CBNet: Minimizing Adjustments in Concurrent Demand-Aware Tree Networks

Otavio Augusto de Oliveira Souza[1] Olga Goussevskaia[1] Stefan Schmid[2]
[1] Universidade Federal de Minas Gerais, Brazil [2] University of Vienna, Austria

***Abstract*—This paper studies the design of demand-aware network topologies: networks that dynamically adapt themselves toward the demand they currently serve, in an online manner. While demand-aware networks may be significantly more efficient than demand-oblivious networks, frequent adjustments are still costly. Furthermore, a centralized controller of such networks may become a bottleneck.**

We present CBNet (Counting-Based self-adjusting Network), a demand-aware network that relies on a distributed control plane supporting concurrent adjustments, while significantly reducing the number of reconfigurations, compared to related work. CBNet comes with formal guarantees and is based on concepts of self-adjusting data structures. We evaluate CBNet analytically and empirically and we find that CBNet can effectively exploit locality structure in the traffic demand.

***Index Terms*—Self-adjusting networks; decentralization; concurrency, online algorithms**

I. Introduction

Empirical studies show that communication patterns in datacenters feature much spatial and temporal structure [1]–[4], i.e., traffic is bursty, and traffic matrices skewed. This structure represents an untapped potential for building more efficient communication networks: today's datacenter networks are designed in a manner which is entirely *oblivious* to the communication pattern. In contrast, a demand-aware network, e.g., based on emerging reconfigurable optical technologies, may self-adjust to better serve the elephant flows in the network [2], [5]–[10].

A key challenge in the design of self-adjusting networks is to strike a balance between the benefits and costs of reconfigurations: while reconfigurations allow to reduce communication costs, by moving frequently communicating nodes (e.g., top-of-rack switches) topologically closer, such reconfigurations should be used in moderation: reconfigurations take time and may temporarily lead to packet loss. A second challenge is related to the collection of data about the demand as well as the decision making based on this data: these are two inherent operations in any demand-aware network, but may become an operational bottleneck. Furthermore, communication patterns may not be perfectly predictable, and hence, reconfiguration decisions need to be taken *in an online manner*.

This paper investigates the algorithmic problem underlying such demand-aware and self-adjusting networks. In particular, to address the above challenges, we introduce CBNet, a distributed and concurrent demand-aware network based on tree topologies, which aims to reduce reconfigurations compared to the state-of-the-art solutions, such as SplayNet [7] and DiSplayNet [11].

CBNet is based on concepts from self-adjusting data structures, and in particular, CBTrees [12]. CBNet gradually adapts the network topology toward the communication pattern in an online manner, i.e., without previous knowledge of the demand distribution. At the same time, *bidirectional semi-splaying* and counters are used to maintain state, minimize reconfiguration costs and maximize concurrency.

Contributions. Our main contribution is CBNet, a counting-based self-adjusting network, which adapts to the unknown traffic pattern in a demand-aware and online manner, while minimizing reconfiguration costs. CBNet relies on decentralized and concurrent algorithms and comes with provable (worst-case) performance guarantees, and also fares well under realistic workloads: we report on extensive simulations using both real and synthetic workloads, and find that CBNet can indeed outperform state-of-the-art, by better leveraging locality and reducing costs.

Related Work and Novelty. Traditionally, network designs rely on static topologies, such as the Clos topology [13]–[15], hypercubic topologies like BCube and MDCube [16], [17], or expander-based networks [18], [19] in datacenters. Recently, reconfigurable topologies have received much attention, which come in two flavors, demand-oblivious e.g. [20]–[22] and demand-aware, e.g. [2], [7], [8], [11], [23]–[27]. Most existing demand-aware architectures rely on an estimation of traffic matrices [8], [25], [26], [28], [29] which can limit the granularity and reactivity of the network. We in this paper consider more distributed approaches, such as ProjecToR [2] or DiSplayNet [11] supporting per-flow or even per-packet reconfiguration. The most closely related papers to our work are [7], [11], which also rely on concepts from self-adjusting data structures (see [6] for an overview of this approach), namely on self-adjusting binary search trees called splay trees. However, while [7] is still centralized, [11] comes with significant adjustment costs.

Paper Organization. In Section II we present the model and problem definitions. In Section III we present an overview of CBNet. In Section IV we describe the concept of counting-based reconfiguration. In Section V we present the sequential version of CBNet algorithm, and in Section VI we analyze its performance. In Section VII we present and analyze the concurrent version of CBNet. In Section VIII we describe the workloads used in our experiments and analyze them in terms of temporal and non-temporal locality. In Section IX we report on simulations, and in Section X present our conclusions.

II. MODEL

In this work the objective is to design and formally analyze the performance of distributed algorithms for self-adjusting networks. The network should connect a set V of n communication nodes (e.g., top-of-rack switches or peers). The input to the problem is given by a sequence σ of m messages $\sigma_i(s,d) \in V \times V$ occurring over time, with source s and destination d; m can be infinite. We denote by b_i the time when a message σ_i is generated, and by e_i the time in which it is delivered. The time between successive requests arrivals is assumed to be at least one. The sequence σ is revealed over time, in an online manner: the algorithm does not have any information about the future requests σ_j at time $t < b_j$. Moreover, the sequence σ can be arbitrary: in our formal analysis we consider a worst-case scenario, where σ is chosen *adversarially*, as to maximize the cost of a given algorithm.

We will focus on networks based on *Binary Search Trees* (BST), because trees are a basic graph family and we envision that the self-adjusting links constitute only a subset of the topology, a usual assumption in such networks [2]. Moreover, BSTs are *locally routable*, i.e., dynamic topological changes do not require the global recomputation of routes. We denote the family of BST networks by $\mathcal{T} = T_0, T_1, \ldots$.

Distributed reconfiguration. In order to minimize the communication cost and adjust the topology smoothly over time, the tree is reconfigured locally through rotations that preserve the BST properties. One rotation updates a constant number of links at constant cost. Accordingly, we will denote the tree at time t computed by a given distributed algorithm (possibly accounting for the communication requests σ_t with $t' < t$) by $T_t \in \mathcal{T}$. From now on, we use the terms *rotation* and (local network) *reconfiguration* interchangeably to refer to local topological updates in the tree.

Bidirectional semi-splaying. Differently from SplayNet [7] and DiSplayNet [11], where the classical *zig, zig-zig* and *zig-zag* splay operations have been employed exclusively in a bottom-up direction, CBNet leverages *bottom-up* and *top-down* semi-splaying (*semi zig-zig* and *semi zig-zag*) operations, first introduced for splay trees [30] and later adapted for top-down communication in CBTrees [12]. Besides being simpler to implement in a distributed setting, the semi-splay operations have a lower communication cost than splaying. Note that (semi) splaying not only moves a node upwards in the tree, preserving BST properties, but also roughly halves the depth of every node along the communication path. This halving effect makes splaying efficient in an amortized sense and is a property not shared by other, simpler rotation heuristics, such as move-to-root [31].

Refined cost model. In CBNet, as messages travel from the source to the destination, rotations are traded with routing operations. Therefore, we distinguish between the work needed to forward a message and the work needed to perform local reconfigurations. In practice, the cost of performing a network topology reconfiguration is typically higher than that of forwarding a message over a communication link. We assume that routing a message incurs a cost of 1 unit per hop and that a rotation incurs a cost of $R = O(1)$.

