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

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 paper, 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 datacenter 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 are the driving force behind 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], arising also in Big Data applications, e.g., within MapReduce frameworks.

In an effort 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 there is (probably) no free lunch [13] 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 one used in our model lay at the core of various popular architectures for distributed machine-learning use cases, implementing, e.g., AllReduce operations [11], [14], [15].

We consider the constrained in-network processing problem [16], 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

1Such tree topologies are common as a virtual overlay over a physical network or as sub-topologies in a data center.
the most congested link throughout the network, where link congestion if defined as the ratio between the number of 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 [17]–[23].

We assume each aggregating switch deployed in the network provides the ability of aggregating 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 placement. Additionally, 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 [24]. 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 wordcount 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 the paper 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.

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 $V = \mathcal{S} \cup \{d\}$ and $E = E' \cup \{(r, d)\}$ for some $E' \subseteq S^2$ forming a tree over the set of switches $\mathcal{S}$. Let $\omega : E \mapsto \mathbb{R}^+$ be 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 the $d$. 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
Algorithm 1 Reduce \((T, L, U)\)

的要求: 一个树 T, 一个网络负载 L, 一个蓝色节点集 U

确保: 在目的地 d 处生成一个汇总信息

1: 对于每个节点 v ∈ T
2: 而 not received 所有消息从所有子节点 do
3: 处理传入的消息 (按开关类型: B, R)
4: 如果需要发送消息到 p(v) (按开关类型: B, R)

允许在 T 中。我们将通常指代 U ⊆ Λ 为 T 中蓝色节点的集合，并要求 |U| ≤ k.

给定一个有向图网络 \(T = (V, E, \omega)\) 与一个网络加载 \(L : S \mapsto \mathbb{N}\), 一个蓝色节点集 U ⊆ Λ, 我们考虑一个简单的减少操作在 T 上，具体见算法 [Ⅲ]。每个开关在树过程中接收所有消息的顺序，所有其他开关（即，不在 U）是非聚合开关。该操作结束时，destination 接收的总共（可能聚合的）信息来自所有节点，这些节点有明确的正负载。

对于每个边 \(e = (s, p(s)) \in E\), 我们定义 link 负载，\(\text{ms}_e (T, L, U)\)，作为传递通过 Reduce 操作的结点 L, 以及 U。我们进一步定义 link 负载 \(\psi_e (T, L, U) = \text{ms}_e (T, L, U) - \tau(e)\) 值，来指代

\[
\psi(T, L, U) = \max_{e \in T} \{\psi_e (T, L, U)\}
\]

作为网络负载。我们的工作考虑了 Congestion-minimization with Bounded In-network Computing (C-BIC) 问题，旨在最小化网络负载，如形式所示。

定义 1 (C-BIC)。给定一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]

在尝试解决 C-BIC 问题时，一种可能使用暴力方法，即枚举所有可能的子集 Λ 大小为 k。这可能会工作得很好，因为一个常数大小 k，但它变得难以用任意大小 k 为常数。下一段我们将描述并讨论我们提供的解决方案，SMC, 到 C-BIC 问题。

### III. Motivating Example

我们先考虑一个动机示例，强调事实，即简单，但合理，的方法可能会由于找到最优解而失败 C-BIC 问题。具体来说，我们考虑以下三种分配策略来确定蓝色节点集：(i) Top 策略，选择 k 的蓝色节点集作为最接近根的蓝色节点。这种方法旨在减少传递到最顶部部分网络的消息数量，其中网络负载最大。这种方法在启发式方法中有效，可以考虑超参数设置，逐级扫描所有可能的子集，这种方法可能有积极的效应在整体网络负载。这种方法是为完全二叉树定义的，其他不在线开关（即，不在 U）是非聚合开关。该操作结束时，destination 接收的总（可能聚合的）信息来自所有节点，这些节点有明确的正负载。

### III. Motivating Example

我们考虑一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]

在尝试解决 C-BIC 问题时，一种可能使用暴力方法，即枚举所有可能的子集 Λ 大小为 k。这可能会工作得很好，因为一个常数大小 k，但它变得难以用任意大小 k 为常数。下一段我们将描述并讨论我们提供的解决方案，SMC, 到 C-BIC 问题。

