Approximate Load Balancing on Dynamic and Asynchronous Networks

William Aiello
Bell Communications Research
445 South Street
Morristown, NJ 07960

Baruch Awerbuch *
Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139

Bruce Maggs
NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Satish Rao
NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

This paper presents a simple local algorithm for load balancing in a distributed network. The algorithm makes no assumption about the structure of the network. It can be executed on a synchronous network with fixed topology, a synchronous network with dynamically changing topology, or an asynchronous network. It works quickly and balances well when the network has an expansion property. In particular, we show that in an n-node network with maximum degree d whose live edges, at every time step, form a μ-expander, the algorithm will balance the load to within an additive $O(d \log n/\mu)$ term in $O(\Delta \log(n\Delta)/\mu)$ time, where Δ is the initial imbalance. The algorithm improves upon previous approaches that yield $O(n)$ time bounds in dynamic and asynchronous networks.

1 Introduction

One of the most fundamental problems to solve on a parallel computer or distributed network is to balance the load or work that must be performed among the various processors. This paper analyzes a simple version of this problem in which every processor has a number of "tokens", representing independent jobs of unit size that must be executed. Ideally, a load balancing algorithm would distribute the tokens so that each processor receives the same number. In many contexts, however, it is sufficient for a load balancing algorithm to approximately evenly distribute the tokens among the processors. For example, on a parallel computer good speedup can be achieved as long as all processors have jobs to execute and no processor's memory capacity is exceeded. As another example, approximate load balancing has been used to solve the problem of self-stabilizing end-to-end communication [AGR92, APV91] on a distributed network.

This paper presents a simple and natural algorithm for approximate load balancing on distributed networks. The algorithm can be executed on synchronous networks with fixed topology, synchronous networks with dynamically changing topology, and asynchronous networks. It works quickly and balances well when the network has an expansion property, regardless of the type of network. In particular, we show that the time required to approximately balance any network is within a logarithmic factor of the optimal time.

The performance of the algorithms in this paper depend on the number of nodes in the network, n, the maximum degree of any node, d, the initial imbalance Δ, and the expansion, μ. The *initial imbalance* is the maximum difference in the number of tokens between any two nodes. An n-node network has *expansion* μ if every set of $k \leq n/2$ nodes has at least μk neighbors outside of the set. The expansion is constant for "regular" expanders [PV76] and is constant to within a poly-logarithmic factor for special-case networks such as shuffle-exchange, Beneš, or butterfly networks [Lei92]. We show that in any network whose live edges form a μ-expander at every step, the algorithms

*Supported by Air Force contract TNDGAFOSR-86-0078, ARO contract DAAL03-86-K-0171, NSF contract CCR8611442, DARPA contract N00014-89-J-1988, and a special grant from IBM.

balance the load to within an additive $O(d \log n/\mu)$ term in $O(\Delta \log(n\Delta)/\mu)$ time. The time is within a logarithmic factor of optimal in the sense that there exists a network with expansion μ and initial imbalance Δ in which the time to balance the network to within an additive term of even $\Delta/2$ is $\Omega(\Delta/\mu)$.

The network models In this paper we consider three types of distributed networks: synchronous networks with fixed topology, synchronous networks with dynamically changing topology, and asynchronous network. In a *synchronous* network, the nodes share a global clock. In each time step, each edge can transmit a message in each direction. In a network with *dynamically changing topology* (a dynamic network), the set of edges in the network may vary at each time step. In any time step, a *live* edge is one that can transmit one message in each direction. We assume that at each time step, each node in a synchronous dynamic network knows which of its edges are live. In an *asynchronous* network, the topology is fixed, but an adversary determines the speed at which each edge operates at every instant of time. For every undirected edge between two nodes, we allow at most two messages to be in transit at any instant in time. These messages may travel in opposite directions across the edge, or both may travel in one direction, while no message travels in the opposite direction. An edge is said to be *live* for a unit interval of time if every message that was in transit across the edge (in either direction) at the beginning of the interval is guaranteed to reach the end of the edge by the end of the interval.

1.1 Related work.

Algorithms for synchronous networks with fixed topology Various load balancing algorithms have been derived for use on expander graphs [PU89] and counting networks [AHS91, HSW91, KP92, HLS92]. In both cases, the algorithms rapidly balance the tokens in a completely even fashion. In the first case, however, the algorithm fails if the expansion is not sufficiently large, and uses a specific sorting network (AKS) to perfectly balance the network. In the second case, the algorithms can be executed only on specific networks. Our algorithm does not perfectly balance the tokens, but it can be executed on any network. On networks with small expansion the algorithm balances slowly, but within a logarithmic factor of the optimal time.

None of the algorithms designed for specific networks can be used on networks with dynamically changing topology. In fact, even the algorithm designed for use on any expander graph [PU89] fails in a dynamic network because it sets up fixed paths through the network on which tokens are transmitted.

