Rapid Convergence of a Local Load Balancing Algorithm for Asynchronous Rings

Johannes E. Gehrke [1,4] C. Greg Plaxton [2,4] Rajmohan Rajaraman [3,4]

January 26, 1998

Abstract

We consider the problem of load balancing in a ring network. We present an analysis of the following local algorithm. In each step, each node of the ring examines the number of tokens at its clockwise neighbor and sends a token to the neighbor if the neighbor has fewer tokens. We show that in a synchronous model, for any initial token distribution b on an n-node ring, the algorithm converges to a completely balanced distribution within $4OPT(b)+n$ steps, where $OPT(b)$ is the time taken by the optimal centralized algorithm to balance b completely. Our main result is an analysis of the algorithm in an asynchronous model in which local computations and messages may be arbitrarily delayed, subject to the constraint that each message is eventually delivered and each computation is eventually performed. By generalizing our analysis for the synchronous model, we show that for any initial token distribution b, the algorithm converges to a completely balanced distribution within $8OPT(b) + 2n$ rounds, where a round is a minimal sequence of steps in which every component of the network is scheduled at least once. We also show that for every initial token distribution, the message complexity of the algorithm is asymptotically optimal among all algorithms that move tokens in the clockwise direction.

1 Introduction

An important problem in a distributed system is to balance the total workload over the processors. Such load balancing problems arise in a number of parallel and distributed

[1]Department of Computer Science, University of Wisconsin-Madison, Madison, WI 53706. Email: johannes@cs.wisc.edu.

[2]Department of Computer Science, University of Texas at Austin, Austin, TX 78712. Supported by the National Science Foundation under Grant No. CCR–9504145. Email: plaxton@cs.utexas.edu.

[3]DIMACS Center, Rutgers University, Piscataway, NJ 08854. DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999 and partially supported by the New Jersey Commission on Science and Technology. Part of this work was done when the author was at the University of Texas at Austin, with support from the National Science Foundation under Grant No. CCR–9504145. Email: rraj@dimacs.rutgers.edu.

[4]Part of this work was done while the authors were visiting the University of Paderborn.

applications including job scheduling in operating systems (e.g., see [12]), adaptive mesh partitioning (e.g., see [27]), and packet routing (e.g., see [22]). A natural approach to the load balancing problem is to have each node periodically poll the other nodes to which it is connected, and send some of its load to neighbors with lesser load. Indeed, such local balancing algorithms have been studied extensively on different models of computation (e.g., see [1, 8, 10]).

We address the static version of the load balancing problem: we assume that each processor has an initial collection of "tokens" (i.e., units of load), and that no tokens are created or destroyed while the tokens are being balanced. In each step, each node can communicate with each of its neighbors and send (resp., receive) at most one token along each of its incident edges. The problem is to design a distributed algorithm that converges to a balanced distribution quickly, where a distribution is said to be *balanced* if the difference between the number of tokens at any two nodes is at most one.

In this paper, we study static load balancing on a ring network. The ring network has been studied extensively in both theory and practice. Several problems arising in distributed computing have been addressed on the ring (see [5, 13, 14, 19] for a variety of examples). From a practical perspective, the ring is an essential component of several parallel and distributed architectures [17, 25].

Our main contribution is a tight analysis of a simple algorithm that is based on the local balancing approach. We show that this algorithm, which for the sake of brevity we denote by $\mathcal{A}$, converges to a balanced distribution in near-optimal time for *every* initial distribution on both *synchronous* and *asynchronous* rings. We are not aware of any other load balancing algorithm that has been shown to achieve such universal near-optimality with respect to a non-trivial family of networks (e.g., rings). All previous optimality results known for load balancing are worst-case results.

1.1 Our results

Let R be a ring network with the set $[n] = \{0, 1, \ldots, n-1\}$ of nodes and the set $\{(i, (i+1) \bmod n\}$ of edges. The local balancing algorithm $\mathcal{A}$ is defined as follows. In each step, for all i in $[n]$, node i sends a token to node $(i+1) \bmod n$ if and only if i has more tokens than $(i+1) \bmod n$. (See Section 2 for a message-passing implementation of $\mathcal{A}$.) We note that $\mathcal{A}$ only moves tokens in a single direction around the ring, say clockwise. We refer to algorithms that only move tokens in the clockwise direction as *unidirectional* algorithms.

We first consider a synchronous model in which each step of the computation consists of the following three parts:

1. Delivery of messages. Any message sent in the last part of the previous step is delivered. Each node receives at most one such message along each of its incident edges.

2. Local computation. Each node performs a constant number of local operations on local data.

3. Sending of messages. Each node sends zero or one messages to each adjacent node.

We show that:

- The number of steps taken by $\mathcal{A}$ to balance any distribution b on a synchronous ring is at most $4OPT(b) + n$, where $OPT(b)$ is the time taken by an optimal centralized algorithm to balance b. The proof is given in Section 4.

We note that the optimal centralized algorithm need not be a unidirectional algorithm; that is, $OPT(b)$ is the time taken to balance b by the best algorithm among *all* algorithms that send and/or receive at most one token along each edge in each step. In fact, if $OPT(b)$ were instead defined as the time taken by an optimal centralized *unidirectional* algorithm to balance b, then the factor of 4 in the stated bound could be replaced by 2.

Our next result concerns an asynchronous model of computation, in which local computations may be performed at arbitrary speeds and messages may be delayed arbitrarily, subject to the constraint that each message is eventually delivered and each computation is eventually performed [18]. In order to measure time complexity in the asynchronous model, we define a *round* to be a minimal sequence of steps in which each component of the ring (i.e., each node or edge) is scheduled at least once. The time complexity of an algorithm is then defined as the maximum number of rounds taken over all possible schedulings of the components. (See Section 5 for a formal description of the asynchronous model.)

The above notion of time is based on the model proposed in [2] for shared memory systems. An analogous model for message-passing systems was studied in [4]. Moreover, our model is equivalent to that proposed in [20], where the time complexity of an algorithm is defined to be the longest amount of elapsed real time from the start to the completion of the algorithm, assuming at most a unit time delay between two steps of the same network component [2]. (The model proposed in [20] has been subsequently used in the study of several distributed computing problems [6, 7].)

We generalize our result for the synchronous model to the asynchronous model at the expense of a factor of 2; in particular, we show that:

- The number of rounds taken by $\mathcal{A}$ to balance any distribution b on an asynchronous ring is at most $8OPT(b) + 2n$. The proof is given in Section 5.

We remark that if $OPT(b)$ were instead defined as the time taken by an optimal centralized *unidirectional* algorithm for b, then the factor of 8 in the stated bound could be replaced by 4. We also show that in both the synchronous and asynchronous models, for every initial token distribution, the message complexity of $\mathcal{A}$ is asymptotically optimal among all unidirectional algorithms.

1.2 Previous and related work

A number of researchers have studied load balancing problems under different models of computation. These models can be classified on the basis of three characteristics: (i) centralized control (e.g., [21, 26]) versus distributed control (e.g., [8, 10]), (ii) uniform communication (e.g., [26]) versus fixed-connection network communication (e.g., [1, 11]), and (iii) unbounded edge capacity (e.g., [8, 10]) versus bounded edge capacity (e.g., [1, 15]) (the capacity of an edge is the maximum number of tokens it can transmit per step). In the discussion that follows, we restrict our attention to results for models of computation with the same basic characteristics as the model considered in the present paper, namely: distributed control, fixed-connection network communication, and bounded edge capacity.

