HAL
archives-ouvertes.fr

Maintaining Balanced Trees For Structured Distributed Streaming Systems

Frédéric Giroire, Remigiusz Modrzejewski, Nicolas Nisse, Stéphane Pérennes

To cite this version:

Frédéric Giroire, Remigiusz Modrzejewski, Nicolas Nisse, Stéphane Pérennes. Maintaining Balanced Trees For Structured Distributed Streaming Systems. 20th Colloquium on Structural Information and Communication Complexity (SIROCCO), Jul 2013, Ischia, Italy. pp.177-188, 10.1007/978-3-319-03578-9_15 . hal-00845536

HAL Id: hal-00845536
https://hal.archives-ouvertes.fr/hal-00845536

Submitted on 17 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Maintaining Balanced Trees For Structured Distributed Streaming Systems [⋆]

F. Giroire, R. Modrzejewski, N. Nisse, and S. Pérennes

COATI, joint project I3S (CNRS & UNS) and INRIA, Sophia Antipolis, France.
{frederic.giroire, remigiusz.modrzejewski, stephane.perennes, nicolas.nisse} @inria.fr

Abstract. In this paper, we propose and analyze a simple localized algorithm to balance a tree. The motivation comes from live distributed streaming systems in which a source diffuses a content to peers via a tree, a node forwarding the data to its children. Such systems are subject to a high churn, peers frequently joining and leaving the system. It is thus crucial to be able to repair the diffusion tree to allow an efficient data distribution. In particular, due to bandwidth limitations, an efficient diffusion tree must ensure that node degrees are bounded. Moreover, to minimize the delay of the streaming, the depth of the diffusion tree must also be controlled. We propose here a simple distributed repair algorithm in which each node carries out local operations based on its degree and on the subtree sizes of its children. In a synchronous setting, we first prove that starting from any n-node tree our process converges to a balanced tree in $O(n^2)$ turns. We then describe a more restrictive model, adding a small extra information to each node, under which we adopt our algorithm to converge in $\Theta(n \log n)$ turns. We then exhibit by simulation that the convergence is much faster (logarithmic number of turns in average) for a random tree.

Keywords: Distributed algorithms, tree balancing, live streaming, peer-to-peer.

1 Introduction

Trees are inherent structures for data dissemination in general and particularly in peer-to-peer live streaming networks. Fundamentally, from the perspective of a peer, each atomic piece of content has to be received from some source and forwarded towards some receivers. Moreover, most of the actual streaming mechanisms ensure that a piece of information is not transmitted again to a peer that already possesses it. Therefore, this implies that dissemination of a single fragment defines a tree structure. Even in *unstructured* networks, whose main characteristic is lack of defined structure, many systems look into perpetuating such underlying trees, e.g. the second incarnation of Coolstreaming [7] or PRIME [9].

[⋆] The research leading to these results has received funding from the European Project FP7 EULER, ANR CEDRE, ANR AGAPE, Associated Team AlDyNet, project ECOS-Sud Chile and région PACA.

Unsurprisingly, early efforts into designing peer-to-peer video streaming concentrated on defining tree-based structures for data dissemination. These have been quickly deemed inadequate, due to fragility and unused bandwidth at the leaves of the tree. One possible fix to these weaknesses was introduced in SplitStream [3]. The proposed system maintains multiple concurrent trees to tolerate failures, and internal nodes in a tree are leaf nodes in all other trees to optimize bandwidth. The construction of intertwined trees can be simplified by a randomized process, as proposed in Chunkyspread [11], leading to a streaming algorithm performing better over a range of scenarios.

As found in [7], node churn is the main difficulty for live streaming networks, especially those trying to preserve structure. On the other hand, in [12] authors embrace change. Their stochastic optimization approach relies on constant random creating and breaking of relationships. To ensure network connectivity, nodes are said to keep open connections with hundreds of potential neighbours. Another approach, displayed in [8], is churn-resiliency by maintaining redundancy within the network structure. Although concentrating on a different field, authors of [10] face a similar to our own problem of maintaining balanced trees, needed for connecting wireless sensors. However, their solution is periodical rebuilding the whole tree from scratch. Our solution aims at minimizing the disturbance of nodes, whose ancestors were not affected by recent failures, as well as minimizing the redundancy in the network.