### III. Motivating Example

我们考虑一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]

在尝试解决 C-BIC 问题时，一种可能使用暴力方法，即枚举所有可能的子集 Λ 大小为 k。这可能会工作得很好，因为一个常数大小 k，但它变得难以用任意大小 k 为常数。下一段我们将描述并讨论我们提供的解决方案，SMC, 到 C-BIC 问题。

### III. Motivating Example

我们考虑一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]

在尝试解决 C-BIC 问题时，一种可能使用暴力方法，即枚举所有可能的子集 Λ 大小为 k。这可能会工作得很好，因为一个常数大小 k，但它变得难以用任意大小 k 为常数。下一段我们将描述并讨论我们提供的解决方案，SMC, 到 C-BIC 问题。

### III. Motivating Example

我们考虑一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]

在尝试解决 C-BIC 问题时，一种可能使用暴力方法，即枚举所有可能的子集 Λ 大小为 k。这可能会工作得很好，因为一个常数大小 k，但它变得难以用任意大小 k 为常数。下一段我们将描述并讨论我们提供的解决方案，SMC, 到 C-BIC 问题。

### III. Motivating Example

我们考虑一个有向图网络 \(T = (V, E, \omega)\)，一个网络加载 \(L : S \mapsto \mathbb{N}\)，一个可用开关集 Λ，以及一个预算 k，Congestion-minimization with Bounded In-network Computing (C-BIC) 问题是在 T 上寻找一个由最小化网络负载 \(\psi(T, L, U)\) 的集合 U ⊆ Λ 为大小至少 k 的子集，使网络负载最小化。

\[
\text{C-BIC}(T, L, k) = \arg \min_{U \subseteq \Lambda} \psi(T, L, U) \quad \text{(2)}
\]
Algorithm 2 SMC(T, L, Λ, k)

Require: A tree T, load L, availability Λ, k blue nodes
1. \( X = \frac{1}{\min_{e \in E} L(v)} \sum_{v} L(v) \) \hspace{1em} \triangleright \text{init. congestion upper bound}
2. \( S = \frac{1}{\max_{e \in E} L(v)} \)
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 \( \beta^* \)
4. run SMC-Color\( (k) \) using \( \beta^* \)

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 \( k = 4 \).

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 the paper is the following theorem.

Theorem 1. Given a weighted tree network \( T \) with rates \( \omega \), a load \( L \), availability \( \Lambda \), and a bound \( k \) on the number of allowed blue switches, algorithm SMC solves the C-BIC problem in time \( O(n \cdot k^3 \cdot \log \left( \frac{\max \omega}{\min \omega} \sum_{v} L(v) \right)) \).

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 node \( i \leq k \) 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 \( \beta_i \)), 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 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 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 3, 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 \((v, p(v))\). Further let \( T_v = T^i_{C(v)} \) denote the subtree rooted at \( v \) (containing all subtrees of all children of \( v \)), 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 \( U \subseteq \Lambda, |U| = k \) that satisfies the congestion constraint \( \psi(T_v, L, U) \leq X \) (or infinity if no such set exists).

Given some value \( X \), algorithm SMC-Gather uses the following concepts for non-leaf nodes: (i) variables \( \beta^m_v(i, \text{color}) \) that should represent the minimum number of messages traversing link \((v, p(v))\) in the tree \( T^v_m \), 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 \( X \), and (ii) variables \( \beta_v(i) = \min \{ \beta^m_v(i, B), \beta^m_v(i, R) \} \). In the following lemma we prove that the semantics we attribute to \( \beta^m_v(i, \text{color}) \) are indeed correct, and that SMC-Gather indeed computes \( \beta(T_v, L, i, X) \) correctly.