Local algorithms restricted to particular networks have been studied on hypercubes [10, 24], meshes [16, 21], and expanders [22, 23]. All of these papers analyze the worst-case complexity of certain local algorithms. More recently, it has been shown that a simple local algorithm is optimal in the worst-case on arbitrary networks [15]. Application of the preceding result to the special case of the ring implies that if the initial imbalance (i.e., the difference between the maximum number of tokens at any node and the minimum number of tokens at any node) is Δ, then the local algorithm balances in $O(n\Delta)$ steps. While there exists a distribution with imbalance Δ for which any algorithm takes $\Omega(n\Delta)$ steps to balance, it is not the case that every distribution with imbalance Δ requires $\Omega(n\Delta)$ steps to balance. In fact, it is easy to construct distributions with imbalance Δ that can be balanced in $O(\Delta)$ steps.

In recent work [3], asynchronous balancing algorithms on several networks including the ring have been studied. However, the results of [3] are geared towards establishing eventual convergence in the presence of dynamic network changes, while we are interested in determining the time to convergence for static load balancing. Also related is the result of [9], where a worst-case bound on the number of token movements is given for a model in which tokens can be transferred between any two nodes.

Our result for the asynchronous model is similar in spirit to that of [7], in that our asynchronous algorithm is not obtained by using a general synchronizer [6] in conjunction with an algorithm optimized for a synchronous model. Instead, we show that $\mathcal{A}$ is directly implementable on asynchronous rings, thus avoiding the overhead and complexity of a synchronizer while achieving near-optimal bounds.

1.3 Outline

The remainder of the paper is organized as follows. Section 2 presents a message-passing implementation of $\mathcal{A}$. Section 3 introduces notations and definitions for our analysis. Sections 4 and 5 present our analysis of $\mathcal{A}$ for the synchronous and asynchronous models, respectively. Section 6 discusses issues related to detecting the termination of $\mathcal{A}$. We conclude with some remarks in Section 7.

2 The unidirectional algorithm $\mathcal{A}$

In this section, we give a message-passing implementation of the unidirectional algorithm $\mathcal{A}$ introduced in Section 1. Recall that the nodes of the ring are assigned unique labels from the set $[n]$. For convenience, we adopt the following notational convention: any arithmetic expression referring to a node is interpreted modulo n. For example, we will often refer to the neighbors of an arbitrary node i as node $i-1$ and node $i+1$, rather than node $(i-1) \bmod n$ and node $(i+1) \bmod n$.

In $\mathcal{A}$, each node i repeatedly communicates with node $i+1$ and sends a token to $i+1$ whenever the number of tokens at i exceeds that at $i+1$. A straightforward implementation of $\mathcal{A}$ for the synchronous model is as follows. In even steps, each node i informs node $i-1$ of the number of tokens at i, and in odd steps, each node i sends a token to $i+1$ if and only if the number of tokens at i exceed the number of tokens at $i+1$. The preceding implementation does not apply for the asynchronous model, however, because the nodes are required to execute in lock-step. In what follows, we describe a simple efficient implementation of $\mathcal{A}$ that applies for both synchronous as well as asynchronous models.

Each node i maintains three variables related to the number of tokens at $i+1$: (i) a count $x(i)$ of the number of tokens that i has sent to $i+1$ since the start of the algorithm, (ii) an estimate $y(i)$ of the number of tokens that $i+1$ has sent to $i+2$ since the start of the algorithm, and (iii) the number $z(i)$ of tokens initially at $i+1$. At the current point in the execution of the algorithm, let $w(i)$ denote the number of tokens at i. Thus, $w(i)$ equals $w_0(i)$ initially. Also, at the current point in the execution of the algorithm, the expression $z(i)+x(i)-y(i)$ represents the estimate at node i of the number of tokens at node $i+1$.

In $\mathcal{A}$, the nodes communicate with their neighbors using three types of messages: (i) *height*, a message that i sends to $i-1$ indicating the number of tokens at i, (ii) *update*, a message that i sends to $i-1$ indicating that i has sent a new token to $i+1$, and (iii) *token*, a message consisting of a token sent by i to $i+1$. Using these three messages, the algorithm can be implemented as follows.

- In the initial step, i performs the following operations: (i) set $x(i)$ and $y(i)$ to zero and set $z(i)$ to ∞, and (ii) send a *height* message with value $w(i)$ to $i-1$.

- In each subsequent step, i performs the following operation. If $w(i) > z(i)+x(i)-y(i)$, then: (i) decrement $w(i)$ by 1, (ii) increment $x(i)$ by 1, (iii) send a *token* message to $i+1$, and (iv) send an *update* message to $i-1$.

- On receipt of a *height* message, i sets $z(i)$ to the value of the message. On receipt of an *update* message, i increments $y(i)$. On receipt of a token, i increments $w(i)$.

While a *height* message and an *update* message are both intended to inform the destination node of the number of tokens at the source node, there may be a significant difference in the sizes of these messages. The size of a *height* message depends on the number of tokens at the source node. On the other hand, an *update* message always consists of only a constant number of bits. We optimize the communication complexity of $\mathcal{A}$ by having each node send a *height* message only in its initial step.

Algorithm $\mathcal{A}$ is analyzed in Sections 4 and 5 under the synchronous and asynchronous models, respectively. In our analysis, we assume for simplicity that the average number of tokens in the initial distribution is an integer. See Section 7 for further discussion of this assumption.

3 Preliminaries

Let $\mathbf{Z}$ and $\mathbf{N}$ denote the integers and nonnegative integers. Let $V = \mathbf{Z}^n$ denote the set of n-tuples of integers. For any t in $\mathbf{N}$ and i in $[n]$, let w_t be such that $w_t(i)$ is the number of tokens at node i at the start of step t. (We number the steps from 0.) For any b in V, let $\mu(b) = \frac{1}{n}\sum_{i\in[n]} b(i)$ denote the average number of tokens in b. We say that the ring is *balanced* in step t if $w_t(i)$ is $\lfloor \mu(b) \rfloor$ or $\lceil \mu(b) \rceil$ for all i in $[n]$, where b is the initial distribution. (Given our assumption that $\mu(b)$ is an integer, this condition simplifies to $w_t(i) = \mu(b)$ for all i in $[n]$.) For any subset S of $[n]$, let $w_t(S)$ denote the total number of tokens in S at the start of step t.

For any i and j in $[n]$, let $d(b, i, j)$ denote the total "imbalance" associated with the set of contiguous nodes obtained when going from node i to node j in the clockwise direction (i and j included). Formally, we have:

$$d(b, i, j) = \sum_{0 \le k \le (j-i) \bmod n} (b(i+k) - \mu(b)).$$

(In other words, if $i \le j$, then $d(b, i, j)$ equals $\sum_{i \le k \le j}(b(k) - \mu(b))$; otherwise, $d(b, i, j)$ equals $[\sum_{j \le k < n}(b(k) - \mu(b)) + \sum_{0 \le k \le i}(b(k) - \mu(b))]$.) Let $\ell(b)$ and $m(b)$ be two integers such that $d(b, \ell(b), m(b))$ is $\max_{i,j} d(b, i, j)$. Without loss of generality, we assume for the remainder of this paper that $\ell(b)$ is zero as we can relabel the nodes appropriately otherwise.