The analysis of these systems focus on the feasibility, construction time and properties of the established overlay network, see for example [3, 11] and [4] for a theoretical analysis. But these works usually abstract over the issue of tree maintenance. Generally, in these works, when some elements (nodes or links) of the networks fail, the nodes disconnected from the root execute the same procedure as for initial connection. To the best of our knowledge, there are no theoretical analysis on the efficiency of tree maintenance in streaming systems, reliability is estimated by simulations or experiments as in [3].

In this paper, we tackle this issue by designing an efficient maintenance scheme for trees. Our distributed algorithm ensures that the tree recovers fast to a "good shape" after one or multiple failures occur. We give analytic upper bounds of the convergence time. To the best of our knowledge, this is the first theoretical analysis of a repair process for live streaming systems. While the $O(n^2)$ worst case bound seems high, simulations shown in Section 5 suggest that the average case is closer to $O(\log n)$, which is lower than the conceivable time of rebuilding a tree from scratch.

The problem setting is as follows. A single source provides live media to some nodes in the network. This source is the single reliable node of the network, all other peers may be subject to failure. Each node may relay the content to further nodes. Due to limited bandwidth, both source and any other node can provide media to a limited number $k \geq 2$ of nodes. The network is organized into a logical tree, rooted at the source of media. If node x forwards the stream towards node y, then x is the parent of y in the logical tree. Note that the delay between broadcasting a piece of media by the source and receiving by a peer is given by its distance from the root in the logical tree. Hence our goal is to minimize the tree depth, while following degree constraints.

As shown in [7], networks of this kind experience high rate of node joins and leaves. Leaves can be both graceful, where a node informs about imminent departure and network rearranges itself before it stops providing to the children, or abrupt (e.g. due to connection or hardware failure). In this work, we assume a *reconnection process*: when a node leaves, its children reattach to its parent. This can be done locally if each node stores the address of its grandfather in the tree. Note that this process is performed independently of the bandwidth constraint, hence after multiple failures, a node may become the parent of many nodes. The case of concurrent failures of father and grandfather can be handled by reattaching to the root of the tree. Other more sophisticated reconnection processes have been proposed, see for example [6].

This process can leave the tree in a state where either the bandwidth constraints are violated (the degree of a node is larger than k) or the tree depth is not optimal. Thus, we propose a distributed *balancing process*, where based on information about its degree and the subtree sizes of its children, a node may perform a local operation at each turn. We show that this balancing process, starting from any tree, converges to a balanced tree and we evaluate the convergence time.

Related Work. Construction of spanning trees has been studied in the context of self-stabilizing algorithms. Herault et al. propose in [6] a new analytic model for large scale systems. They assume that any pair of processes can communicate directly, under condition of knowing receiver's identifier, what is the case in Internet Protocol. They additionally assume a discovery service and a failure detection service. Under this model they propose and prove correctness of an algorithm constructing a spanning tree over a set of processes. Similar assumptions have been used by Caron et al. in [2] to construct a distributed prefix tree and by Bosilca et al. in [1] to construct a binomial graph (Chord-like) overlay.

In this paper we assume the results of these earlier works: nodes can reliably communicate, form connections and detect failures. We do not analyze these operations at message level. Furthermore, we analyze the overlay assuming it is already a spanning tree. However, it may have an arbitrary shape, e.g. be a path or a star (all nodes connected directly to the root). This can be regarded as maintaining the tree after connection or failure of an arbitrary number of nodes.

Our results. In Section 2, we provide a formal definition of the problem and propose a distributed algorithm for the balancing process. The process works in a synchronous setting. At each turn, all nodes are sequentially scheduled by an adversary and must execute the process. In Section 3, we show that the balancing process always succeeds in $O(n^2)$ turns. Then, in Section 4, we study a restricted version of the algorithm in which a node performs an operation only when the subtrees of its children are balanced. In this case, we succeeded in obtaining a tight bound of $\Theta(n \log n)$ on the number of turns for the worst tree. Finally, we show that the convergence is in fact a lot faster in average for a random tree and takes a logarithmic number of turns.

Due to space limitations, only intuitions of some proofs are presented here. The full proofs can be found in [5].

2 Problem and Balancing Process

In this section, we present the main definitions and settings used throughout the paper, then we present our algorithm and prove some simple properties of it.

2.1 Notations

This section is devoted to some basic notations.