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 \( \beta^m_v \) as computed by SMC-Gather \((T_v, L, k, X) \) satisfies

\[
\beta^m_v(i, R) = \beta(T^v_m, L, i, X) \text{ where } v \text{ is colored } R
\]

and

\[
\beta^m_v(i, B) = \beta(T^v_m, L, i, X) \text{ where } v \text{ is colored } B,
\]

where

\[
\beta_v(i, B) = \begin{cases} 1, & \text{if } \beta_{c_1}(i - 1) < \infty \\ \infty, & \text{otherwise} \end{cases}
\]
\begin{align*}
\beta_v^1(i, R) &= \begin{cases} 
\beta_c(i) + L(v), & \text{if } (\beta_c(i) + L(v)) \cdot \tau(v) \leq X \\
\infty, & \text{otherwise}
\end{cases} \\
\text{and for } m > 1 & \IEEEyesnumber
\beta_v^m(i, B) = \begin{cases} 
1, & \text{if } \min_{0 \leq j < i} (\beta_v^{m-1}(i - j, B) + \beta_c(j)) < \infty \\
\infty, & \text{otherwise}
\end{cases} \\
\beta_v^m(i, R) &= \min_{0 \leq j < i} (\beta_v^{m-1}(i - j, R) + \beta_c(j)), \quad \text{if Eq. \ref{eq:beta_v^m(i,R)}} \text{ holds} \\
&\quad \text{otherwise}
\end{align*}

where
\begin{equation}
\min_{0 \leq j < i} (\beta_v^{m-1}(i - j, R) + \beta_c(j)) \cdot \tau(v) \leq X. \label{eq:beta_v^m(i,R)}
\end{equation}

Overall,
\begin{equation}
\beta_v(i) = \min (\beta_v^C(v)(i, B), \beta_v^C(v)(i, R)) = \beta(T_v, L, i, X) \label{eq:beta_v(i)}
\end{equation}

\textbf{Proof:} The proof is by double induction on the height of \( T_v \) and the number of children \( m \) for which \( \beta_v^m(i, R) \) and \( \beta_v^m(i, B) \) have been computed correctly.

For the base case, we observe that for any leaf node \( v \) the following holds: (i) For \( i > 0 \), \( v \) can be colored blue, and this minimizes the load on link \( (v, p(v)) \) implying that \( \beta_v(i) = 1 \).

(ii) For \( i = 0 \), \( v \) cannot be colored blue, the load on the outgoing link is \( L(v) \cdot \tau(v, p(v)) \), implying that:
\begin{equation}
\beta_v(0) = \begin{cases} 
L(v), & \text{if } L(v) \cdot \tau(v, p(v)) \leq X \\
\infty, & \text{otherwise}
\end{cases} \label{eq:beta_v(0)}
\end{equation}

It follows that for every leaf node \( v \),
\begin{equation}
\beta_v(i) = \beta(T_v, L, i, X), \label{eq:beta_v(i)_leaf}
\end{equation}
which proves the base case.

Let \( v \) be a non-leaf and assume that \( \beta_v(i) \) has been computed correctly for all nodes \( v' \) at height less then node \( v \)'s height, and for all \( i \). In particular, this is true for every child \( c_m \) of node \( v \), \( m = 1, \ldots, C(v) \). Consider first \( m = 1 \), where we have two cases:

(i) Assume \( v \) is blue and \( i > 0 \). By the induction hypothesis, if \( \beta_c(i-1) < \infty \), i.e. satisfies the congestion constraint, then \( \beta_v^1(i, B) = 1 \). Otherwise, again by the induction hypothesis, if \( \beta_c(i - 1) = \infty \) both \( \beta_c(i - 1) \) and \( \beta_v^1(i, B) \) don't satisfy the congestion constraint. Eq. \ref{eq:beta_v^1(i,B)} follows.

(ii) Assume \( v \) is red. By the induction hypothesis, if \( \beta_c(i) < \infty \), then there is a solution that satisfies the congestion constraint using blue nodes in \( T_{c_i} \).

If \( (\beta_c(i) + L(v)) \cdot \tau(v) \leq X \) then the congestion constraint is also satisfied on \( (v, p(v)) \) in \( T_v \), implying that \( \beta_v^1(i, R) = \beta_c(i) + L(v) \). Otherwise the congestion constraint is violated either in Eq. \ref{eq:beta_v^1(i,R)} follows.

Now consider \( m > 1 \), where we assume that for all \( m' < m \), \( \beta_v^{m'}(i, R) \) and \( \beta_v^{m'}(i, B) \) have been computed correctly, and in particular, satisfy Eq. \ref{eq:beta_v^m(i,R)} and \ref{eq:beta_v^m(i,B)}.