We now introduce the notion of *discrepancy*, which plays an important role in our analysis. The discrepancy of a distribution denotes the maximum imbalance among all sets of contiguous nodes of the ring. More formally, the discrepancy $D(b)$ of b is given by $d(b, 0, m(b))$. In Section 4.3, we show that $OPT(b)$ is at least $D(b)/2$ (Lemma 4.1).

In the remainder of this paper, we will be concerned with applying the functions μ, d, and m with respect to the initial token distribution. Therefore, as a shorthand, we let μ, $d(i, j)$, and m, denote $\mu(w_0)$, $d(w_0, i, j)$, and $m(w_0)$, respectively.

4 Analysis for synchronous rings

In this section, we analyze $\mathcal{A}$ under the synchronous model of computation. Section 4.1 gives an informal overview of our analyis. Section 4.2 introduces the notion of a "partial" algorithm. Section 4.3 develops the formal machinery needed to establish our main synchronous upper bound, which is stated in Section 4.4.

4.1 Informal overview

Three key ideas underly our analysis of $\mathcal{A}$ for synchronous rings. We now briefly discuss these ideas in order give the reader some intuition for the formal arguments that follow. This subsection may be skipped without loss of continuity.

The first key idea is to prove that for any token distribution b, there exists a pair of adjacent nodes i and $i+1$ in the ring such that i never sends a token to $i+1$. This effectively reduces the problem of analyzing $\mathcal{A}$ on a ring to the somewhat simpler problem of analyzing $\mathcal{A}$ on a linear array (obtained by "breaking" the ring between nodes i and $i+1$).

The second key idea is the use of the "prefix sum vector" (formally defined in Section 4.3) to characterize the state of imbalance of the linear array alluded to above. We are able to show that if some token distribution b has a prefix sum vector that is component-wise at most as large as the prefix sum vector of some other token distribution c, then $\mathcal{A}$ balances b at least as quickly as c.

The third key idea is to show that for any given discrepancy D, there a "worst-case" token distribution that simultaneously achieves the highest possible value in each component of the prefix sum vector. Using the claim of the preceding paragraph, it follows that we can upper-bound the performance of $\mathcal{A}$ on any token distribution with discrepancy D by upper-bounding the performance of $\mathcal{A}$ on the worst-case token distribution with discrepancy D. Because this worst-case token distribution happens to have a very simple structure, the latter task turns out to be straightforward.

4.2 Partial algorithms

In the synchronous model, each node executes in a lock-step manner, and each message is transmitted in a single step. By the definitions of $x(i)$, $y(i)$, and $z(i)$, we obtain that the value of $z(i) + x(i) - y(i)$ at the start of step t equals $w_t(i+1)$ for any $t > 0$. Therefore, each step of node i can be expressed as follows: if $w_t(i) > w_t(i+1)$, then send a token to $i+1$. For our analysis, it is helpful to consider a generalization of $\mathcal{A}$ given by Definition 4.2 below.

Definition 4.1 *We say that a step t of an algorithm is an S-**step**, where S is a subset of $[n]$, if each node not in S is idle in step t and each node i in S performs the following operation: if $w_t(i) > w_t(i+1)$, then i sends a token to $i+1$.*

Definition 4.2 *A **partial algorithm** $\mathcal{B}$ is one in which each step is an S-step for some subset S of $[n]$. For any t in $\mathbf{N}$, we let $\mathcal{B}(t)$ denote the set S such that step t of $\mathcal{B}$ is an S-step.*

It follows from Definitions 4.1 and 4.2 that $\mathcal{A}$ is a partial algorithm in which each step after step 0 is an $[n]$-step. We bound the running time of $\mathcal{A}$ by providing a general analysis in Section 4.3 that applies to all partial algorithms.

4.3 Analysis of partial algorithms

While the number of tokens present at each node of the ring after any number of steps of $\mathcal{A}$ (or any other partial algorithm) is easy to calculate, the particular token distribution obtained does not directly provide a simple measure of progress of the algorithm. A simple approach to measure the progress of a load balancing algorithm is the following: (i) assign to each node a potential that grows with the imbalance at the node, and (ii) determine the rate at which the sum of the potentials of the nodes decreases with time. While this approach simplifies the worst-case analysis for general networks (see [15]), it appears to be inadequate for our purposes since information about the particular distribution of imbalance is lost.

By exploiting the simple structure of ring networks, we are able to capture the precise distribution of the imbalance of the network in a measure, referred to as the *prefix sum vector*. For each t in $\mathbf{N}$, let p_t be defined as follows:

$$p_t(i) = \sum_{0 \le j \le i} (w_t(j) - \mu) \tag{1}$$

for all i in $[n]$. (In other words, p_t is the n-tuple of the prefix sums of the difference between the number of tokens at each node at the start of step t and the average.) Given an initial token distribution $w_0 = b$, let $T(b)$ denote $\sum_{i \in [n]} p_0(i)$.

The following lemma gives a lower bound on the time complexity of any balancing algorithm and the number of token transmissions of any unidirectional balancing algorithm in terms of $D(b)$ and $T(b)$, respectively, where b is the initial token distribution. Recall that $D(b)$, which is formally defined in Section 3, is the discrepancy of b.

Lemma 4.1 *Any algorithm takes at least $D(b)/2$ steps to balance b. Any unidirectional algorithm incurs at least $T(b)$ token transmissions to balance b.*

Proof: Consider the set $S = \{i : 0 \leq i \leq m(b)\}$ of nodes. By definition, $w_0(S)$ is $D(b) + \mu|S|$. (Recall that $\ell(b)$ is 0.) If the ring is balanced in t steps, then for each node i in S, $w_t(i)$ is μ. Therefore, $w_t(S)$ is at most $\mu|S|$, and hence, at least $D(b)$ tokens are sent out of S in t steps. Since at most two tokens can be sent out of S per step, t is at least $D(b)/2$.

For each i in $[n]$, the number of token transmissions across edge $(i, i+1)$ required by any unidirectional algorithm is at least $p_0(i)$ since $p_0(i)$ is the excess number of tokens in the interval $[0, i]$. Therefore, the total number of token transmissions needed by any unidirectional algorithm to balance b is at least $T(b)$. ∎

The remainder of this section is devoted to deriving an upper bound on the time taken by any partial algorithm to balance a given distribution b. We begin by determining the effect of a step of a partial algorithm on the prefix sum vector. For this purpose, it is useful to define a partial order $\preceq$ on V as follows: $b \preceq c$ if and only if $b(i) \leq c(i)$ for all i in $[n]$. For convenience, we use 0 to denote the "all-zeros" n-tuple $(0, \ldots, 0)$. Lemma 4.2 expresses a partial algorithm as a recurrence relation among the prefix sum vectors.