Let $n \in \mathbb{N}^*$. Let $T = (V, E)$ be a n-node tree rooted in $r \in V$. Let $v \in V$ be any node. The *subtree* T_v *rooted at* v is the subtree consisting of v and all its descendants. In other words, if $v = r$, then $T_v = T$ and, otherwise, let e be the edge between v and its parent, T_v is the subtree of $T \setminus e = (V, E \setminus \{e\})$ containing v. Let $n_v = |V(T_v)|$.

Let $k \geq 2$ be an integer. A node $v \in V(T)$ is *underloaded* if it has at most $k-1$ children and at least one of these children is not a leaf. v is said *overloaded* if it has at least $k+1$ children. Finally, a node v with k children is *imbalanced* if there are two children x and y of v such that $|n_x - n_y| > 1$. A node is *balanced* if it is neither underloaded, nor overloaded nor imbalanced. Note that a leaf is always balanced.

A tree is a *k-ary tree* if it has no nodes that are underloaded or overloaded, i.e., all nodes have at most k children and a node with $< k$ children has only leaf-children. A rooted k-ary tree T is *k-balanced* if, for each node $v \in V(T)$, the sizes of the subtrees rooted in the children of v differ by at most one. In other words, a rooted tree is k-balanced if and only if all its nodes are balanced.

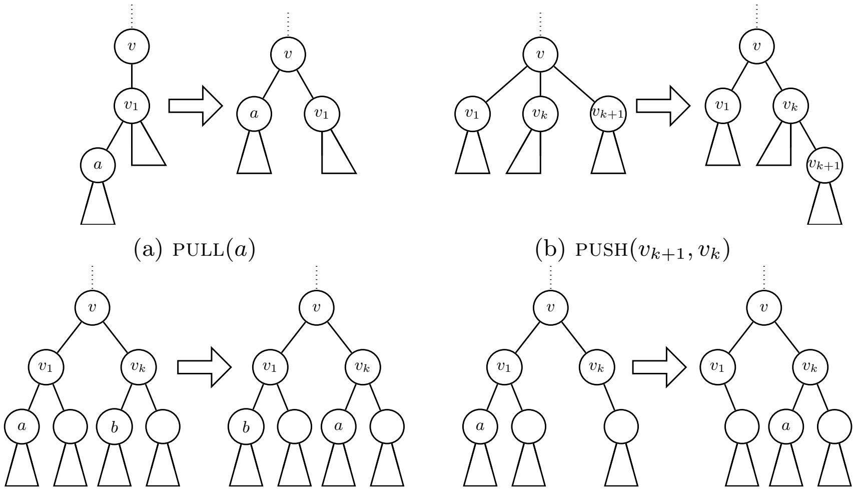
As formalized by the next claim, k-balanced trees are good for our live streaming purpose since such overlay networks (k being small compared with n) ensure a low dissemination delay while preserving bandwidth constraints.

Claim 1 *Let T be a n-node rooted tree. If T is k-balanced, then each node of T is at distance at most $\lfloor \log_k n \rfloor$ from r.*

2.2 Distributed Model and Problem

Nodes are autonomous entities running the same algorithm. Each node v has a local memory where it stores the size n_v of its subtree, the size of the subtrees of its children and the size of the subtrees of its grand-children, i.e., for any child x of v and for any child y of x, v knows n_x and n_y.

Computations performed by the nodes are based only on the local knowledge, i.e., the information present in the local memory and that concerns only nodes at distance at most 2. We consider a synchronous setting. That is, the time is slotted in turns. At each turn, any node may run the algorithm based on its knowledge and, depending on the computation, may do one of the following



(a) PULL(a)

(b) PUSH(v_{k+1}, v_k)

(c) SWAP(a, b)

(d) SWAP($a, \emptyset$)

Fig. 1: Operations performed by node v in the balancing process

operations. In the algorithm we present, each operation done by a node v consists of rewiring at most two edges at distance at most 2 from v. More precisely, let v_1, v_k and v_{k+1} be children of v, a be a child of v_1 and b be a child of v_k (if any). The node v may

- replace the edge $\{v_1, a\}$ by the edge $\{v, a\}$. A grand-child a of v then becomes a child of v. This operation is denoted by PULL(a) and illustrated in Figure 1a;
- replace the edge $\{v, v_{k+1}\}$ by the edge $\{v_k, v_{k+1}\}$. A child v_{k+1} of v then becomes a child of another child v_k of v. This operation is denoted by PUSH (v_{k+1,v_k}), see Figure 1b;
- replace the edges $\{v_1, a\}$ and $\{v_k, b\}$ by the edges $\{v_1, b\}$ and $\{v_k, a\}$. The children v_1 and v_k of v exchange two of their own children a and b. This operation is denoted by SWAP(a,b) and an example is given in Figure 1c. Here, a or b may not exist, in which case, one of v_1 and v_k "wins" a new child while the other one "looses" a child. This case is illustrated in Figure 1d.

