linkages in the network. Then, starting with an empty system, for all $t \geq 1$ we have*

$$\hat{L}^t(\mathcal{P}) = \sum_{i=1}^{n} \hat{\ell}_i^t \leq 2\Delta n^2(n+1).$$

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 [4] (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 $f : \{1, \ldots, n\} \to \mathbb{N}$ 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 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. This is the statement of 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 $R = \{w : w \notin S \text{ and } \exists v \in S_{in} \text{ such that } \alpha^t_{wv} > 0\}$. 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 $L_{in} = \sum_{w \in R, v \in S_{in}} \alpha^t_{wv}$ denote the total flow from R to S during load balancing. We aim to find sets $R_k \subseteq R$ and $S_k \subseteq S_{in}$ with

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

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

To do this, let $E_1 = \{(w, v) : w \in R, v \in S_{in}, \alpha^t_{wv} > 0\}$. Consider an edge $e_1 = (w_1, v_1) \in E_1$ where $\alpha^t_{e_1}$ is largest. From the definition of $\alpha^t_{w_1 v_1}$, we see that $\bar{\ell}^t_{w_1} \geq 2 \max\{d_{w_1}, d_{v_1}\} \alpha^t_{w_1 v_1} + \bar{\ell}^t_{v_1}$. 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. Note also that the quantity $\bar{\ell}^t_{w_1}$ includes the number of tasks that v_1 might happen to pass to other elements in S, since this is included 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. set $R_2 := R_1 \cup \{w_2\}$, add v_2 to S_1, so $S_2 := S_1 \cup \{v_2\}$, and delete the appropriate set of edges from E_2. Thus, $E_3 := E_2 \backslash (\{(w_2, v') : v' \in S_{in}\} \cup \{(w', v_2) : w' \in R\})$.

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 $E_{k+1} = \emptyset$. It is possible that this procedure terminates at a step when $R_k = R$ or $S_k = S_{in}$ (or both), or with one or both of R_k, S_k being proper subsets of their respective sets. In any case, we have constructed sets R_k and S_k (each with $k \leq \min\{|S_{in}|, |R|\}$ elements), so that by taking $S' = (S \backslash S_k) \cup R_k$, this set S' satisfies the two conditions of the theorem. □

From the previous lemma, 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

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

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

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

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

$$\begin{aligned} \hat{L}^t(S \cup P_j) - f(|S \cup P_j|) + n &\geq \bar{L}^{t+1}(S) - f(|S|) + n & (7) \\ \hat{L}^t(S \cup P_j) - f(|S \cup P_j|) &\geq \bar{L}^{t+1}(S) - f(|S|) > 0. & (8) \end{aligned}$$

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

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

The last two inequalities use the facts that $x \leq n$ and $\Delta \geq 2$. We see this final ratio is at least n (with our assumption that $n \geq 2$). 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). □

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 $\bar{L}^{t+1}(S) - n \leq f(|S|)$. Our goal now is to show that any set S' of i processors will satisfy $L^{t+1}(S') \leq \bar{L}^{t+1}(S) - n$, 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 $|S_{new}| = |S_{old}| \geq 1$. 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 maximized. Assume (for simplicity) that $j = 1$ and $k = 2$. As in Lemma 2.3, we conclude that $\bar{\ell}_1^{t+1} \geq 2\max\{d_1, d_2\}\alpha_{1,2}^{t+1} + \bar{\ell}_2^{t+1}$. 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 $\bar{\ell}_1^{t+1} \geq 2\max\{d_1, d_2\}\alpha_{1,2}^{t+1} + \bar{\ell}_2^{t+1} \geq A_{1,2} + \bar{\ell}_2^{t+1}$. 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. □

On the first glance it might look as if the our proof strategy is overly complicated and that there is a much simpler proof. In the course of proving our result, we show that there is a gap of $4n\Delta$ tasks between a processor in the bad set S and a processor outside of the bad set before balancing whenever S is bad after balancing. Hence, at least n tasks were sent away from S in this step and the invariant could not have been violated by S. But unfortunately it is possible to create a different bad set of processors during load balancing (possibly with a different number of processors), and we have to discount this case too. Hence, we have to show that if we can find a bad set after load balancing, then there was another bad set S' before load balancing, which leads us to a contradiction through our series of lemmas above.

