Tree Gradient Coding

Amirhossein Reisizadeh* UC Santa Barbara Email: reisizadeh@ucsb.edu
Saurav Prakash* USC Email: sauravpr@usc.edu
Ramtin Pedarsani UC Santa Barbara Email: ramtin@ece.ucsb.edu
Amir Salman Avestimehr USC Email: avestimehr@ee.usc.edu

***Abstract*—Scaling up distributed machine learning systems face two major bottlenecks – delays due to stragglers and limited communication bandwidth. Recently, a number of coding theoretic strategies have been proposed for mitigating these bottlenecks. In particular, the Gradient Coding (GC) scheme was proposed to speed up distributed gradient descent algorithm in a synchronous master-worker setting by providing robustness to stragglers. A major drawback of the master-worker architecture for distributed learning is however, the bandwidth contention at the master, which can significantly deteriorate the performance as the cluster size increases. In this paper, we propose a new framework named Tree Gradient Coding (TGC) for distributed gradient aggregation, which parallelizes communication over a tree topology while providing straggler robustness. As our main contribution, we characterize the minimum computation load for TGC for a given tree topology and straggler resiliency, and design a tree gradient coding algorithm that achieves this optimal computation load. Furthermore, we provide results from experiments over Amazon EC2, where TGC speeds up the training time by up to $18.8\times$ in comparison to GC.**

I. Introduction

Modern machine learning problems comprise of large-scale models trained over massive data sets. With increasing availability of commodity hardware for computing, there has been a growing interest towards developing scalable solutions for carrying out distributed data processing in large-scale computing clusters. These clusters, however, suffer from two major bottlenecks – (1) slow computing nodes or stragglers, (2) communication bandwidth due to large data transfers.

We focus on the commonly used synchronous gradient descent (GD) paradigm, where parallelization can be achieved by scaling out gradient computations to workers, and the model is updated synchronously among all the workers. In the common master-worker setting, the workers compute partial gradients on their local data batches and upload their results to the master, which aggregates the full gradient and updates the model. This, however, is bottlenecked by the slowest workers, which makes the master-worker setup prone to stragglers [1].

Recently, coding theoretic approaches have been proposed for mitigating stragglers as well as reducing communication load in distributed computation [2], [3]. Since then, the use of coding theory for speeding up distributed computation, also known as "coded computation", has been considered in a number of works, e.g. [4]–[17]. The most related work to this paper is Gradient Coding (GC) [18], where the authors proposed a straggler mitigation scheme for gradient aggregation in master-worker setup. Particularly, for a given data set, N workers and a straggler toleration parameter S, data is redundantly placed among the workers such that the master needs to collect local gradient computations from only $N-S$ workers to aggregate the full gradient, i.e. the scheme is robust to S stragglers.

There has, however, been a recent shift from the master-worker setting for distributed machine learning to other architectures and algorithms, such as AllReduce over rings and trees [19]–[21] mainly for the following reason. As the cluster size increases, the master-worker setup suffers from bandwidth bottleneck, since all the workers need to communicate with the master simultaneously. However, the alternate approaches allow for multiple simultaneous communications to take place, thus increasing the bandwidth utilization.

In this paper, we propose the use of a tree communication topology, and focus on the optimal data allocation and coding design for such topology that achieves a target straggler resiliency. As our main contribution, we propose a Tree Gradient Coding (TGC) framework for distributed gradient aggregation that is both bandwidth efficient and straggler tolerant. Specifically, the compute nodes are arranged in a logical tree topology for communication, with master at the root, which recovers the full gradient from the computation results of the children. In this topology, each compute node only interacts with its parent and children nodes. The tree topology thus alleviates the communication bottleneck, as each node only communicates with neighbours in the tree. During execution, each worker uploads a coded partial gradient to its parent, by combining the partial gradients obtained from its local data batches with the coded partial gradients received from the children.

For a given tree topology we characterize the minimum computation load for TGC, as well as provide an achievable task allocation and coding strategy. As we demonstrate, TGC achieves significantly smaller computation load compared to GC. Moreover, we conduct experiments over Amazon EC2 cluster, where we demonstrate up to $18.8\times$ speedups for TGC compared to GC.

Notation. For a natural number m, $[m]$ represents the set $\{1, 2, \ldots, m\}$. The all ones matrix with dimension $u \times v$ is denoted by $\mathbf{1}_{u\times v}$.