In all cases, the local memory of the at most $k^2 + 1$, including the parent of v, nodes that are concerned are updated. Note that each of these operations may be done using a constant number of messages of size $O(\log n)$.

In this setting, at every turn, all nodes sequentially run the algorithm. In order to consider the worst case scenario, the order in which all nodes are scheduled during one turn is given by an adversary. The algorithm must ensure that after a finite number of turns, the resulting tree is k-balanced. We are interested in time complexity of the worst case scenario of the repair. That is, the performance of the algorithm is measured by the maximum number of turns after which the tree becomes k-balanced, starting from any n-node tree.

2.3 The Balancing Process

In this section, we present our algorithm, called *balancing process*. We prove some basic properties of it. In particular, while the tree is not k-balanced, the balancing process ensures that at least one node performs an operation. In the next sections, we prove that the balancing process actually allows to reach a k-balanced tree after a finite number of steps.

At each turn, a node v executes the algorithm described on Figure 2. To summarize, an underloaded node does a PULL, an overloaded node does a PUSH and an imbalanced node (whose children are not overloaded) does a SWAP operation. Note that a SWAP operation may exchange a subtree with an empty subtree, but cannot create an overloaded node. Intuitively, the children affected by PUSH and PULL are chosen to get probably the least imbalance (reduce the biggest or merge the two small). It is important to emphasise that the balancing process requires no memory of the past operations.

```
Algorithm executed by a node $v$ in a tree $T$. If $v$ is not a leaf, let $(v_1, v_2, \cdots, v_d)$ be
the $d \geq 1$ children of $v$ ordered by subtree-size, i.e., $n_{v_1} \geq n_{v_2} \geq \cdots \geq n_{v_d}$.

1. If $v$ is underloaded (then $d < k$), let $a$ be a child of $v_1$ with biggest subtree size.
   Then node $v$ executes PULL$(a)$.                    // That is, $a$ becomes a child of $v$.
2. Else if $v$ is overloaded (then $d > k \geq 2$), then node $v$ executes PUSH$(v_{k+1}, v_k)$.
                                                        // That is, $v_{k+1}$ becomes a child of $v_k$.
3. Else if $v$ is imbalanced (then $d = k$) and if $v_1$ and $v_k$ are not overloaded, let $a$ and
   $b$ be two children of $v_1$ and $v_k$ respectively such that $|n_{v_1} - n_a + n_b - (n_{v_k} - n_b + n_a)|$
   is minimum ($a$ (resp. $b$) may be not defined, i.e., $n_a = 0$ (resp., $n_b = 0$), if $v_1$
   (resp $v_2$) is underloaded).
   Then node $v$ execute SWAP$(a, b)$.        // That is, $a$ and $b$ exchange their parent.
```

Fig. 2: Balancing Process

Note that if the tree if k-balanced, no operation are performed, and that, if the tree is not, at least one operation is performed.

Claim 2 *If T is not k-balanced, and all nodes execute the balancing process, then at least one node will do an operation.*

In the next section, we prove that, starting from any tree, the number of operations done by the nodes executing the balancing process is bounded. Together with the previous claim, it allows to prove

Theorem 1. *Starting from any tree T where each node executes the balancing process, after a finite number of steps, T eventually becomes k-balanced.*

Before proving the above result in next Section, we give a simple lower bound on the number of turns required by the Balancing Process. A *star* is a rooted tree where any non root-node is a leaf.

Lemma 1. *If the initial tree is a n-node star, then at least $\Omega(n)$ turns are needed before the resulting tree is k-balanced.*

3 Worst case analysis

In this Section we obtain an upper bound of $O(n^2)$ turns needed to balance the tree. We prove it using a potential function, whose initial value is bounded, integral and positive, may rise in a bounded number of turns and, otherwise, strictly decreases. For clarity of presentation we assume we want to obtain a 2-balanced tree. The proofs can be extended to larger k. Due to lack of space, most of them are only sketched here and can be found in [5].