Consider a sequence σ of m messages, an algorithm $\mathcal{A}$, a *BST* T_0, and a message $\sigma_i(s,d) \in \sigma$. Let us define the reconfiguration cost ρ_i as the number of rotations performed by $\mathcal{A}$ to deliver σ_i. The routing cost will be equal to $d_{e_i}(s,d)$, the length of the path $\mathcal{P}_{e_i}(s,d)$ in the resulting tree T_{e_i}, i.e., after σ_i has been delivered. Note that $d_{e_i}(s,d)$ is not necessarily one, as in [7], [11], but is equal to the number of times the message was forwarded along $\mathcal{P}_{b_i}(s,d)$, instead of triggering a rotation. We assume that the value of the routing cost to deliver a message, even when it is addressed to itself ($\sigma_i(v,v)$), is at least one.

Definition 1. ***Work cost:*** *Consider any initial binary tree T_0, a sequence of m messages $\sigma_i(s,d) \in \sigma$ and algorithm $\mathcal{A}$. We define the* ***total routing cost****,* ***total reconfiguration cost****, and* ***total work cost****, respectively, as follows:* $\mathcal{D}(\mathcal{A}, T_0, \sigma) = \sum_{i=1}^{m} (d_{e_i}(s,d)(\sigma_i) + 1)$, $\mathcal{R}(\mathcal{A}, T_0, \sigma) = R \times \sum_{i=1}^{m} \rho_i$, $\mathcal{C}(\mathcal{A}, T_0, \sigma) = \mathcal{D}(\mathcal{A}, T_0, \sigma) + \mathcal{R}(\mathcal{A}, T_0, \sigma)$.

We assume that time is divided into synchronous time slots, in which a message can travel a constant number of hops in the network or a local reconfiguration might be performed. In order to study concurrency, we divide the execution time in *rounds*: in a round, multiple (independent) nodes can make local reconfigurations (steps) concurrently. We consider that nodes and communication between them are reliable and synchronous. In terms of time, we aim to minimize the makespan:

Definition 2. ***Time cost:*** *Consider any initial binary tree T_0, a sequence of m messages σ and algorithm $\mathcal{A}$.* $Makespan(\mathcal{A}, T_0, \sigma) = \max_{1 \le i \le m} e_i - \min_{1 \le i \le m} b_i$.

Our objective is to *minimize* the communication cost both in terms of work and time. We are interested in the worst-case performance over arbitrary sequences of operations (rather than individual operations), and hence, conduct an *amortized analysis*. In our model, the amortized cost can be described as the average cost per message for a given sequence σ.

Definition 3. ***Amortized cost:*** *Given a sequence of m messages σ, if $\mathcal{C}(\sigma_i)$ is the (time or work) cost of the $\sigma_i \in \sigma$, the amortized cost is defined with respect to the worst sequence σ and initial tree T_0 as:* $\mathcal{C}_A = \max_{\sigma, T_0} \frac{1}{m} \sum_{\sigma_i \in \sigma} \mathcal{C}(\sigma_i)$.

Another useful concept for the analysis is empirical entropy.

Definition 4. ***Empirical entropy:*** *Given a sequence of m messages σ and initial tree T_0 on n nodes, the empirical entropy is defined with respect to $\hat{S}$ as:* $H(\hat{S}) = \sum_{i=1}^{n} f_s(v_i) \log \frac{1}{f_s(v_i)}$, *where $\hat{S} = \{f_s(v_1), \ldots, f_s(v_n)\}$ are the frequencies that a node $v_i \in V$ is a source in σ. Similarly, $H(\hat{D})$ is defined for the set of destination frequencies $\hat{D}$.*

III. CBNET OVERVIEW

State-of-the-art distributed self-adjusting networks, such as SplayNet [7] and DiSplayNet [11] are based on the self-adjusting binary search trees *splay trees* [30]. They gradually adapt the network topology toward the communication pattern in an online manner. These self-adjusting networks, however, face challenges when it comes to their application in practice.

Problematic inheritance from data structures. The communication model underlying SplayNet and DiSplayNet is not entirely distributed nor realistic. They adopt an aggressive reconfiguration strategy: the source and the destination nodes of each message execute a sequence of bottom-up rotations, until they meet at their *LCA*, at which point they finally exchange the data. Behind this approach lie two problematic assumptions: (1) the destination node knows when a message is generated toward it and starts rotations simultaneously with the source node; and (2) the destination node travels in the network without carrying any data in order to meet the source node, so both the source and the destination nodes are locked until the message is delivered, limiting concurrency.

Less adjustments. CBNet is based on a different, concurrent self-adjusting data structure, the CBTree [12], in which rotations are traded for routing operations. CBNet performs rotations infrequently, an amortized subconstant $o(1)$ per operation, drastically reducing the reconfiguration cost, while preserving the amortized communication cost guarantees.

More realistic distributed communication. CBNet enables a more realistic and fully distributed communication model. Bidirectional semi-splaying and the ability to forward messages allow for the natural behavior of a message traveling through the network: a message moves bottom-up in the tree when it is navigating towards the root and top-down, otherwise. To the extent of our knowledge, CBNet is the first *message-oriented* self-adjusting network.

More concurrency. Frequent network reconfigurations limit the potential for concurrent execution of self-adjusting networks because rotations might result in conflicts between concurrent communication requests. By drastically reducing the number of reconfigurations and by freeing the source and the destination nodes of each message, CBNet scales better with the concurrency level.

IV. COMMUNICATION HISTORY THROUGH COUNTERS

CBNet keeps track of the communication history through counters and weights, a strategy inspired by CBTrees [12]. Each node $v \in V$ maintains a local variable $c_t(v)$ that accounts for the number of times v has been the source or the destination of some communication request: $c_t(v) = |\{\sigma_i | \sigma_i \in \sigma,\ v = src(\sigma_i) || v = dst(\sigma_i),\ e_i \leq t\}|$, and $c(v) = c_{e_m}(v), \forall v \in V$. We define **weights** as:

$$W_t(v) = \sum_{x \in T_t(v)} c_t(x), W(v) = W_{e_m}(v), \forall v \in V, \quad (1)$$

where e_m is the time of delivery of the last message $\sigma_m \in \sigma$ and $T_t(v)$ is the subtree rooted at v in time-slot $0 \leq t \leq e_m$. In our implementation of CBNet, nodes maintain only their weights, and the counter value is obtained as: $c_t(v) = W_t(v) - W_t(v.l) - W_t(v.r), \forall v \in V, t \geq 0$.