Algorithms for dynamic and asynchronous networks The best previous bounds for algorithms in dynamic and asynchronous networks [AMS89, AGR92] only guarantee a linear gap between the upper and lower bounds on the time to approximately load balance. The reason for this gap is that the techniques in [AMS89, AGR92] do not attempt to take any advantage of parallelism, and give bounds on time that are no better than the total communication complexity. We are not aware of past work capturing the effects of parallelism in a dynamic or asynchronous network model, without assuming any structure in the network.

Relationship to fault tolerance An asynchronous or dynamic network can be viewed as a network in which some nodes or edges fail during the execution of an algorithm. The fault tolerance properties of many different specific networks have been analyzed. Typically the goal is to show that a network with faults can emulate a fault-free network with little slowdown. For most of these networks, load balancing is straightforward if there are no faults. Typically the locations of the faults are assumed to be known or at least locally detectable. Hypercubes [HLN87, HLN89, AL91], arrays [KKL$^+$90], butterflies [LMS92], and multibutterflies [LM89] have been studied in this context. When the fault pattern is not known and not locally detectable it is assumed that the nodes fail independently with fixed probability. Again, these results hold for specific networks such as the hypercube [Rab89, Lyu90] or the mesh [AB92]. Our algorithm makes no assumptions on either the pattern of "faults" or the topology of the network.

Continuous methods. The techniques that we use to prove our results are related to those used by Mihail in [Mih89] to bound the convergence rate of random walks on graphs in terms of the expansion of the graph. (Other previous work [Ald83, Ald87, Alo86, AD87, Mih89, JS88] related the convergence rate to the second eigenvalue of the underlying graph and then related the second eigenvalue to the conductance.) Informally, Mihail showed that a step in a random walk corresponds to a type of averaging of probabilities across edges in the

graph. By the expansion properties of the graph, many high probability nodes will be averaged with low probability nodes. While we will also rely on the expansion properties of the network to show that nodes with many tokens will send tokens to nodes with few, Mihail's techniques differ in two fundamental ways from ours. First, she is able to average very different probabilities across an edge in one time step. This is tantamount to sending an arbitrary number of tokens across an edge in a time step. This is not possible in our model since edges represent constant capacity links. Second, she is able to average probabilities that are arbitrarily close and thus to prove a uniform rate of convergence no matter how close the distribution is to uniform. The fact that our tokens are discrete seems to limit the rate of convergence once the network is approximately balanced.

1.2 Structure of this paper

We proceed by analyzing the behavior of the algorithm on synchronous networks with fixed topologies in Section 2. In Section 3, we show that the algorithm can also be applied to dynamic synchronous networks. Finally, in Section 4 we discuss asynchronous dynamic networks.

2 A Synchronous Algorithm and Its Performance

In this section, we show that the natural algorithm balances the load in a synchronous network with fixed topology. In later sections, we extend our results to dynamic and asynchronous networks.

2.1 The Model

In the synchronous network model, each edge of the network can transmit a single token and communicate enough information to specify the number of tokens at any node in a single step. Furthermore, we assume that each node can read the head of a queue for each incoming edge in one step. And we assume that there is a global clock so that each node is executing at the same speed.

We define the state of a *load balancing problem* to be a graph, $G = (V, E)$, and a state function, $p : V \rightarrow I$, where I is the set of positive integers. The graph represents the distributed network on which the algorithm operates, and $p(v)$ represents the number of tokens at node v. We refer to $p(v)$ as the *height* of v.

We call a state, p, *totally balanced* if for any two vertices, $u, v \in V$, $|p(v) - p(u)| \leq 1$. We call a state, p, *locally balanced* if for any edge, $(u, v) \in E$, $|p(v) - p(u)| \leq 1$. We call a state x*-balanced* if for any two nodes, $u, v \in V$, $|p(v) - p(u)| \leq x$. Notice that if p is totally balanced then p is locally balanced. If p is locally balanced then p is $diam(G)$-balanced, where $diam(G)$ is the diameter of the graph.

2.2 The Algorithm

The algorithm proceeds in rounds. In each round:

- Each node, u, sends a message consisting of $p(u)$ over every incident edge.
- Each node, u, sends a token to each neighbor, v, from which it received a message, $p(v)$, where $p(v) + 2d + 1 \leq p(u)$.
- Each node, u, computes $p(u)$ by subtracting one for each token it sent and adding one for each token that it received.

Informally, each node is trying to send its tokens to neighbors with significantly fewer tokens. If a single token moves from a node v with height $p(v) = k$ to some node, u, with $p(u) = l$ then the token can be seen as moving from height k to height $l+1$. If several tokens move into u and its new height is m then one token can be seen as moving to height $l + 1$, and another can be seen as moving to height $l + 2$, and so on up to height m. Since only tokens at height greater than $l + 2d + 1$ move into u and only d tokens can enter u, we can infer that tokens only drop in height.

2.3 Analysis

We bound the time needed to approximately balance a network in terms of two measures. First, we define $\Delta(p) = \max_v(|p(v) - \frac{\sum_v p(v)}{|V|}|)$, i.e., the difference between the maximum or minimum height and the average height in the network. This is a measure of the imbalance of state p. Second, we define the expansion of a graph $G = (V, E)$ to be the minimum over $S \subset V$, where $|S| \leq |V|/2$, of $|N(S)|/|S|$, where $N(S)$ is the set of neighbors of nodes in S that are not in S. We denote the expansion of a graph G by $\mu(G)$. Notice that $\mu(G)$ is always less than or equal to 1. (We observe that the expansion provides an upper bound on the diameter of a graph. That is, any graph has diameter at most $2\log_{1+\mu(G)} |V|$.)