Lemma 4.2 *For any partial algorithm $\mathcal{B}$, we have: if i is in $\mathcal{B}(t)$ and $2p_t(i) > p_t(i-1) + p_t(i+1)$, then $p_{t+1}(i) = p_t(i) - 1$; otherwise, $p_{t+1}(i) = p_t(i)$.*

The proof of Lemma 4.2 follows from Lemmas 4.3 and 4.5 below. In Lemma 4.3, we show that $0 \preceq p_t$ is the only precondition for Lemma 4.2. Lemma 4.5 establishes this precondition by induction on t.

Lemma 4.3 *For any partial algorithm $\mathcal{B}$, if $0 \preceq p_t$, then: if i is in $\mathcal{B}(t)$ and $2p_t(i) > p_t(i-1) + p_t(i+1)$, then $p_{t+1}(i) = p_t(i) - 1$; otherwise, $p_{t+1}(i) = p_t(i)$.*

Proof: Since $0 \preceq p_t$, $p_t(0)$ is nonnegative, and hence $w_t(0)$ is at least μ. Moreover, by definition, $p_t(n-1)$ is 0. Since $p_t(n-2)$ is nonnegative, $w_t(n-1)$ is at most μ. Therefore, no token is sent from node $n-1$ to node 0. It follows that for each i in $[n]$, if node i sends a token to node $i+1$, then $p_{t+1}(i)$ is $p_t(i) - 1$; otherwise, $p_{t+1}(i)$ is $p_t(i)$. Node i sends a token to $i+1$ if and only if i is in $\mathcal{B}(t)$ and $p_t(i) - p_t(i-1)$ is greater than $p_t(i+1) - p_t(i)$. The desired claim follows. ∎

Lemma 4.4 forms the induction base for Lemma 4.5.

Lemma 4.4 *For any token distribution, we have $0 \preceq p_0$.*

Proof: The proof is by contradiction. Let i be the smallest nonnegative integer such that $p_0(i)$ is negative. From the definition of i, it follows that $d(i+1, m)$ equals $d(0, m) - d(0, i)$. Since $d(0, i) = p_0(i) < 0$, we obtain that $d(i+1, m)$ is greater than $d(0, m)$, contradicting the definition of m. ■

Lemma 4.5 *For any partial algorithm $\mathcal{B}$ and all t in $\mathbf{N}$, we have $0 \preceq p_t$.*

Proof: The proof is by induction on t. The base case follows from Lemma 4.4. The induction hypothesis is that $0 \preceq p_t$. For the induction step, we consider step t and argue that $0 \preceq p_{t+1}$. By Lemma 4.3, we have: if $2p_t(i) > p_t(i-1) + p_t(i+1)$, then $p_{t+1}(i)$ is $p_t(i) - 1$; otherwise, $p_{t+1}(i)$ is $p_t(i)$. In either case, since $p_t(i)$ is nonnegative (by the induction hypothesis) and is an integer for all i, we obtain that $p_{t+1}(i)$ is nonnegative for all i, thus completing the induction step. ■

Lemma 4.6 shows that each step of a partial algorithm, when viewed as a function on the prefix sum vector, is monotonic with respect to $\preceq$.

Lemma 4.6 *Let S be an arbitrary subset of $[n]$. Let p and q denote the prefix sum vectors associated with token distributions b and c, respectively. Let p' and q' denote the prefix sum vectors associated with the token distributions obtained after performing an S-step on distributions b and c, respectively. If $p \preceq q$ then $p' \preceq q'$.*

Proof: Consider any i in $[n]$. If $p(i)$ is less than $q(i)$, then $p'(i) \leq q(i) - 1 \leq q'(i)$. Otherwise, we have $p(i) = q(i)$. By Lemma 4.2, if $q'(i)$ is $q(i) - 1$, then $2q(i) > q(i-1) + q(i+1)$. It then follows from the hypothesis of the lemma that $2p(i) > p(i-1) + p(i+1)$, which together with Lemma 4.2 implies that $p'(i)$ is $p(i) - 1$. Thus, the desired claim holds. ■

Corollary 4.6.1 *Consider a partial algorithm $\mathcal{B}$. Let p_0 and q_0 denote the prefix sum vectors at the start of step 0 when the initial token distributions are b and c, respectively. If $p_0 \preceq q_0$, then the number of steps taken by $\mathcal{B}$ to balance b is at most that taken to balance c.* ■

Given a nonnegative integer h, consider the set $U(h)$ of token distributions with discrepancy h. Let $P(h)$ denote the set of prefix sum vectors associated with the distributions in $U(h)$. Using a straightforward argument, we prove in Lemma 4.7 below that $f(h) = (2h, h, \ldots, h, 0)$ is the distribution whose prefix sum vector $g(h) = (h, h, \ldots, h, 0)$ is the unique least upper bound (with respect to $\preceq$) of $P(h)$.

Lemma 4.7 *For any initial token distribution b, we have $p_0 \preceq g(D(b))$.*

Proof: By the definition of D and p_0, for each i in $[n]$, $p_0(i)$ is at most $D(b)$. Moreover, since μ is an integer, $p_0(n-1)$ is zero. It thus follows from the definition of g that $p_0 \preceq g(D(b))$. ■

It follows from Corollary 4.6.1 and Lemma 4.7 that the number of steps taken by a partial algorithm $\mathcal{B}$ to balance any distribution in $U(h)$ is at most the number of steps taken by $\mathcal{B}$ to balance $f(h)$. We derive an upper bound on the time taken by a given partial algorithm $\mathcal{B}$ to balance $f(h)$ by relating $\mathcal{B}$ to a different partial algorithm $\mathcal{C}$. Given partial algorithms $\mathcal{B}$ and $\mathcal{C}$, we say that $\mathcal{B}$ *covers* (resp., *is covered by*) $\mathcal{C}$ if $\mathcal{B}(t)$ is a superset of (resp., subset of) $\mathcal{C}(t)$ for all t. In the following we prove that if $\mathcal{B}$ covers $\mathcal{C}$, then $\mathcal{B}$ balances any distribution at least as quickly as $\mathcal{C}$ does.

Lemma 4.8 *Let $\mathcal{B}$ and $\mathcal{C}$ be two partial algorithms such that $\mathcal{B}$ covers $\mathcal{C}$. Given an initial token distribution, let p_t and q_t denote the prefix sum vectors at the start of step t of $\mathcal{B}$ and $\mathcal{C}$, respectively. Then, for each step t, $p_t \preceq q_t$.*

Proof: The proof is by induction on t. The induction base is trivial since $p_0 = q_0$. For the induction hypothesis, we assume that $p_t \preceq q_t$. Consider step t of $\mathcal{B}$ and $\mathcal{C}$. Let $\mathcal{D}$ be a partial algorithm that is identical to $\mathcal{C}$ except that $\mathcal{D}(t) = \mathcal{B}(t)$. Let r represent the prefix sum vector obtained after step t of $\mathcal{D}$. Since $\mathcal{D}(t) \supset \mathcal{C}(t)$, it follows from Lemma 4.2 that $r \preceq q_{t+1}$. By Lemma 4.6 and the induction hypothesis, it follows that $p_{t+1} \preceq r$. By the transitivity of $\preceq$, it follows that $p_{t+1} \preceq q_{t+1}$. ■

Corollary 4.8.1 *Let $\mathcal{B}$ and $\mathcal{C}$ be two partial algorithm such that $\mathcal{B}$ covers $\mathcal{C}$. For any initial token distribution b, the number of rounds taken by $\mathcal{B}$ to balance b is at most that taken by $\mathcal{C}$.* ■

We are now ready to place an upper bound on the time taken by any partial algorithm to balance $f(h)$. Let $\mathcal{B}$ be a partial algorithm. For i in $\mathbf{N}$, let r_i be defined as follows: r_0 is -1 and for all $i > 0$, r_i is the smallest integer greater than r_{i-1} such that $\cup_{r_{i-1} < j \le r_i} \mathcal{B}(j) = [n]$. We define the *ith round* of $\mathcal{B}$ to be the sequence of steps in the interval $[r_i + 1, r_{i+1}]$.