3 A Matching Lower Bound

In this section we provide a simple example that asymptotically matches the upper bound from Section 2. Fix some $n \geq 3$ and consider the linear array $G = (V, E)$ with $V = \{P_0, \ldots, P_{n-1}\}$ and $E = \{(P_i, P_{i+1}) | 0 \leq i < n - 1\}$. Furthermore, suppose that during every time step, n new

tasks are generated on processor P_{n-1}. The idea of the proof essentially follows from a few simple observations, which we state without formal proof.

Observation 3.1

1. *The system is periodic since it is stable and thus 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 processor P_{n-1} to all other processors), followed by periodical behavior (notice that we consider a strictly deterministic system).*

2. *Another obvious fact is that once the system has finished the initial run-in phase, every processor must delete one task in every round. 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).*

3. *Suppose the period length is T. Then we see that once the system is periodic, during any T rounds, processor P_i ($i > 0$) must send exactly $T \cdot i$ many tasks to processor P_{i-1} (some of which gets spread to the other processors $P_{i-2}, \ldots, P_0$), because that is just the number of tasks that processors $P_{i-1}, \ldots, P_0$ delete in T rounds. In other words, on average processor P_i sends i many tasks during any of those rounds (it does, in fact, send* exactly *i tasks to processor P_{i-1}, thus $T = 1$; more about that later).*

4. *In our setting, load will never be sent toward processor P_{n-1}.*

Theorem 3.2 below implies that the preceding 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 processor P_i will never send more than i tasks to processor P_{i-1}; the proof is by induction on time. The claim is trivially true in round 1. Let $\alpha_i^t = \alpha_{i,i-1}^t$ denote the number of tasks that processor P_i sends to processor P_{i-1} in round 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 all $i \in \{1, \ldots, n-1\}$. Let ℓ_i^t denote the load of processor P_i before the balancing step in round t, $0 \leq i < n$. From Observation 3.1 (2), 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 $\ell_{i-1}^t = \ell_{i-1}^{t-1} + \alpha_i^{t-1} - \alpha_{i-1}^{t-1} - 1$. 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

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

$$= \alpha_i^{t-1} + \frac{3}{4} + \frac{\alpha_{i+1}^{t-1} - 2\alpha_i^{t-1} + \alpha_{i-1}^{t-1}}{4} = \frac{2\alpha_i^{t-1} + \alpha_{i+1}^{t-1} + \alpha_{i-1}^{t-1}}{4} + \frac{3}{4}$$
$$\leq \frac{2i + (i+1) + (i-1)}{4} + \frac{3}{4} = i + \frac{3}{4}.$$

From the above we know that processor P_i will never send more than i tasks to processor P_{i-1} during each round (i.e. $\alpha_i^t \leq i$ since fractional tasks are not allowed 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", processor P_i will always send i tasks to processor P_{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 processor P_0 is empty at the end of a round, receives one task from processor P_1 in the next round, 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, the load of processor P_i is $\sum_{j=0}^{i} 4j = 2i(i+1)$, 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 $2\Delta n^2(n+1) = 4n^2(n+1)$ from Theorem 2.1 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*.

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

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 [8] 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 [8], 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 $1 \leq i \leq n - 1$. 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. □

In a similar manner, under adversarial injection schemes, it is easy to show that the work stealing protocol will not be stable for many classes of graphs, even under a randomized injection pattern. For example, we can simply define the process in a way such that the expected load of a processor increases between two load 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 neighbors of u. Then none of the neighbors 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 randomized injection patterns. Define the process in a way that the expected load of u increases between two load balancing actions.

5 A Different Model for Task Generation/Deletion

In this section we define a load generation model similar to [20] and [4]. Rather than bounding the total number of tasks that are generated per round, we bound the *load change* in any subset of the processors. During each round, tasks can be added or deleted from processors, subject to the restriction in Inequality (10) below. The processors then balance load amongst themselves as before.