We define the **rank** of a node as: $r_t(v) = \log W_t(v), \forall v \in V, t \geq 0$, assuming $r_t(v) = 0$ if $W_t(v) = 0$. And we define the **potential** of a network by using the potential function presented in [12] as:

$$\Phi_t = \Phi(T_t) = \sum_{v \in V} r_t(v), t \geq 0. \quad (2)$$

Local computation of potential change. The potential of the network starts with the value of zero. It increases over time due to the arrival of new messages or decreases as a result of a rotation. In order to decide whether a given message will cause a rotation or will be forwarded, the network potential difference $\Delta\Phi_t$ that would result from that rotation has to be computed locally in time-slot t. Since the network potential is the sum of the ranks of each node in the tree, and since the ranks of all nodes that do not participate in a rotation have their ranks unchanged, $\Delta\Phi_t$ depends only on the type of rotation (top-down or bottom-up, *semi zig-zig* or *semi zig-zag*) and the rank variation of the neighboring nodes that would participate in that rotation [12]. If the rotation has been performed, the weights of the participating nodes are updated to reflect the change in network potential.

V. SEQUENTIAL CBNET

When a message enters the network, it moves from the source toward the destination node by means of two kinds of operations, referred to as routing steps or rotation steps. Routing steps move the message between nodes while keeping the network's topology unchanged. Rotation steps move the message by restructuring the topology of the network. We define the *current node* of a message as the node currently holding the message at a given point in time. Below we formally define a *step*.

Definition 5. *$Step_t(\sigma_i, x_t)$: Given a message $\sigma_i(s,d) \in \sigma$ with begin and end times b_i and e_i, respectively, and a BST instance $T_t, b_i \leq t \leq e_i$, a **step** $step_t(\sigma_i(s,d), x_t)$ is an operation performed by the message's current node, $x_t \in \mathcal{P}_t(s,d)$, that reduces the message's distance to its destination d, while preserving the BST properties of the network. A sequence of steps, starting at the source node, $x_{b_i} = s$, deliver the message to the destination node $x_{e_i} = d$. Each step can move the message along the path $\mathcal{P}_t(s,d)$ in a **bottom-up** or **top-down** direction and can trigger an operation of type **routing** ($forward(T_t, \sigma_i, direction)$) or of type **rotation** ($splay(T_t, \sigma_i, direction, splayType)$), where $splayType \in \{$semi zig-zag, semi zig-zig$\}$.*

Since there is no central information accessible to nodes about the network structure, to determine the direction of a message, each node $v \in V$ stores the identifiers of its direct neighbors in the tree, i.e., its parent ($v.p$), its left child ($v.l$), its right child ($v.r$), as well as the smallest ($v.smallest$) and the largest ($v.largest$) identifiers currently present in the subtree rooted at v. With this information, given the destination

identifier, a node can decide if the message must be forwarded to its parent ($d \leq x.smallest$ and $d \leq x.largest$), its right child ($x < d \leq x.largest$), or left child ($x.smallest \leq d < x$) in the tree. The rotation type of the step is determined by the position of the current node relative to its neighbors and the message's direction, as defined in [12]. Whether the step is of type rotation or routing is determined by the network potential difference that a rotation would cause, as described below.

Algorithm 1: While a message $\sigma_i(s,d) \in \sigma$ travels from source to destination, each current node $x_t \in T_t, b_i \leq t \leq e_i$ executes the step routine, shown in Algorithm 1. If the $LCA_t(s,d)$ has been reached (lines 2-3) a weight-update message is sent in the bottom-up direction, toward the root r of the tree. This update message is a small control message that carries no data and increments the weights of all nodes $v \in \mathcal{P}_t(LCA_t(s,d), r)$ by two. Note that it does trigger rotations on its way, like a regular message; we therefore include it in the work cost analysis of CBNet. Then, the current node decides upon which type of rotation (line 4) and direction (line 5) to perform. It then computes the network potential difference that the latter would cause (line 6). All these computations are performed without global knowledge about the network topology information from two hops ahead in the path $\mathcal{P}_t(x_t, d)$ is enough, as described in Section IV.

If the rotation decreases the total potential of the network by more than a constant $\delta \in (0,2]$ (we used $\delta = 2$ in our implementation), then a rotation is performed, updating the tree topology (lines 7-8); otherwise, the topology remains unchanged and the message is forwarded to the next current node (line 10). Finally, the weights of the nodes that participated in this step and remained on the path from s to d after the step are incremented by one (line 11);

Algorithm 1 Sequential CBNet $step_t(\sigma(s,d), x_t)$ executed by the current node $x_t \in T$ of message $\sigma_i(s,d)$ in time slot t:

Require: $T_t, \sigma_i(s,d), \delta \in (0,2]$

1: $T_{t+1} \leftarrow T_t$;
2: **if** $x_t = LCA_t(s,d)$ **then**
3: $\quad sendUpdateWeights(T_t, \textit{bottom-up}, \sigma_i)$;
4: $splayType \leftarrow getSplayType(T_t, \sigma_i)$;
5: $direction \leftarrow getDirection(T_t, \sigma_i)$;
6: $\Delta\Phi_t \leftarrow get\Delta\Phi(T_t, splayType, direction)$;
7: **if** $\Delta\Phi_t < -\delta$ **then**
8: $\quad T_{t+1} \leftarrow splay(T_t, \sigma_i, direction, splayType)$;
9: **else**
10: $\quad forward(T_t, \sigma_i, direction)$;
11: $updateWeights(T_t, splayType, direction)$;

VI. Sequential CBNet analysis

We start the analysis of CBNet by presenting an auxiliary lemma, proved in [30] and [12]. We proceed by bounding the amortized routing cost in Theorem 1, the total reconfiguration cost in Theorem 2 and total work cost in Theorem 3.

Potential method. In the potential method, the amortized cost $\hat{c}_t(step_t)$ of an operation $step_t(\sigma_i, x_t)$ is the actual cost $\mathcal{C}_t(step_t)$, plus the increase in potential $\Delta\Phi_t(step_t) = \Phi(T_t) - \Phi(T_{t-1})$, due to $step_t$: $\hat{c}_t(step_t) = \mathcal{C}_t(step_t) + \Phi(T_t) - \Phi(T_{t-1})$. If $\sigma_i \in \sigma$ consists of p_i steps of cost $O(1)$, the actual cost to fulfill σ_i is $\sum_{t=1}^{p_i} O(1)$, causing a potential change of $\sum_{t=1}^{p_i} \Phi(T_t) - \Phi(T_{t-1})$. This summation results in a telescoping series in which all terms cancel except the first and the last, resulting in amortized cost $\hat{c}(\sigma_i) = p_i + \Phi(T_{e_i}) - \Phi(T_{b_i})$ per message. We use the potential function defined in (2) to amortize actual costs in our analysis.

Consider an initial *BST* network instance T_0 of n nodes, a message sequence σ of m requests, a message $\sigma_i(s,d) \in \sigma$ generated at time b_i and delivered at time e_i.

Lemma 1 (Access Lemma (bottom-up [30] and top-down [12] semi-splays))**.** *Consider a $step_t(\sigma_i, x_t)$, executed by the current node $x_t \in T_t$ of rank $r_t(x_t), b_i \leq t \leq e_i$, of type rotation with $splayType \in \{$semi zig-zig, semi zig-zag$\}$. Let $\Delta\Phi_t$ be the net decrease in the potential of T_t, x_{t+1} be the new current node of the message, and $r_{t+1}(x_{t+1})$ be the rank of the latter, right after $step_t$, respectively. If the direction of the step is bottom-up, we have that $x_{t+1} = x_t.p.p$ and $\Delta\Phi_t + 2 \leq 2(r_{t+1}(x_{t+1}) - r_t(x_t))$. Otherwise, if the direction is top-down, we have that $x_t = x_{t+1}.p.p$ and $\Delta\Phi_t + 2 \leq 2(r_t(x_t) - r_{t+1}(x_{t+1}))$.*

The first amortized analysis of self-adjusting networks was presented for SplayNet [7]. In Theorem 1 we prove that the amortized routing cost of CBNet is asymptotically the same as the amortized reconfiguration cost of SplayNet.