As stated in the introduction, these two measures can be used to prove an existential lower bound on the time needed to reach a balanced state in a graph with expansion μ and a state with imbalance Δ. That is, for any Δ and μ there is a graph G with expansion μ, and a state p with imbalance Δ where the time to even $\Delta/2$-balance is $\Omega(\Delta/\mu)$.

We provide upper bounds that are within a logarithmic factor of this existential lower bound in the theorem below. We denote $\Delta(p)$ by Δ and we denote $\Delta(p^t)$ by Δ^t.

Theorem 1 *Given an undirected load balancing state, p, on an n-node graph, $G = (V, E)$, the algorithm above will yield an $O(d \log n/\mu)$ balanced state, p', in time*

$$O\left(\frac{\Delta}{\mu}\log(n\Delta)\right),$$

where μ is the minimum expansion of G, and d is the maximum degree of any vertex in G.

Proof: In this proof, we denote the average number of tokens by ρ, i.e. $\rho = \frac{\sum_v p(v)}{n}$. We denote the state after the tth step of the algorithm by p^t. We denote the vertices with $p^t(v) \geq \rho$ by V_+^t, and the vertices with $p^t(v) \leq \rho$ by V_-^t. We let $N^t(S)$ denote the nodes that are connected to nodes in S that are not in S. When the time step is clear from the context we may abbreviate $N^t(S)$ by $N(S)$.

For the time being, we analyze only the reduction in the number of tokens whose heights are above average. At the end of the proof we explain how to analyze both this reduction and the reduction in the number of "holes" whose heights are below average. Additionally, we assume that at any step of the algorithm, the number of nodes with height above the average is at most $n/2$. Later we show how to remove this assumption.

We define a token's position at step t of the algorithm to be the node where it resides. We define its height in the node as follows. Initially, we arbitrarily number the tokens in a node, v, from 1 to $p(v)$. When a token is transmitted from a node we take it to be the token with the highest height in the node and when a token enters a node it will have the highest height in the node. (If many tokens enter or leave a node in a single step we arbitrarily order them.) We denote the height of a token τ at time t to be $h^t(\tau)$. Henceforth, we may use $h(\tau)$ to denote $h^t(\tau)$ or V_+ to denote V_+^t when the value of t is clear from the context.

We prove our theorem with the use of a potential function. We define the positive potential at time t, Φ_+^t, of a state p^t to be the sum over tokens, τ, with $h^t(\tau) > \rho$ of $h^t(\tau) - \rho$. This can also be expressed as

$$\Phi_+^t = \sum_{v \in V_+^t} (p(v) - \rho)(p(v) - \rho + 1)/2.$$

Our load balancing algorithm greedily decreases the potential function above. That is, since the potential is the sum of the heights of the tokens, whenever a token moves it drops height by at least d. The more height a token drops the more the potential drops. Intuitively, we will argue that due to the expansion of the graph there will be many edges from high height nodes to low height nodes so that there will be significant potential drop every round of the algorithm.

Claim 1 *Consider any partition, $\{S_0, ..., S_{l-1}\}$ of V_+^t and vector $a[0] = \rho, a[1], ..., a[l-1], a[l] = \infty$, such that for all $v \in S_k$, $a[k] < p^t(v) \leq a[k+1]$, and $a[k] - a[k-1] > 2d$. For any $l > b \geq 1$, the potential drop in the $t+1$st step of the algorithm by tokens, τ, where $h(\tau) > a[b+1]$ is at least*

$$\sum_{b \leq k < l-1} (a[k+1] - a[k] - 2d) C(S_{<k}, S_{>k}),$$

where $C(S, S')$ is the number of edges between nodes in S and nodes in S'. ($S_{<i}$ denotes the union of all sets S_i where $j < i$. $S_{>i}$ denotes the union of all sets S_j where $j > i$.)

Proof: We can prove a lower bound on the potential drop by examining the live edges spanning two levels of the partition. We observe from the algorithm and the fact that the maximum node degree is d, that every live edge connecting a node in S_j to a node in S_k where $k < j - 1$ transmits a token in step i of the algorithm. In addition, for every live edge, (u, v), the transmitted token comes from a height of at least $p(u) - d + 1$ and drops to height at most $p(v) + d$ since at most d messages enter v in one time step. Thus, the total potential drop on tokens, τ, with $h(\tau) \geq a[b]$ is at least

$$\sum_{(u,v) \in E_S} p(u) - p(v) - 2d + 1$$

where $E_S = \{(u, v) : (u, v)$ is live and there is a $k \geq b$ such that $u \in S_{>k}$ and $v \in S_{<k}\}$.