∗ Authors have equal contribution.

This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053, ARO award W911NF1810400, NSF grants CCF-1703575 and CCF-1755808, ONR Award No. N00014-16-1-2189, and CCF-1763673. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

II. Background

In many machine learning algorithms, a model $\theta \in \mathbb{R}^{p\times 1}$ is estimated by minimizing an empirical loss function over the training data set. Specifically, for the training data set $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^{p+1}; j = 1, \cdots, d\}$, the goal is to find the optimal θ that minimizes the following loss function:

$$\sum_{\mathbf{x}\in\mathcal{D}} \ell(\theta; \mathbf{x}) + \lambda R(\theta), \tag{1}$$

where $\ell(\cdot)$ is the underlying loss function, $R(\cdot)$ denotes the regularization function, and λ is the regularization parameter.

In GD algorithm, the optimal model is estimated via an iterative procedure that makes the following update at each iteration t: $\theta^{(t+1)} = \theta^{(t)} - \mu(\mathbf{g} + \lambda\nabla R(\theta^{(t)}))$. Here $\mathbf{g} = \sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\theta^{(t)}; \mathbf{x})$ is the gradient of the empirical loss function in (1) on model at iteration t and μ denotes the step size. We denote the partial gradient with respect to a subset $\mathcal{P} \subseteq \mathcal{D}$ by $\mathbf{g}_{\mathcal{P}} = \sum_{\mathbf{x}\in\mathcal{P}} \nabla\ell(\theta^{(t)}; \mathbf{x})$. We also denote by $\mathbf{g}$ the full gradient, i.e. $\mathbf{g} = \mathbf{g}_{\mathcal{D}}$.

At scale, the computations cannot be carried out at a single computing node; thus, the data set needs to be distributed across multiple nodes. Moreover, computation redundancy can be introduced in order to provide robustness to stragglers in the distributed setting. Recently, Gradient Coding (GC) was proposed in [18] to robustify distributed gradient aggregation against stragglers in a master-worker setup. Next we illustrate the main ideas behind GC.

Example 1 (Illustration of GC)**.** Consider the master-worker setup with $N = 3$ workers in Fig. 1. The goal is to make gradient aggregation robust to $S = 1$ straggler. Let $\mathcal{D}$ be partitioned into three batches $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$. As depicted in Fig. 1, each worker computes partial gradients corresponding to two data batches, and then communicates a linear combination of them to the master. The master can recover the full gradient $\mathbf{g}_{\mathcal{D}} = \mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2} + \mathbf{g}_{\mathcal{D}_3}$ from the results of *any* $N - S = 2$ workers. For instance if node 2 straggles, master recovers the full gradient by adding the results of workers 1 and 3.

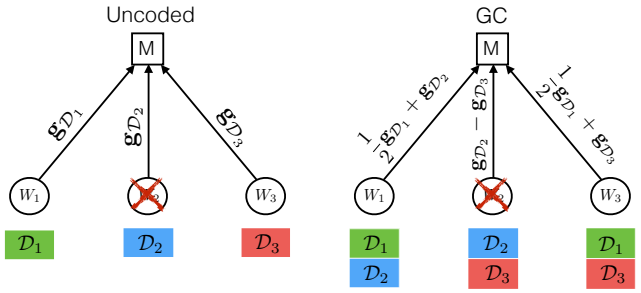


Fig. 1: GC handles stragglers by redundant computation allocation.

More generally, for a given data set $\mathcal{D}$ with k partitions $\{\mathcal{D}_1, \cdots, \mathcal{D}_k\}$ and maximum number of stragglers S, the goal of GC is to design gradient encoding and decoding matrices $\mathbf{B} \in \mathbb{R}^{N\times k}$ and $\mathbf{A} \in \mathbb{R}^{f\times N}$, such that the following holds:

$$\mathbf{AB} = \mathbf{1}_{f\times k}. \tag{2}$$

Here, $f = \binom{N}{S}$ denotes the number of combinations of non-straggling workers. For the i^{th} row $\mathbf{b}_i$ of $\mathbf{B}$, $\|\mathbf{b}_i\|_0$ denotes the number of partitions assigned to i^{th} worker, $\text{supp}(\mathbf{b}_i)$ denotes the data partitions that are allocated to i^{th} worker, while the values in b_i denote the coding coefficients for combining the corresponding partial gradients. After receiving coded partial gradients from any $N - S$ workers, the master finds the corresponding row $\mathbf{a}_i$ in $\mathbf{A}$, and then linearly combines the results to obtain the full gradient $\mathbf{g}_{\mathcal{D}}$ as follows:

$$\mathbf{a}_i\mathbf{B}\bar{\mathbf{g}} = \sum_{u\in\text{supp}(\mathbf{a}_i)} \mathbf{a}_i(u)(\mathbf{b}_u\bar{\mathbf{g}}) = \mathbf{1}_{1\times k}\bar{\mathbf{g}} = \mathbf{g}_{\mathcal{D}}. \tag{3}$$

where $\bar{\mathbf{g}} = [\mathbf{g}_{\mathcal{D}_1}; \ldots; \mathbf{g}_{\mathcal{D}_k}]$. Optimal GC schemes for general N and S are described in [18], which achieve optimal computation load $r_{\mathsf{GC}} = \frac{S+1}{N}$. Thus, each worker is allocated $\|\mathbf{b}_i\|_0 = \frac{S+1}{N}k$ partitions for processing. As demonstrated in Example 1, the following matrices solve the GC problem for $N = 3$ and $S = 1$.

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & -1 & 0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 1/2 & 1 & 0 \\ 0 & 1 & -1 \\ 1/2 & 0 & 1 \end{pmatrix}. \tag{4}$$

III. Motivating Example and Main Result

We consider a general tree topology for distributed gradient aggregation and describe a Tree Gradient Coding (TGC) problem which generalizes GC. As we demonstrate via next example, TGC achieves a much lower computation load compared to GC for the same *fraction* of straggling children per parent node.

Example 2 (TGC via a motivating example)**.** Consider $N = 12$ nodes on a tree topology depicted in Figure 2(a). The goal is to develop a data allocation and communication design such that the gradient aggregation at master has a resiliency of $1/3$ per parent, i.e. the master can recover the full gradient while *any* of the 3 children per *any* parent straggle. Note that for a classical master-worker setup, a resiliency of $1/3$ implies $S = 4$ and $r_{\mathsf{GC}} = 5/12$. As we illustrate here, using a tree topology can significantly reduce the computation load.

Consider the data set $\mathcal{D}$ of size d partitioned to $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$. To recover $\mathbf{g}_{\mathcal{D}} = \mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2} + \mathbf{g}_{\mathcal{D}_3}$ at the master from any two of its three children, it suffices to ensure that the partial gradients uploaded from the three nodes in the first layer to the master are $\mathbf{m}_{(1,1)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}$, $\mathbf{m}_{(1,2)} = \mathbf{g}_{\mathcal{D}_2} - \mathbf{g}_{\mathcal{D}_3}$, and $\mathbf{m}_{(1,3)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_3}$.

Next, we explain how TGC ensures such recovery guarantees. Consider the sub-tree consisting of the parent node $(1, 1)$ and its three children denoted by $T(1, 1)$. To construct $\mathbf{m}_{(1,1)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}$ from the results of its local partial gradient computation and those of any two of its children, we assign $\mathcal{D}^{T(1,1)} = \frac{1}{2}\mathcal{D}_1 \cup \mathcal{D}_2$ to sub-tree $T(1, 1)$[1]. Node $(1, 1)$ then picks $r_{\mathsf{TGC}}d = \frac{4}{15}d$ data points from $\mathcal{D}^{T(1,1)}$ as $\mathcal{D}(1, 1)$. To do so, $\mathcal{D}^{T(1,1)}$ is partitioned to 5 sub-sets as $\mathcal{D}^{T(1,1)} = \mathcal{D}_1^{T(1,1)} \cup \cdots \cup \mathcal{D}_5^{T(1,1)}$ and node $(1, 1)$ picks the first two sub-sets, i.e. $\mathcal{D}(1, 1) = \mathcal{D}_1^{T(1,1)} \cup \mathcal{D}_2^{T(1,1)}$ and the rest $\mathcal{D}_{T(1,1)} = \mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_4^{T(1,1)} \cup \mathcal{D}_5^{T(1,1)}$ is passed to layer 2. Note

[1]To ease the notations, we associate a real scalar b to all the data points in a generic data set $\mathcal{D}$, denoting it by $b\mathcal{D}$, and define the gradient over $b\mathcal{D}$ as $\mathbf{g}_{b\mathcal{D}} = b\mathbf{g}_{\mathcal{D}} = b\sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\theta^{(t)}; \mathbf{x})$.

