Maintaining Balanced Trees For Structured Distributed Streaming Systems

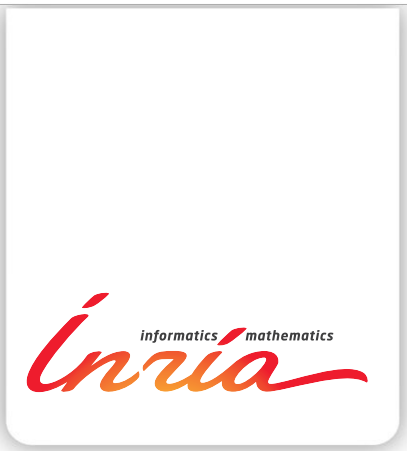
Frédéric Giroire, Remigiusz Modrzejewski, Nicolas Nisse, Stéphane Pérennes

Maintaining Balanced Trees For Structured Distributed Streaming Systems

F. Giroire , R. Modrzejewski , N. Nisse, S. Perennes

RESEARCH REPORT
N° 8309
May 21, 2013
Project-Team COATI

ISSN 0249-6399 ISRN INRIA/RR--8309--FR+ENG



Maintaining Balanced Trees For Structured Distributed Streaming Systems*

F. Giroire †, R. Modrzejewski† ‡, N. Nisse†, S. Perennes†

Project-Team COATI

Research Report n° 8309 — version 1 — initial version May 21, 2013 — revised version Mai 2013 — 19 pages

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, for which the convergence is reached in $O(n \log n)$ turns and this bound is tight. We then exhibit by simulation that the convergence is much faster (logarithmic number of turns in average) for a random tree.

Key-words: Distributed algorithms, tree balancing, live streaming, peer-to-peer

* This work was supported by European Project FP7 EULER.

† {frederic.giroire, remigiusz.modrzejewski, stephane.perennes, nicolas.nisse} @inria.fr - MASCOTTE Project, I3S (CNRS & UNS) and INRIA, INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis Cedex France.

‡ This work was partially supported by région PACA.

RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Equilibrage d'arbres pour les systèmes de diffusion en temps réel distribués

Résumé : Dans cet article, nous proposons et analysons un algorithme local simple pour équilibrer un arbre. La motivation vient des systèmes distribués de diffusion en temps réel, dans lesquels une source diffuse un contenu vers des récepteurs via un arbre, un noeud transmettant les données à ses enfants. Ces systèmes sont soumis à un niveau élevé d'arrivées et de départs (churn). Il est donc crucial d'être en mesure de réparer efficacement l'arbre de diffusion afin de permettre une distribution efficace des données. En particulier, en raison de limitations de bande passante, un arbre de diffusion efficace doit veiller à ce que les degrés des noeuds soient bornés. Par ailleurs, pour minimiser le délai de la diffusion en continu, la profondeur de l'arbre de diffusion doit également être contrôlée. Nous proposons ici un algorithme de réparation distribué simple dans lequel chaque nœud exécute des opérations locales en fonction de son degré et de la taille des sous-arbres de ses enfants. Dans un cadre synchrone, nous montrons tout d'abord, qu'à partir de n'importe quel arbre avec n nœuds, notre processus converge vers un arbre équilibré en $O(n^2)$ tours. Nous décrivons ensuite un modèle plus restrictif, en ajoutant une petite information supplémentaire pour chaque nœud, pour lequel la convergence est atteinte en $O(n \log n)$ tours, cette borne étant serrée. Nous mettons ensuite en évidence par simulation que la convergence est beaucoup plus rapide (nombre logarithmique de tours en moyenne) pour un arbre aléatoire.

Mots-clés : Algorithmes distribués, équilibrage d'arbres, diffusion en temps réel, pair-a-pair

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 [6] or PRIME [7].

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 [8], leading to a streaming algorithm performing better over a range of scenarios.

The analysis of these systems focus on the feasibility, construction time and properties of the established overlay network, see for example [3, 8] 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.

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 [6], 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 [5].

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 [5] 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 noes 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.

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.*

Proof. We prove the claim by induction. The hypothesis of induction $H_d, d \geq 1$, is: if $n < \sum_{i=0}^{d} k^i + 1$, a node is at distance at most $d-1$ from the root. H_1 is clearly true.