We define *the level* of a node, v, to be the j such that $v \in S_j$ and we denote it by $l(v)$. By recalling that $a[l(u)] + 1 \leq p(u)$ and $p(v) \leq a[l(v)+1]$, we can write a lower bound for the sum above as

$$\sum_{(u,v)\in E_S} a[l(u)] - a[l(v)+1] - 2d + 2 \geq \sum_{(u,v)\in E_S} \sum_{l(u)>k>l(v)} a[k+1] - a[k] - 2d + 2.$$

(The second expression is no higher than the first due to the fact that d is subtracted from every term of the telescoping expression for $a[l(u)] - a[l(v)+1]$.)

By noticing that a live edge, (u, v), contributes to the quantity, $C(S^t_{<k}, S^t_{>k})$, if and only if $l(u) > k > l(v)$, we can rewrite the above lower bound on potential drop as

$$\sum_{k\geq b} (a[k+1] - a[k] - 2d + 2)C(S^t_{<k}, S^t_{>k}) \geq \sum_{k\geq b} (a[k+1] - a[k] - 2d)C(S^t_{<k}, S^t_{>k}).$$

Thus, the claim holds. □

We prove our theorem by producing a partition for each time step t and using Claim 1 to show that the potential higher than a certain point drops in at least half the time steps as long as there is excess height.

For each time step, t, in $\{1 \ldots T\}$, where $T = 1536\Delta \log(n\Delta)/\mu$, we produce a partition for a state, p^t, as follows. We set $W = 4d$. Then, we set $a^t[0] = \rho$, and $a^t[i] = \rho + iW$ for i less than $\lceil \Delta^t/W \rceil$, and $a[l] = \infty$, for $l = \lceil \Delta^t/W \rceil$. We also define the corresponding partition, $\{S^t_0, \ldots, S^t_{l-1}\}$, as in Claim 1.

We define a sequence of sets, I^t_0,..,$I^t_{\lceil \Delta^t \mu/8W \rceil}$, for each time step t where

$$I^t_k = \bigcup_{k\frac{8}{\mu}\leq j<(k+1)\frac{8}{\mu}} S^t_j.$$

We mark a set, I^t_i, *good for a time step, t,* if $|I^t_i| < |I^t_{>i}|$, where $I^t_{>i} = \cup_{j>i} I^t_j$. We say that an index, k, is *good for the interval,* $\{1...T\}$, if set I^t_k is good for at least $1/2$ the values, $t \in \{1...T\}$.

We state the following claim concerning the existence of good set indices.

Claim 2 *At least one index, k, with value less than $2\log n$ is good for the interval $\{1...T\}$ as long as $\Delta^t \geq 2\frac{8}{\mu} W \log n$, for all t.*

Proof: By the definition of good we see that at most $\log n$ I^t_i's are not good at any time step, since working from $k = 0$ upwards each set I^t_k that isn't good divides the number of nodes in the higher sets by 2 and there are only $n/2$ nodes. So of the first $2\log n$ sets at least $\log n$ of the sets are good at any one time step. By averaging over time, we see that at least one of the indices is good for the interval. □

We pick an arbitrary (e.g., the lowest) good index, q, that is good for the interval. We define $b = q\frac{8}{\mu}$, i.e., the smallest index for a set S^t_i included in I^t_q. We define the s-potential for a state p^t as

$$\Phi^t_s = \sum_{v:p^t(v)>s} (p^t(v) - s)(p^t(v) - s + 1)/2,$$

or equivalently as the sum of the positive differences of token heights and $a[b]$. Reducing $\Phi^t_{a[b]}$ to zero would imply that there were no tokens with height greater than $a[b]$ at time t.

Lemma 2 *As long as $\Delta^t > 64d \log n/\mu$ for a time step t, the potential in the tokens with height larger than $a[b]$ decreases by a factor of $(1 - \mu/768\Delta^t)$ in every time step where I^t_q is good.*

The lemma implies that either Δ^t must have dropped below $128d \log n/\mu$ at some time step in $\{1...T\}$ or the b-potential drops by one half every $768\Delta/\mu$ good steps since $(1 - 1/x)^x < 1/2$ and $\Delta \geq \Delta^t$, and the $a[b]$-potential in the tokens above height $a[b+1]$ will have dropped to 0 after $T/2 = 768\Delta \log(n\Delta)/\mu$ such good steps (since $2^{-\log(n\Delta)} \leq 1/n\Delta \leq 1/\Phi_{a[b]}$) which means again that Δ^t must have dropped below $64d \log n/\mu$.

Proof: To start, we state the following central claim that follows from the expansion properties of the graph.

Claim 3 *For any i, if $|I^t_i| < 7|I^t_{>i}|$ then*

$$\sum_{i\frac{8}{\mu}\leq k<(i+1)\frac{8}{\mu}} (a[k+1] - a[k] - 2d)C(S^t_{<k}, S^t_{>k}) \geq 2d|I^t_{>i}|.$$

Proof: Since $I^t_{\geq i} < 8|I^t_{>i}|$, we can deduce that at most $6/\mu$ of the sets $S^t_k \subset I^t_i$ have size greater than $\mu/2|S^t_{>k}|$. (Otherwise the size would grow by a factor of $(1 + \mu/2)^{6/\mu} \geq 8$.)