We distinguish between two cases:

(i) Assume \( v \) is blue and \( i > 0 \). If there exists a \( j \) such that, \( \beta_c(j) < \infty \) and \( \beta_v^{m-1}(i - 1 - j) < \infty \), by the induction hypothesis, this means that the congestion constraint is satisfied both in \( T_{c_m} \) with \( j \) blue nodes and \( T_{v}^{m-1} \) with \( i - 1 - j \) blue nodes.

This implies that there exists a partition of \( i \) that satisfies the congestion constraint, and \( \beta_v^m(i, B) = 1 \). Otherwise, the congestion constraint cannot be satisfied by any partition, in which case \( \beta_v^m(i, B) = \infty \). Eq. \ref{eq:beta_v^m(i,B)} and \ref{eq:beta_v^m(i,R)} thus follow.

(ii) Assume \( v \) is red, and that there exists a \( j \) such that, \( \beta_c(j) < \infty \) and \( \beta_v^{m-1}(i - j) < \infty \). For each such \( j \), by the induction hypothesis, the congestion constraint is satisfied by this partition both in \( T_{c_m} \) with \( j \) blue nodes and \( T_{v}^{m-1} \) with \( i - j \) blue nodes. If, additionally, \( (\beta_v^{m-1}(i - j) + \beta_c(j)) \cdot \tau(v) \leq X \) then the congestion constraint is also satisfied on \( (v, p(v)) \) by this partition. Taking the minimum over all such partitions ensures that the number of messages traversing \( (v, p(v)) \) is minimized, while satisfying the congestion constraint in \( T_{v}^{m-1} \). To see this, assume by contradiction that there exists a way to have less messages traverse \( (v, p(v)) \) while satisfying the congestion constraint. In particular, such a solution places some \( j \) blue nodes in \( T_{c_m} \), and \( (i - j) \) blue nodes in \( T_{v}^{m-1} \). Since the additional load on \( (v, p(v)) \) due to \( L(v) \) is independent of any such placement, it follows that having a smaller number of messages traverse \( (v, p(v)) \) implies that either the number of messages traversing \( c_m, v \) is smaller than \( \beta_c(j) \) or smaller than \( \beta(T_{v}^{m-1}, L, i, X) \), contradicting the correctness of \( \beta_v^m(j) \) or \( \beta_v^m(i-j, R) \), respectively, which follows from the induction hypothesis. This shows the validity of Eq. \ref{eq:beta_v^m(i,R)} and \ref{eq:beta_v^m(i,B)} which completes the proof.

\section*{In the second phase of SMC, SMC-Color essentially traces back the allocation of blue nodes along the optimal path in the dynamic programming performed by SMC-Gather. To show that SMC-Color indeed produces an optimal solution to the C-BIC problem we make use of the following lemma.}

\textbf{Lemma 3.} Assume \( \beta \) is the output of SMC-Gather for the network congestion upper bound \( X \), such that \( \psi(r) \) is finite. Then, SMC-Color colors blue a set \( U \), such that \( |U| \leq k \), and \( \psi(T, L, U) \leq X \).

\textbf{Proof:} In what follows, we say a node \( v \) is correctly assigned if: (i) it is colored so as to satisfy with the congestion constraint of the system, (ii) it is allotted the number of blue nodes for \( T_v \) so as to satisfy with the congestion constraint of the system. We prove by induction on the order of handling nodes by SMC-Color that if node \( v \) is correctly assigned then each of its children \( c_m \), \( m = 1, \ldots, C(v) \) is correctly assigned.