Lemma 4.9 *For any nonnegative integer h, the number of rounds taken by any partial algorithm $\mathcal{B}$ to balance $f(h)$ is at most $2h + n - 1$.*

Proof: For any i, let the ith round of $\mathcal{B}$ consist of the steps $[r_i + 1, r_{i+1}]$. In order to establish the desired claim, we construct a partial algorithm $\mathcal{C}$ that is covered by $\mathcal{B}$. Since the rounds of $\mathcal{C}$ may differ from those of $\mathcal{B}$, to avoid ambiguity, we refer to $[r_i + 1, r_{i+1}]$ as interval i.

Given interval i and a node j, let $y_{i,j}$ be the smallest integer such that j is in $\mathcal{B}(y_{i,j})$. We now define $\mathcal{C}$ as follows. For each interval i, and each step t in interval i, node j is in $\mathcal{C}(t)$ if and only if: (i) t is $y_{i,j}$ and (ii) j has the same parity as i. In other words, for algorithm $\mathcal{C}$, a node j participates in an interval i only if i and j have the same parity. Moreover, if j has the same parity as i, then node j participates for $\mathcal{C}$ only in the first step t of interval i in which j participates for $\mathcal{B}$. It follows directly from the definition that $\mathcal{C}$ is a partial algorithm and that $\mathcal{B}$ covers $\mathcal{C}$. We now show that $\mathcal{C}$ balances $f(h)$ before the start of interval $2h + n - 1$.

We mark the h excess tokens on node 0 with the labels 0 through $h - 1$ from the top. We show that during the execution of $\mathcal{C}$ the following property holds: at the start of interval i, if $k \le \min\{\lceil i/2 \rceil, h\}$, token k is at node $\min\{i - 2k, n - 1\}$; otherwise, token k is at node 0. The proof is by induction on $i \le 2h + n$. The induction base is trivial. For the induction hypothesis, we assume that the above statement holds at the start of interval i.

Consider interval i. By the definition of $\mathcal{C}$, if $i - j$ is even in interval i, then node j sends token $(i - j)/2$ to $j + 1$; otherwise, node j does not send any token. Thus, each node j sends at most one token to $j + 1$ in any interval. Furthermore, by the induction hypothesis, if $i - j$ is even then node j has token $(i - j)/2$ while node $j + 1$ has no marked token, thus completing the induction step.

By the aforementioned property, $\mathcal{C}$ balances $f(h)$ before the start of interval $2h + n - 1$. The lemma then follows by Corollary 4.8.1. ■

The upper bound on the time complexity of a partial algorithm now follows from Corollary 4.6.1, Lemma 4.7, and Lemma 4.9.

Lemma 4.10 *Given any initial token distribution b, the number of rounds taken by any partial algorithm to balance b is at most $2D(b) + n - 1$.*

Proof: By Lemma 4.7, $p_0 \preceq g(D(b))$. Therefore, by Corollary 4.6.1, the number of rounds taken to balance b is at most that taken to balance $f(D(b))$. By Lemma 4.9, the number of rounds taken to balance $f(D(b))$ is at most $2D(b) + n - 1$. The desired claim follows. ■

We now place a bound on the number of token transmissions before balancing a distribution b. Whenever a node i sends a token in step t, we have $p_{t+1}(i) = p_t(i) - 1$. Therefore, the total number of token transmissions by any partial algorithm is exactly $T(b)$, which, by Lemma 4.1, is optimal with respect to all unidirectional algorithms.

Lemma 4.11 *Given an initial token distribution b, the number of token transmissions by any partial algorithm is $T(b)$.* ■

4.4 Complexity of $\mathcal{A}$

Every step of $\mathcal{A}$ after step 0 is an $[n]$-step. It thus follows from Lemma 4.10 that the number of steps taken to balance any distribution b with an integral average is at most $2D(b) + n$.

We now consider the message complexity of $\mathcal{A}$. In step 0, n *height* messages are transmitted. The number of *update* messages transmitted is at most the total number of token transmissions since an *update* message is sent by a node i in step t only if i sends a token in step t. By Lemma 4.11, the number of token transmissions is at most $T(b)$. Hence, the number of *update* messages is at most $T(b)$.

Theorem 1 *Consider the synchronous model of a ring network with n processors. If the initial token distribution is b, then the number of steps taken by $\mathcal{A}$ to balance b is at most $2D(b)+n$. The number of* height *messages,* update *messages, and* token *messages transmitted by $\mathcal{A}$ are n, $T(b)$, and $T(b)$, respectively.* ■

5 Analysis for asynchronous rings

In this section, we analyze $\mathcal{A}$ under an asynchronous model of computation. We consider the ring network as consisting of $3n$ different components: n nodes given by the set $[n]$ and $2n$ directed edges given by the set $\{(i, i+1), (i, i-1)\}$. As defined in Section 2, each step of a node consists of sending a constant number of messages to its neighbors together with performing a small number of local operations. Each edge (i, j) is a directed channel that transmits messages from i to j in FIFO order. At any instant, there may be several messages in transit from i to j on edge (i, j). Each step of edge (i, j) consists of delivering the first message (if any) in FIFO order among the messages currently in transit from i to j.

We model asynchrony by means of an adversary $\mathcal{X}$ that schedules the components of the network over a sequence of steps. In step t, each component in a set $\mathcal{X}(t)$ of components chosen by the adversary executes its next step simultaneously. Given adversaries $\mathcal{X}_1$ and $\mathcal{X}_2$, we say that $\mathcal{X}_1$ is *weaker* (resp., *stronger*) than $\mathcal{X}_2$ if for all t, $\mathcal{X}_1(t)$ is a superset (resp., subset) of $\mathcal{X}_2(t)$. The notions of an adversary and that of weakness generalize the notions of a partial algorithm and that of covering defined in Section 4. Indeed, we establish our results for the asynchronous model by generalizing some of the claims of Section 4.

As mentioned above, when an edge is scheduled, the first message (if any) in FIFO order is delivered to the destination node. In the definition of $\mathcal{A}$, there are some operations that are performed at the node on receipt of a message. (An example of such an operation is changing the value of $y(i)$ at node i on receipt of an *update* message.) Such operations can be executed either during the scheduling of the edge delivering the particular message, or at the next scheduling of the destination node, as determined by the adversary.

