Constrained In-network Computing with Low Congestion in Datacenter Networks

Raz Segal, Chen Avin, and Gabriel Scalosub
School of Electrical and Computer Engineering
Ben Gurion University of the Negev, Israel
razseg@post.bgu.ac.il, avin@bgu.ac.il, sgabriel@bgu.ac.il

IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796980

***Abstract*—Distributed computing has become a common practice nowadays, where recent focus has been given to the usage of smart networking devices with in-network computing capabilities. State-of-the-art switches with near-line rate computing and aggregation capabilities enable acceleration and improved performance for various modern applications like big data analytics and large-scale distributed and federated machine learning.**

In this work, we formulate and study the theoretical algorithmic foundations of such approaches, and focus on how to deploy and use constrained in-network computing capabilities within the data center. We focus our attention on reducing the network congestion, i.e., the most congested link in the network, while supporting the given workload(s). We present an efficient optimal algorithm for tree-like network topologies and show that our solution provides as much as an x13 improvement over common alternative approaches. In particular, our results show that having merely a small fraction of network devices that support in-network aggregation can significantly reduce the network congestion, both for single and multiple workloads.

I. Introduction

As online applications and services increase in popularity, distributed data processing capabilities and data center networks have become a major part of the infrastructure of modern society. Moreover, due to the vast growth in the amount of data processed by such applications, recent work shows that the *bottleneck* for efficient distributed computation is now the underlying communication *network* and not the computational capabilities at the servers [1]–[3], as was traditionally the case.

For example, distributed machine learning (ML) tasks, which drive some of the most exciting technological developments of recent years, are significantly constrained by such bottlenecks [4]. Frequently, communication-intensive and network-wide operations like *AllReduce* are essential for such applications to sustain the ever-increasing volumes of data they have to process. Other examples are scenarios giving rise to the *incast* problem [5], [6] arising in Big Data applications, e.g., within MapReduce frameworks.

To improve the performance of such tasks, a recent line of work, both by academia and industry, proposed the usage of *in-network computing* [7]–[10]. This approach tries to offload as much of the computation as possible onto "smart" networking devices achieving two goals: (i) possibly reducing the amount of data that traverses the network, and (ii) reducing or even eliminating some of the computational tasks from servers and end hosts. By that, in-network computing aims to significantly improve performance and cost.

This effort is bearing fruit and cutting-edge networking devices like switches and SmartNICs actually perform local computation on streams of traffic, like reduce operations, even at line rate [10], [11]. By using SDN and programmable network elements (e.g., P4) [12], such in-network *computing devices* are being deployed, and have been shown to greatly improve both networks, and applications, performance, as well as resource usage efficiency [10], [11].

As could be expected when using in-network computing, deploying such capable devices in a network comes at a cost (e.g., usage of computing resources, power consumption, or availability). Hence, such capabilities might not be ubiquitous throughout the network, or at all times, or for every workload. For example, when such a service is bundled in a service-level agreement (SLA), or when multiple tenants and multiple workloads call for such in-network computation abilities, it might be that the available resources that are required to support such in-network computation might not be sufficient for satisfying all pending requirements.

In this work, we focus our attention on the task of *data aggregation* as it occurs in, e.g., MapReduce frameworks, or distributed machine learning frameworks making use of, e.g., a parameter server, or gradient aggregation and distribution. We study such in-network computing paradigms in tree-based (overlay) topologies consisting of a tree network of switches, each connected to some number of servers (e.g., switches can be viewed as Top-of-Rack switches).[1] Our goal is to perform a *Reduce* operation, where the data aggregated from all servers should reach a special *destination* server d (which can be logically viewed as simply the root switch). It should be noted that tree-based topologies as the ones used in our model lay at the core of various popular architectures for distributed machine-learning use cases, implementing, e.g., AllReduce operations [11], [13], [14].

We consider the *constrained in-network processing* problem [15], where we have at our disposal a limited *budget* of k aggregation switches, which we can deploy (or activate) in some k locations throughout the network. Our objective in this work is to minimize the *network congestion*, i.e., minimizing the *most congested link* throughout the network, where link congestion if defined as the ratio between the number of

[1]Such tree topologies are common as a virtual overlay over a physical network or as sub-topologies in a data center.

messages traversing the link (i.e., the link load) and the rate of the link. Minimizing congestion is notably a key objective in networking, as it bears significant consequences for network and applications performance alike [16]–[22].

We assume each aggregating switch deployed in the network provides the ability to aggregate multiple incoming messages onto a *single* outgoing message. For cases where all switches can perform aggregation, one obtains the minimum congestion possible (as each link carries a single message). On the other extreme, when none of the switches has aggregation capabilities, congestion is extremely high, since essentially all messages must traverse the very few links entering the root.

However, for non extremal values of k, finding the optimal placement of a limited number of aggregation switches so as to minimize network congestion, is not a trivial task, even for trees, which is the case considered in this work. This is due to the fact that such an optimal placement of aggregation switches is affected by various network and workload factors, including the specific tree topology, the rates of the links, the load distribution at the servers, and the availability of resources for supporting such aggregation at the switches. Nevertheless, we present *an optimal algorithm* for performing such placements. Addtionally, our results show that placing relatively few aggregation nodes may drastically reduce network congestion, if judiciously placed in the proper locations.

Our model and results seem to be especially tailored for cloud environments, where providers may offer in-network aggregation with congestion guarantees as part of their business offerings. This can be viewed as part of their Network-as-a-Service (NaaS) suite, allowing the dynamic allocation, and reallocation, of in-network computing capabilities *on-demand*.

A. Our Contribution

We formulate the *Congestion-Minimization with Bounded In-network Computing* (C-BIC) problem, and present an optimal and time-efficient algorithm for solving the problem for a single workload on tree networks with heterogeneous link rates. Such topologies are common in datacenter networks, e.g., fat-tree topologies [23]. Our solution uses a hybrid search-and-dynamic-programming approach.

We further extend our framework to support *multiple tenants/workloads*, and adapt our algorithms to settings where workloads arrive in an *online* fashion. In these settings each switch may support a limited number of workloads, according to its *aggregation capacity*. Each new workload may use (some) in-network aggregation capabilities, and the aggregation capacities of the switches should be carefully allocated.

We discuss and present various properties of our resulting solutions, and evaluate their performance for various server load distribution, network sizes, workload arrivals, aggregation capacities, and network characteristics. In our study, we further consider two main *use cases*: (i) MapReduce (using word-count as an illustration), and (ii) gradient aggregation for distributed machine learning. We further show the benefits of using our algorithm when compared with several natural allocation strategies. Our results indicate that a small fraction of aggregation switches can already significantly reduce the network congestion in data aggregation tasks.

The paper is structured as follows. In Sec. II we introduce our formal system model. Sec. III provides a motivating example highlighting various aspects of the C-BIC problem. Sec. IV presents an overview of our optimal algorithm SMC and the main theoretical results. We evaluate our algorithm experimentally in Sec. V. We conclude our work with related work and discussion in Secs. VI and VII, respectively. We note that due to space constraints, we provide merely proof sketches for some of the proofs. The full proofs are available in [24].