Lemma 2. *Starting from any n-node rooted tree T, after having executed the Balancing Process during $O(n)$ turns, no node will do a* PUSH *operation anymore.*

This lemma is proved by tracking a potential function $\Phi(T) = \sum_{v \in V(T)} \max \{0, d_v - 3\}$, where d_v is the number of children of node v. Note that any node who started a turn with degree at least three, will perform a PUSH and receive at least one new child, thus finishing the turn with degree not greater than in the beginning. Thus, no operation can increase Φ. In each turn, either Φ decreases, or a node with no overloaded ancestors performs its last PUSH. As the value of Φ is bounded by the number of nodes, the lemma holds.

Let $\mathcal{Q}$ be the sum over all nodes $u \in T$ of the distance between u and the root.

Lemma 3. *Starting from any n-node rooted tree T, there are at most $O(n^2)$ distinct (not necessarily consecutive) turns with a* PULL *operation. More precisely, the sum of the sizes of the subtrees that are pulled during the whole process does not exceed n^2.*

Proof. First, by Lemma 2, there are no PUSH operations after $O(n)$ turns. Note that a SWAP operation does not change $\mathcal{Q}$. Moreover, a PULL operation of a subtree T_v makes $\mathcal{Q}$ decrease by n_v. Since $\mathcal{Q} = \sum_{u \in V(T)} d(u, r) \leq n^2$, the sum of the sizes of the subtrees that are pulled during the whole process does not exceed n^2. □

Potential function. To prove the main result of this section, we define a potential function and show that: (1) the initial value of the potential function is bounded; (2) its value may raise due to PULL operations, but in a limited number of turns and by a bounded amount; (3) a SWAP operation may not increase its value; (4) if no PUSH nor PULL operation are done, there exists at least one node doing a SWAP operation, strictly decreasing the potential function.

We tried simple potential functions first. However, they led either to an unbounded number of turns with non-decreasing value, or to a larger upper bound. For example, it would be natural to define the potential of a node as the difference between its subtree sizes. For this potential function, (1) (2) and

(3) are true, but, unfortunately, for some trees the potential function does not decrease during a turn. This function can be patched so that each operation makes the potential decrease: multiplying the potential of a node by its distance to the root. However, the potential in this case can reach $O(n^3)$.

The potential function giving the $O(n^2)$ bound is defined as follows. Recall that we consider a n-node tree T rooted in r such that all nodes have at most two children. Let $E_0 = n$ and, for any $0 \leq i \leq \lceil \log(n+1) \rceil$, let $E_i = 2E_{i+1} + 1$. Note that $(E_i)_{i \leq \lceil \log(n+1) \rceil}$ is strictly decreasing, and $0 < E_{\lceil \log(n+1) \rceil} \leq 1$. Intuitively, E_i is the mean-size of a subtree rooted in a node at distance i from the root in a balanced tree with n nodes.

Let K_i be the set of nodes of T at distance exactly $i \geq 0$ from the root and $|K_i| = k_i$, and, for any $0 \leq i \leq \lceil \log(n+1) \rceil$, let $m_i = 2^i - k_i$. Intuitively, m_i represents the number of nodes, at distance i from the root, missing compared to a complete binary tree.

For any $v \in V(T)$ at distance $0 \leq i \leq \lceil \log(n+1) \rceil$ from the root, the *default* of v, denoted by $\mu(v)$, equals $n_v - \lceil E_i \rceil$ if $n_v > E_i$ and $\lfloor E_i \rfloor - n_v$ otherwise. Note that $\mu(v) \geq 0$ since n_v is an integer.

Let the *potential at distance i from r* , $0 \leq i \leq \lceil \log(n+1) \rceil$, be $P_i = m_i \cdot \lfloor E_i \rfloor + \sum_{u \in K_i} \mu(u)$. Finally, let us define the *potential* $\mathcal{P} = \sum_{0 \leq i \leq \lceil \log(n+1) \rceil} P_i$. Since $\mu(u) \leq n$ for any $u \in V(T)$, and $\sum_{0 \leq i \leq \lceil \log(n+1) \rceil} m_i + k_i \leq 2n$, then $\mathcal{P}(T) = O(n^2)$.