\footnote{When \( i = 0 \) then \( \beta_v^m(i, B) = \infty \).}
Algorithm 3 SMC-Gather($T, L, \Lambda, k, X$) at node $v$

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

Ensure: Correct potential functions, $\beta_v$, at each node $v$

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

Algorithm 4 SMC-Color($k$) at node $v$

Require: $\beta$

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 $p(v)$
4: if $v$ is a leaf node and $i > 0$ then
5: color $v$ blue and return
6: if $\beta_v(i, B) < \infty$ then
7: color $v$ blue
8: for $m = C(v), \ldots, 2$ do
9: $j = \text{mSplit}(i, \beta_v^{m-1}, \beta_{c_m}, \text{color of } v)$
10: send $j$ to $c_m$
11: $i = i - j$
12: if $v$ is blue then
13: send $i - 1$ to $c_1$
14: else
15: send $i$ to $c_1$
16: return

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

For the base case, consider node $d$, which should have $k$ blue nodes in its subtree, its color is trivially not blue (since $d$ is a server). So $d$ is correctly assigned. $d$ has a single child, $r$, and by line 6 of SMC-Color, along with Eq. (3) and Eq. (4), $r$ is colored correctly, since by line 27 of SMC-Gather, its color is the one satisfying congestion constraint. Clearly by line 2 in SMC-Color $r$ is correctly.

Assume the claim holds for all nodes handled before node $v$, and consider node $v$ which is correctly assigned. First, since $v$ is correctly colored, by induction on the number of children of $v$ from $C(v)$ to 1, it is easy to show that each child $c$ is assigned the correct number of blue nodes to be distributed in its subtree $T_c$. This follows from the fact that the mSplit procedure in lines 17-21 of SMC-Color essentially extract the value $j$ obtaining the minimum considered also by the mCost procedure in lines 29-34 in SMC-Gather. Since each child $c$ is assigned correctly the correct number of blue nodes, by Lemma 2 and line 27 of SMC-Gather, $c$ will be also colored correctly.

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

**Lemma 4.** If $\beta(T, L, k, X)$ can be computed in $\alpha$ time, then C-BIC($T, L, \Lambda, k$) is solved in time $\alpha \cdot \log(\sum_v L(v) \cdot \frac{1}{\omega_{\text{min}}})$.

**Proof:** The proof follows directly from applying a binary search over the upper bound $X$ on the network congestion, where the maximum such value is no larger than $\frac{1}{\omega_{\text{min}}} \cdot \sum_v L(v)$, and the granularity is at least $\frac{1}{\omega_{\text{min}}}$. Where in each iteration we check whether or not $\beta(T, L, k, X)$ is finite, using Algorithm SMC-Gather.

We can now prove Theorem 1.

**Proof of Theorem 1** The correctness of the algorithm follows from Lemmas 2. 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 $\text{SMC-Gather } O\left(\log(\sum_v L(v) \cdot \frac{\omega_{\text{max}}}{\omega_{\text{min}}})\right)$ times, resulting in a total running time for solving the C-BIC problem of $O\left(n \cdot k^2 \cdot \log\left(\frac{\omega_{\text{max}}}{\omega_{\text{min}}} \sum_v L(v)\right)\right)$. 

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 the MapReduce paradigm, 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 serves as top-of-the-rack (ToR) switches connected to servers (workers) that generate load. The remaining network switches model the higher levels of a datacenter 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. 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 distinct workload distributions, where the number $k$ of blue nodes we are allowed to use takes values in $k = 1, 2, 4, 8, 16, 32$. The figure also shows the network congestion for the all-blue and 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 a significantly 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 perform 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 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 $A_t = \{ s \mid a_t(s) > 0 \}$.

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. 3a), linearly increasing (in Fig. 3b), and exponentially increasing (in Fig. 3c). The figure shows the normalized network congestion, where normalized to the congestion obtained by the all-red solution. Namely, if the performance of an algorithm is normalized to the congestion obtained by the all-red solution.

Any 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 section 5 we normalized 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 α(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 [3]. We refer to this use case as the parameter server (PS) use case.

We evaluate the performance of SMC for WC, and PS, using the constant rates regime, which better highlights the differences in the performance, and using the uniform distribution which is more challenging for reducing congestion.

Fig. 6 shows the results of our evaluation, where the congestion attained by SMC is normalized to that of the all-red scenario. This figure highlights 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 impacts on the perceived 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. 1, 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]. Various approaches have been suggested to modify ML methodologies in order to improve upon the network induced performance of distributed ML [4], [35], [36]. Additional network- and system-level adaptations have been suggested to improve upon ML performance of such systems [38], [39]. 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 work 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.

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.
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.