Theorem 1. *Let $H(\hat{S})$ and $H(\hat{D})$ be the source and destination empirical entropies of σ, as defined in (4). The **amortized routing cost** incurred by Algorithm 1 to deliver all messages in σ is $O(H(\hat{S}) + H(\hat{D}))$.*

Proof. To bound the routing cost of CBNet, we analyze the number of routing steps performed to deliver a message $\sigma_i(s,d) \in \sigma$ and the respective weight update message sent from the $LCA_t(s,d)$ to the root of the tree (recall from Section IV that this update message carries no data but might trigger rotations). Consider a $step_t(\sigma_i(s,d), x_t)$ that *has not caused a rotation* in the bottom-up direction (the top-down direction case is analogous), executed by the current node x_t. Since no rotation was performed during this step, by Algorithm 1, $\Delta\Phi_t \geq -\delta, \delta \in (0,2)$, $\Delta\Phi_t$ being the decrease in total potential of the tree immediately after the referred step. By Lemma 1, we have that: $2 - \delta < 2 + \Delta\Phi_t \leq 2(r_{t+1}(x_{t+1}) - r_t(x_t))$, where $x_{t+1} = x_t.p.p$ and $r_{t+1}(x_{t+1})$ is the rank of the new current node after the step. Let $\delta' = 1 - \delta/2$, $t = e_i$, $\alpha = LCA_t(s,d)$ and $r \in V$ be the root of T_t. Then $r_t(z) - r_t(x) > \delta'$ for each pair of consecutive edges (x,y) and (y,z) on the (bottom-up) path $\mathcal{P}_t(s,r)$ (and analogously on the (top-down) path $\mathcal{P}_t(\alpha,d)$). Let $h_i = \lfloor |\mathcal{P}_t(s,d) \cup \mathcal{P}_t(\alpha,r)|/2 \rfloor$. Summing over all h_i pairs of edges, we have that the sum

telescopes to $(r_t(r) - r_t(s)) + (r_t(\alpha) - r_t(d)) > h_i\delta'$, and:

$$\begin{aligned} h_i &< \frac{(r_t(r) - r_t(s)) + (r_t(\alpha) - r_t(d))}{\delta'} \\ &\leq \frac{(r_t(r) - r_t(s)) + (r_t(r) - r_t(d))}{\delta'} \\ &= O\left(\log \frac{W_t(r)}{W_t(s)} + \log \frac{W_t(r)}{W_t(d)}\right) \\ &= O\left(\log \frac{W(r)}{c(s)} + \log \frac{W(r)}{c(d)}\right), \forall \sigma_i \in \sigma \end{aligned}$$

where $W(v)$ and $c(v)$ are the total weight and count of node v, respectively. The last equality follows from the fact that $W_t(v) \geq c_t(v), \forall v \in T_t, 0 \leq t \leq e_m$ (Def. (1)).

Using the fact that $W(r) = 2m$, since each of the m messages in σ increases two counters in the tree, the source and the destination node's, by one, we have that $\mathcal{D}_A(CBNet, T_0, \sigma)$

$$\begin{aligned} &= \frac{1}{m}\sum_{i=1}^{m} h_i \\ &= O\left(\frac{1}{m}\left(\sum_{i=1}^{m} \log \frac{m}{c(src(\sigma_i))} + \sum_{i=1}^{m} \log \frac{m}{c(dst(\sigma_i))}\right)\right) \\ &= O\left(\frac{1}{m}\left(\sum_{i=1}^{n} s(v_i) \log \frac{m}{c(v_i)} + \sum_{j=1}^{n} d(v_j) \log \frac{m}{c(v_j)}\right)\right) \\ &= O\left(\frac{1}{m}\left(\sum_{i=1}^{n} s(v_i) \log \frac{m}{s(v_i)} + \sum_{j=1}^{n} d(v_j) \log \frac{m}{d(v_j)}\right)\right) \\ &= O(H(\hat{S}) + H(\hat{D})), \end{aligned}$$

where $s(v_i)$ and $d(v_i)$ are the number of times a node $v_i \in V$ was source and destination in σ, respectively, and $src(\sigma_i(s,d)) = s$ and $dst(\sigma_i(s,d)) = d, \forall \sigma_i \in \sigma$. Note that $c(v_i) \geq s(v_i)$ *or* $d(v_i), \forall i \in \{1 \dots n\}$. □

Having upper bounded the amortized routing cost, we turn our attention to the reconfiguration cost of CBNet. In Theorem 2 we present a generalization of the analysis of CBTrees [12], a self-adjusting data structure, for self-adjusting networks, while keeping the bound on the number of rotations.

Theorem 2. *The **total reconfiguration cost** incurred by Algorithm 1 to deliver all messages in σ is $O\left(n \log \frac{m}{n}\right)$.*

Proof. Each rotation performed by Algorithm 1 decreases the total potential of the tree by at least $\delta \in (0, 2]$. The potential decreases only by means of rotations and cannot be negative, by definition. Hence, the number of rotations, performed by the sequence of all current nodes of each message, is upper bounded by the sum of potential increases throughout the message sequence σ, i.e., $\mathcal{R}(CBNet, T_0, \sigma) = O(\Delta\Phi^+(\sigma))$. The potential of the network increases when the counters $c_t(s)$ and $c_t(d)$, $1 \leq t \leq e_m$ are incremented upon the delivery of each message $\sigma_i(s,d) \in \sigma, 1 \leq i \leq m$, as well as the weights of the nodes that belong to the path traversed by σ_i and the respective weight-update message sent toward the root. Let $t = e_i$, $\alpha = LCA_t(s,d)$, $r \in V$ be the root of the tree instance T_t, and $\Delta\Phi^+(\sigma_i)$ be the network *potential increase* caused by each message $\sigma_i \in \sigma$, then

$$\begin{aligned} \Delta\Phi^+(\sigma_i) &\leq \sum_{u \in \mathcal{P}_t(s,r) \cup \mathcal{P}_t(d,\alpha)} \log\left(W_t(u) + 2\right) - \log W_t(u) \\ &= \sum_{u \in \mathcal{P}_t(s,r) \cup \mathcal{P}_t(d,\alpha)} \log\left(1 + \frac{2}{W_t(u)}\right) \\ &\leq \sum_{u \in \mathcal{P}_t(s,r) \cup \mathcal{P}_t(d,\alpha)} \log\left(\frac{2}{W_t(u)}\right) \\ &\leq \sum_{u \in \mathcal{P}_t(s,r) \cup \mathcal{P}_t(d,\alpha)} \frac{2}{W_t(u)}. \end{aligned} \tag{3}$$

We can split the sum in (3) into two sums, from s to r and from d to *LCA*:

$$\Delta\Phi^+(\sigma_i) \leq \sum_{u \in \mathcal{P}_t(s,r)} \frac{2}{W_t(u)} + \sum_{v \in \mathcal{P}_t(d,\alpha)} \frac{2}{W_t(v)}. \tag{4}$$

As we argued in Theorem 1, for every pair of consecutive edges $(x,y),(y,z)$ on the path traversed by message σ_i, $W_t(z)/W_t(x) > 2^{\delta'}, \delta' \in (0,1]$. It follows that $2/W_t(u)$ for $u \in \mathcal{P}_t(s,r)$ and $\mathcal{P}_t(d,\alpha)$ are both geometric series. So the two summation terms in (4) converge to $O(1/W_t(s))$ and $O(1/W_t(d))$, respectively.