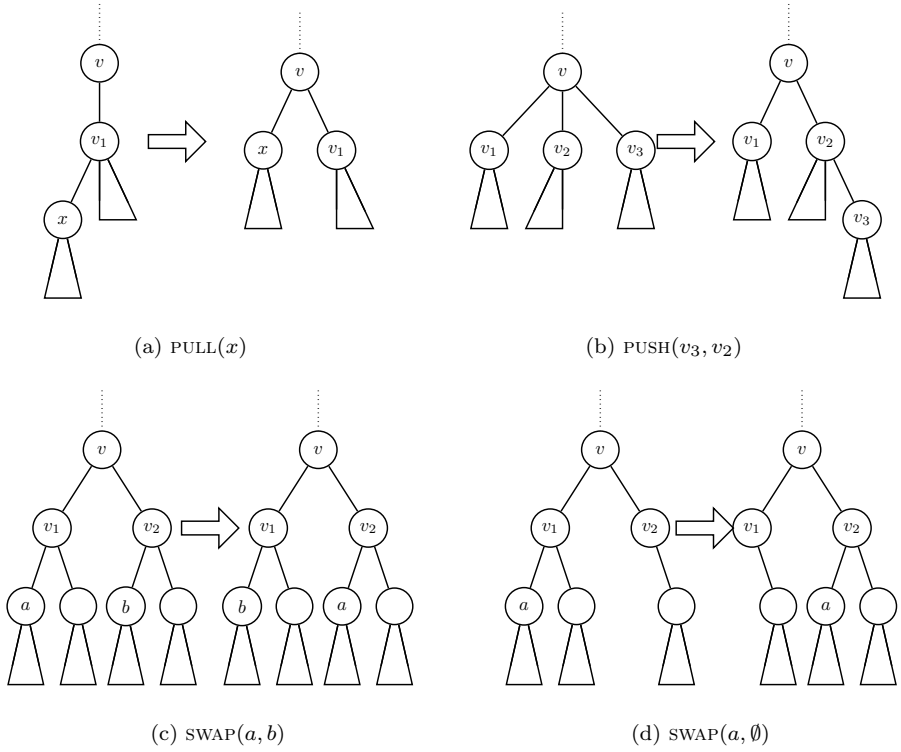
Consider now H_{d+1}. Let T be a tree of size $n < \sum_{i=0}^{d+1} k^i + 1$. Consider a child v of its root r. The size of its subtree T_v is $n_v < \sum_{i=0}^{d} k^i + 1$. Otherwise, the subtrees of the other children of r would be of size larger or equal than $\sum_{i=0}^{d} k^i$ as the root is k-balanced. The tree would be of size larger or equal than $k(\sum_{i=0}^{d} k^i) + 2 = \sum_{i=0}^{d+1} k^i + 1$. Contradiction.

By hypothesis of induction, a node of T_v is at distance at most $d-1$ from v, and thus, any node of T is at distance at most d from the root. □

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 presents 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 *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 x and y be two children of v, a be a child of x and b be a child of y (if any). The node v may



(a) PULL(x) (b) PUSH(v_3, v_2)

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

Figure 1: Operations performed by node v in the balancing process

- replace the edge $\{x, a\}$ 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 $\{v, x\}$ by the edge $\{y, x\}$. A child x of v then becomes a child of another child y of v. This operation is denoted by PUSH(x,y), see Figure 1b;
- replace the edges $\{x, a\}$ and $\{y, b\}$ by the edges $\{x, b\}$ and $\{y, a\}$. The children x and y 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 x and y "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

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 x be a child of v_1 with biggest subtree size. **Then** node v executes PULL(x). *// That is, x becomes a child of v.*

2. **Else if** v is overloaded (then $d > k \geq 2$), **then** node v executes PUSH(v_3, v_2).

   // That is, v_3 becomes a child of v_2.

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.*

Figure 2: Balancing Process

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. It is important to emphasise that the balancing process requires no memory of the past operations.

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.*

Proof. If T is not k-balanced, there exists a not k-balanced node v. According to the balancing process, v will perform an operation except if it has degree k and either its children v_1 or v_k are overloaded. In this case, the overloaded node will perform 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 to prove 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.*

Proof. Initially, the degree of the root is $n-1$. While $n > k$, the only operation that may modify the degree of the root is when the root itself does a PUSH, in which case its degree decreases by one. Hence the degree of the root decreases by at most one per turn. Since the tree cannot become k-balanced while the degree of the root is at least $k+1$, at least $n-1-k = \Omega(n)$ turns are required. □

3 Worst case analysis

In this Section we obtain an upper bound of $O(n^2)$ turns needed to balance the tree. For clarity of presentation we assume we want to obtain a 2-balanced tree. The proofs extend to larger k.

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.*

Proof. Let v be any node with degree $d \geq 3$ at the beginning of the turn and let us study how its degree evolves during one turn. We show that its degree cannot have increased at the end of the turn.

First, a simple case analysis proves that the degree of v may increase due to operations done by other nodes only in two cases. Either the parent p of v does a PUSH operation on v or p does a SWAP operation (a child of a sibling of v becoming a new child of v). In both cases, the degree of v increases by at most one. Moreover, at most one of these cases occurs in one given turn since the parent p of x is scheduled only once and if the parent of v changes during a turn, it means either that it has been pulled by a node who has already been scheduled or that it has been swapped by its grandparent g. In the last case, it cannot be swapped or pulled by the already scheduled g, and its new parent will not carry out a PUSH as it is not overloaded by definition of the balancing process.

Now, let us consider the contribution of the operation performed by v itself during this turn. If its degree is still at least three when v is scheduled, then v has to execute a PUSH operation, reducing its degree by one. Otherwise, the degree of v must be at most 2 after its executes its operation: either v had degree 1 and did a PULL operation, or it had degree 0 or 2 and its degree remains unchanged after its operation.

Let us define a potential function Φ, where $\Phi(T) = \sum_{v \in V(T)} \max\{0, d_v - 3\}$, with d_v being the number of children of node v. The above paragraphs show that the potential function Φ is not increasing during the Balancing Process, Note that $\Phi(T) \leq n$ for any n-node tree T. Therefore, there are at most $O(n)$ turns where the function strictly decreases.

To conclude, we show that during a turn, either Φ strictly decreases, or at least one node executes its last PUSH operation.