For the other $2/\mu$ values of k,

$$C(S^t_{<k}, S^t_{>k}) > (\mu/2)|S^t_{>k}| > (\mu/2)|I^t_{>i}|.$$

The first inequality follows since $S^t_{>k}$ has at least $\mu|S^t_{>k}|$ neighbors in total by the definition of expansion and has fewer than $\mu/2|S^t_{>k}|$ neighbors in S^t_k by

the fact that $\mu/2|S^t_{>k}| > |S^t_k|$. The second inequality follows since $|S^t_{>k}| \geq |I^t_{>i}|$.

□

We proceed by evaluating the b-potential and the potential drop in a good time step for $I^t_{\geq q}$. We do this by dividing the potential into at most $\log_8 n$ groups by defining the following integers.

$$g_0 = q - 1$$

$$g_i = \text{ minimum } k > g_{i-1} \text{where } |I^t_{g_{i-1}<j\leq k}| \geq 7|I^t_{>k}|$$

where $I^t_{g_{i-1}<j\leq k} = \cup_{g_{i-1}<j\leq k} I^t_j$. We define our groups as

$$G^t_i = I^t_{g_{i-1}<j\leq g_i}.$$

We proceed by deriving a lower bound on $\sum_{k\geq b}(a[k+1] - a[k] - 2d)C(S^t_{<k}, C(S^t_{>k})$, which is a lower bound on the potential drop for tokens, τ, with $h^t(\tau) \geq a^t[b]$ by Claim 1.

By Claim 3 there is a contribution of $2d|I^t_{>j}| \geq d|G^t_i|/4$ for each set I^t_j in G_i except for $|I^t_{g_i}|$, since $|I^t_j| < 7|I^t_{>j}|$ by the definitions of g_i and G_i. There are $(g_i - g_{i-1} - 1)$ such sets I^t_j in G_i, and the contributions for different G_i's are disjoint, thus

$$\sum_{k\geq b}(a[k+1] - a[k] - 2d)C(S^t_{<k}, C(S^t_{>k}) \geq \sum_{i\geq 1}(g_i - g_{i-1} - 1)d|G^t_i|/8.$$

We refer to the series on the right above as *the potential drop series,* and to the i term in the series as $D(i)$. That is, $D(i) = (g_i - g_{i-1} - 1)d|G^t_i|/4$.

We rewrite the series as $\sum_{i>1} R(i)$ where $R(1) = D(1)/2$ and $R(i) = (D(i) + R(i-1))/2$ for later use. It can easily be verified that $\sum_{i>1} R(i) = \sum_{i>1} D(i)$.

The total potential in the groups is at most

$$\sum_{i\geq 1}((g_i - g_0 + 1)(32d/\mu))^2|G^t_i|,$$

since $(g_i - g_0 + 1)(32d/\mu)$ is the maximum height of any node in G^t_i.

We refer to the series above as the *total potential series* and to the ith term in the series above as $U(i)$. That is $U(i) = ((g_i - g_0 + 1)(32d/\mu))^2|G^t_i|$.

We proceed to derive a lower bound on the ratio of the potential drop to the total potential by inductively lower bounding the ratio of $R(i)$ to $U(i)$ for each i.

We start with $i = 1$. If we take the ratio of $R(1) = D(1)/2$ to $U(1)$ we get

$$\frac{(g_1 - g_0 - 1)d}{8((g_1 - g_0 + 1)(32d/\mu))^2} \geq \frac{(g_1 - g_0 - 1)\mu}{((g_1 - g_0 + 1)(256))\Delta}.$$

Since we know that $(g_1 - g_0) > 1$, by the fact that I^t_q is good for time step t, we can conclude that $(g_1 - g_0 - 1)/(g_1 - g_0 + 1) \geq 1/3$ and that the ratio of $R(1)$ to $U(1)$ is at least

$$\frac{\mu}{768\Delta}.$$

We proceed by proving the induction step for i assuming that the ratio of $R(i-1)$ to $U(i-1)$ is at least $\frac{\mu}{768\Delta}$. Recall, that $R(i) = D(i)/2 + R(i-1)/2$.

We analyze two cases. The first is that $U(i)$ is at most 1/2 as much as $U(i-1)$. In this case, the ratio of just $R(i-1)/2$ to $U(i)$ is sufficiently large to meet the inductive condition.

If $U(i)$ is more than $1/2$ of $U(i-1)$, we can show that just $D(i)/2$ is enough to meet the inductive condition as follows. By computing $U(i)$ and $U(i-1)$ we get

$$\begin{aligned} |G_{i-1}|(g_{i-1} - g_0 + 1)^2 &< 2|G_i|(g_i - g_0 + 1)^2 \\ &\leq 2/7|G_{i-1}|(g_i - g_0 + 1)^2. \end{aligned}$$

The second inequality holds since by the definition of the G_i's we have $|G_{i-1}| \geq 7|G_i|$.