Now consider all communication requests $\sigma_i \in \sigma$. Let $v \in V$ be a particular node in the network. After the k-th time v has been a source or a destination of some message in σ, its counter becomes $c_t(v) = k$. Thus, the potential increase due to all communications over $v \in V$ is

$$\begin{aligned} \Delta\Phi^+(\sigma) &= \sum_{i=1}^{m} O\left(\frac{1}{c_{e_i}(src(\sigma_i))} + \frac{1}{c_{e_i}(dst(\sigma_i))}\right) \\ &= \sum_{v \in V} \sum_{j | v = src(\sigma_j) || v = dst(\sigma_j)} O\left(\frac{1}{c_{e_j}(v)}\right) \\ &= \sum_{v \in V} O(\log c(v)) = O\left(n \log \frac{m}{n}\right) \end{aligned} \tag{5}$$

where (5) comes from the fact that $\sum_{v \in V} \log c(v)$ is maximized when $c(v) = m/n, \forall v \in V$.

□

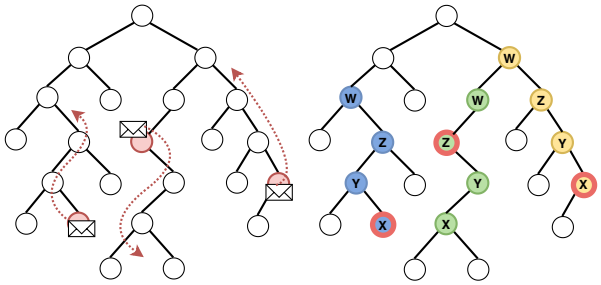
Theorem 3. *The **total work cost** incurred by Algorithm 1 to deliver all messages in σ is $O\left(m \log n + n \log \frac{m}{n}\right)$.*

Proof. The result follows from Theorems 1 and 2 and the fact that the empirical entropy is maximized when the source and destination distributions are uniform, i.e., it is $O(\log n)$. □

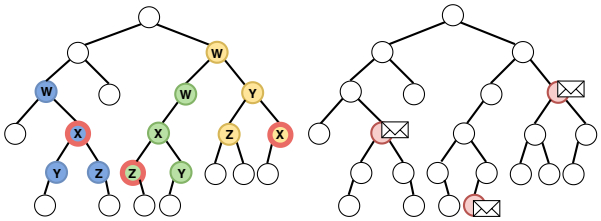
Since the execution of Algorithm 1, is sequential, the time needed to deliver all messages in σ has the same asymptotic upper bound as the total communication cost.

VII. Concurrent CBNet

We now turn our attention to the concurrent CBNet implementation and analysis. Firstly, we highlight some of the assumptions made when modeling concurrent network communication. In Section VII-A we explain the main algorithm



(a) Bottom-up and top-down steps (b) Splay type determines clusters



(c) Rotations are performed (d) Messages made progress

Fig. 1. Three concurrent not conflicting clusters: (a)-(b) blue and yellow nodes participate in bottom-up steps with current nodes x, green nodes participate in a top-down step with current node z. (c)-(d) Three rotations occurred in parallel, and the three messages have made progress in the tree while staying at the same current nodes.

design principles that we used. In Section VII-B, we prove the liveness of and analyze the communication cost incurred by the concurrent CBNet.

A. Concurrent network reconfiguration

To design and implement the concurrent CBNet we adopt the design principles of DiSplayNet [11], which follow an optimistic approach, in the sense that conflicts are dealt with upon occurrence, and are based on the following two principles:

1) **Prioritization:** In order to ensure deadlock and starvation freedom, concurrent steps are executed according to a *priority*. Given two messages σ_i and σ_j, we say that σ_i has a higher priority than σ_j if $b_i < b_j$.
2) **Independent clustering:** To ensure consistency among concurrent steps, local independent clusters of nodes that participate in a single step are computed in a distributed manner. A cluster is formally defined below.

Definition 6. ***Cluster*** *$\mathcal{K}_t(\sigma_i, x_t)$: Consider the set of links $(v, w) \in T_t$ that would be modified as a result of a rotation executed by a $step_t(\sigma_i, x_t)$. The **parent node** v of each such link belongs to a set of nodes that is referred to as the cluster $\mathcal{K}_t(\sigma_i, x_t)$. In each cluster, the current node of the respective step, $x_t \in T_t$, is the only node that can perform a step, even if it is decided that it will be of type routing; the other nodes in the cluster are* locked *in round t.*

Figure 1 presents an example execution of three concurrent (not conflicting) clusters.

B. Analysis of concurrent execution

In the concurrent scenario, the analysis of CBNet is more challenging than in the sequential setting, because there is only guarantee that the message with the highest priority has consecutive progress towards the destination. For the remaining messages, the consecutive progress can be interrupted, resulting in several consecutive progress sequences. An interruption can cause the message to travel through a different path.

The proof of liveness of CBNet follows the same lines as that of DiSplayNet [11], due to the prioritization rule and the independent clustering approach.

We now turn our attention to the analysis of the communication cost of the concurrent version of CBNet. Firstly, we introduce the concept of a conflict between a pair of concurrent steps.

Definition 7. ***Conflict****: Consider a sequence of messages σ and two messages $\sigma_i = \{s_i, d_i\} \in \sigma$ and $\sigma_j = \{s_j, d_j\} \in \sigma$, such that σ_i has higher priority, i.e., $i < j$. Moreover, consider two steps $step_{t_i}(\sigma_i, x_{t_i}), b_i \leq t_i \leq e_i$ and $step_{t_j}(\sigma_j, x_{t_j}), b_j \leq t_j \leq e_j$ with the respective clusters $\mathcal{K}_{t_i}(\sigma_i, x_{t_i})$ and $\mathcal{K}_{t_j}(\sigma_j, x_{t_j})$. We say that a conflict occurs in the concurrent execution of σ if $t_i = t_j$ and $\mathcal{K}_{t_i}(\sigma_i, x_{t_i}) \cap \mathcal{K}_{t_j}(\sigma_j, x_{t_j}) \neq \emptyset$. A conflict can be of type **pause**, if both steps are of type **routing**, or of type **bypass** if $step_{t_i}(\sigma_i, x_{t_i})$ is of type **rotation**.*

Note that only a bypass can generate a work cost overhead in the network, given that it reconfigures the network topology in the local neighborhood of the current node carrying a message, whereas a pause can generate only a time cost overhead.