Lemma 4. *For any n-node rooted tree T, a* PULL *operation of a subtree T_v may increase the potential $\mathcal{P}$ by at most $2n_v$.*

This lemma is proved by case analysis. Let u be the node performing the operation, x its unique child and v the node being pulled. We show that the default increases by at most n_v for x, $\lfloor E_{j-1} \rfloor - \lfloor E_j \rfloor$ for nodes below it whose distance j from the root is $j \leq \lceil \log(n+1) \rceil$ and by at most n_w for every node whose distance from root is $\lceil \log(n+1) \rceil + 1$. Calculating the new potential, using all those inequations, the lemma holds.

Let v be a node at distance $\lceil \log(n+1) \rceil > i \geq 0$ from the root r of T. v is called *i-median* if it has one or two children a and b and $n_a > E_{i+1} > n_b$ (possibly v has exactly one child and $n_b = 0$).

Lemma 5. *For any n-node rooted tree T, a* SWAP *operation executed by any node v does not increase the potential $\mathcal{P}$. Moreover, if v is $(i-1)$-median then $\mathcal{P}$ strictly decreases by at least one.*

This lemma is proved by calculating the new potential, in all the possible cases of relative sizes of the children and E_i before and after the operation.

Let v be a node at distance $0 \leq i < \lceil \log(n+1) \rceil - 1$ from the root r of T. v is called *i-switchable* if it has one or two children a and b and $n_a > E_{i+1} > n_b$ (possibly v has only exactly child, and $n_b = 0$), $n_a - n_b \geq 2$ and none of its ancestors can execute a SWAP operation. Note that, if a node is *i-switchable*, then it is i-median.

Lemma 6. *Let T be a tree where no* PUSH *nor* PULL *operation is possible. If a node v is i-switchable, then either v can do a* SWAP *operation, or $0 \leq i < \lceil \log(n+1) \rceil - 2$ and it has a $(i+1)$-switchable child.*

To prove this lemma we first take care of nodes at distance $\lceil \log(n+1) \rceil$ from r, showing that in all the possible cases of its children sizes a SWAP can be performed. Then, for nodes at smaller distances to r, if an i-switchable node can not perform a SWAP, then in all possible cases one of its children is $(i+1)$-switchable.

Lemma 7. *At each turn when no* PULL *nor* PUSH *operations are done, if the tree is not balanced, then there is a i-switchable node, $0 \leq i < \lceil \log(n+1) \rceil - 1$.*

To prove this lemma, we define a $\mathcal{S}_i$*-situation*: for any $j < i$, all nodes at distance j from the root cannot do a SWAP operation, and for any $j \leq i$, $k_j = 2^j$ and, f or any node v at distance i from the root, $n_v \in \{\lceil E_i \rceil, \lfloor E_i \rfloor\}$. If the tree is in a $\mathcal{S}_{\lceil \log(n+1) \rceil - 1}$-situation, then it is balanced. Let j be the smallest integer such that T is not in a $\mathcal{S}_j$*-situation.* Then there is a node at distance $j-1$ from the root, which in all possible cases is $(j-1)$-switchable.

Theorem 2. *Starting from any n-node rooted tree, the balancing process reaches a 2-balanced tree in $O(n^2)$ turns.*

Proof. By Lemma 2, after $O(n)$ turns, no PUSH operations are executed anymore and all nodes have at most two children. From then, there may have only pull or SWAP operations. Moreover, by Claim 2, there is at least one operation per turn while T is not balanced. From Lemma 3, there are at most $O(n^2)$ turns with a PULL operation. Once no PUSH operations are executed anymore, from Lemmata 3, 4 and 5, potential $\mathcal{P}$ can increase by at most $O(n^2)$ in total (over all turns). Moreover, by Lemma 5, if a i-median node executes a SWAP operation, the potential $\mathcal{P}$ strictly decreases by at least one.

By Lemma 7, at each turn when no pull nor PUSH operations are done, there is an i-switchable node, $0 \leq i < \lceil \log(n+1) \rceil - 1$. Thus, by Lemma 6, at each such turn, there is an i-switchable that can execute a SWAP operation. Since a i-switchable node is i-median ($0 \leq i < \lceil \log(n+1) \rceil - 1$), by Lemma 5, the potential $\mathcal{P}$ strictly decreases by at least one.