Simplifying the equation above, we have

$$\begin{aligned} (g_{i-1} - g_0 + 1) &\leq \sqrt{2/7}(g_i - g_0 + 1) \\ &\leq 1/2(g_i - g_0 + 1). \end{aligned}$$

If we subtract $1/2(g_{i-1} - g_0 + 1)$ from both sides of the inequality above and simplify we get

$$(g_{i-1} - g_0 + 1) \leq (g_i - g_{i-1}).$$

Using the fact that $(g_i - g_0 + 1) = (g_i - g_{i-1}) + (g_{i-1} - g_0 + 1)$, we can conclude that

$$\frac{(g_i - g_{i-1})}{(g_i - g_0 + 1)} \geq 1/2.$$

Furthermore, since $g_1 - g_0 \geq 2$, we derive that

$$\frac{(g_i - g_{i-1} - 1)}{(g_i - g_0 + 1)} \geq 1/3.$$

We compute the ratio of $D(i)/2$ to $U(i)$ to be

$$\frac{(g_i - g_{i-1} - 1)\mu}{(g_i - g_0 + 1)256\Delta}.$$

Combining the equations above we get that the ratio of $D(i)/2$ to $U(i)$ is greater than

$$\frac{\mu}{768\Delta}.$$

That is, the ratio of $R(i)$ to $U(i)$ is at least $\mu/768\Delta$ for every i. Thus, the ratio of the two series is at least $\mu/768\Delta$ and the potential drop is at least a $\mu/768\Delta$ fraction of the total potential.
End of proof of lemma.

□

To complete the proof of the theorem, we must remove the assumption that at most $n/2$ nodes have height above the average, and analyze the reduction in the potential of both the tokens with height above the average and *holes* with height below average. A hole is the absence of a token. A hole at height $h \leq \rho$ has potential $\rho + 1 - h$. Note that a node v with height $p(v) \leq \rho$ has holes at heights $p(v)+1, p(v)+2, \ldots, \rho$. The potential Φ_- of the holes can be expressed as

$$\Phi_- = \frac{1}{2}\sum_{V_-}(\rho - p(v) + 1)(\rho - p(v)).$$

Note that Φ_- is non-negative.

Now consider any interval $\{1 \ldots 2T\}$ of time steps. Over this interval, either V_+ contains $n/2$ or fewer nodes on at least T steps, or V_- contains $n/2$ fewer nodes on at least T steps. Select the one that does. Without loss of generality, let it be V_+. We now apply our analysis of the reduction in the potential Φ_+ in these T steps. One concern is that during the steps in which V_+ contains more than $n/2$ nodes, Φ_+ might increase. But this can't happen. Observe that for any $\rho' \geq \rho$, at no step of the algorithm does the sum of the potential of the tokens in V_+ with height ρ' or larger increase. (Similarly, for any $\rho'' \leq \rho$, the sum of the potential of the holes in V_- with height ρ'' or smaller never increases.) Thus by our previous analysis, after $2T$ steps, there will be no more tokens in V_+ with height larger than $16W/\mu \log n$. Once all of the tokens with large height have been eliminated from V_+, it must be the case that at most $n/2$ nodes in V_- contain holes with height less than $\rho - 16W/\mu \log n$. Thus, if we run the algorithm for T more steps, the holes with height less than $\rho - 16W/\mu \log n$ will be eliminated.

We can also prove that we $O(\log n/\mu)$ balance (rather than $O(d \log n/\mu)$ balance) a synchronous network using only a factor of d more time. However, since it does not apply to dynamic or asynchronous networks we do not pursue this direction in this paper.

3 Dynamic Synchronous Networks

3.1 The Model

The model differs from the previous model in that edges may go up and down dynamically. An edge is *live* in a time step if it can transmit a message in each direction. We assume in this synchronous case that each node knows which of its edges are live. We also assume that the set of live edges at any step forms a graph that has expansion at least μ.

3.2 The Algorithm

We present an algorithm for dynamic synchronous networks below. At any point in time, each node u contains an estimate, $e_u(v)$ of the height of each of its neighbors, v. Initially, this estimate is set to zero. At each time step, each node u does the following.

1. For each live neighbor v adjacent to node u, if $e_u(v) < p(v) - 8d$, u sends a message to v consisting of a token and its current height $p(u)$, otherwise u sends a message to v consisting only of $p(v)$.

2. For each message received from a live neighbor, v, $e_u(v)$ is updated according to the message and if the message also contains a token then $p(u)$ is incremented by one.

3.3 Analysis

We analyzed the algorithm in the previous section with the understanding that each node operates based on "perfect" information about the state of its neighbors, and that at each time step, tokens move across edges whose endpoints differ sufficiently in height. We capitalized on this event by assuming that there was significant potential drop for each edge where the endpoints differed significantly in height.

The dynamic algorithm bases its decision on whether to send a token to a neighbor on a previous transmission from that neighbor that might have arrived long ago, and thus could be completely outdated. Consequently, the

real algorithm will deviate from the "idealized" algorithm in some cases.

One such case is when a token moves on edge in the "wrong" direction. That is, it moves from a lower node to a higher node. Another case is that that no packet motion takes place on an edge from a high node to a low node due to the fact that the previous height report from the low node was much higher than its current height. We can argue that the algorithm continues to make progress in spite of these deviations by using amortized analysis.