Indeed, let v be an overloaded node that is closest to the root. First, we notice that no ancestor of v can become overloaded (simple induction on the distance between v and the root). We show that the degree of v strictly decreases during this turn.

The parent p of v cannot do any PUSH operation since it cannot become overloaded anymore. A SWAP operation can increase the degree of v to at most 2, which is a decrease from $d \geq 3$ at the beginning of the turn. When v is scheduled, if its degree is at least 3, then it will perform a PUSH operation and decrease its degree, otherwise no operation will increase its degree over 2. In any case, the degree of v decreases.

Hence, either the degree of v was at least 4 and the contribution of v in Φ decreases during this turn, i.e., Φ strictly decreases. Or, v had degree 3 before the turn and is not overloaded anymore at the end of the turn. Since all its ancestors are not overloaded, v will never be overloaded again and therefore will never do another PUSH operation. □

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. 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. □

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, missing compared to a balanced 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 level i*, $0 \leq i \leq \lceil \log(n+1) \rceil$, 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 $2n_v$.*

Proof. Let us consider a PULL operation executed by node u. Let x be its unique child and let v be the child of x such that T_v is pulled by u. Let i be

the distance between x and the root. For any $j \geq i$, let L_j be the set of nodes of T_v at distance j from the root before the PULL operation and $|L_j| = \ell_j$ (note that $L_i = \emptyset$ and $\ell_i = 0$).

For any $0 \leq j \leq \lceil \log(n+1) \rceil$, let P_j be the potential at level j before the PULL operation and P'_j be this potential after the operation. Note that for any $j < i$, $P'_j = P_j$. For any node $w \in V(T)$, let $\mu(w)$ the default of w before the PULL operation and $\mu'(w)$ its default after the operation. For any $w \notin V(T_v) \cup \{x\}$, $\mu(w) = \mu'(w)$.

Moreover, either $\mu(x) = \lfloor E_i \rfloor - n_x$ and then $\mu'(x) = \lfloor E_i \rfloor - (n_x - n_v)$, or $\mu(x) = n_x - \lceil E_i \rceil$ and either $\mu'(x) = n_x - n_v - \lceil E_i \rceil$ or $\mu'(x) = \lfloor E_i \rfloor - (n_x - n_v)$. In any case, $\mu'(x) - \mu(x) \leq n_v$.

For any $w \in L_j$, $i < j \leq \lceil \log(n+1) \rceil$, there are several cases to be considered. Either $\mu(w) = \lfloor E_j \rfloor - n_w$ and then $\mu'(w) = \lfloor E_{j-1} \rfloor - n_w$. In that case, $\mu'(w) - \mu(w) \leq \lfloor E_{j-1} \rfloor - \lfloor E_j \rfloor$. Or $\mu(w) = n_w - \lceil E_j \rceil$ and $\mu'(w) = n_w - \lceil E_{j-1} \rceil$. In that case, since $\lceil E_j \rceil \leq \lceil E_{j-1} \rceil$, $\mu'(w) - \mu(w) \leq \lfloor E_{j-1} \rfloor - \lfloor E_j \rfloor$. Otherwise, $\mu(w) = n_w - \lceil E_j \rceil$ and $\mu'(w) = \lfloor E_{j-1} \rfloor - n_w$. This case occurs if $E_{j-1} > n_w > E_j$. In that case, $\mu'(w) - \mu(w) \leq \lfloor E_{j-1} \rfloor + \lceil E_j \rceil - 2n_w \leq \lfloor E_{j-1} \rfloor - \lfloor E_j \rfloor$. To summarize, in any case, $\mu'(w) - \mu(w) \leq \lfloor E_{j-1} \rfloor - \lfloor E_j \rfloor$.

Finally, for any $w \in L_{\lceil \log(n+1) \rceil + 1}$, either $\mu'(w) = n_w - \lfloor E_{\lceil \log(n+1) \rceil} \rfloor$, or $\mu'(w) = \lceil E_{\lceil \log(n+1) \rceil} \rceil - n_w \leq 1$, i.e., in any case, $\mu'(w) \leq n_w$.

For any j, $i < j \leq \lceil \log(n+1) \rceil$, $P'_j = P_j + (\ell_j - \ell_{j+1}) \lfloor E_j \rfloor + \sum_{w \in L_{j+1}} \mu'(w) - \sum_{w \in L_j} \mu(w)$. That is, $P'_j = P_j + \sum_{w \in L_{j+1}} (\mu'(w) - \lfloor E_j \rfloor) - \sum_{w \in L_j} (\mu(w) - \lfloor E_j \rfloor)$. Moreover, $P'_i = P_i - \lfloor E_i \rfloor + \mu'(v) + \mu'(x) - \mu(x)$. Finally, let $\mathcal{P}$ be the potential before the PULL operation and let $\mathcal{P}'$ be the potential after the PULL operation. Summing the previous formulas, we obtain:

$\mathcal{P}' = \mathcal{P} + \mu'(x) - \mu(x) + \sum_{i < j \leq \lceil \log(n+1) \rceil} \sum_{w \in L_j} (\mu'(w) + \lfloor E_j \rfloor - \mu(w) - \lfloor E_{j-1} \rfloor) + \sum_{w \in L_{\lceil \log(n+1) \rceil + 1}}$
$(\mu'(w) - \lfloor E_{\lceil \log(n+1) \rceil} \rfloor)$. By previous inequations, $\mathcal{P}' \leq \mathcal{P} + n_v + \sum_{w \in L_{\lceil \log(n+1) \rceil + 1}} n_w \leq \mathcal{P} + 2n_v$. □

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 only exactly 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.*

Proof. Let $i - 1$ be the distance from v to r. Let x and y be the children of v. Let a and b be the children of x and let c and d be the children of y. Without loss of generality, $n_a \geq n_b \geq 0$, and $n_c \geq n_d \geq 0$ and $n_a + n_b \geq n_c + n_d$. Because the SWAP operation is executed, then $n_a + n_d - n_b - n_c < n_a + n_b - n_c - n_d = \delta_u$ and $\delta_u > 1$, $n_b > n_d$ and $n_a > n_c$. In particular, b and d are exchanged.