II. Preliminaries & System Model

We consider a system comprising a set of n switches $\mathcal{S}$, a set of servers (workers) $\mathcal{W}$, and a special destination server $d \notin \mathcal{W}$. We assume there exists a pre-specified *root* switch $r \in \mathcal{S}$, and a *weighted tree network* $T = (V, E, \omega)$, where $V = \mathcal{S} \cup \{d\}$, $E = E' \cup \{(r, d)\}$ for some $E' \subseteq \mathcal{S}^2$ forming a tree over the set of switches $\mathcal{S}$, and $\omega : E \mapsto \mathbb{R}^+$ is the rate function of the links (in message per second). For $e \in E$ let $\tau(e) = \frac{1}{\omega(e)}$. The tree T thus consists of the underlying network topology connecting the switches, and connecting the root r to the destination d.

We further assume that all links in E are directed towards d. In particular, every switch $s \in \mathcal{S}$ has a unique *parent* switch $p(s) \in \mathcal{S}$ defined as the neighbor of s on the unique path from s to d. In such a case we say s is a *child* of $p(s)$, and we let $C(s)$ denote the number of children of switch s.

We assume each server $w \in \mathcal{W}$ is connected to a single switch $s(w) \in \mathcal{S}$, and let $L : \mathcal{S} \mapsto \mathbb{N}$ be the function matching each switch s with the number of servers connected to s. We refer to L as the *network load*. Each server w produces a single message, which is forwarded to $s(w)$, where we assume every message has size at most M, for some (large enough) constant M. Each switch s can be of one of two types, or operates at one of two modes:

(i) an *aggregating* switch (blue), which can aggregate messages arriving from its children (each of size at most M), to a *single* message (also of size at most M) and forwards it to its parent switch $p(s)$, or

(ii) a *non-aggregating* switch (red), which cannot aggregate messages, and simply forwards each message arriving from any of its children to its parent switch $p(s)$.

We denote by $\Lambda \subseteq \mathcal{S}$ the set of switches that are *available* as aggregation switches. Our view of aggregating switches is applicable to devices which compute, e.g., separable functions [25]. In particular, this holds true for aggregation functions computing, e.g., the average, or sum, of the values contained in the messages being sent by the servers.

In what follows we will be referring to aggregating switches as *blue* nodes in T, and to non-aggregating switches as *red* nodes in T. Our *budget* is denoted by a non-negative integer k, which serves as an upper bound on the number of blue nodes allowed in T. We will usually refer to $U \subseteq \Lambda$ as the set of blue nodes in T and require that $|U| \leq k$.

Algorithm 1 Reduce (T, L, U)

Require: A tree T, A network load L, A set of blue node U
Ensure: An aggregate information at destination d
1: For each node v in T do:
2: **while** not received all messages from all children **do**
3: process incoming message (by switch type: B, R)
4: if needed send message to $p(v)$ (by switch type: B, R)

Given a weighted tree network $T = (V, E, \omega)$ with a network load $L : \mathcal{S} \mapsto \mathbb{N}$, and a set of blue nodes $U \subseteq \Lambda$, we consider a simple Reduce operation on T as detailed in Algorithm 1. Every switch in the tree processes all messages received from its children and forwards message(s) to its parent. Every blue node (i.e., a node in U) is an aggregation switch and all other switches (i.e., nodes not in U) are non-aggregation switches. The operation ends when the *destination* receives the overall (possibly aggregated) information from all the nodes that have a strictly positive load.

For every link $e = (s, p(s)) \in E$, we define the *link load*, $\mathrm{msg}_e(T, L, U)$, as the number of messages traversing link e, given the Reduce operation on T, L, and U. $\psi_e(T, L, U) = \mathrm{msg}_e(T, L, U) \cdot \tau(e)$ denotes the *link congestion* and we let

$$\psi(T, L, U) = \max_{e \in T} \{\psi_e(T, L, U)\} \tag{1}$$

denote the *network congestion*. Our work considers the Congestion-minimization with Bounded In-network Computing (C-BIC) problem, which aims at minimizing the network congestion, formally defined as follows.

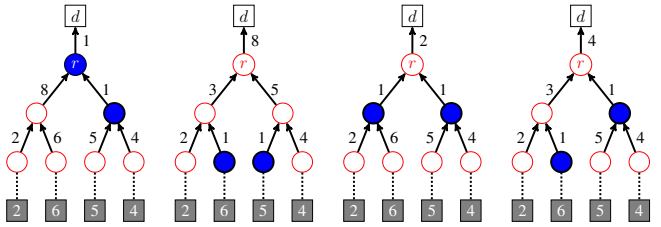
Definition 1 (C-BIC)**.** Given a weighted tree network $T = (V, E, \omega)$, a network load $L : \mathcal{S} \mapsto \mathbb{N}$, a set of available switches Λ, and a budget k, the *Congestion-minimization with Bounded In-network Computing* (C-BIC) problem is finding a set of switches $U \subseteq \Lambda$ of size at most k that minimizes the network congestion $\psi(T, L, U)$. Formally,

$$\text{C-BIC}(T, L, \Lambda, k) = \arg \min_{\substack{U \subseteq \Lambda \\ |U| = k}} \psi(T, L, U) \tag{2}$$

In trying to solve the C-BIC problem, one may use a brute-force approach, and enumerate over all possible subsets of Λ of size k. This may work well for a small constant k, but it becomes quickly intractable for arbitrary values of k. In what follows we will describe and discuss our *efficient* solution, SMC, to the C-BIC problem.

III. Motivating Example

We now turn to consider a motivating example highlighting the fact that simple, yet reasonable, approaches might fall short of finding an optimal solution to the C-BIC problem. Specifically, we consider the following three allocation strategies for determining the set of blue nodes: (i) The *Top* strategy, which picks the set of k blue nodes as the set closest to the root. This approach targets reducing the number of messages transmitted in the topmost part of the network, where congestion is expected to be largest. (ii) The *Max* strategy, which picks the set of blue nodes as the k switches with the largest load. This approach is motivated by the fact that one should aim at reducing link congestion "at the bud", which would presumably have a positive effect on overall congestion. (iii) The *Level* strategy, defined for complete binary trees, which aims at partitioning the network into subtrees of similar size, where all the messages within a subtree are aggregated. This is done by picking a whole level in the complete binary tree as the set of blue nodes. This approach, which essentially targets load balancing, strives to "equalize" congestion in distinct sub-trees in the network.



(a) Top (b) Max (c) Level (d) SMC

Figure 1: Example of solutions produced by 4 allocation algorithms for a simple load over a weighted tree network, with constant rates of 1 and $k = 2$ aggregation switches (blue nodes).

Consider a tree network with $n = 7$ switches which induces a complete binary tree topology on the set of switches which are all available for aggregation, with a constant rate of 1 for all links. Servers are connected only to leaf switches. Such a topology can be viewed as if the leaf switches are effectively top-of-rack (ToR) switches in a small data center topology, where each rack accommodates a distinct number of servers (or VMs). Fig. 1 provides an illustration of the network. Each leaf switch is connected to a rack of several worker servers where the number of workers in the rack is marked in the gray square. In particular, the load handled by the 4 leaf switches is $(2, 6, 5, 5)$ (from left to right). In our example the maximum number of blue switches allowed is set to $k = 2$. Each link e is marked with its link congestion, $\psi_e(T, L, U)$.