The result then follows from the fact that $\mathcal{P} \leq n^2$. □

4 Adding an extra global knowledge to the nodes

In this section, we assume an extra global knowledge: each node knows whether it has a descendant that is not balanced. This extra information is updated after each operation. Then, our algorithm is modified by adding the condition that any node v executing the balancing process can do a PULL or SWAP operation only if all its descendants are balanced. Adding this property allows to prove better upper bounds on the number of steps, by avoiding conflict between an operation performed by a node and an operation performed by one of its not balanced descendant. We moreover prove that this upper bound for our algorithm is asymptotically tight, reached when input tree is a path. The approach presented in this section is specific for $k = 2$. I.e., the objective of the Balancing Process is to reach a 2-balanced tree.

First, we define a function f used to bound the number of turns needed to balance a tree consisting of two balanced subtrees and a common ancestor. Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the function defined recursively as follows.

$$\begin{array}{ll}
\forall a \geq 0, & f(a,a) = 0 \\
\forall a \geq 1, & f(a,a-1) = 0 \\
\forall a \geq 2, & f(a,0) = 1 + f(\left\lfloor \frac{a-1}{2} \right\rfloor, 0) \\
\forall a > 2, \forall 1 \leq b < a-1, & f(a,b) = 1 + \max\left(f(\left\lceil \frac{a-1}{2} \right\rceil, \left\lfloor \frac{b-1}{2} \right\rfloor), f(\left\lfloor \frac{a-1}{2} \right\rfloor, \left\lceil \frac{b-1}{2} \right\rceil)\right)
\end{array}$$

Lemma 8. *For any $a \geq 0$, $a \geq b \geq 0$, $f(a,b) \leq \max\{0, \log_2 a\}$.*

This lemma is proved by a simple induction on a. Now, we give a function bounding the number of turns needed to balance any tree of a given size. Let $g : \mathbb{N} \to \mathbb{N}$ be the function defined recursively as follows.

$$\begin{array}{ll}
\forall n \in \{0,1\}, & g(n) = 0 \\
\forall n > 1, & g(n) = \max_{a \geq b \geq 0, a+b=n-1}(\max\{g(a), g(b)\} + f(a,b))
\end{array}$$

Using a simple induction on n, we obtain that:

Lemma 9. *For any $n \geq 0$, $g(n) \leq \max\{0, n \log_2 n\}$.*

We now state our main results:

Theorem 3. *Starting from any n-node rooted tree, the balancing process with global knowledge reaches a 2-balanced tree in $O(n \log n)$ turns.*

Note first that Lemma 2 still holds with the new balancing process, that is: no node is overloaded after $O(n)$ turns. Let now x be a node with all descendants balanced. Let y and z be the children of x. We show by induction on n_y that x becomes balanced in at most $f(n_y, n_z)$ turns. Then, by induction on n, we show that T can be balanced in at most $g(n)$ turns.

Next theorem shows that there are trees starting from which the balancing process actually uses a number of turns of the order of the above upper bound.

Theorem 4. *Starting from an n-node path rooted in one of its ends, the balancing process with global knowledge reaches a 2-balanced tree in $\Omega(n \log n)$ turns.*

The proof is done by an induction on the tree size.

5 Simulations

In the previous sections we obtained upper and lower bounds for the maximum number of turns needed to balance a tree of a given size. A significant gap

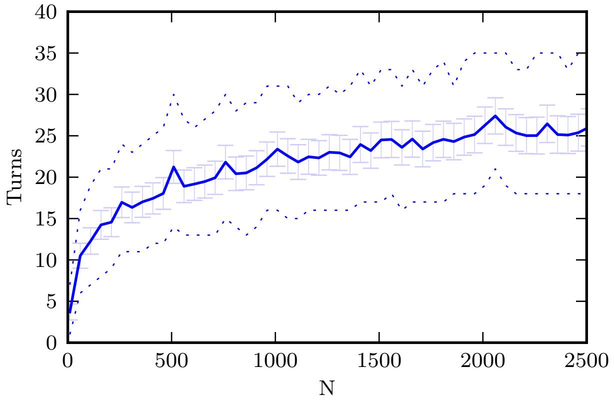


Fig. 3: Balancing a random tree

between those bounds raises the question: which bound is closer to what happens for random instances? We investigate the performance of the algorithm running an implementation under a discrete event simulation. Scheduling of nodes within a turn is given by a simple adversary algorithm. First, it detects which nodes can perform no operation. It schedules them to move first, to ensure that they do not perform operations enabled by operations of other nodes. Then, it schedules the remaining nodes in a random order.