Using the following lemma, the proof of the previous section can be modified for the dynamic setting. Recall the concept of s-potential,

$$\Phi_s^t = \sum_{v:p^t(v)>s} (p^t(v) - s)(p^t(v) - s + 1)/2.$$

Lemma 3 *By time step t, there has been at least*

$$-2d|\mathcal{T}_s| + \sum_{\substack{(u,v)\in E \\ t}} \delta((u,v),t)(p^t(u) - \max(p^t(v), s) - 12d)$$

s-potential drop since the beginning of the algorithm, where $\delta((u,v),t)$ is one for each time when (u,v) is live and $(p^t(u) - \max(p^t(v), s) - 12d) \geq 0$ and zero otherwise, and $\mathcal{T}_s$ is the set of tokens with height greater than s initially.

Proof sketch of lemma: We use a credit assignment on the incoming edges of each node to make up for the deviations caused by the differences in the real and estimated heights of node's neighbors. We assign credit whenever there is a token transmission (and thus potential drop). We use credit when there was an "uphill" token transmission and when there is no token transmission when there should have been one.

For each node u and each incoming edge (v,u), we will maintain an integer weight $c(v,u)$, which we will refer to as an *incoming edge credit.* Initially we set $c(u,v) = 2\max(p(u) - s, 0)$ and $c(v,u) = 2\max(p(v) - s, 0)$ for each edge, (u,v). This corresponds to taking $d|\mathcal{T}_s|$ credit up front for potential drop that doesn't happen. This is reflected in the additive term in the lemma.

The credit functions and our lower bound on potential drop evolve as follows.

Each time a token traverses an edge, (u,v), it releases $p^t(u) - \max(p^t(v), s) + c(u,v)$ potential (and it sets $c(u,v)$ to zero.) If u has height greater than $s + 4d$ then we add two to each incoming edge credit for u and we add two to each incoming edge credit for v. The remainder is counted as potential drop.

Each time $\delta^t((u,v),t)$ is one, $c(u,v)$ is set to zero and $c(u,v)$ potential drop is counted.

We will analyze two cases: a token moves uphill over an edge and a token does not move over an edge that it should have moved over. (The situation where a token moves downhill over an edge automatically contributes the appropriate amount of potential drop and credits.) In the sketch below, we assume that there was an initial packet transmission on every edge. This assumption can be removed by arguing about the initial assignment of incoming edge credits.

In the first case a node u sends a token uphill to a node v. This results in an increase in s-potential of $\max(p(v) - \max(p(u), s), 0)$. We will argue that $c(u,v) \geq p(v) - \max(e_u(v), s) + 4d$, so that the total potential released is enough to cover the new assignment of $4d$ credit. That is, $c(u,v)$ allows us to cover both the increase in s-potential and the new assignment of credit.

We assume without loss of generality that $p(v)$ is greater than s. Furthermore, since $e_u(v) < p(u) - 8d$ at least $8d$ tokens entered v since (u,v) was last operating. At most d of the tokens that entered v after $p(v)$ was larger than $s - 4d$ could have come from nodes with height less than $s + 4d$, i.e., one for each edge. Two units of credit were assigned for each of the remaining $p(v) - \max(e_u(v), s - 4d)$ tokens. That is,

$$\begin{aligned} c(u,v) &\geq 2(p(v) - \max(e_u(v), s - 4d) - d) \\ &\geq 2(p(v) - \max(e_u(v), s)) \\ &\quad +2(\min(4d, \max(s - e_u(v), 0)) - 2d \\ &\geq p(v) - \max(e_u(v), s) + 4d \end{aligned}$$

The third inequality follows from $p(v) \geq s$ and $p(v) - e_u(v) \geq 8d$.

As we stated above this is sufficient to cover the increase in s-potential ($\max(p(v) - \max(p(u), s), 0) \leq \max(p(v) - \max(e_u(v), s))$ caused by sending a token from u to v and the cost for assigning new credit ($4d$).

We proceed with the second case: there is a live edge (u,v) at time t on which no motion takes place and the difference in heights of u and v is sufficiently high. Let

$$\text{diff}^t(u,v) = p^t(u) - \max(p^t(v), s) > 12d.$$

We will show that $c(u,v) \geq \text{diff}^t(u,v) - 12d$, which is sufficient to give the potential drop claimed in the lemma for this edge.

Then, in order for link (u, v) to stay idle at time t, it must be the case that

$$p^t(u) - e_u(v) \leq 8d.$$

Consequently,

$$p^t(v) - e_u(v) > \text{diff}^t(u, v) - 8d$$

Since the last time (u, v) was live, at least $\text{diff}^t(u, v) - 8d$ tokens left node v. Of these, at most $4d$ left when the height of v was less than $s + 4d$. Thus, $c(u, v)$ is at least $2(\text{diff}^t(u, v) - 12d)$. We removed this credit from the incoming edge and count it as potential drop.