For any $w \in \{x, y\}$, let $\mu(w)$ be the default of w before the SWAP operation and let $\mu'(w)$ be its default after the operation. Let $\mathcal{P}$ be the potential before the SWAP operation and let $\mathcal{P}'$ be the potential after the SWAP operation. Clearly, $\mathcal{P}' = \mathcal{P} - \mu(x) - \mu(y) + \mu'(x) + \mu'(y)$.

Clearly, if $i - 1 \geq \lceil \log(n+1) \rceil$, $\mathcal{P}' = \mathcal{P}$. Therefore, let us assume that $0 \leq i - 1 < \lceil \log(n+1) \rceil$. There are several cases to be considered.

- Case $n_x \leq E_i$. Then, $\mu(x) = \lfloor E_i \rfloor - (n_a + n_b + 1)$, $\mu'(x) = \lfloor E_i \rfloor - (n_a + n_d + 1)$ because $n_a + n_d + 1 < n_x \leq E_i$, $\mu(y) = \lfloor E_i \rfloor - (n_c + n_d + 1)$ because $n_y < n_x \leq E_i$, and $\mu'(y) = \lfloor E_i \rfloor - (n_c + n_b + 1)$ because $n_c + n_b + 1 < n_x \leq E_i$.

  $\mathcal{P}' = \mathcal{P} + \lfloor E_i \rfloor - (n_a + n_d + 1) - \lfloor E_i \rfloor + (n_a + n_b + 1) + \lfloor E_i \rfloor - (n_c + n_b + 1) - \lfloor E_i \rfloor + (n_c + n_d + 1) = \mathcal{P}$.

- Case $n_y \geq E_i$. Then, $\mu(x) = (n_a + n_b + 1) - \lceil E_i \rceil$ because $n_x > n_y \geq E_i$, $\mu'(x) = (n_a + n_d + 1) - \lceil E_i \rceil$ because $E_i \leq n_y < n_a + n_d + 1$, $\mu(y) = (n_c + n_d + 1) - \lceil E_i \rceil$, and $\mu'(y) = (n_c + n_b + 1) - \lceil E_i \rceil$, because $E_i \leq n_y < n_c + n_b$.

  Again, $\mathcal{P}' = \mathcal{P}$.

- Case $n_y < n_c + n_b + 1 \leq E_i \leq n_a + n_d + 1 < n_x$. Then, $\mu(x) = (n_a + n_b + 1) - \lceil E_i \rceil$, $\mu'(x) = (n_a + n_d + 1) - \lceil E_i \rceil$, $\mu(y) = \lfloor E_i \rfloor - (n_c + n_d + 1)$ and $\mu'(y) = \lfloor E_i \rfloor - (n_c + n_b + 1)$.

  Thus, $\mathcal{P}' = \mathcal{P} + (n_a + n_d + 1) - \lceil E_i \rceil - (n_a + n_b + 1) + \lceil E_i \rceil + \lfloor E_i \rfloor - (n_c + n_b + 1) - \lfloor E_i \rfloor + (n_c + n_d + 1) = \mathcal{P} + 2n_d - 2n_b \leq \mathcal{P} - 1$.

- Case $n_y < n_a + n_d + 1 \leq E_i \leq n_c + n_b + 1 < n_x$. Then, $\mu(x) = (n_a + n_b + 1) - \lceil E_i \rceil$, $\mu'(x) = \lfloor E_i \rfloor - (n_a + n_d + 1)$, $\mu(y) = \lfloor E_i \rfloor - (n_c + n_d + 1)$ and $\mu'(y) = (n_c + n_b + 1) - \lceil E_i \rceil$. Thus, $\mathcal{P}' = \mathcal{P} + \lfloor E_i \rfloor - (n_a + n_d + 1) - (n_a + n_b + 1) + \lceil E_i \rceil + (n_c + n_b + 1) - \lceil E_i \rceil - \lfloor E_i \rfloor + (n_c + n_d + 1) = \mathcal{P} - 2n_a + 2n_c \leq \mathcal{P} - 1$.

- Case $n_y < \max\{n_c + n_b + 1, n_a + n_d + 1\} \leq E_i < n_x$. Then, $\mu(x) = (n_a + n_b + 1) - \lceil E_i \rceil$, $\mu'(x) = \lfloor E_i \rfloor - (n_a + n_d + 1)$, $\mu(y) = \lfloor E_i \rfloor - (n_c + n_d + 1)$ and $\mu'(y) = \lfloor E_i \rfloor - (n_c + n_b + 1)$.