In the following analysis, consider an initial *BST* network instance T_0 of n nodes, a message sequence σ of m requests, and the concurrent CBNet algorithm.

Theorem 4. *The **total reconfiguration cost** incurred by concurrent CBNet to deliver all messages in σ is $O\left(n \log \frac{m}{n}\right)$.*

Proof. As has been shown in Theorem 2, the number of rotations performed by CBNet is upper bounded by the maximum network potential increase during the execution of σ, which is determined by the total number of messages in σ and is independent of the order or time in which each request $\sigma_i \in \sigma$ is generated or delivered. Given that concurrent CBNet applies the same rule to perform a rotation as the sequential CBNet, the total reconfiguration cost of concurrent CBNet has the same upper bound as in the sequential execution. □

Theorem 5. *The **total routing cost** incurred by concurrent CBNet to deliver all messages in σ is $O\left(m \log n + n^2 \log \frac{m}{n}\right)$.*

Proof. Since only a bypass can generate additional work to deliver a message in σ, and a bypass is associated with at least one rotation, the routing cost overhead due to concurrency is bounded by the total number of rotations in the concurrent execution of σ, multiplied by the additional routing cost incurred by the lower-priority messages that have been bypassed,

which is $O(n)$ per bypassed message, in the worst case. The result follows by adding the concurrency overhead to the total sequential routing cost (see Theorem 1 and Def. (3)). □

Theorem 6. *The **total work cost** incurred by concurrent CBNet to deliver all messages in σ is $O\left(m \log n + n^2 \log \frac{m}{n}\right)$.*

Proof. The result follows from Theorems 1, 4 and 5. □

In Theorem 7 we provide a worst-case upper bound on the time cost of concurrent CBNet.

Theorem 7. *The **makespan** of concurrent CBNet to deliver all messages in σ is $O\left(m \log n + n^2 \log \frac{m}{n}\right)$.*

Proof. We know that at least one message (of the highest priority) is delivered without being paused or bypassed at a time in the concurrent execution of CBNet. This implies that the makespan is upper bounded by the total communication cost of the concurrent execution, computed in Thm. 6. □

Comparison to related work. The bounds on work and time costs of concurrent CBNet are a significant improvement compared to the cost of DiSplayNet, which has worst-case total work and time costs $O(m(m+n)\log n)$ [11].

VIII. Workload traces

In this work, before running each experiment, we measure and classify the locality of the input in terms of its temporal and non-temporal components. Temporal locality is the trend of continuous communication between nodes for some time (Bursty). Non-temporal locality is the tendency for some pairs of nodes to communicate more frequently than the others pairs (Skewed). This classification is useful for evaluating self-adjusting networks because some algorithms may operate better with one component than another. For example, because CBNet performs network reconfigurations only when they decrease the network potential, we expect it to perform better on inputs with high non-temporal locality. More aggressive reconfiguration strategies, such as DiSplayNet [11], have the potential to obtain more benefits from high temporal locality. Real-world workloads can present an intermix of these localities at various levels, and identifying them can facilitate the interpretation of the results.

Measuring locality. To measure the locality of reference present in a workload, we use the definition of *trace complexity*, introduced in [1], which leverages only randomization and data compression operations. Given a sequence of communication requests σ, we generate two transformations: $\Gamma(\sigma)$, comprised of a sequence of requests where the relative order of requests is shuffled, and temporal relationships are lost; and $\mathcal{U}(\sigma)$, a sequence of requests of the same size and domain, but collected from a uniform distribution, removing the non-temporal locality as a result. The temporal complexity of a trace is given by the ratio between the entropy $C(\sigma)$ contained in the sequence σ and the entropy contained in the temporal transformation $C(\Gamma(\sigma))$: $T(\sigma) = \frac{C(\sigma)}{C(\Gamma(\sigma))}$. Similarly, the non-temporal component of a trace is given by the ratio of the entropy of the temporal transformation $\Gamma(\sigma)$ to the entropy of the uniform sequence $\mathcal{U}(\sigma)$: $NT(\sigma) = \frac{C(\Gamma(\sigma))}{C(\mathcal{U}(\sigma))}$. The entropy $\mathcal{C}(\cdot)$ can be measured through the size of the compressed file. Note that $T, NT \in [0,1]$, since $C(\sigma) \leq C(\Gamma(\sigma)) \leq C(\mathcal{U}(\sigma)))$.

Definition 8. ***Trace complexity [1]:*** *The complexity of a workload σ is given by the product of temporal and non-temporal complexity: $\Psi(\sigma) = T(\sigma) \times NT(\sigma)$. Note that $\Psi(\sigma) = \frac{C(\sigma)}{C(\Gamma(\sigma))} \times \frac{C(\Gamma(\sigma))}{C(\mathcal{U}(\sigma))}$, and therefore $\Psi(\sigma) = \frac{C(\sigma)}{C(\mathcal{U}(\sigma))}$*

We can compare workloads of different sizes using a graphical representation on a two-dimensional plane, as shown in Figure 2. The x and y-axis represent the temporal and non-temporal components, respectively. At each point, the area of the circle corresponds to the trace complexity value $\Psi(\cdot)$. Note that the lower the temporal or non-temporal complexity of a trace, the higher its temporal and non-temporal locality. We ran experiments with all the workloads shown in Figure 2, but due to space constraints, we will focus only on a subset, as described below.

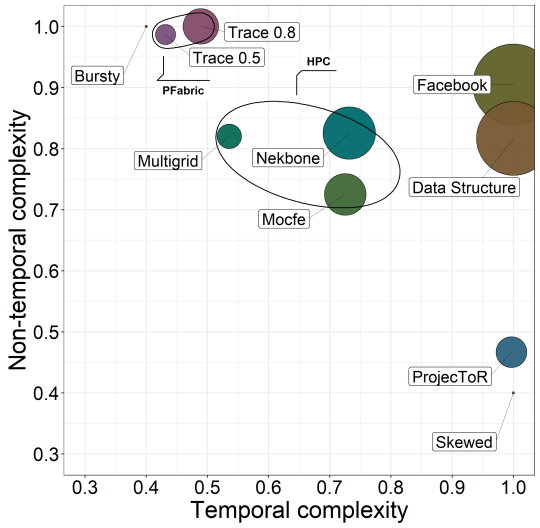


Fig. 2. Trace map of all workloads used in the experiments. The x and y axes represent the temporal and non-temporal components, respectively, and the area of each circle corresponds to the trace's complexity, defined in (8).

High non-temporal and low temporal locality (ProjecToR and Skewed): The ProjecToR workload [32] describes the distribution of communication probability between $8,367$ pairs of nodes in a network of $n = 128$ nodes (top of racks), randomly selected from 2 production groups, running between Map Reduce operations, index builders, database and systems storage. We sampled a sequence of $m = 10,000$ independent and identically distributed requests ($i.i.d.$) in time by the communication matrix provided and repeated each experiment 30 times. As can be seen in the Figure 2, this data set has a high non-temporal locality compared to other real data sets. On the other hand, due to the way the messages are sampled ($i.i.d.$), it presents almost no temporal locality.