In the following, $\bar{\ell}_i^t$ (respectively, $\bar{L}^t(S)$) denotes the load of processor P_i (resp. the total load of all processors in set the S) after we have generated and deleted tasks, and ℓ_i^t (resp. $L^t(S)$) is the load of processor P_i (resp. the total load of all processors in the set S) immediately after the load balancing phase. Let $\text{avg}(t)$ be the average load of the processors in round t after load generation and deletion, i.e. $\text{avg}(t) = \frac{1}{n} \cdot \sum_{i=1}^{n} \bar{\ell}_i^t$. Again, $L^t(\mathcal{P})$ denotes the total system load at the end of step t. One round looks now as follows:

1. Tasks are generated and deleted according to the following *generation restriction*:

$$\bar{L}^t(S) - L^{t-1}(S) \leq |S| \cdot (\text{avg}(t) - \text{avg}(t-1)) + n \quad \text{for every subset } S. \tag{10}$$

2. Every processor balances its load with some or all its neighbors in the network using the diffusion operation defined in Section 1.2.

We can show the following result.

Theorem 5.1 *Let $n \geq 2$ denote the number of processors in the system. Let $\Delta \geq 2$ denote the maximum degree of the connected graph G that specifies the communication linkages in the network. Assume the load generation and deletion fulfills the generation restriction in (10). Then, starting with an empty system, for all $t \geq 1$ and all $S \subseteq \mathcal{P}$ we have*

$$L^t(S) \leq |S| \cdot avg(t) + 5\Delta n^3.$$

Furthermore, the maximum number of tasks per processor is $avg(t) + 5\Delta n^2$.

Proof: The proof of this theorem follows the proof of Theorem 2.1. Here, we will concentrate on the parts that have to be changed compared to that proof. We redefine f as follows.

$$f(k) = \sum_{i=n-k+1}^{n} i \cdot (5\Delta) \cdot n. \tag{11}$$

Our new invariant is

$$L^t(S) \leq |S| \cdot \text{avg}(t) + f(|S|) = |S| \cdot \text{avg}(t) + \sum_{i=n-|S|+1}^{n} i \cdot (5\Delta) \cdot n. \tag{12}$$

Similar to the previous section, we call a set S *bad* if $L^t(S) > |S| \cdot \text{avg}(t) + f(|S|)$, and *good* otherwise. Since Lemma 2.3, Corollary 2.5, and Corollary 2.4 only depend on the load balancing scheme and not on the underlying load generation and deletion, they still can be applied. Because Lemma 2.6 depends on the actual load of the processors and, therefore, on the load generation model, we have to adjust it. The new version is presented below.

Lemma 5.2 *Suppose that at the end of round t, every set $S \subseteq \mathcal{P}$ satisfies (12). Further, suppose that after the load generation and deletion phase in round $t+1$, there is some set $S \subseteq \mathcal{P}$ such that $\bar{L}^{t+1}(S) > |S| \cdot avg(t+1) + f(|S|)$. Then, at the end of round $t+1$, S again satisfies Inequality (12).*

Proof: We only consider the parts of the proof that are different from the proof of Lemma 2.6.

We first show that

$$\text{if } P_j \in S \text{ then } \bar{\ell}_j^{t+1} \geq (n - |S| + 1)(5\Delta)n + \text{avg}(t+1) - n. \tag{13}$$

In the case when $S = \{P_i\}$ for some i (that is, $|S| = 1$), this statement is clear, since we must have $\bar{\ell}_i^{t+1} > n(5\Delta)n + \text{avg}(t+1)$ to violate Inequality (12).

As in Lemma 2.6, when $|S| \geq 2$ we can prove (13) by contradiction. So assume that some $P_j \in S$ satisfies $\bar{\ell}_j^{t+1} < (n-|S|+1)(5\Delta)n + \text{avg}(t+1) - n$. Since S was good before load generation, but not after, we know that $\bar{L}^{t+1}(S) - f(|S|) > |S| \cdot \text{avg}(t+1)$. 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

$$\begin{aligned}
\bar{L}^{t+1}(S \backslash P_j) &> \bar{L}^{t+1}(S) - (n - |S| + 1)(5\Delta)n - \text{avg}(t+1) + n \\
\bar{L}^{t+1}(S \backslash P_j) - f(|S \backslash P_j|) &> \bar{L}^{t+1}(S) - f(|S \backslash P_j|) - (n - |S| + 1)(5\Delta)n - \text{avg}(t+1) + n \\
\bar{L}^{t+1}(S \backslash P_j) - f(|S \backslash P_j|) &> \bar{L}^{t+1}(S) - f(|S|) - \text{avg}(t+1) + n > (|S| - 1) \cdot \text{avg}(t+1) + n.
\end{aligned}$$