  $\mathcal{P}' = \mathcal{P} + \lfloor E_i \rfloor - (n_a + n_d + 1) - (n_a + n_b + 1) + \lceil E_i \rceil + \lfloor E_i \rfloor - (n_c + n_b + 1) - \lfloor E_i \rfloor + (n_c + n_d + 1) = \mathcal{P} + \lfloor E_i \rfloor + \lceil E_i \rceil - 2(n_a + n_b + 1) = \mathcal{P} + \lfloor E_i \rfloor + \lceil E_i \rceil - 2n_x$.

  Since $n_x > \lceil E_i \rceil \geq \lfloor E_i \rfloor$, $\mathcal{P}' \leq \mathcal{P} - 1$.

- Case $n_y < E_i \leq \min\{n_c + n_b + 1, n_a + n_d + 1\} < n_x$. Then, $\mu(x) = (n_a + n_b + 1) - \lceil E_i \rceil$, $\mu'(x) = (n_a + n_d + 1) - \lceil E_i \rceil$, $\mu(y) = \lfloor E_i \rfloor - (n_c + n_d + 1)$ and $\mu'(y) = (n_c + n_b + 1) - \lceil E_i \rceil$.

  $\mathcal{P}' = \mathcal{P} + (n_a + n_d + 1) - \lceil E_i \rceil - (n_a + n_b + 1) + \lceil E_i \rceil + (n_c + n_b + 1) - \lceil E_i \rceil - \lfloor E_i \rfloor + (n_c + n_d + 1) = \mathcal{P} + 2(n_c + n_d + 1) - \lceil E_i \rceil - \lfloor E_i \rfloor = \mathcal{P} + 2n_y - \lceil E_i \rceil - \lfloor E_i \rfloor \leq \mathcal{P} - 1$

Since v is $(i-1)$-median if and only if one of the last four cases is concerned, this concludes the proof. □

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.*

Proof. Let v be a i-switchable node ($0 \leq i < \lceil \log(n+1) \rceil - 1$) and let x be its greatest child and y its other child if any (possibly $n_y = 0$).

Because no PUSH operation is possible, all nodes have at most two children.

First, let us assume that $i = \lceil \log(n+1) \rceil - 2$. By definition, $n_y < E_{i+1} = E_{\lceil \log(n+1) \rceil - 1} \leq 3$. Hence, either $n_y = 0$ and $n_x \geq 2$ and v must do a PULL operation which is not possible, or $n_y = 1$ and $n_x \geq 3$, or $n_y = 2$ and $n_x \geq 4$. For the last two cases, x cannot have only one child, since otherwise he should execute a PULL operation which is not possible. Therefore, it is easy to check that, in the last two cases, v can execute a SWAP operation.

Now, assume that $i < \lceil \log(n+1) \rceil - 2$. Because $n_v > n_x > E_i \geq 3$ and no PULL operation is executed by v, then v has two children x and y. Let a and b be the two children of x (if any) and let c and d be the two children of y (if any). Without loss of generality, $n_x \geq n_y$, $n_a \geq n_b$ and $n_c \geq n_d$. Because v is i-median, then $n_x > E_{i+1} > n_y$.

Let us assume that v cannot do any SWAP operation. Then, either $n_x - n_y \leq 1$, or $n_d \geq n_b$ (and then $n_c \leq n_a$), or $n_c \geq n_a$ (and then $n_b \geq n_d$). The first case is not possible since v is i-switchable and $n_x - n_y \geq 2$. Therefore, there are only two cases to be considered.

- If $n_a \geq n_c \geq n_d \geq n_b$, then $2n_a + 1 \geq n_a + n_b + 1 = n_x > E_{i+1} = 2E_{i+2} + 1 > n_y = n_c + n_d + 1 \geq 2n_b + 1$ and $n_a > E_{i+2} > n_b$. Moreover, $1 + n_a + n_b = n_x \geq n_y + 2 = 3 + n_c + n_d \geq 2n_b + 3$ and $n_a \geq n_b + 2$. Hence, x is $i+1$-switchable.
- If $n_c \geq n_a \geq n_b \geq n_d$, then $2n_c + 1 \geq n_a + n_b + 1 = n_x > E_{i+1} = 2E_{i+2} + 1 > n_y = n_c + n_d + 1 \geq 2n_d + 1$ and $n_c > E_{i+2} > n_d$. Moreover, $2n_c + 1 \geq n_a + n_b + 1 = n_x \geq n_y + 2 = n_c + n_d + 3$ and $n_c \geq n_d + 2$. Hence, y 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, $0 \leq i < \lceil \log(n+1) \rceil - 1$.*

Proof. Let a and b be the two children of the root (r has two children since otherwise a PULL operation may be done or the tree has two nodes and is balanced). Recall that $E_0 = n = n_a + n_b + 1 = 2E_1 + 1$.

- If $n_a = n_b$, the root is balanced and cannot execute a SWAP operation. Moreover, $E_1 = n_a = n_b = (n-1)/2$.
- Otherwise, assume without loss of generality, $n_a > n_b$, then $n_a > E_1 = (n_a + n_b)/2 > n_b$.
  - If $n_a > E_1 > n_b$ and $n_a - n_b \geq 2$, then the root is 0-switchable.
  - If $n_a > E_1 > n_b$ and $n_a - n_b \leq 1$ then the root cannot execute a SWAP operation (since no such operation can decrease the difference between its subtrees).