that data points in $\mathcal{D}(1,1)$ carry on the linear combination coefficients associated with $\mathcal{D}^{T(1,1)} = \frac{1}{2}\mathcal{D}_1 \cup \mathcal{D}_2$. Figure 2(b) demonstrates each node in sub-tree $T(1,1)$ with its allocated data set along with the encoding coefficients. Moving to layer 2, $\mathcal{D}_{T(1,1)}$ is partitioned to 3 subsets and according to $\mathbf{B}$ in (4), the allocations to nodes $(2,1)$, $(2,2)$ and $(2,3)$ are as follows:

$$\mathcal{D}(2,1) = \frac{1}{2}\mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_4^{T(1,1)},$$
$$\mathcal{D}(2,2) = \mathcal{D}_4^{T(1,1)} \cup (-1)\mathcal{D}_5^{T(1,1)},$$
$$\mathcal{D}(2,3) = \frac{1}{2}\mathcal{D}_3^{T(1,1)} \cup \mathcal{D}_5^{T(1,1)}.$$

Similarly for other sub-trees, each node now is allocated with a data set for which each data point is associated with a scalar. For instance, node $(2,1)$ uploads $\mathbf{m}_{(2,1)} = \mathbf{g}_{\mathcal{D}(2,1)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_3^{T(1,1)}} + \mathbf{g}_{\mathcal{D}_4^{T(1,1)}}$ to its parent $(1,1)$. Node $(1,1)$ can recover from any 2 surviving children, e.g. from $(2,1)$ and $(2,1)$ and using the first row in $\mathbf{A}$, it uploads

$$\mathbf{m}_{(1,1)} = [2,-1,0][\mathbf{m}_{(2,1)};\mathbf{m}_{(2,2)};\mathbf{m}_{(2,3)}] + \mathbf{g}_{\mathcal{D}(1,1)}$$
$$= 2\mathbf{m}_{(2,1)} - \mathbf{m}_{(2,2)} + \mathbf{g}_{\mathcal{D}(1,1)} = \frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}$$

to the master. Similarly for other nodes, the master can recover the full gradient from any two children, e.g. using the second row of decoding matrix $\mathbf{A}$ and surviving children $(1,1)$ and $(1,3)$:

$$[1,0,1][\mathbf{m}_{(1,1)};\mathbf{m}_{(1,2)};\mathbf{m}_{(1,3)}] = \mathbf{m}_{(1,1)} + \mathbf{m}_{(1,3)}$$
$$= \left(\frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_2}\right) + \left(\frac{1}{2}\mathbf{g}_{\mathcal{D}_1} + \mathbf{g}_{\mathcal{D}_3}\right) = \mathbf{g}_{\mathcal{D}}.$$

More generally, we consider an (n,L)–regular tree topology for communication defined as follows:

In an (n,L)–regular tree topology T, the computing nodes are arranged in a multi-level tree architecture for communication, where the master is at the root node and the N workers are arranged in L layers of the tree with each *parent* node having n *children* nodes, i.e. $N = n + \cdots + n^L$. We identify each node with a pair (l,j), where $l \in [L]$ denotes the corresponding layer, and $j \in [n^l]$ denotes the node's index in that layer. We also let $(0,1)$ denote the root node. Furthermore, $T(l,j)$ denotes the sub-tree with the root node (l,j).

Definition 1. For a given (n,L)–regular tree topology, a distributed gradient aggregation strategy is $\alpha-$resilient for a given $0 \leq \alpha < 1$ if the master can recover the full gradient while any $s = \alpha n$ children per any parent may straggle.

Tree Gradient Coding (TGC) Problem: For a given (n,L)–regular tree topology, data set $\mathcal{D}$ and resiliency α, the TGC problem is to design $\alpha-$resilient gradient aggregation strategies, while uniformly loading each node with r fraction of the total data set $\mathcal{D}$ (named as *computation load*). In the following, we formally formulate the TGC problem:

- Starting from the master node, let $\mathcal{D}^{T(1,j)}$ denote the collective data set assigned to sub-tree $T(1,j)$ according to encoding vectors $\mathbf{b}_j^{\mathsf{M}} \in \mathbb{R}^{1\times k}$ for $j \in [n]$. That is, the total data set $\mathcal{D}$ is partitioned to $\{\mathcal{D}_1, \cdots, \mathcal{D}_k\}$ and $\mathcal{D}^{T(1,j)}$ collects the partitions with non-zero corresponding elements in $\mathbf{b}_j^{\mathsf{M}}$, i.e. for all $j \in [n]$

$$\mathcal{D}^{T(1,j)} = \cup_{\kappa=1}^{k} \mathbb{1}\{\mathbf{b}_j^{\mathsf{M}}(\kappa) \neq 0\}\mathcal{D}_\kappa. \quad (5)$$

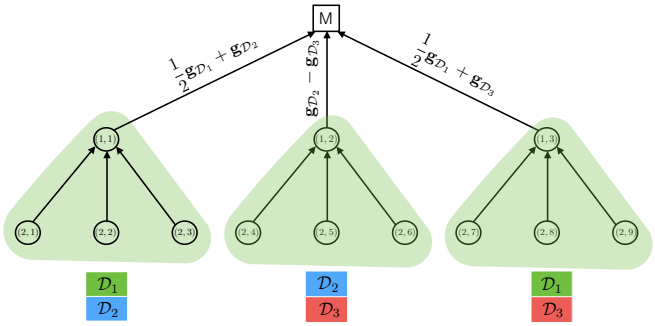
To recover the full gradient at the master from any $n-s$ of its sub-trees, the design vectors $\{\mathbf{b}_1^{\mathsf{M}}, \cdots, \mathbf{b}_n^{\mathsf{M}}\}$ should satisfy the following property for every subset $I \subseteq [n], |I| = n-s$:

$$\mathbf{1}_{1\times k} \in \text{span}\{\mathbf{b}_i^{\mathsf{M}} | i \in I\}. \quad (6)$$

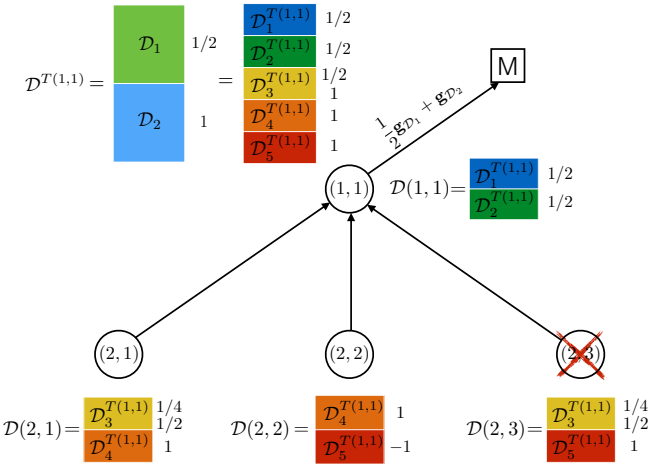
- For layer $l = 1$, consider the node $(1,j)$, $j \in [n]$, with local data set $\mathcal{D}(1,j) \subseteq \mathcal{D}^{T(1,j)}$ corresponding to the local encoding vector $\mathbf{b}_0^{(1,j)}$. To satisfy the computation load constraint, $\mathbf{b}_0^{(1,j)}$ should impose computation load $|\mathcal{D}(1,j)| \leq rd$. Let $\mathcal{D}_{T(1,j)}$ denote the collective data set for the sub-trees of the children of node $(1,j)$, i.e. $T(2,n(j-1)+1), \cdots, T(2,nj)$. Treating $\mathcal{D}_{T(1,j)}$ as a data set over a sub-tree with $(1,j)$ as its master, data sets $\mathcal{D}^{T(2,n(j-1)+i)}$ are determined by the encoding vectors $\mathbf{b}_i^{(1,j)}$, $i \in [n]$. That is, $\mathcal{D}_{T(1,j)}$ is partitioned to $\{\mathcal{D}_{T(1,j),1}, \cdots, \mathcal{D}_{T(1,j),k}\}$ and

$$\mathcal{D}^{T(2,n(j-1)+i)} = \cup_{\kappa=1}^{k} \mathbb{1}\{\mathbf{b}_i^{(1,j)}(\kappa) \neq 0\}\mathcal{D}_{T(1,j),\kappa}. \quad (7)$$

Straggler resiliency condition requires the encoding vectors



(a) Illustration of the partial gradients communicated to the master from the sub-trees attached to it.