Figs. (1a), (1b), and (1c) show the results of applying strategies Top, Max, and Level, respectively, to such a network and load, obtaining a network congestion of 8, 9, and 6, respectively. The optimal approach attaining a network congestion of 5, which is obtained by our proposed algorithm, SMC (formally described and analyzed in Sec. IV), ends up picking a non-trivial set of blue nodes, as can be seen in Fig. (1d). As we show in the sequel, our algorithm is optimal, and thus is guaranteed to have the minimum congestion possible.

A further observation, which hinders the applicability of greedy approaches, is that the optimal solution is not necessarily monotone in k. For the network in Fig. 1, one may consider the optimal placement for $k = 2, 3, 4$.. There is no way to add a single blue node to the optimal solution for $k = 2$ and obtain an optimal set of blue nodes for $k = 3$, that is a subset of the optimal solution for $k = 4$.

Algorithm 2 SMC(T, L, Λ, k)

Require: A tree T, load L, availability Λ, k blue nodes

1: $X = \frac{1}{\min_e \omega(e)} \sum_v L(v)$ ▷ init. congestion upper bound
2: $S = \frac{1}{\max_e \omega(e)}$
3: **run** binary search in the range $[0, X]$ with step size S, using SMC-Gather, finding the *minimal* congestion upper bound X^*, returning the corresponding β^*
4: **run** SMC-Color(k) using β^*

IV. SMC: An Optimal Algorithm

In this section, we describe our algorithm, Search for Minimal Congestion (SMC), that produces an optimal solution to the C-BIC problem. The main technical contribution of our work is the following theorem.

Theorem 1. *Given a weighted tree network T with rates ω, a load L, availability Λ, and a bound k on the number of allowed blue switches, algorithm SMC solves the C-BIC problem in time $O\left(n \cdot k^2 \cdot \log\left(\frac{\omega_{\max}}{\omega_{\min}} \cdot \sum_v L(v)\right)\right)$.*

A. Overview of SMC

In this section, we provide a bird's-eye view of SMC, which is formally defined in Algorithm 2. The algorithm runs a binary search for the minimal congestion for which a feasible solution $U \subseteq \Lambda$ exists. Given the bound k on the number of blue nodes allowed in the network, for each potential upper bound X on the congestion, SMC uses dynamic programming, and is split into two phases.

The algorithm used during the binary search in the first phase, dubbed SMC-Gather, consists of scanning the switches in the tree in DFS-order. In every switch node v we effectively consider all *potentially efficient partitions* of any number $i \leq k$ of blue nodes across all children of the node. For every such i, the partition that minimizes the number of messages leaving the node is retained (maintained by the vector β_v), and information is passed on to the parent of the node. We note that the algorithm finds such a partition efficiently. The main property satisfied by SMC-Gather is shown in Lemma 2. The information disseminated upwards by SMC-Gather is then used in the second phase to compute the optimal solution (and place the blue nodes). SMC-Gather is formally defined in Algorithm 3, where it is described as an asynchronous distributed algorithm, with synchronization induced by messages sent from a node to its parent.

In the second phase we apply algorithm SMC-Color, which scans the nodes of the tree in reverse-DFS-order, and essentially tracks the feasible allocation satisfying the upper bound X on the congestion (if such an allocation exists). Initially a node is considered red, and during the scan SMC-Color sets a node as blue only when it is necessary for minimizing the congestion on its outgoing link, while satisfying the congestion constraint determined by the upper bound X (if possible). A node then informs each of its children as to the number of (remaining) blue nodes that can be distributed in the subtree rooted at that child. To this end, SMC-Color uses the information obtained by SMC-Gather, and in particular the partition that ensures that the congestion constraint is satisfied (if possible). SMC-Color is formally defined in Algorithm 4, where it is also described as an asynchronous distributed algorithm. Here synchronization is induced by messages received by a node from its parent.

B. Analysis of SMC

We begin by introducing some notation that would be used throughout our proofs. For very node v, we let $c_1, \ldots, c_{C(v)}$ denote the children of v (in some arbitrary fixed order). For every $m = 1, \ldots, C(v)$ we let T_v^m denote the subtree rooted at v containing only the subtrees rooted at children $c_1, \ldots, c_m$, and let $\tilde{T}_v^m$ denote the *extended subtree* of T_v^m, which is extended by adding the link $(v, p(v))$. We further let $T_v = T_v^{C(v)}$ denote the subtree rooted at v and let $\tilde{T}_v$ be the extended subtree of T_v.

Let X be a real value, representing an upper bound on *network congestion*. We define $\beta(T_v, L, k, X)$ as the minimum number of messages traversing link $(v, p(v))$ for which there exists a set $U \subseteq T_v, |U| = k$ that satisfies the congestion constraint $\psi(\tilde{T}_v, L, U) \leq X$ (or infinity if no such set exists).

Given some value X, algorithm SMC-Gather uses the following concepts: (i) variables $\beta_v^m(i, \text{color})$ denote the minimum number of messages traversing link $(v, p(v))$ in the tree $\tilde{T}_v^m$, where v is colored by color and at most i nodes in T_v^m are blue, while ensuring that the congestion in $\tilde{T}_v^m$ is at most X if feasible, and $\beta_v^m(i, \text{color}) = \infty$ otherwise. (ii) variables $\beta_v(i) = \min\{\beta_v^m(i, B), \beta_v^m(i, R)\}$. The following lemma shows that the semantics we attribute to $\beta_v^m(i, \text{color})$ are correct, and that SMC-Gather correctly computes $\beta(T_v, L, i, X)$.

Lemma 2. *For every node v, every $m = 1, \ldots, C(v)$, and every $i = 0, \ldots, k$, $\beta_v(i)$ as computed by SMC-Gather satisfies $\beta_v(i) = \beta(T_v, L, i, X)$, where if v is not a leaf then β_v^m as computed by SMC-Gather(T, L, Λ, k, X) satisfies*

$$\beta_v^m(i, R) = \beta(T_v^m, L, i, X) \text{ where } v \text{ is colored } R \tag{3}$$

and

$$\beta_v^m(i, B) = \beta(T_v^m, L, i, X) \text{ where } v \text{ is colored } B, \tag{4}$$

where

$$\beta_v^1(i, B) = \begin{cases} 1, & \text{if } \beta_{c_1}(i-1) < \infty \\ \infty, & \text{otherwise,} \end{cases} \tag{5}$$

$$\beta_v^1(i, R) = \begin{cases} \beta_{c_1}(i) + L(v), & \text{if } (\beta_{c_1}(i) + L(v)) \cdot \tau(v) \leq X \\ \infty, & \text{otherwise} \end{cases} \tag{6}$$

and for $m > 1$

$$\beta_v^m(i, B) = \begin{cases} 1, & \text{if } \min\limits_{0 \leq j < i} (\beta_v^{m-1}(i-1-j, B) + \beta_{c_m}(j)) < \infty \\ \infty, & \text{otherwise} \end{cases} \tag{7}$$

$$\beta_v^m(i, R) = \begin{cases} \min\limits_{0 \leq j \leq i} (\beta_v^{m-1}(i-j, R) + \beta_{c_m}(j)), & \text{if (9) holds} \\ \infty, & \text{otherwise} \end{cases} \tag{8}$$

where

$$\min_{0\le j\le i}\left(\beta_v^{m-1}(i-j,R)+\beta_{c_m}(j)\right)\cdot\tau(v)\le X. \quad (9)$$

Overall,

$$\beta_v(i)=\min\left(\beta_v^{C(v)}(i,B),\beta_v^{C(v)}(i,R)\right)=\beta(T_v,L,i,X) \quad (10)$$

Proof sketch: First, we note that, by definition, SMC-Gather computes $\beta_v^m(i,\text{color})$ and $\beta_v(i)$ according to equations (6)-(9). Eq. (6) corresponds to lines 16-18, Eq. (5) corresponds to lines 19-22, Eq. (8) and Eq. (7) corresponds to lines 25 and 24, respectively, and finally Eq. (10) corresponds to line 27. It therefore suffices to show that the above equations imply Eqs. (3) and (4).