On the other hand, Inequality (10) tells us that

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

Putting this together with our last inequality above, we see that

$$\begin{aligned} L^t(S\backslash P_j) + (|S|-1)\cdot(\text{avg}(t+1)-\text{avg}(t)) + n - f(|S\backslash P_j|) &\geq \bar{L}^{t+1}(S\backslash P_j) - f(|S\backslash P_j|) \\ &\geq (|S|-1)\cdot\text{avg}(t+1)+n \\ \implies L^t(S\backslash P_j) - f(|S\backslash P_j|) &> (|S|-1)\cdot\text{avg}(t). \end{aligned}$$

This is a contradiction to the hypothesis that all sets satisfied (12) at the end of round t. Hence, we conclude what we wanted to show, namely Inequality (13).

When $S = \mathcal{P}$, then our lemma follows immediately since the load of S is exactly $n \cdot \text{avg}(t+1)$. hence, we can assume that $S \neq \mathcal{P}$. Then, in a similar manner as before, we can show

$$\text{if } P_j \notin S, \text{ then } \bar{\ell}_j^{t+1} \leq (n-|S|)(5\Delta)n + \text{avg}(t+1) + n. \tag{14}$$

To see this, again assume the contrary, so that some $P_j \notin S$ satisfies

$$\bar{\ell}_j^{t+1} > (n-|S|)(5\Delta)n + \text{avg}(t+1) + n.$$

Then we have the following inequalities

$$L^t(S\cup P_j) + n + (\text{avg}(t+1)-\text{avg}(t))(|S|+1) \geq \bar{L}^{t+1}(S\cup P_j) \tag{15}$$

$$\bar{L}^{t+1}(S\cup P_j) - f(|S\cup P_j|) > \bar{L}^{t+1}(S) - f(|S|) + \text{avg}(t+1) + n. \tag{16}$$

Inequality (15) is due to the generation restriction. Inequality (16) 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

$$\begin{aligned} & L^t(S\cup P_j) - f(|S\cup P_j|) + (\text{avg}(t+1)-\text{avg}(t))(|S|+1) + n \\ \geq\ & \bar{L}^{t+1}(S) - f(|S|) + \text{avg}(t+1) + n \end{aligned} \tag{17}$$

$$\begin{aligned} & L^t(S\cup P_j) - f(|S\cup P_j|) + (\text{avg}(t+1)-\text{avg}(t))(|S|+1) \\ \geq\ & \bar{L}^{t+1}(S) - f(|S|) + \text{avg}(t+1) \ > \ 0 \end{aligned} \tag{18}$$

$$L^t(S\cup P_j) - f(|S\cup P_j|) + (\text{avg}(t+1)-\text{avg}(t))(|S|+1) \geq |S+1|\cdot\text{avg}(t+1) \tag{19}$$

$$\hat{L}^t(S\cup P_j) - f(|S\cup P_j|) \geq |S+1|\cdot\text{avg}(t). \tag{20}$$

Inequality in (18) comes from our assumption that $\bar{L}^{t+1}(S) > |S|\cdot\text{avg}(t+1) + f(|S|)$. Again, we have a contradiction and obtain the upper bound on the load of elements not in S, as expressed in (14).