The Skewed workload corresponds to an artificial sequence generated by the approach presented in [1]. The non-temporal locality component was produced using the *Zipf* distribution that belongs to the family of power-law distributions. This distribution has its entropy defined according to its parameters, so it is possible to calculate its parameters analytically given the entropy value that we want to reproduce. This trace corresponds to a sequence of $m = 10,000$ communication requests in a network of $n = 1024$ nodes.

High temporal and low non-temporal locality (PFabric and Bursty): The traces of PFabric [33] were generated by executing simulation scripts in NS2. We sample a sequence of $m = 1,000,000$ communication requests from a network of $n = 144$ nodes. As shown in Figure 2, this workload presents the largest temporal locality among the real data sets. The Bursty workload was generated artificially ($m = 10,000, n = 1024$) to have an extremely high temporal locality and almost no non-temporal locality.

High temporal and non-temporal locality (HPC [34]): This workload consists of high-performance computing applications, such as solutions of Poisson's equations, hyperbolic components of the Navier-Stokes equation, and solution for elliptical linear system models. We collected a sample of $m = 1,000,000$ requests for a network of $n = 1024$ nodes. A shown in Figure 2, this workload presents considerable levels of locality, both temporal and non-temporal.

Low locality (Data Structure): Since CBNets are an extension of self-adjusting data structures, it is interesting to analyze their performance on request sequences encountered in self-adjusting data structures. The Data Structure workload consists of artificial communication sequences ($m = 10,000, n = 128$), where each message's destination is the root node of the network and the source is chosen randomly, following a normal distribution ($std = 1.6$) over the remaining $n-1$ nodes.

IX. Simulations

In this section, we will evaluate the performance of CBNet relative to state-of-the-art baselines on real-world and synthetically generated data traces, described in Section VIII.

A. Baselines

We implemented the following baselines:

BT: A balanced static BST, with no reconfigurations. This baseline corresponds to the optimum topology when all pairs of nodes are equally likely to communicate;

OPT: An optimal static BST, computed using the dynamic program presented in [7]. Note that, like in the BT, there is no reconfiguration cost, but only routing cost. From a practical point of view, this baseline is not realistic since it requires prior knowledge of the distribution of communication requests;

SN: Our implementation of SplayNet [7]. Note that SplayNet is not fully distributed, since messages are scheduled sequentially, which would require a global scheduler.

DSN: Our implementation of DiSplayNet, a concurrent version of SplayNet, which is slightly different and more realistic than the one in [11]. We implemented a 3-way handshake procedure for the source and destination nodes of each message to start rotating toward their *LCA* simultaneously;

SCBN: A sequential version of CBNet. Like SplayNet, it is not entirely distributed and serves as a reference to assess the benefits of concurrent reconfigurations.

B. Experimental setup

All our experiments were performed using the Sinalgo simulation platform [35], which provides a network abstraction for message passing in a synchronous communication model.

To space the requests in time and make the timestamp sequences more realistic, we used a Poisson distribution with $\lambda = 0.05$ to determine the time of the entrance of each message in the network. In all experiments, we assumed that the reconfiguration cost of one step equals the cost of forwarding a message through one link, i.e., $R = 1$ unit. Note that this does not occur in reality, as the cost of a reconfiguration is typically much higher than the routing cost. In practice, the advantage of CBNet in terms of reconfiguration cost reduction would be significantly higher than depicted in our plots. We made empirical measurements of the total routing and reconfiguration work and the time required for different communication patterns. We divided the input workloads into test groups according to their type of locality.

C. Results

Since CBNet uses the communication history of the nodes to improve source-destination paths, we expect them to perform especially well on traces with high non-temporal locality, as these sequences present more opportunities for path length optimization between high-frequency source-destination pairs, similarly to the static optimal network (OPT). In contrast, for sequences with a low non-temporal locality, we expect CBNet to perform similarly to the static balanced (BT), since the communication matrix is approximately uniform.

High non-temporal locality (ProjecToR and Skewed): In Figure 3 we analyze the work cost, which is comprised of the reconfiguration (rotations) and message forwarding (routing) components. In both ProjecToR and Skewed workloads, CBNet performed the least amount of work among the baselines, and less rotations compared to other self-adjusting networks. Compared to BT, it is possible to see that CBNet took advantage of the non-temporal locality present in both workloads, bringing the network closer to the optimal static tree configuration. The difference in work between BT and OPT is coherent with the presence of non-temporal locality in the traces, as shown in both plots. CBNet and SCBN had similar work costs to DSN and SN, respectively. However, unlike the latter, in which most of the work is due to reconfiguration, CBNet performed routing steps almost exclusively, which supports our analytical results.

Figure 4 presents the results in terms of makespan and throughput. Note that we did not include the static networks, as there is no defined time model for them. The first point that we highlight is the superior performance, in makespan and

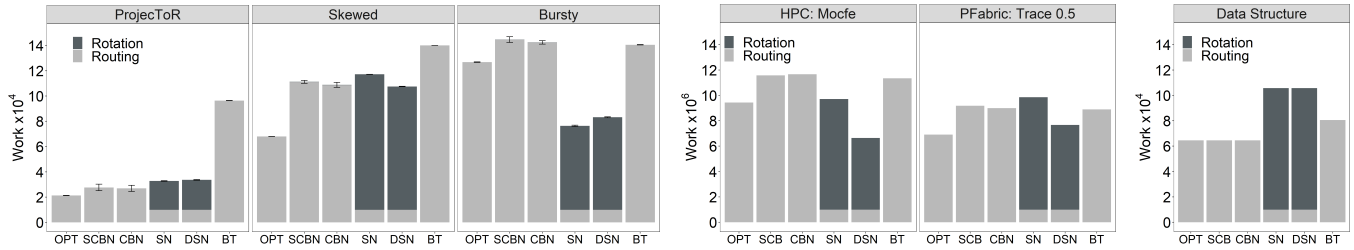


Fig. 3. Work cost: For CBNet, almost 100% of total work cost is comprised of routing, as opposed to mostly reconfiguration for SN and DSN.

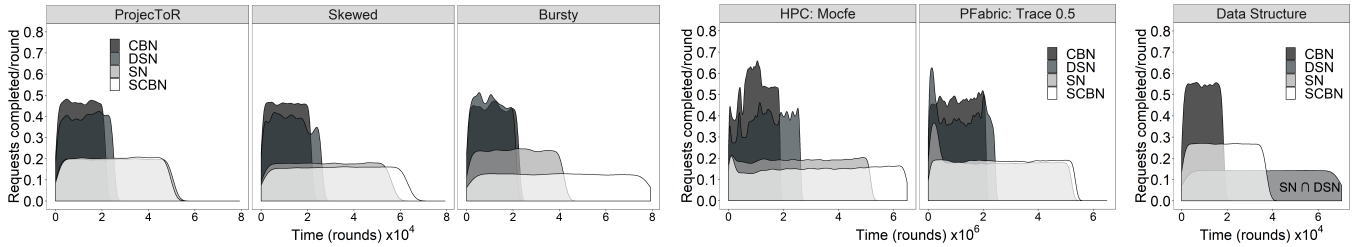


Fig. 4. Throughput: the time cost advantage of concurrent (CBN and DSN) relative to sequential (SN and SCBN) networks is evident in all workloads.

throughput, of the two concurrent versions of self-adjusting networks over the sequential ones, showing that they took advantage of opportunities for parallelism. The second point is the superior performance of CBNet compared to DiSplayNet, in terms of makespan and throughput. This gain comes from the way CBNet transmits messages. Unlike DSN, where both the source and destination nodes must be dedicated exclusively to a single communication request, CBNet does not lock the destination nodes, so they are able to work on other messages in parallel.