The proof is by double induction on the height of the subtree T_v rooted at any node v, and indices $m=1,\ldots,C(v)$ of the children of a node v. First observe that for any leaf node v, $\beta_v(i)=1$ if $i>0$ (since v can be colored blue, and this minimizes the load on link $(v,p(v))$). If $i=0$, since v cannot be colored blue we have $\beta_v(0)=L(v)$ if $L(v)\cdot\tau(v,p(v))\le X$, and $\beta_v(0)=\infty$ otherwise.

For $m=1$ the claim holds by the induction hypothesis on the height of v and specifically, the correctness of $\beta_{c_1}(i)$ (using Eq. (6)-(5)). For $m>1$, the claim holds by the induction hypothesis on both the number of children of v, as well as the height of v. Specifically, by the fact that the claim holds for $\beta_v^{m-1}(\ell-j,\text{color})$ (induction on m) and $\beta_{c_m}(j)$ (induction on height) for all $\ell\ge j$.

Intuitively, for the case, e.g., where v is colored R, this means that there exists a partition of $\ell=i$ blue nodes across the subtree T_v^m inducing a congestion of at most X, and *minimizing* the sum of messages on $(v,p(v))$. Such a partition is obtained for distributing some $0\le j\le\ell$ blue nodes in T_{c_m}, and the remaining $\ell-j$ blue nodes in T_v^{m-1}. This similarly holds if v is colored B.

If $\beta_{c_m}(j)=\infty$ then by the induction hypothesis it is impossible to maintain congestion at most X in the subtree T_{c_m}, which implies the same result for T_v^m. On the other hand, if for every child c_m of v, $\beta_{c_m}(j)$ is finite, then having congestion at most X in $\tilde{T}_v$ depends merely on the number of messages traversing link $(v,p(v))$. Since by the induction hypothesis, $\beta_{c_m}(j)$ is the *minimal* number of messages entering v from c_m, satisfying the congestion constraint within $\tilde{T}_{c_m}$, and $\beta_v^{m-1}(\ell-j,\text{color})$ is the *minimal* number of messages traversing $(v,p(v))$ in $\tilde{T}_v^{m-1}$ while satisfying the congestion constraint within $\tilde{T}_v^{m-1}$, this implies that the *total* number of messages traversing link $(v,p(v))$ would also be minimal, subject to the congestion constraint on T_v^m. It follows that if the congestion on link $(v,p(v))$ is at most X (as verified by Eq. (9)), then Eq. (3) has been computed correctly; It captures the minimum number of messages traversing the link $(v,p(v))$ while satisfying $\psi(\tilde{T}_v^m,L,U)\le X$. If the congestion on this link is more than X, then since the number of messages traversing the link is the minimal possible while satisfying the congestion constraint in T_v^m, it is impossible to have congestion at most X on both link $(v,p(v))$ and T_v^m, and thus, $\beta_v^m(i,R)$ is set to ∞. The same intuition applies to the case where v is colored B (using $\ell=i-1$), implying that Eq. (4) holds true. The proof follows. ■

Algorithm 3 SMC-Gather(T,L,Λ,k,X) at node v

Require: A tree T, load L, availability Λ, k # of blue nodes and X maximal link utilization.

Ensure: Correct potential functions, β_v, at each node v

1: **if** v is a leaf node **then**
2: $\quad\beta_v(0)=L(v)$
3: $\quad$**if** $\beta_v(0)\cdot\tau(v,p(v))>X$ **then**
4: $\quad\quad\beta_v(0)=\infty$
5: $\quad$**for** $i=1,\ldots,k$ **do** ▷ v can be blue
6: $\quad\quad$**if** $v\in\Lambda$ **then** ▷ v is available
7: $\quad\quad\quad\beta_v(i)=1$
8: $\quad\quad$**else**
9: $\quad\quad\quad\beta_v(i)=\beta_v(0)$
10: $\quad$send β_v to $p(v)$ and **return** ▷ inform parent
11: wait to receive β_c from each child c of v
12: **for** $m=1,\ldots,C(v)$ **do**
13: $\quad c_m\leftarrow$ the m'th child of v
14: $\quad$**for** $i=0,\ldots,k$ **do**
15: $\quad\quad$**if** $m=1$ **then**
16: $\quad\quad\quad\beta_v^m(i,R)=\beta_{c_m}(i)+L(v)$
17: $\quad\quad\quad$**if** $\beta_v^m(i,R)\cdot\tau(v,p(v))>X$ **then**
18: $\quad\quad\quad\quad\beta_v^m(i,R)=\infty$
19: $\quad\quad\quad$**if** $i>0$ **and** $\beta_{c_m}(i-1)\le X$ **and** $v\in\Lambda$ **then**
20: $\quad\quad\quad\quad\beta_v^m(i,B)=1$
21: $\quad\quad\quad$**else**
22: $\quad\quad\quad\quad\beta_v^m(i,B)=\infty$
23: $\quad\quad$**else** ▷ $m>1$
24: $\quad\quad\quad\beta_v^m(i,B)=\text{mCost}(i-1,\beta_v^{m-1},\beta_{c_m},X,B)$
▷ when $i=0$ then $\beta_v^m(i,B)=\infty$
25: $\quad\quad\quad\beta_v^m(i,R)=\text{mCost}(i,\beta_v^{m-1},\beta_{c_m},X,R)$
26: **for** $i=0,\ldots,k$ **do**
27: $\quad\beta_v(i)=\min\left\{\beta_v^{C(v)}(i,B),\beta_v^{C(v)}(i,R)\right\}$
28: send β_v to $p(v)$ and **return**

29: **procedure** mCost$(i,\beta_v^{m-1},\beta_{c_m},X,\text{color})$
30: $\quad\beta=\min_{0\le j\le i}[\beta_v^{m-1}(i-j,\text{color})+\beta_{c_m}(j)]$
31: $\quad$**if** $\beta\cdot\tau(v,p(v))>X$ **then**
32: $\quad\quad$**return** ∞
33: $\quad$**else**
34: $\quad\quad$**return** β

In the second phase of SMC, SMC-Color traces back the allocation of blue nodes along the optimal path in the dynamic programming performed by SMC-Gather. The following lemma provides the grounds for the optimality of the solution produced by SMC-Color.