That is, we can count at least $p^t(u) - \max(p^t(v), s) - 12d$ potential drop at every time, t, where the edge, (u, v), is live and $p^t(u) - \max(p^t(v), s) \geq 12d$. □

4 The Asynchronous Algorithm

In this section, we describe an algorithm for load balancing on an asynchronous unreliable distributed system. We briefly describe the changes that need to be made to the analysis of the previous section to apply to this situation.

4.1 The Model

In an *asynchronous* network, the topology is fixed, but an adversary determines the speed at which each edge operates at every instant of time. An edge is said to be *live* for a unit interval of time if every message that was in transit across the edge (in either direction) at the beginning of the interval is guaranteed to reach the end of the edge by the end of the interval.

4.2 The Algorithm

Initially each node marks each neighbor as "ready." Each node u repeats the following step.

1. For each ready neighbor, v, adjacent to u if $e_u(v) < p(v) - 4d$, u sends a message to v consisting of a token and its current height $p(u)$, otherwise u sends a message v consisting only of $p(v)$. It marks each such neighbor as "not ready."

2. For each message received from a neighbor, v, $e_u(v)$ is updated according to the message and if the message also contains a token then $p(u)$ is incremented by one. It marks any neighbor from which it received a message to be ready.

4.3 Analysis

We argue that the algorithm above essentially implements a synchronous algorithm on a dynamic network. We define completed transmissions as follows. We initialize the estimated heights so that two messages are sent over any edge, (u, v), at the start of the algorithm. One from u to v and one from v to u. The first transmission is *complete* when both messages arrive. The arrival of these messages allows both u and v to send new messages. The ith transmission is complete when both u and v have received i messages.

The analysis of the dynamic synchronous networks can now be used for asynchronous networks by simply replacing the concept of a live time step for an edge in Lemma 3 by the concept of a completed transmission on an edge. Using a simple induction and the definition of an asynchronous edge being live, it is easy to show that the number of completed transmissions on the edge is at least the number of time intervals where the edge is live.

References

[AB92] Y. Aumann and M. Ben-Or. Computing with faulty arrays. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 162–169, May 1992.

[AD87] D. Aldous and P. Diaconis. Strong uniform times and finite random walks. *Advances in Applied Mathematics*, 8:69–97, 1987.

[AGR92] Yehuda Afek, Eli Gafni, and Adi Rosen. Slide - a technique for communication in unreliable networks. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, pages 35–46, August 1992.

[AHS91] J. Aspnes, M. Herlihy, and N. Shavit. Counting networks and multi-processor coordination. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 348–358, May 1991.

[AL91] W. Aiello and T. Leighton. Coding theory, hypercube embeddings, and fault tolerance. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 125–136, July 1991.

[Ald83] D. Aldous. Random walks on finite groups and rapidly mixing markov chains. *Seminaire de Probabilities XVII Lecture Notes in Mathematics*, 986, 1983.

[Ald87] D. Aldous. On the markov chain simulation method for uniform combinatorial distributions and simulated annealing. *Probability in Eng. and Inf. Sci.*, 1:33–46, 1987.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[AMS89] Baruch Awerbuch, Yishay Mansour, and Nir Shavit. End-to-end communication with polynomial overhead. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 358–363, October 1989.

[APV91] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 268–277, October 1991.

[HLN87] J. Hastad, T. Leighton, and M. Newman. Reconfiguring a hypercube in the presence of faults. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 274–284, May 1987.

[HLN89] J. Hastad, T. Leighton, and M. Newman. Fast computation using faulty hypercubes. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 251–263, May 1989.

[HLS92] Maurice Herlihy, Beng-Hong Lim, and Nir Shavit. Low contention load balancing on large-scale multiprocessors. In *4'th ACM Symposium on Parallel Algorithms and Architectures*, pages 219–227, June 1992.

[HSW91] Maurice Herlihy, Nir Shavit, and Orly Waarts. Low contention linearizable counting. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 526–535, October 1991.

[JS88] M.R. Jerrum and A. Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of the permanent resolved. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 235–244, 1988.

[KKL+90] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, and A. Tsantilas. Asymptotically tight bounds for computing with faulty arrays of processors. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 285–296, October 1990.

[KP92] Michael Klugerman and Greg Plaxton. Small-depth counting networks. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 417–428, May 1992.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan-Kaufman, San Mateo, CA, 1992.

[LM89] T. Leighton and B. Maggs. Expanders might be practical: Fast algorithms for routing around faults in multibutterflies. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 384–389, October 1989.

[LMS92] T. Leighton, B. Maggs, and R. Sitaraman. On the fault tolerance of some popular bounded-degree networks. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 542–552, October 1992.

[Lyu90] Y.–D. Lyuu. Fast fault-tolerant parallel communication and on-line maintenance using information dispersal. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 378–387, July 1990.

[Mih89] Milena Mihail. Conductance and convergence of markov chains - a combinatorial treatment of expanders. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 526–531, October 1989.

[PU89] D. Peleg and E. Upfal. The token distribution problem. *SIAM J. on Comput.*, 18:229–243, 1989.

[PV76] N. Pippenger and L. G. Valiant. Shifting graphs and their applications. *J. of the ACM*, 23:423–432, 1976.

[Rab89] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. of the ACM*, 36(3):335–348, 1989.