Therefore, either the root is 0-switchable, or we are in a $\mathcal{S}_1$-*situation*: the two children a and b of the root are such that $n_a = n_b = E_1$ or $n_a > E_1 > n_b$

and $n_a - n_b \leq 1$, and in both cases, $n_a, n_b \in \{\lceil E_1 \rceil, \lfloor E_1 \rfloor\}$ and the root cannot perform a SWAP operation.

Let $i \geq 1$. Assume that we are in 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, for any node v at distance i of the root, $n_v \in \{\lceil E_i \rceil, \lfloor E_i \rfloor\}$.

First, note that if the tree is in a $\mathcal{S}_{\lceil \log(n+1) \rceil - 1}$-situation, then it is balanced. Therefore, let $j \leq \lceil \log(n+1) \rceil - 1$ be the smallest integer such that T is not in a $\mathcal{S}_j$-situation. For any node u at distance $j-1$ from the root, $n_u \geq \lfloor E_{j-1} \rfloor \geq \lfloor E_{\lceil \log(n+1) \rceil - 2} \rfloor \geq 3$. Therefore, u has exactly two children since if it has more children, a PUSH operation would be possible, and if it has only one child, a PULL operation would be possible (note that, such a PULL operation would actually be done during the turn since all ancestors of u cannot do a SWAP operation).

Since the tree is not in a $\mathcal{S}_j$-situation, there is a node u at distance $j-1$ from the root and with two children a and b such that, without loss of generality, $n_a \notin \{\lceil E_j \rceil, \lfloor E_j \rfloor\}$. However, $n_a + n_b + 1 = n_u \in \{\lceil E_{j-1} \rceil, \lfloor E_{j-1} \rfloor\} = \{\lceil 2E_j + 1 \rceil, \lfloor 2E_j + 1 \rfloor\}$.

Assume first that $n_a > \lceil E_j \rceil$. Then, $n_b = n_u - n_a - 1 \leq n_u - 2 - \lceil E_j \rceil \leq \lceil 2E_j + 1 \rceil - 2 - \lceil E_j \rceil \leq 2\lceil E_j \rceil - 1 - \lceil E_j \rceil \leq \lceil E_j \rceil - 1 < E_j$. Hence, $n_a > E_j > n_b$ and $n_a - n_b \geq 2$ and u is $j-1$-switchable.

Similarly, if $n_a < \lfloor E_j \rfloor$, then $n_b = n_u - n_a - 1 \geq \lfloor 2E_j + 1 \rfloor - \lfloor E_j \rfloor \geq \lfloor E_j \rfloor + 1 > E_j$. Again, u 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 Corollary 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 is asymptotically tight.

Again, for ease of presentation, we assume in this section that $k = 2$, i.e., the objective of the Balancing Process is to reach a 2-balanced tree.

Let $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the function defined recursively as follows.

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

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

Proof. The proof is by induction on a. If $a \leq 1$, then $f(a,b) = 0$ and $f(2,b) \leq 1$ for any $0 \leq b \leq a$ and the result holds. Let $a > 2$ and assume the result is true for any $0 \leq a' < a$. Then, $f(a,0) = 1 + f(\lfloor \frac{a-1}{2} \rfloor, 0) \leq 1 + \log_2 \lfloor \frac{a-1}{2} \rfloor \leq \log_2 a$ and the result holds. Finally, for any $b < a-1$, $f(a,b) = 1 + \max\left(f(\lceil \frac{a-1}{2} \rceil, \lfloor \frac{b-1}{2} \rfloor), f(\lfloor \frac{a-1}{2} \rfloor, \lceil \frac{b-1}{2} \rceil)\right)$. Because $a > 2$ and $b < a-1$, then $\lfloor \frac{a-1}{2} \rfloor \geq \lceil \frac{b-1}{2} \rceil$. Therefore, the induction hypothesis applies and $f(a,b) \leq 1 + \log_2 \lceil \frac{a-1}{2} \rceil \leq \log_2 a$. □

Let $g : \mathbb{N} \to \mathbb{N}$ be the function defined recursively as follows.

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

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

Proof. The proof is by induction on n. If $n \leq 1$, then $g(n) = 0$ and $g(2) = f(1,0) = 0$. Let $n > 2$ and assume that $g(n') \leq n' \log_2 n'$ for any $2 \leq n' < n$. Then, for any $0 \leq b \leq a$ with $a + b = n - 1$, the induction hypothesis implies that $\max\{g(a), g(b)\} \leq (n-1)\log_2(n-1)$ and, by Lemma 8, $f(a,b) \leq \log_2 a \leq \log_2(n-1)$, so $g(n) \leq n \log_2(n-1) \leq n \log_2 n$ and the result holds. □

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.*

Proof. Let $B(n)$ be the maximum number of turns that is needed to reach a 2-balanced tree starting from any tree with at most n nodes. Recall that we consider that all nodes execute the Balancing Process with the extra constraint that a node can execute a PULL or SWAP operation only if all its descendants are balanced.

In this setting, the result of Lemma 2 still holds and starting from any tree with at most $O(n)$ nodes, there are no overloaded anymore and no PUSH operation will never been execute again. Actually, in this setting, the proof of Lemma 2 becomes easier since the parent of an overloaded node cannot execute a SWAP or a PULL operation.