Lemma 3. *Assume β is the output of SMC-Gather for the network congestion upper bound X, such that $\beta_r(k)$ is finite. Then, SMC-Color colors blue a set U, such that $|U|\le k$, and $\psi(T,L,U)\le X$.*

Algorithm 4 SMC-Color(k) at node v

Require: β
Ensure: Optimal coloring
1: **if** v is the destination d **then**
2: send k to r and **return**
3: color v red and wait for i from $p(v)$
▷ i: number of blue nodes in T_v
4: **if** v is a leaf node and $i > 0$ **then**
5: color v blue and **return**
6: **if** $\beta_v^{C(v)}(i, B) < \infty$ **then** ▷ $\beta_v^{C(v)}(i, B) \leq \beta_v^{C(v)}(i, R)$
7: color v blue
8: **for** $m = C(v), \ldots, 2$ **do** ▷ children in reverse order
9: $j = \mathrm{mSplit}(i, \beta_v^{m-1}, \beta_{c_m}, \text{color of } v)$
10: send j to c_m
11: $i = i - j$
12: **if** v is blue **then** ▷ handle c_1 last
13: send $i - 1$ to c_1
14: **else**
15: send i to c_1
16: **return**

17: **procedure** $\mathrm{mSplit}(i, \beta_v^{m-1}, \beta_{c_m}, \mathrm{color})$
18: **if** color $==$ R **then**
19: **return** $\arg\min_{0 \leq j \leq i}[\beta_v^{m-1}(i - j, \mathrm{color}) + \beta_{c_m}(j)]$
20: **else** ▷ color $==$ B
21: **return** $\arg\min_{0 \leq j < i}[\beta_v^{m-1}(i - j, \mathrm{color}) + \beta_{c_m}(j)]$

Proof sketch: The proof is by induction on the order in which node v receives the message from its parent $p(v)$. Assume v receives i blue nodes to distribute from $p(v)$. In a nutshell, if v is a leaf, it is colored blue if $i > 0$, which implies that in its (empty) subtree the congestion is at most X. If v is not a leaf but colored blue, this implies that the condition in line 6 holds true. In such a case, by Lemma 2, v can be colored blue (leaving $i - 1$ blue nodes to distribute across the subtrees rooted at its children): the load on link $(v, p(v))$ would be minimal (i.e., 1), and the maximum congestion in $\tilde{T}_v$ can be made at most X. If v is colored red, this means that the condition in line 6 is false. By Lemma 2, it is impossible to obtain congestion at most X in $\tilde{T}_v$ by distributing merely i blue nodes in T_v, with v colored blue. Furthermore, by the induction hypothesis, v received i since it is possible to maintain the congestion bound of X by distributing i blue nodes in T_v. Since by using the mSplit procedure, v partitions i blue nodes to the subtrees rooted at its children according to β, by the assumption on β and Lemma 2, this would satisfy the congestion constraint in $\tilde{T}_v$, and minimize the load (and the congestion) on link $(v, p(v))$. ■

We now show that the C-BIC problem can be reduced to computing $\beta(T, L, k, X)$.

Lemma 4. *If $\beta(T, L, k, X)$ is computed in α time, then C-BIC(T, L, Λ, k) can be solved in time $\alpha \cdot \log(\sum_v L(v) \cdot \frac{\omega_{\max}}{\omega_{\min}})$.*

Proof: The proof follows from applying a binary search over the upper bound X on the network congestion, where the maximum such value is no larger than $\frac{1}{\omega_{\max}} \cdot \sum_v L(v)$, and the granularity is at least $\frac{1}{\omega_{\min}}$. In each iteration we check whether $\beta(T, L, k, X)$ is finite using SMC-Gather. ■

We can now prove Theorem 1.

Proof of Theorem 1: The correctness of the algorithm follows from Lemmas 2-4. For the running time of SMC, we note that it is dominated by the running time of SMC-Gather, which, in turn, is dominated by the for-loops in lines 12-25. This loop handles every edge $(v, p(v))$ once, and for each edge the running time is $O(k^2)$, resulting in a total running time for SMC-Gather of $O(n \cdot k^2)$. By Lemma 4, performing the binary search requires running SMC-Gather $O\left(\log(\sum_v L(v) \cdot \frac{\omega_{\max}}{\omega_{\min}})\right)$ times, resulting is a total running time for solving the C-BIC problem of $O\left(n \cdot k^2 \cdot \log\left(\frac{\omega_{\max}}{\omega_{\min}} \cdot \sum_v L(v)\right)\right)$. ■

V. Evaluation

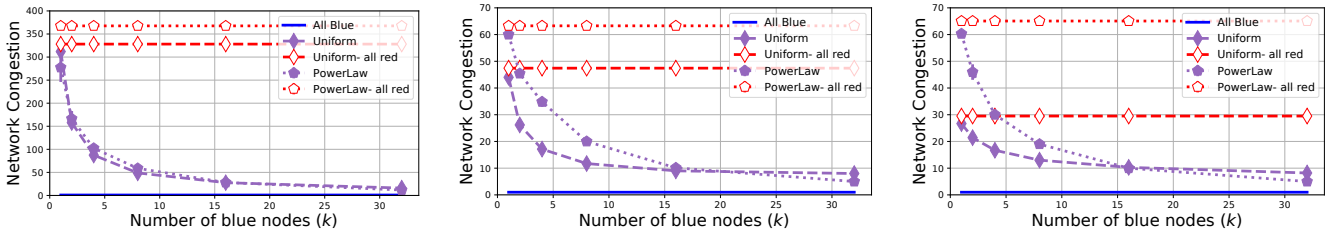
In this section, we report the results of our extensive evaluation of SMC. Our results shed light on various aspects pertaining to its performance, and also on the problem it is designed to solve. In our evaluation, we examine both the network congestion induced by SMC, as well as that obtained by contending strategies. We also show the result of running distributed application, including *word count* (using MapReduce), and gradient aggregation in distributed machine learning. These results essentially perform the Reduce operation on real workloads, thus highlighting real-world benefits.

We use the following setup for most of our evaluation (unless explicitly stated otherwise). Our network is a complete binary *tree* with 255 nodes (and 128 leaves), where links have weights denoting their capacity. We place load only in the *leaves* of the tree, which serve as *top-of-the-rack (ToR) switches* connected to servers (workers) that generate load. The remaining network switches model the higher levels of a data center network, which facilitates a flow of information from the worker to the destination, serving as the aggregation server, that is connected to the root of the tree.

We consider two distributions for the load generated at the leaves, both with an average load of 5 workers per ToR switch: (i) an almost *uniform* load, where the load of each node is picked u.a.r. in the range of integers $[1, 9]$ (with variance 2.6), and (ii) a *power-law* load, where the (integer) load of each node is picked from a power-law distribution in the range $(1, 63)$ (with variance 97.1).