High temporal locality (PFabric and Bursty): In the PFabric and Bursty workloads, the communication matrix is close to uniform, and consequently, the total work of BT and OPT is almost the same, as can be seen in Figure 3. Among all the algorithms, SplayNet and DiSplayNet had the best performance in terms of work, even better than OPT, showing how well these algorithms exploit temporal locality by performing rotations aggressively. In contrast, in this scenario, CBNet approached the OPT work cost, since it takes into account the entire communication history of the nodes to perform reconfigurations, which makes it less reactive to temporal locality of the workload. In Figure 4 we can see that, in terms of makespan and throughput, however, CBNet performed as well as DiSplayNet, or better, despite the unfavorable trace complexity of the PFabric and Bursty workloads. This shows that CBNet achieved higher parallelism. Note that due to the artificially high temporal locality in the Bursty workload, where the request sequence was created globally and is mostly comprised of consecutive request repetitions, the execution becomes sequential at some nodes, limiting the opportunities for concurrency.

High non-temporal and temporal locality (HPC): As shown in Figure 3, DSN and SN took significant advantage of the temporal locality present in the HPC Mocfe workload. Furthermore, since OPT and BT presented similar values, we can conclude that the non-temporal locality is not sufficient, resulting in higher work cost for CBNet and SCBN. Nevertheless, CBNet achieved the best makespan and throughput due to higher parallelism, as shown Figure 4. Note that CBNet's makespan was significantly lower than SCBNet's, which reinforces the advantages of concurrent execution.

Low locality (Data Structure): As shown in Figure 4, CBNet and SCBN obtained the best results in in terms of throughput and makespan, while DiSplayNet had no advantage relative to the baselines, presenting similar performance to SplayNet. Interestingly, CBNet performed similarly to OPT in terms of total work cost, as shown in Figure 3. DSN and SN, on the other hand, performed significantly more work, which can be explained by the fact that this workload contains no spatial locality, which increases the number of conflicts that result from a high number of rotations.

D. Final remarks

Benefits of concurrency: The advantages of concurrent (CBN and DSN) versus sequential (SN, SCBN) self-adjusting networks in time cost were evident in all workloads (Figure 4). Moreover, CBNet presented superior throughput to DiSplayNet in all scenarios, including the most challenging ones in terms of trace complexity, such as the Bursty and PFabric workloads. Finally, the total work overhead due to concurrency was insignificant or negative in all experiments (Figure 3).

CBNet and temporal locality: CBNet has the property that messages follow an expected path length proportional to the entropy of the communication distribution, which is

computed based on the counting history stored by each node. One of the consequences of this approach is that network reconfigurations are less reactive to the temporal locality of the demand. Considering that the request sequence is infinite in a real network, in order for the topology not to become too static, a counter resetting mechanism should be implemented, so that older requests contribute less to the current weights used in the potential computations. In our experiments we did not evaluate any counter resetting strategy to keep the experiments coherent with the theoretical analysis.

X. Conclusion

We presented CBNet, a novel self-adjusting network that relies on a fully decentralized control plane and significantly reduces adjustment costs, compared to prior work. Despite its concurrent nature, CBNet comes with formal guarantee. While our contribution is still theoretical in nature, we believe that these properties together constitute an interesting step forward toward practical self-adjusting networks.

Our work opens several interesting avenues for future research. In particular, it will be interesting to further analyze the optimal trade-off between the benefits and the costs of adjustments. It will also be interesting to generalize the network topology beyond trees and consider implications on the network and the transport layers.

References

[1] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," in *Proc. ACM SIGMETRICS*, 2020.

[2] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference*. ACM, 2016, pp. 216–229.

[3] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. 9th ACM Internet Measurement Conference (IMC)*, 2009, pp. 202–208.

[5] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," *Proc. ACM HotNets*, 2009.

[6] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," in *ACM SIGCOMM Computer Communication Review (CCR)*, 2018.

[7] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 24, no. 3, pp. 1421–1433, 2016.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339–350, 2011.

[9] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," in *Proc. International Symposium on Distributed Computing (DISC)*, 2017.

[10] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, and A. Tanwer, "Firefly: A reconfigurable wireless data center fabric using free-space optics," in *ACM SIGCOMM Computer Communication Review*, vol. 44. ACM, 2014, pp. 319–330.

[11] B. Peres, O. Souza, O. Goussevskaia, S. Schmid, and C. Avin, "Distributed self-adjusting tree networks," in *Proc. IEEE INFOCOM*, 2019.

[12] Y. Afek, H. Kaplan, B. Korenfeld, A. Morrison, and R. E. Tarjan, "The cb tree: A practical concurrent self-adjusting search tree," *Distrib. Comput.*, vol. 27, no. 6, p. 393–417, Dec. 2014.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[14] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[15] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 399–412.

[16] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 39, no. 4, pp. 63–74, 2009.

[17] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "Mdcube: a high performance network structure for modular data center interconnection," in *Proc. ACM CONEXT*, 2009, pp. 25–36.

[18] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, "Beyond fat-trees without antennae, mirrors, and disco-balls," in *Proc. ACM SIGCOMM*, 2017, pp. 281–294.

[19] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 12, 2012, pp. 17–17.

[20] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proc. ACM SIGCOMM*, 2017, pp. 267–280.

[21] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," *arXiv preprint arXiv:1903.12307*, 2019.

[22] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. ACM SIGCOMM*, 2020, pp. 782–797.

[23] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "Osa: An optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking*, vol. 22, no. 2, pp. 498–511, April 2014.

[24] C. Avin, A. Hercules, A. Loukas, and S. Schmid, "rdan: Toward robust demand-aware network designs," in *Information Processing Letters (IPL)*, 2018.

[25] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," *Queueing Systems*, vol. 88, no. 3-4, pp. 311–347, 2018.

[26] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 447–458, Aug. 2013.

[27] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[28] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, and S. Zhong, "Enabling wide-spread communications on optical fabric with megaswitch," in *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'17, USA, 2017, pp. 577–593.

[29] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 327–338, 2011.

[30] D. Sleator and R. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[31] B. Allen and I. Munro, "Self-organizing binary search trees," *J. ACM*, vol. 25, no. 4, pp. 526–535, Oct. 1978.

[32] "Projector dataset," www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect, 2016.

[33] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "Pfabric: Minimal near-optimal datacenter transport," in *Proc. ACM Conference on SIGCOMM*, 2013, p. 435–446.

[34] U. DOE, "Characterization of the doe mini-apps." https://portal.nersc.gov/project/CAL/overview.htm, 2016, accessed 11-February-2020.

[35] D. C. Group, "Sinalgo - simulator for network algorithms," http://disco.ethz.ch/projects/sinalgo/index.html, 2007, accessed 10-January-2020.