Given an adversary, we define a *round* to consist of a minimal sequence of steps in which each component of the network is scheduled at least once by the adversary. The sequence of steps is partitioned into a sequence of non-overlapping rounds. The time complexity of an algorithm is defined to be maximum, over all adversaries, of the number of rounds taken to balance the ring. The message complexity of an algorithm is the maximum, over all adversaries, of the number of messages transmitted by the algorithm.

We now begin the analysis of $\mathcal{A}$ under the asynchronous model defined above. For any $t \geq 0$ and any i in $[n]$, let $u_t(i)$ denote the number of tokens in transit along edge $(i, i+1)$ at the start of step t. In analogy to Equation 1, we define two notions of prefix sums. For each t in $\mathbf{N}$, we define p_t and q_t as follows:

$$\begin{aligned} p_t(i) &= \sum_{0 \leq j \leq i} (w_t(j) + u_t(j) - \mu) \text{ and} \\ q_t(i) &= p_t(i) - u_t(i) \end{aligned}$$

for all i in $[n]$. We refer to p_t and q_t as the *upper prefix sum vector* and the *lower prefix sum vector*, respectively. Let $\pi_t(i)$ denote the last step $t' < t$ such that a *height* or an *update* message sent by $i+1$ in step t' is received by i in some step before step t. If no *height* or *update* message is received by i in any of the first t steps, we set $\pi_t(i)$ to -1. For convenience, we let $q_{-1}(i)$ equal ∞ for all i.

Lemmas 5.1, 5.2, 5.3, and 5.4 generalize Lemmas 4.3, 4.5, 4.2, and 4.8, respectively. The proofs of Lemmas 5.1 and 5.2 follow the same lines as the proofs of Lemmas 4.3 and 4.5, respectively.

Lemma 5.1 *Consider the execution of $\mathcal{A}$ against an adversary $\mathcal{X}$. Assume that $0 \preceq q_s$ for all $s \leq t$. If i is in $\mathcal{X}(t)$ and $2q_t(i) > p_t(i-1) + q_{\pi_t(i)}(i+1)$, then $q_{t+1}(i)$ is $q_t(i) - 1$; otherwise, $q_{t+1}(i)$ is $q_t(i)$.*

Proof: Consider any $s \leq t$. Since $0 \preceq q_s$, $q_s(0)$ is nonnegative, and hence $w_s(0)$ is at least μ. Moreover, by definition, $p_s(n-1)$ is 0. Since $0 \leq q_s(n-1) \leq p_s(n-1)$, $q_s(n-1)$ and $u_s(n-1)$ are both 0. Since $p_s(n-2)$ is nonnegative, $w_s(n-1)$ is at most μ. Therefore, no token is sent from node $n-1$ to node 0 in step t. It follows that for each i in $[n]$, if node i sends a token to node $i+1$, then $q_{t+1}(i)$ is $q_t(i) - 1$; otherwise, $q_{t+1}(i)$ is $q_t(i)$.

We now show that node i sends a token to node $i+1$ if and only if i is in $\mathcal{X}(t)$ and $q_t(i) - p_t(i-1)$ is greater than $q_{\pi_t(i)}(i+1) - q_t(i)$. It follows from the definitions of q_t and

p_t that $w_t(i)$ equals $q_t(i) - p_t(i-1)$. If $\pi_t(i)$ equals -1, then it follows from the definition of $\pi_t(i)$ that the value of the variable $z(i)$ (see Section 2) at the start of step t is ∞. Since the values of $x(i)$, $y(i)$, and $q_t(i)$ are all finite, we obtain that if $\pi_t(i)$ is -1 then at the start of step t, $z(i) + x(i) - y(i) = \infty = q_{\pi_t(i)}(i+1) - q_t(i)$. We now consider the case when $\pi_t(i)$ does not equal -1. By the definition of $\pi_t(i)$, the value of $z(i)$ at the start of step t is finite. Moreover, by the definitions of the variables $x(i)$, $y(i)$, and $z(i)$ of Section 2 and the fact that no token is sent from node $n-1$ to node n in any step $s \le t$, we obtain that the value of the expression $z(i) + x(i) - y(i)$ at the start of step t equals

$$\begin{aligned} & q_0(i+1) - q_0(i) + q_0(i) - q_t(i) - (q_0(i+1) - q_{\pi_t(i)}(i+1)) \\ = \; & q_{\pi_t(i)}(i+1) - q_t(i). \end{aligned}$$

Therefore, in any step t, if node i is scheduled by the adversary, then node i sends a token to $i+1$ if and only if $q_t(i) - p_t(i-1)$ is greater than $q_{\pi_t(i)}(i+1) - q_t(i)$. The desired claim follows. ■

Lemma 5.2 *Given any adversary, $0 \preceq q_t$ and $0 \preceq p_t$ hold for all t in $\mathbf{N}$.*

Proof: The proof is by induction on t. The induction basis follows from Lemma 4.4.

For the induction step, consider step t. By the induction hypothesis, $0 \preceq q_t$. We first show that $0 \preceq q_{t+1}$. By Lemma 5.1, we have: if $2q_t(i) > p_t(i-1) + q_{\pi_t(i)}(i+1)$, then $q_{t+1}(i)$ is $q_t(i) - 1$; otherwise, $q_{t+1}(i)$ is $q_t(i)$. In either case, the induction hypothesis implies that $p_t(i)$, $p_t(i-1)$, and $q_{\pi_t(i)}(i+1)$ are all nonnegative integers, and hence that $q_{t+1}(i)$ is nonnegative for all i. Since $q_{t+1} \preceq p_{t+1}$, it follows that $0 \preceq p_{t+1}$, thus establishing the induction step. ■

Lemmas 5.1 and 5.2 together imply the following lemma.

Lemma 5.3 *Given any adversary $\mathcal{X}$, if i is in $\mathcal{X}(t)$ and $2q_t(i) > p_t(i-1) + q_{\pi_t(i)}(i+1)$, then $q_{t+1}(i)$ is $q_t(i) - 1$; otherwise, $q_{t+1}(i)$ is $q_t(i)$.* ■

Given a fixed initial distribution of tokens and two different adversaries, we now relate the prefix sum vectors obtained after t steps of $\mathcal{A}$ against the two adversaries. Lemma 5.4 states that both the upper and lower prefix sum vectors associated with the weaker adversary are lower bounds (with respect to $\preceq$) on the upper and lower prefix sum vectors associated with the stronger adversary.

Lemma 5.4 *Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two adversaries such that $\mathcal{X}_1$ is weaker than $\mathcal{X}_2$. Given an initial token distribution, let p_t^1 (resp., q_t^1) denote the upper (resp., lower) prefix sum vector at the start of step t of $\mathcal{A}$ against adversary $\mathcal{X}_1$, and let p_t^2 (resp., q_t^2) denote the upper (resp., lower) prefix sum vector at the start of step t of $\mathcal{A}$ against adversary $\mathcal{X}_2$. For each step t, we have $q_t^1 \preceq q_t^2$ and $p_t^1 \preceq p_t^2$.*

Proof: Let $\alpha_t(i)$ and $\beta_t(i)$ denote the value of $\pi_t(i)$ under adversaries $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively. We prove by induction on t that: (i) $q_t^1 \preceq q_t^2$, (ii) $p_t^1 \preceq p_t^2$, and (iii) for all i, $q^1_{\alpha_t(i)}(i+1) \le q^2_{\beta_t(i)}(i+1)$. The induction base is trivial since $q_0^1 = q_0^2$, $p_0^1 = p_0^2$, and $\alpha_t(i) = \beta_t(i) = -1$ for all i. For the induction hypothesis we assume that (i), (ii), and (iii) hold for all steps less than or equal to t.