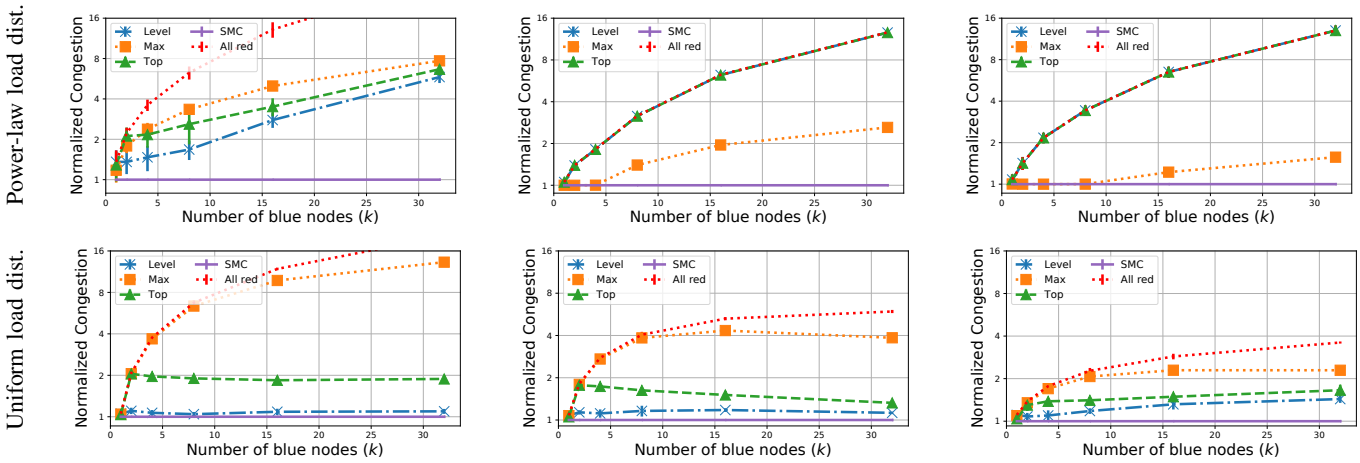
We further consider three different rate schemes for the links in the tree: (i) *constant* rates, were all link rates are equal to 1, (ii) *linear* rates, were $\omega(e)$ increases linearly, by adding 1, from leaf edges (rate 1) towards the root, with a maximum rate of 7 in links entering the root, and (iii) *exponential* rates, were $\omega(e)$ increases exponentially with base 1.5, from leaf edges (rate 1), towards the root, with a maximum rate of 17 in links entering the root.

Each experiment was repeated ten times and we present the average performance for each such set of experiments.



(a) constant ($\omega = 1$) (b) linear increasing ($\omega = i$) (c) exponentially increasing ($\omega = (1.5)^i$)

Figure 2: Limited In-network aggregation, SMC congestion gains with limited resources



(a) constant ($\omega = 1$) (b) linear increasing ($\omega = i$) (c) exponentially increasing ($\omega = (1.5)^i$)

Figure 3: SMC vs. other strategies for distinct rates (Fig. 3a-3c), and load distributions (top plot vs. bottom plot).

For clarity we present error bars only where we encountered significant variance in the results.

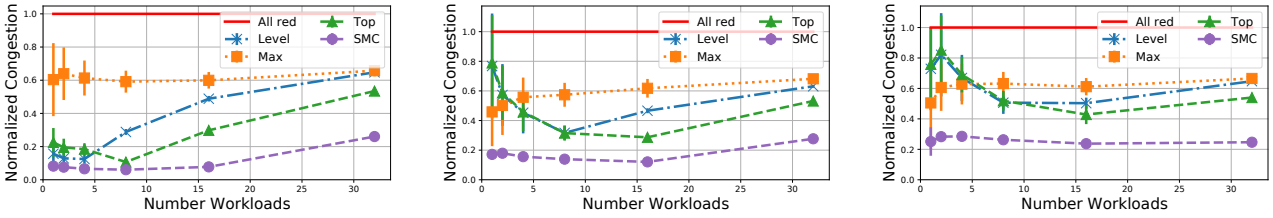
The Gains from Limited In-network Aggregation : We first consider the network congestion reduction when using limited in-network aggregation resources. Fig. 2 presents the network congestion of SMC for the three rate schemes and the two workload distributions, where we increase the number k of blue nodes. The figure also shows the network congestion for the *all-blue* and the *all-red* scenarios, which provide upper- and lower-bounds on the possible congestion. The main takeaway from this figure is that in-network aggregation reduces the network congestion, and does that at a fast pace; Even with a small number of aggregation switches significant reduction is achieved. Specifically, in all cases using merely 32 aggregation switches, which are about 12% of the nodes, induces a x10 reduction in network congestion, which is close to the congestion obtained in the all-blue scenario.

Comparing SMC with Other Strategies: We now consider the performance of SMC compared to the performance of several contending strategies for solving the C-BIC problem. Specifically, we focus our attention on the simple strategies described in our motivating example in Sec. III, namely, (i) *Top*, (ii) *Max*, and (iii) *Level*.

Fig. 3 presents the performance of SMC alongside the performance of the contending strategies in the three rate scheme (left to right), for the two different workload distribution (top and bottom), where we consider $k = 1, 2, 4, 8, 16, 32$. and the network congestion of each algorithm is *normalized* to the network congestion achieved by our algorithm, SMC, which was shown to be optimal in Sec. IV-B. We further plot the performance of the *all-red* solution for reference. As would be expected (by the optimality of SMC), all strategies preform worse then SMC, sometimes as much as x13 worse.

One can note that with the power-law workload distribution, and with constant rates, Max performs worse than Top and Level (3a, top), while for the linear and exponentially increasing rates it outperforms them (3b and 3c, top). This is due to the *location* where maximum link congestion is encountered. In the constant rate regime the maximum link congestion occurs closer to the root of the tree. In contrast, when link rates are higher, the maximum congested link is "pushed" farther from the root, towards the leaves. However, this phenomena does not assist Max under the *uniform load distribution*, since, due to the smaller variance of this distribution, Max is unable to reduce all heavily loaded ToR switches.

Since SMC is optimal, it exhibits the best performance in all scenarios. This serves to show that using SMC ensures robustness regardless of load distribution or link rates. However, the second-best strategy strongly depends on the load distribution, or the link rates. The power-law load distribution favors the



(a) constant ($\omega = 1$) (b) linearly increasing ($\omega = i$) (c) exponentially increasing ($\omega = (1.5)^i$)

Figure 4: SMC vs. other strategies when aggregating more workloads ($k = 16$, switch aggregation capacity is 4)

Max strategy, since high-load ToR switches that perform aggregation induce a significant reduction in congestion. For the uniform distribution, however, the Level strategy fares best, since it manages to load balance the uniform loads at the leaf-switches throughout the network. The Top strategy is the most sensitive to the link rates, where having higher rates towards the root of the network implies that performing in-network aggregation further up provides very little benefits compared to performing aggregation closer to the leaves.

Multiple Workloads: We now turn to address the problem of handling *multiple workloads*, and determining where aggregation should take place for each such workload. We note that this serves as an extension of our framework that goes beyond the model described in Sec. II. Each workload L_t is determined by its time, $t = 0, 1, 2, \ldots$. We consider a sequence of workloads, L_t, $t = 0, 1, 2, \ldots$, arriving in an *online* fashion, such that determining the aggregating switches for workload L_t should be settled before handling workload L_{t+1}.

We further assume each switch s has a predetermined *aggregation capacity* $a(s)$ which bounds the number of workloads for which s can be assigned as an aggregating switch. We let $a_t(s)$ denote the *residual aggregation capacity* remaining at s before handling workload L_t. If switch s is designated as an aggregation switch when handling workload L_t, then $a_{t+1}(s) = a_t(s) - 1$, and $a_{t+1}(s) = a_t(s)$ otherwise.