Since we aim at proving that $B(n) = O(n \log n)$, the first $O(n)$ turns are negligible and we may consider only starting trees without overloaded nodes.

Let T be any n-node tree rooted in r. Let x be any not balanced node with two children y and z such that $n_y - 1 > n_z \geq 0$. Note that, because x is not

balanced, $n_y \geq 2$. Note also that possibly $n_z = 0$ (i.e., $T_z = \emptyset$) in which case, x is underloaded.

Assume that all descendants of x are balanced, we first show that after at most $f(n_y, n_z)$ turns, all nodes in T_x are balanced. It is important that while there is at least one node that is not balanced in T_x, no operation done by a node in $V(T) \setminus V(T_x)$ will affect T_x. Hence, we can consider only the operations executed by nodes in T_x.

The proof is by induction on n_y. If $n_y = 2$, then x executes a PULL operation after which all nodes in T_x become balanced. Since $f(2, 0) = 1$, the result holds. Hence, let $n_y > 2$. There are two cases to be considered.

- if x is underloaded, let u and v be the two children of y. Because $n_y > 2$ and all nodes in T_y are balanced, u and v actually exist and $|n_u - n_v| \leq 1$ and $n_u + n_v = n_y - 1$. W.l.o.g. $n_u \geq n_v$ and therefore, $n_v \leq \lfloor \frac{n_y - 1}{2} \rfloor$. Then x executes PULL(u). Then, u is now a child of x and T_u are still balanced. x is balanced as well, as $|n_v + 1 - n_u| \leq 1$. On the other hand, y has now a single child v and all its descendants are balanced. By induction, all nodes in T_y become balanced after at most $f(n_v, 0)$ turns, i.e., by Lemma 8, after at most $\log_2 n_v$ turns.

  In total, all nodes in T_x become balanced after at most $1 + \log_2 n_v \leq 1 + \log_2 \lfloor \frac{n_y - 1}{2} \rfloor = f(n_y, 0)$.

- if x is imbalanced, then $n_y - n_z > 1$. Let y_1 and y_2 be the two children of y and let z_1 and z_2 be the two children of z. Because y and z are balanced, $|y_1 - y_2| \leq 1$ and $|z_1 - z_2| \leq 1$. W.l.o.g., $y_1 \geq y_2$ and $z_1 \geq z_2$. Then, x executes SWAP(y_2, z_2). Now, y is the parent of y_1 and z_2 and all its descendants are balanced. Similarly, z is the parent of y_2 and z_1 and all its descendants are balanced.

  Because $|y_1 - y_2| \leq 1$ and $|z_1 - z_2| \leq 1$, then x becomes balanced but y and z may now be not balanced anymore.

  Note that, while not all nodes in T_y and T_z are balanced, the operations executed in one of these trees does not affect the other one. By induction, all nodes in T_y become balanced after at most $f(n_{y_1}, n_{z_2})$ turns and all nodes in T_Z become balanced after at most $f(n_{y_2}, n_{z_1})$ turns.

  In total, all nodes in T_x become balanced after one SWAP operation and the maximum number of turns for all nodes in T_y and T_z to become balanced. Therefore, it takes at most $1 + \max\{f(n_{y_2}, n_{z_1}), f(n_{y_1}, n_{z_2})\} \leq f(n_y, n_z)$ turns.

Now, we are ready to prove the theorem. We prove by induction on n that $B(n) \leq g(n)$ and the theorem directly follows from Lemma 9. The result clearly holds for $n \leq 1$.

Let T be a tree rooted in r with at most n nodes. Let a and b the children of the root. While some descendant of r is not balanced, r does not execute any action. By definition of B, as both T_a and T_b behave as independent trees, all descendants of r become balanced after $\max\{B(n_a), B(n_b)\}$ turns, i.e., by the induction hypothesis, after $\max\{g(n_a), g(n_b)\}$ turns. Finally, by the above paragraph, at most $f(n_a, n_b)$ additional turns are sufficient for all nodes to become balanced. Hence, T becomes 2-balanced after at most $\max\{g(n_a), g(n_b)\} + f(n_a, n_b)$ turns.

Then, $B(n) \leq \max_{a \geq b \geq 0, a+b=n-1}(\max\{g(a), g(b)\} + f(a,b)) = g(n)$ turns. □ □

Next lemma 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.*

Proof. Let $h : \mathbb{N} \to \mathbb{N}$ be the function defined as:

$$h(d) = 2^d + \max(0, \sum_{i=1}^{d-2} 2^i).$$

Let $d \geq 1$. Let $\mathcal{T}_d$ be the set of trees defined as follows. For any $T \in \mathcal{T}_d$, T has $n+1$ nodes where $h(d-1) \leq n < h(d)$ and consists of a root r with a unique child u and u is the root of an n-node balanced tree.

We first prove by induction on $d \geq 1$ that, starting from any tree in $\mathcal{T}_d$, there is a schedule for the adversary such that the balancing process with global knowledge reaches a 2-balanced tree in exactly $d-1$ turns. This is clearly true for $d = 1$. The balanced subtree rooted in u is of size at most 1, hence the tree is already balanced. So no operations ($d - 1 = 0$) are needed.