The process starts in a random tree. It is obtained by assigning random weights to a complete graph and building a minimum weight spanning tree over it. Figure 3 displays the number of turns it took to balance trees of progressing sizes. For each size the numbers are aggregated over 10000 different starting trees. The solid line marks the average, dotted lines the minimum and maximum numbers of turns and error bars show the standard deviation.

What can be seen from this figure, is that the number of turns spent to balance a random tree progresses logarithmically in regard to the tree size. This holds true both for average and the worst cases encountered. This is significantly less even than the lower bound on maximum time. This is because that comes from the particular case of star as the starting tree, which is randomly obtained with probability $\frac{1}{n!}$ and did not occur in our experiments for bigger values of n.

6 Conclusions and future research

We have proposed a distributed tree balancing algorithm and shown following properties. The algorithm does stop only when the tree is balanced. After at most $\Omega(n)$ turns there are no overloaded nodes in the tree, what corresponds to a broadcast tree where every node receives content. This bound is reached when the starting tree is a star. Balancing process after there are no overloaded nodes lasts at most $O(n^2)$ turns. With the additional restriction that a node acts only if all of its descendants are balanced, the number of turns to balance any tree is $O(n \log n)$. This bound is reached when the starting tree is a path.

An obvious, but probably hard, open problem is closing the gap between the $O(n^2)$ upper bound and the $\Omega(n)$ lower bound on balancing time. Another possibility is examination of the algorithm's average behaviour, which as hinted by simulations should yield $O(\log n)$ bound on balancing time.

The algorithm itself can be extended to handle well the case of trees that are not regular. Furthermore, in order to approach a practical system, moving to multiple trees would be highly beneficial. Allowing the algorithm to stop with more imbalance, where children are allowed to differ by a given threshold instead of one, could lead to a faster convergence.

References

1. G. Bosilca, C. Coti, T. Herault, P. Lemarinier, and J. Dongarra. Constructing resiliant communication infrastructure for runtime environments. In *International Conference in Parallel Computing*, 2009.
2. E. Caron, A. Datta, F. Petit, and C. Tedeschi. Self-stabilization in tree-structured peer-to-peer service discovery systems. In *IEEE Symposium on Reliable Distributed Systems*, pages 207–216, 2008.
3. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Split-Stream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, page 313, 2003.
4. G. Dan, V. Fodor, and I. Chatzidrossos. On the performance of multiple-tree-based peer-to-peer live streaming. In *26th IEEE International Conference on Computer Communications*, pages 2556–2560, 2007.
5. F. Giroire, M. Remigiusz, N. Nisse, and S. Pérennes. Maintaining Balanced Trees For Structured Distributed Streaming Systems. Research Report RR-8309, INRIA, May 2013.
6. T. Herault, P. Lemarinier, O. Peres, L. Pilard, and J. Beauquier. A model for large scale self-stabilization. In *IEEE Parallel and Distributed Processing Symposium*, pages 1–10, 2007.
7. B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, and X. Zhang. Inside the new coolstreaming: Principles, measurements and performance implications. In *27th IEEE International Conference on Computer Communications*, 2008.
8. Z. Li, G. Xie, K. Hwang, and Z. Li. Churn-resilient protocol for massive data dissemination in p2p networks. *IEEE Parallel and Distributed Systems*, 22(8):1342–1349, 2011.
9. N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. *IEEE/ACM Transactions on Networking*, 17(4):1052–1065, 2009.
10. M.-S. Pan, C.-H. Tsai, and Y.-C. Tseng. The orphan problem in zigbee wireless networks. *IEEE Transactions on Mobile Computing*, 8(11):1573–1584, 2009.
11. V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *14th IEEE International Conference on Network Protocols*, pages 2–11, 2006.
12. S. Zhang, Z. Shao, and M. Chen. Optimal distributed p2p streaming under node degree bounds. In *18th IEEE International Conference on Network Protocols*, pages 253–262, 2010.