We first show that $q_{t+1}^1 \preceq q_{t+1}^2$. Consider any i in $[n]$. If $q_t^1(i) < q_t^2(i)$, then it trivially follows from Lemma 5.3 that $q_{t+1}^1(i) \leq q_{t+1}^2(i)$. Otherwise, $q_t^1(i) = q_t^2(i)$. In this case, by Lemma 5.3, if $q_{t+1}^2(i) = q_t^2(i)-1$, then $2q_t^2(i) > p_t^2(i-1)+q_s^2(i+1)$, where s equals $\beta_t(i)$. Let s' equal $\alpha_t(i)$. By the induction hypothesis, $q_{s'}^1(i+1) \leq q_s^2(i+1)$ and $p_t^1(i-1) \leq p_t^2(i-1)$. Since $q_t^1(i) = q_t^2(i)$, we thus obtain $2q_t^1(i) > p_t^1(i-1) + q_{s'}^1(i+1)$, which together with Lemma 5.3 implies that $q_{t+1}^1(i) = q_t^1(i) - 1$. Thus, we have $q_{t+1}^1 \preceq q_{t+1}^2$.

We next show that $p_{t+1}^1 \preceq p_{t+1}^2$. In order to prove that $p_{t+1}^1(i) \leq p_{t+1}^2(i)$, we need only consider the case in which $(i, i+1)$ is in $\mathcal{X}_1(t)$, as otherwise the desired claim follows directly from the induction hypothesis. Accordingly, assume that $(i, i+1)$ is in $\mathcal{X}_1(t)$. Let $u_t^1(i)$ and $u_t^2(i)$ denote the values of $u_t(i)$ associated with adversaries $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively. If $u_t^1(i)$ is positive, then $p_{t+1}^1(i) = p_t^1(i) - 1 \leq p_t^2(i) - 1 \leq p_{t+1}^2(i)$. Otherwise, we have $p_{t+1}^1(i) = p_t^1(i) = q_t^1(i)$, and $p_{t+1}^2(i) \geq p_t^2(i) - u_t^2(i) = q_t^2(i)$. Therefore, $p_{t+1}^1(i)$ is at most $p_{t+1}^2(i)$.

We now complete the induction step by showing that for all i, $q_{\alpha_{t+1}(i)}^1(i+1) \leq q_{\beta_{t+1}(i)}^2(i+1)$. If $(i+1, i)$ is not in $\mathcal{X}_2(t)$, then $\alpha_{t+1}(i) \geq \alpha_t(i)$ and $\beta_{t+1}(i) = \beta_t(i)$, and hence the desired claim follows from the induction hypothesis and the trivial inference from Lemma 5.3 that q_j^1 is nonincreasing as j increases. We now consider the case in which $(i+1, i)$ is in $\mathcal{X}_2(t)$. If $\alpha_{t+1}(i) \neq \alpha_t(i)$, then the desired claim holds since $q_{\alpha_{t+1}(i)}^1(i+1) = q_{\alpha_t(i)}^1(i+1) - 1$, while $q_{\beta_{t+1}(i)}^2(i+1) \geq q_{\beta_t(i)}^2(i+1) - 1$. Otherwise, we consider two subcases: $\alpha_{t+1}(i) \geq \beta_{t+1}(i)$ and $\alpha_{t+1}(i) < \beta_{t+1}(i)$. In the first subcase, since q_j^1 is nonincreasing as j increases, $q_{\alpha_{t+1}(i)}^1(i+1) \leq q_{\beta_{t+1}(i)}^1(i+1)$. For the second subcase, we note that since $\alpha_{t+1}(i) = \alpha_t(i)$, no *update* message is received by i in step t under adversary $\mathcal{X}_1$, which implies that no *update* message was sent by node $i+1$ to node i in the interval $[\alpha_{t+1}(i) + 1, t-1]$ of steps under adversary $\mathcal{X}_1$. Since every token transmission from node $i+1$ to node $i+2$ is accompanied by an *update* message from node $i+1$ to node i, we obtain that for every t' in $[\alpha_{t+1}(i) + 1, t-1]$, $q_{t'}^1(i+1) = q_{\alpha_{t+1}(i)}^1(i+1)$. In particular, $q_{\alpha_{t+1}(i)}^1(i+1) = q_{\beta_{t+1}(i)}^1(i+1)$. Thus, in either of the two subcases, $q_{\alpha_{t+1}(i)}^1(i+1) \leq q_{\beta_{t+1}(i)}^1(i+1)$. The desired claim now follows from the induction hypothesis. ∎

Corollary 5.4.1 *Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two adversaries such that $\mathcal{X}_1$ is weaker than $\mathcal{X}_2$. For any initial token distribution b, the number of rounds taken by $\mathcal{A}$ to balance b against $\mathcal{X}_1$ is at most the number of rounds taken by $\mathcal{A}$ to balance b against $\mathcal{X}_2$.*

We are now ready to establish the main result for asynchronous rings.

Theorem 2 *The number of rounds taken by $\mathcal{A}$ to balance any initial token distribution b is at most $4D(b) + 2n - 2$. The number of token transmissions is at most $T(b)$ and the number of message transmissions is at most $2T(b) + n$.*

Proof: Given any adversary $\mathcal{X}_1$, we construct a stronger adversary $\mathcal{X}_2$ that schedules each component exactly once in each round, as follows: component α is in $\mathcal{X}_2(t)$ if and only if α is in $\mathcal{X}_1(t)$ and t is the first step in the current round such that α is in $\mathcal{X}_1(t)$. We next construct an adversary $\mathcal{X}_3$ that is stronger than $\mathcal{X}_2$ such that each round of $\mathcal{X}_3$ consists of scheduling the components in the following order: first all edges of the form $(i, i-1)$ in any order, then all the nodes in any order, and finally all edges of the form $(i, i+1)$ in any order.

By the definition of $\mathcal{X}_2$, the number of rounds taken by $\mathcal{A}$ against $\mathcal{X}_1$ is at most that taken by $\mathcal{A}$ against $\mathcal{X}_2$. By the definition of $\mathcal{X}_3$, the number of rounds taken by $\mathcal{A}$ against $\mathcal{X}_3$ equals the number taken by $\mathcal{A}$ in the synchronous model, which is at most $2D(b) + n$ by Theorem 1. Moreover, it is easy to see that $\mathcal{X}_3$ can be constructed such that for any t, the number of rounds completed at the start of step t of $\mathcal{X}_3$ is at least half the number completed at the start of step t of $\mathcal{X}_2$. It thus follows from Corollary 5.4.1 that for $\mathcal{A}$, the number of rounds taken against $\mathcal{X}_1$ is at most twice the number taken against $\mathcal{X}_3$. Thus, the number of rounds taken by $\mathcal{A}$ to balance any initial token distribution b is at most $4D(b) + 2n$.