Let $d \geq 1$ and let us assume by induction that any tree in $\mathcal{T}_d$ is balanced in $d-1$ turns. Let $T \in \mathcal{T}_{d+1}$. Note first that, all nodes of T but the root are balanced. Therefore, during the first turn, the best schedule for the adversary is to schedule the root last. When the root is scheduled, it must execute a PULL. Therefore, at the end of the first turn, the tree consists of the root r with two children, u and a new child v (that was a child of u before the PULL). The subtree T_v is balanced and u has a unique child w and T_w is balanced with at most $n_w = \lfloor \frac{n-1}{2} \rfloor$ nodes. Since $h(d) \leq n < h(d+1)$, we have $2^d + \sum_{i=1}^{d-2} 2^i \leq n_w \leq 2^d + \sum_{i=1}^{d-2} 2^i$. That is, $h(d-1) \leq n_w < h(d)$. Hence, $T_u \in \mathcal{T}_d$. Moreover, since the root does not execute any operation while the nodes of T_u are not balanced, we can consider T_u as an independent subtree and the induction hypothesis holds. Therefore, there is a schedule for the adversary such that T_u is balanced after exactly $d-1$ turns. In total, there is an adversary that implies that $T \in \mathcal{T}_{d+1}$ requires d turns to become balanced.

We now prove the Lemma. For any $n \geq 1$, let d_n be the integer such that $h(d_n) \leq n < h(d_n + 1)$.

Consider an n-node path rooted in one of its ends and the following schedule of the nodes. Let us define the beginning of Phase i, $i = 1..n$, as the turn when the tree is composed of a path P^i of length $n - i$ with one end the root r and the other end is a node v attached to a balanced subtree T^i of size i. During the Phase i, the adversary schedules the nodes as follows: at each turn, all nodes of $P^i \setminus \{v\}$ (they don't do anything since they have unbalanced descendant), then the nodes of $T^i \cup \{v\}$ in the same ordering as defined above. Hence, Phase i boils down to balancing a tree composed of a root node attached to a single balanced subtree. By above paragraph, Phase i lasts at least d_i turns.

Hence, the tree will be balanced in $N = \sum_{i=1}^{n} d_i$ turns. Hence,

$$N \geq \sum_{i=1}^{h(d_n)-1} d_i \geq \sum_{i=1}^{d_n-1} (i-1)(h(i+1) - h(i)) \geq (d_n - 2)(h(d_n) - h(d_n - 1))$$

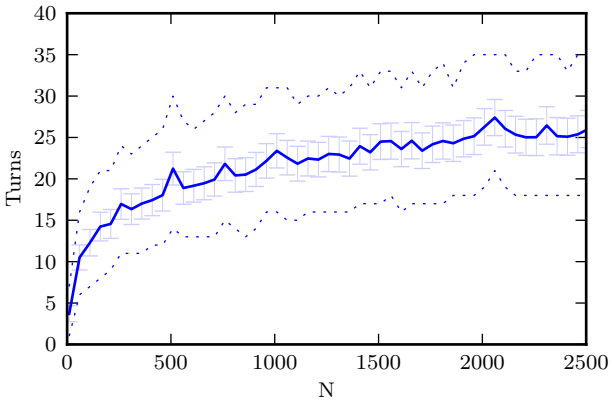


Figure 3: Balancing a random tree

By definition of h, $h(d_n) - h(d_n - 1) = 2^{d_n} + 2^{d_n-2} - 2^{d_n-1} > 2^{d_n-2}$. Hence,

$$N \geq (d_n - 2)2^{d_n-2}.$$

Note now that by definition $n < h(d_n + 1) \leq 2^{d_n+2}$. It implies that $d_n > \log_2 n - 2$. Finally, we obtain $N = \Omega(n \log n)$. □ □

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 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.

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 (ParCo2009)*, 2009.

[2] E. Caron, A. Datta, F. Petit, and C. Tedeschi. Self-stabilization in tree-structured peer-to-peer service discovery systems. In *Reliable Distributed Systems, 2008. SRDS'08. IEEE Symposium on*, pages 207–216. IEEE, 2008.

[3] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, page 313. ACM, 2003.

[4] G. Dan, V. Fodor, and I. Chatzidrossos. On the performance of multiple-tree-based peer-to-peer live streaming. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2556–2560. IEEE, 2007.

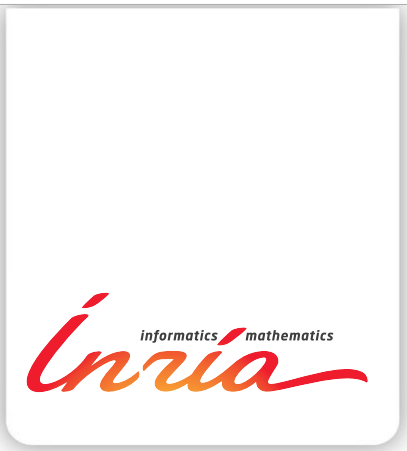
[5] T. Herault, P. Lemarinier, O. Peres, L. Pilard, and J. Beauquier. A model for large scale self-stabilization. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.

[6] 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 *Proc. of IEEE Infocom*, 2008.

[7] N. Magharei and R. Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. *IEEE/ACM Transactions on Networking (TON)*, 17(4):1052–1065, 2009.

[8] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *Network Protocols,*

2006. ICNP'06. Proceedings of the 2006 14th IEEE International Conference on, pages 2–11. IEEE, 2006.



RESEARCH CENTRE
SOPHIA ANTIPOLIS – MÉDITERRANÉE

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399