We examine the performance of the various strategies considered in Sec. V, when applied repeatedly to the sequence of workloads $L_0, L_1, \ldots$, given as input. The set of switches available for aggregation when handling workload L_t is defined by $\Lambda_t = \{s \mid a_t(s) > 0\}$.

We generate our sequence of workloads in an online fashion, by drawing each workload from either the uniform load distribution, or the power-law load distribution, each with probability $\frac{1}{2}$, and use as our baseline the values $k = 16$ and $a(s) = 4$ for every switch s. We evaluate the system's performance when handling more and more workloads, where we specifically consider handling $1, 2, 4, 8, 16, 32$ workloads.

Fig. 4 shows the performance of SMC compared to the performance of the various strategies described in Sec. III. Similarly to our previous results, our evaluation considers 3 scaling laws for link rates: constant (in Fig. 4a), linearly increasing (in Fig. 4b), and exponentially increasing (in Fig. 4c).

The figure shows the normalized network congestion, where congestion is normalized to that obtained by the all-red solution. Namely, if the performance of an algorithm is $\alpha \in [0, 1]$ in some scenario, this means that the algorithm entails a network congestion that is an α fraction of the congestion incurred by the all-red scheme. Notice that as the number of workloads increases, the performance of any strategy would converge to that of the all-red configuration. This follows from the fact that the aggregation capacity is bounded, implying that once the number of workloads is large enough, further workloads cannot benefit from any aggregation, and the initial benefits of aggregating the prefix of the workload arrival sequence become marginal compared to the toll imposed by the entire sequence. This explains the worsening performance exhibited when increasing the number of workloads. Nevertheless, for the exponential rates regime SMC is able to sustain a larger amount of workloads before changing for the worse.

Switch Capacity: We now turn to evaluate the effect of the switch in-network capacity. Similarly to the evaluation of multiple workloads we normalize the results to the all-red scenario, and consider distinct link rates environments.

Fig 5 shows the effect of varying the aggregation capacity on the performance of SMC, while using $k = 16$, 32 workloads, and distinct values $a(s) = 4, 8, 16, 32$ for every switch s. In such a scenario, clearly a capacity of 32 will yield the best performance, as capacity is abundant, and each workload can be aggregated optimally, independently of other workloads. However, as shown in fig 5, SMC actually achieves this optimal performance with significantly smaller switch capacity.

SMC for Different Applications: We now consider two use cases for evaluating the system: (i) *big-data*, using a word-count task [26], where we make use of a wikipedia dump [27], with an overall of 54M words, out of which 800K are unique. We refer to this use case as the *word count (WC)* use case. (ii) *distributed ML*, using distributed gradient aggregation with a parameter server [28], where worker servers independently perform neural-network training, over a 10K feature space, using 0.5 dropout rate [29]. The workers send their updated gradients to a parameter server, which then updates the system model parameters. We refer to this use case as the *parameter server (PS)* use case, where we focus on the tradeoffs between aggregation and congestion (and not model quality).

We evaluate the performance of SMC for WC, and PS, using the constant rates regime, which better highlights the differ-

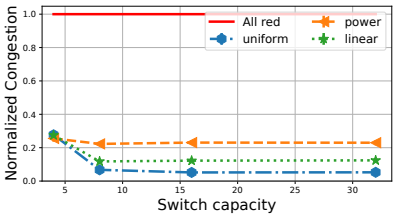


Figure 5: SMC performance when increasing the switch capacity, 32 workloads and $k = 16$ per workload.

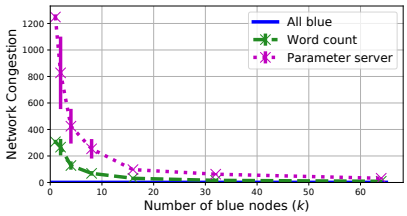


Figure 6: SMC performance for the WC and PS use cases.

ences in the performance, and using the uniform distribution which is more challenging for reducing congestion.

Fig. 6 shows the results of our evaluation, which highlight the significant reduction in network congestion even when using a small number of aggregation switches. The main takeaway here is that the *application* scenario has a significant impact on the network congestion. While in the PS use-case the congestion is very high without aggregation and rapidly improves once (limited) aggregation is deployed, for the WC use-case network congestion is significantly smaller apriori, and the improvement obtained by deploying few aggregation switches is milder.

VI. Related Work

Various studies considered data aggregation [30], covering diverse domains such as wireless networks, scheduling, etc. [31], [32], and studying which functions may be aggregated efficiently [30], [33]. Furthermore, as discussed in Sec. I, data aggregation is a cornerstone of big data tasks, using, e.g., the MapReduce framework [2], [34], and more recently also of distributed machine learning (ML) environments, performing, e.g., the training of deep neural networks.

Specifically for such ML tasks, network performance has been noted as a major bottleneck hindering the efficient usage of such frameworks [3], [35]. Various approaches have been suggested to modify ML methodologies in order to improve upon the network induced performance of distributed ML [4], [36], [37]. Additional network- and system-level adaptations have been suggested to improve upon ML performance of such systems [38]–[40]. A notable use-case which applies to our framework is the usage of a *parameter server* for aggregating and distributing model parameters [28], where various works addressed the networking overheads it entails [35], [41], [42]. Additional approaches focus on *gradient aggregation*, where merely gradients are aggregated and distributed to the workers. This concept has gained significant popularity in frameworks of federated ML [43]. A special emphasis is notably given for supporting large-scale ML in High-Performance Computing (HPC) clusters, including specially tailored protocols for doing in-network aggregation (e.g., Nvidia's SHARP [10]).

More generally, in-network computing has been the focus of much attention, fueling the design of advanced architectures ranging from network HW design [44], through networking services [45], up to various applications [46]–[48], including ML [11], [49], to name but a few.

We note that the majority of these works address the incorporation of specific functionalities within the network, or the application. In contrast, our work considers a more general network-level problem focusing on resource allocation and placement within the network, in scenarios where resources are scarce, in an attempt to optimize system performance, independent of the specific application being served.

Lastly, the work most closely related to ours is [15], where we studied the same model, but developed algorithms which minimize the utilization complexity of the network, i.e., the average link utilization over all network links. While the framework is similar, the different objectives called for a significantly distinct high-level approach.

VII. Discussion and Future Work

This work considers the C-BIC problem, where we need to determine the location of a limited number of aggregation switches performing a reduce operation, within a tree network, so as to minimize the network congestion. This problem lays at the heart of many distributed computing use cases, and most notably in variations of the *AllReduce* operation for distributed and federated machine learning. Our work describes an optimal algorithm, SMC, for solving the C-BIC problem in trees, and provides insights as to the performance of SMC via an extensive simulation study.

Developing solutions that are applicable to *general* networks (i.e., not necessarily tree networks), thus supporting multi-path routing is a challenging task we leave for future research. Obtaining worst-case guarantees for multiple workloads is another interesting open problem. The main challenge there is how to distribute remaining aggregation capacity throughout the network to the various workloads. In general, we may serve every workload using a *different* number of aggregation switches (i.e., there need not be a uniform k for all workloads). Finally we would like to target minimizing the *delay* incurred by the system, and we expect our general algorithmic approach to also be effective for such objectives. The authors have provided public access to their code and/or data at https://github.com/razseg/SMC.