The bounds on the number of token and message transmissions follow as in the synchronous case. ■

6 Termination detection

Thus far, we have not given any mechanism for detecting termination. Fortunately, it is easy to modify our algorithm to efficiently detect termination. For instance, the following simple scheme results in only a constant factor blowup in the established time bounds. We create a dummy object, henceforth referred to as the *marker*, that is initially located at an arbitrary node. If a node i has the marker and if $w(i) = z(i) + x(i) - y(i)$, then i sends the marker to $i+1$. Let the height of the marker at any instant be defined as follows: (i) if the marker is at a node i, then its height is the number of tokens at i, and (ii) if the marker is in transit along an edge $(i, i+1)$, then the height of the marker is the number of tokens at i at the start of the last step when the marker was sent along edge $(i, i+1)$ by node i.

We detect termination on the basis of the following claim: the ring reaches a balanced state if and only if the marker is passed around the n nodes of the ring without a change in its height. We now give a sketch of the proof of the preceding claim.

We first show that if the height of the marker remains the same, say h, through n transfers, then the ring is balanced. Lemma 5.2 implies the existence of a node that has at least μ tokens throughout the execution of the algorithm. It thus follows that h is at least μ. We now show that h is at most μ. Consider the time interval during which the n transfers of the marker take place. During this time interval, whenever the marker is at a node i, no token is sent or received by i. Moreover, since communication within each edge is FIFO, no token "overtakes" the marker while the marker is in transit. Hence, each token in the ring contributes at most once to the height of the marker during the time interval. Since the height of the marker is h throughout the time interval, the total number of tokens in the ring is at least hn, implying that h is at most μ. Therefore, the ring is balanced.

The proof of the other direction is easy. Once the ring is balanced, each node has μ tokens and the marker is passed around the ring without a change in its height. Moreover, this event occurs at most n steps (or rounds) after the ring is balanced and can be detected easily. Hence, the number of steps (resp., rounds) it takes for the algorithm to terminate is $O(n)$ more than the bound established in Theorem 1 (resp., Theorem 2).

7 Concluding remarks

In Sections 4 and 5, we obtained bounds on the time taken for $\mathcal{A}$ to converge to a balanced state. One unfortunate characteristic of the bounds is the additive linear term in the time complexity of $\mathcal{A}$ (see Theorems 1 and 2). We claim that such an additive linear term is unavoidable for any distributed algorithm. To observe this, consider two initial token distributions b and c that are defined as follows. In both distributions, node 0 has two tokens and every other node except node $\lfloor n/2 \rfloor$ has one token. In distribution b, $\lfloor n/2 \rfloor$ has zero tokens, while in c, $\lfloor n/2 \rfloor$ has one token. The optimal centralized algorithm for either distribution takes at most one step. However, any distributed algorithm takes at least linear time to terminate for at least one of the two distributions, since it takes linear time to distinguish between the two distributions.

Throughout the paper we have assumed that the average number of tokens per node, μ, is an integer. A natural question is whether our analysis can be extended to handle the case of nonintegral μ. Intuitively, one might expect that the running time of $\mathcal{A}$ on a token distribution with nonintegral μ could be upper-bounded by the running time of $\mathcal{A}$ on suitably chosen token distributions for which the average number of tokens is equal to $\lfloor \mu \rfloor$ or $\lceil \mu \rceil$. We remark that this intuition can shown to hold for both the synchronous and asynchronous models of computation. In fact, it is possible to prove precisely the same bounds as in Theorems 1 and 2 for the case of nonintegral μ. We have chosen to omit the proofs, however, as they are somewhat tedious and provide no fundamental new insights into the nature of the load balancing problem.

Our model assumes that at most one token can be transmitted along any edge in any step. Our results can be easily generalized to models which allow more than one token, say c tokens, to be transmitted along an edge simultaneously. We can show that a suitable modification of $\mathcal{A}$ balances the ring to within $O(c)$ tokens in time which is optimal up to an additive $O(n)$ term.

Acknowledgments

A large part of this work was done while the authors were visiting the University of Paderborn, Germany. The authors would like to thank Friedhelm Meyer auf der Heide, Brigitte Oesterdiekhoff, and Rolf Wanka for valuable discussions.

References

[1] W. Aiello, B. Awerbuch, B. Maggs, and S. Rao. Approximate load balancing on dynamic and asynchronous networks. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 632–641, May 1993.

[2] E. Arjomandi, M. J. Fischer, and N. A. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *Journal of the ACM*, 30:449–456, 1983.

[3] A. Arora and M. Gouda. Load balancing: An exercise in constrained convergence. In J-M. Hélary and M. Raynal, editors, *Proceedings of the 9th International Workshop on Distributed Algorithms,* Lecture Notes in Computer Science, volume 972, pages 183–197. Springer-Verlag, 1995.

[4] H. Attiya and M. Mavronicolas. Efficiency of semi-synchronous versus asynchronous networks. *Mathematical Systems Theory*, 27:547–571, 1994.

[5] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35:845–875, 1988.

[6] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32:804–823, 1985.

[7] B. Awerbuch, L. Cowen, and M. Smith. Efficient asynchronous distributed symmetry breaking. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 214–223, 1994.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[9] E. Cohen. On the convergence span of greedy load balancing. *Information Processing Letters*, 52:181–182, 1994.

[10] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 2:279–301, 1989.

[11] X. Deng, H. N. Liu, L. Long, and B. Xiao. Competitive analysis of network load balancing. *Journal of Parallel and Distributed Computing*, 40:162–172, 1997.

[12] D. Eager, D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, 12:662–675, 1986.

[13] P. Fizzano, D. Karger, C. Stein, and J. Wein. Job scheduling in rings. In *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 210–219, June 1994.

[14] G. Frederickson and N. Lynch. Electing a leader in a synchronous ring. *Journal of the ACM*, 34:98–115, 1987.

[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. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 548–558, May 1995.

[16] A. Heirich and S. Taylor. A parabolic theory of load balance. Technical Report Caltech-CS-TR-93-22, Caltech Scalable Concurrent Computation Lab, March 1993.

[17] D. Hutchinson. *Local Area Network Architectures.* Addison-Wesley, 1988.

[18] L. Lamport and N. Lynch. Distributed computing: Models and methods. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1157–1199. Elsevier/MIT Press, 1990.

[19] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*. Morgan-Kaufmann, San Mateo, CA, 1991.

[20] N. Lynch and M. Fisher. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13:17–43, 1981.

[21] F. Meyer auf der Heide, B. Oesterdiekhoff, and R. Wanka. Strongly adaptive token distribution. *Algorithmica*, 15:413–427, 1996.

[22] D. Peleg and E. Upfal. The generalized packet routing problem. *Theoretical Computer Science*, 53:281–293, 1987.

[23] D. Peleg and E. Upfal. The token distribution problem. *SIAM Journal on Computing*, 18:229–243, 1989.

[24] C. G. Plaxton. Load balancing, selection, and sorting on the hypercube. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 64–73, June 1989.

[25] A. Tanenbaum. *Computer Networks*. Prentice Hall, 1989.

[26] A. N. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *Journal of the ACM*, 32:445–465, 1985.

[27] R. D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency: Practice and Experience*, 3:457–481, 1991.