Again, we have a load difference of at least $5\Delta n$ between processors on S and processors not in S. The rest of this lemma is a simple calculation and can be done similar to the one in Lemma 5.2. □

Lemma 2.7 only depends on the load difference of the processors and is still valid under the new load generation and deletion model. We have only to show that we still have

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

i.e. if we subtract n from the load of set S after load generation and deletion, set S is good again. This can be done as follows. Due to the generation restriction, we know that the load generated in

S is upper bounded by $|S| \cdot (\text{avg}(t+1) - \text{avg}(t)) + n$. We know that $L^t(S) \leq |S| \cdot \text{avg}(t) + f(|S|)$ since S was good at the end of round t. This gives us

$$\begin{aligned} & \bar{L}^{t+1}(S) \leq L^t(S) + |S| \cdot (\text{avg}(t+1) - \text{avg}(t)) + n \\ \leq \; & |S| \cdot \text{avg}(t) + f(|S|) + |S|(\text{avg}(t+1) - \text{avg}(t)) + n \end{aligned}$$

$$\begin{aligned} \bar{L}^{t+1}(S) &\leq |S| \cdot \text{avg}(t+1) + f(|S|) + n \\ \bar{L}^{t+1}(S) - n &\leq |S| \cdot \text{avg}(t+1) + f(|S|). \end{aligned}$$

Also, the remainder of the proof of Theorem 5.1 can be done similar to the proof of Theorem 2.1. □

5.1 Further Extensions

We can easily generalize our results to other load generation processes, and the proofs of the following results are much like those of Theorem 5.1.

Theorem 5.3 *Let $n \geq 2$ denote the number of processors in the system. Let $\Delta \geq 2$ denote the maximum degree of the graph G that specifies the communication linkages in the network. Assume the load generation and deletion fulfills the generation restriction*

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

Then, starting with an empty system, for all $t \geq 1$ and all $S \subseteq \mathcal{P}$ we have

$$L^t(S) \leq |S| \cdot avg(t) + 5\Delta n^2 K.$$

Furthermore, the maximum number of tasks per processor is $avg(t) + 5\Delta nK$.

Proof: Similar to the proof of Theorem 5.1. All we have to do is to define

$$f(k) = \sum_{i=n-k+1}^{n} i \cdot (5\Delta) \cdot K.$$

□

Furthermore, we can improve our results to a load generation model where the imbalance that we allow to be generated in any set depends on the number of outgoing edges.

Theorem 5.4 *Let $n \geq 2$ denote the number of processors in the system. Let $\Delta \geq 2$ denote the maximum degree of the graph G that specifies the communication linkages in the network. Let $e(S)$ be the number of outgoing edges of the set S. Assume the load generation and deletion fulfills the generation restriction*

$$\bar{L}^t(S) - L^{t-1}(S) \leq |S| \cdot (avg(t) - avg(t-1)) + K \cdot e(S).$$

Then, starting with an empty system, for all $t \geq 1$ and all $S \subseteq \mathcal{P}$ we have

$$L^t(S) \leq |S| \cdot avg(t) + 5\Delta n^2 K.$$

Furthermore, the maximum number of tasks per processor is $avg(t) + 5\Delta nK$.

Proof: Similar to the proof of Theorem 5.1. All we have to do is to define

$$f(k) = \sum_{i=n-k+1}^{n} i \cdot (5\Delta) \cdot K.$$

□

Acknowledgements

The authors wish to thank the anonymous referee for valuable comments.

References

[1] 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.

[2] 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.

[3] A. Anagnostopoulos, A. Kirsch, and E. Upfal. Stability and efficiency of a random local load balancing protocol. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*, pp. 472–481.

[4] 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.

[5] B. Awerbuch, P. Berenbrink, A. Brinkmann, and C. Scheideler. Simple routing strategies for adversarial systems. *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Coumputer Science (FOCS 2001)*, pp. 158–167.

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

[7] 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.

[8] P. Berenbrink, T. Friedetzky, and L.A. Goldberg. The natural work-stealing algorithm is stable. *SIAM J. Computing* **32** (2003), pp. 1260–1279.

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

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

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

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

[13] 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.

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

[15] 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.

[16] B. Ghosh and S. Muthukrishnan. Dynamic load balancing by random matchings. *J. Computer and Systems Science* **53** (1996), pp. 357–370.

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

[18] F.M. auf der Heide, C. Scheideler, and V. Stemann. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoretical Computer Science* **162** (1996) pp. 245–281.

[19] S. Muthukrishnan, B. Ghosh, and M.H. Schultz. First- and second-order diffusive methods for rapid, coarse, distributed load balancing. *Theory of Computing Systems* **31** (1998), pp. 331–354.

[20] 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.

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

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