Acknowledgments: The work was partially funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation program (grant agreement No 864228 - AdjustNet).

REFERENCES

[1] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *SIGCOMM*, 2011, pp. 98–109.

[2] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. R. Pietzuch, and A. L. Wolf, "Netagg: Using middleboxes for application-specific on-path aggregation in data centres," in *CoNEXT*, 2014, pp. 249–262.

[3] R. Viswanathan, A. Balasubramanian, and A. Akella, "Network-accelerated distributed machine learning for multi-tenant settings," in *SoCC*, 2020, pp. 447–461.

[4] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, M. Canini, and P. Kalnis, "Compressed communication for distributed deep learning: Survey and quantitative evaluation," KAUST, Tech. Rep., 2020.

[5] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM*, 2010, pp. 63–74.

[6] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, 2013.

[7] D. R. K. Ports and J. Nelson, "When should the network be the computer?" in *HotOS*, 2019, pp. 209–215.

[8] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *HotNets*, 2017, pp. 150–156.

[9] P. Costa, A. Donnelly, A. I. T. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *USENIX NSDI*, 2012, pp. 29–42.

[10] R. L. Graham, L. Levi, D. Bureddy, G. Bloch, G. Shainer, D. Cho, G. Elias, D. Klein, J. Ladd, O. Maor, A. Marelli, V. Petrov, E. Romlet, Y. Qin, and I. Zemah, "Scalable hierarchical aggregation and reduction protocol (SHARP)^{TM} streaming-aggregation hardware design and evaluation," in *ISC*, 2020, pp. 41–59.

[11] N. Gebara, M. Ghobadi, and C. Paolo, "In-network aggregation for shared machine learning clusters," *MLSys*, vol. 3, 2021.

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: programming protocol-independent packet processors," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[13] S. Jeaugey, "Massively scale your deep learning training with nccl 2.4," 2019, nVIDIA Developer Blog, https://tinyurl.com/ynca94ek.

[14] P. Sanders, J. Speck, and J. L. Träff, "Two-tree algorithms for full bandwidth broadcast, reduction and scan," *Parallel Computing*, vol. 35, no. 12, pp. 581–594, 2009.

[15] R. Segal, C. Avin, and G. Scalosub, "SOAR: minimizing network utilization with bounded in-network computing," in *CoNEXT*, 2021, pp. 16–29.

[16] R. Banner and A. Orda, "Multipath routing algorithms for congestion minimization," *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 413–424, 2007.

[17] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *STOC*, 2008, pp. 255–264.

[18] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the I/O of HPC applications under congestion," in *IPDPS*, 2015, pp. 1013–1022.

[19] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P. Bremer, M. Schulz, and L. V. Kalé, "Identifying the culprits behind network congestion," in *IPDPS*, 2015, pp. 113–122.

[20] N. Bansal, K. Lee, V. Nagarajan, and M. Zafer, "Minimum congestion mapping in a cloud," *SICOMP*, vol. 44, no. 3, pp. 819–843, 2015.

[21] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network design with minimal congestion and route lengths," in *INFOCOM*, 2019, pp. 1351–1359.

[22] L. Gao and G. N. Rouskas, "Congestion minimization for service chain routing problems with path length considerations," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2643–2656, 2020.

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

[24] R. Segal, C. Avin, and G. Scalosub, "Constrained in-network computing with low congestion in datacenter networks," *CoRR*, 2022. [Online]. Available: http://arxiv.org/abs/2201.04344

[25] D. Mosk-Aoyama and D. Shah, "Computing separable functions via gossip," in *PODC*, 2006, pp. 113–122.

[26] "Apache hadoop - mapreduce tutorial," 2021. [Online]. Available: https://tinyurl.com/zd74vfht

[27] "Wikimedia downloads," 2021. [Online]. Available: https://tinyurl.com/ybj2ysya

[28] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *USENIX OSDI*, 2014, pp. 583–598.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] P. Jesus, C. Baquero, and P. S. Almeida, "A survey of distributed data aggregation algorithms," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 381–404, 2015.

[31] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery, "Information fusion for wireless sensor networks: Methods, models, and classifications," *ACM Comput. Surv.*, vol. 39, no. 3, p. 9, 2007.

[32] B. Malhotra, I. Nikolaidis, and M. A. Nascimento, "Aggregation convergecast scheduling in wireless sensor networks," *Wirel. Networks*, vol. 17, no. 2, pp. 319–335, 2011.

[33] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: interfaces and implementations," in *SOSP*, 2009, pp. 247–260.

[34] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *USENIX OSDI*, 2004, pp. 137–150.

[35] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *NIPS*, 2014, pp. 19–27.

[36] A. Dutta, E. H. Bergou, A. M. Abdelmoniem, C.-Y. Ho, A. N. Sahu, M. Canini, and P. Kalnis, "On the discrepancy between the theoretical analysis and practical implementations of compressed communication for distributed deep learning," in *AAAI*, 2020, pp. 3817–3824.

[37] S. Wang, D. Li, and J. Geng, "Geryon: Accelerating distributed CNN training by network-level flow scheduling," in *INFOCOM*, 2020, pp. 1678–1687.

[38] A. M. Abdelmoniem, C. Ho, P. Papageorgiou, M. Bilal, and M. Canini, "On the impact of device and behavioral heterogeneity in federated learning," 2021, arXiv, https://arxiv.org/abs/2102.07500.

[39] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, "Impact of network topology on the performance of dml: Theoretical analysis and practical factors," in *INFOCOM*, 2019, pp. 1729–1737.

[40] S. Ouyang, D. Dong, Y. Xu, and L. Xiao, "Communication optimization strategies for distributed deep neural network training: A survey," *J. Parallel Distributed Comput.*, vol. 149, pp. 52–65, 2021.

[41] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *USENIX HotCloud*, 2015.

[42] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, "Parameter hub: a rack-scale parameter server for distributed deep neural network training," in *SoCC*, 2018, pp. 41–54.

[43] A. Reisizadeh, H. Taheri, A. Mokhtari, H. Hassani, and R. Pedarsani, "Robust and communication-efficient collaborative learning," in *NeurIPS*, 2019, pp. 8386–8397.

[44] H. Eran, L. Zeno, M. Tork, G. Malka, and M. Silberstein, "NICA: An infrastructure for inline acceleration of network applications," in *USENIX ATC*, 2019, pp. 345–362.

[45] P. Shantharama, A. S. Thyagaturu, and M. Reisslein, "Hardware-accelerated platforms and infrastructures for network functions: A survey of enabling technologies and research studies," *IEEE Access*, vol. 8, pp. 132 021–132 085, 2020.

[46] H. T. Dang, P. Bressana, H. Wang, K. Lee, N. Zilberman, H. Weatherspoon, M. Canini, F. Pedone, and R. Soulé, "P4xos: Consensus as a network service," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1726–1738, 2020.

[47] Y. Tokusashi, H. Matsutani, and N. Zilberman, "LaKe: The power of in-network computing," in *ReConFig*, 2018.

[48] S. Vaucher, N. Yazdani, P. Felber, D. E. Lucani, and V. Schiavoni, "ZipLine: in-network compression at line speed," in *CoNEXT*, 2020, pp. 399–405.

[49] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," 2019.