(b) Illustration of the task allocation and communication components of TGC for the sub-tree $T(1,1)$.

Fig. 2: The task allocation and communication for TGC.

$\{\mathbf{b}_0^{(1,j)}, \mathbf{b}_1^{(1,j)}, \cdots, \mathbf{b}_n^{(1,j)}\}$ to satisfy:

$$\mathbf{b}_j^{\mathsf{M}} \in \text{span}\{\mathbf{b}_0^{(1,j)}, \mathbf{b}_i^{(1,j)} | i \in I\}, \tag{8}$$

for every subset $I \subseteq [n], |I| = n - s$. This also implies that each parent will recover from its non-straggling children and combine with its local gradient with the proposed decoding vectors determined by (8).

- Similar spanning conditions should be satisfied for the rest of the layers. For the last layer, the following condition must be satisfied for all $j \in [n^{L-1}]$ and $i \in [n]$:

$$\mathbf{b}_0^{(L,n(j-1)+i)} = \mathbf{b}_i^{(L-1,j)}. \tag{9}$$

The goal of TGC problem is therefore to design encoding and decoding vectors satisfying the conditions (6), (8) and (9). These vectors specify both data allocation and communication strategy to aggregate the gradient while being robust.

As our main contribution, we solve the TGC problem by proposing a resilient gradient aggregation scheme (also named as TGC) which consists of data allocation and coding/communication strategy. We characterize the computation load of the proposed scheme and show its optimality. This is formally stated as follows.

Theorem 1. *For a given (n, L)–regular tree topology, resiliency $0 \leq \alpha < 1$ and $s = \alpha n$, the minimum computation load for the TGC problem is*

$$r_{TGC} = \frac{1}{\left(\frac{n}{s+1}\right) + \cdots + \left(\frac{n}{s+1}\right)^L}. \tag{10}$$

Remark 1. Theorem 1 implies a significant improvement over GC in computation load. To attain resiliency α, GC loads each worker with $r_{\mathsf{GC}} \approx \alpha$ fraction of the total data set, while this is greatly reduced to $r_{\mathsf{TGC}} \approx 1/(\frac{1}{\alpha} + \frac{1}{\alpha^2} + \ldots + \frac{1}{\alpha^L}) \approx \alpha^L \ll \alpha$ for sufficiently large L. For example, for $\alpha = 0.5$ and $L = 4$, r_{TGC} is less than 10% of r_{GC}. Moreover, Theorem 1 states that TGC scheme achieves any target resiliency $\alpha = s/n$ over a fix (n, L)–regular tree topology by the minimum computation load on each node (as in (10)).

Remark 2. TGC makes the distributed GD strategy robust to stragglers patterned according to Definition 1, i.e. any $s = \alpha n$ stragglers per any parent node which sums up to a total of (at least) $S = \alpha N$ stragglers – the same as the worst case number of stragglers in GC. In our experiments over Amazon EC2, stragglers are found to be randomly distributed over the tree (and not adversarially picked), which is aligned with the random stragglers pattern considered in this paper.

Remark 3. While Theorem 1 illustrates the theoretical gains of TGC over GC, we also run experiments over Amazon EC2, where we show speedups of up to $18.8\times$ in comparison to GC. These results demonstrate that TGC can provide significant gains in the overall training time of distributed learning algorithms by optimizing the computation load and improving the bandwidth efficiency.

The proof for Theorem 1 can be concluded from Lemma 1 (Achievability) and Lemma 2 (Converse) in Section IV.

IV. Achievability and Converse

In this section, we describe our proposed distributed gradient aggregation strategy TGC and prove its achievability and optimality as stated in Theorem 1. In particular, we first describe our proposed TGC scheme. Then, in Lemma 1 we characterize the computation load induced by the proposed TGC scheme and conclude the section with proving the scheme's optimality in Lemma 2.

Consider a data set $\mathcal{D}$, an (n, L)–regular tree and resiliency $\alpha = s/n$. Our proposed TGC scheme has two main components:

Task Allocation: The goal of the master is to recover the total gradient $\mathbf{g} = \mathbf{g}_{\mathcal{D}}$ from any of its $n - s$ out of n children's uploaded computation results. In order to ensure this, the data set is allocated in a recursive manner as follows:

1) The encoding matrix $\mathbf{B}$ is generated using the GC algorithm for the master-worker setup with n nodes and straggling parameter s (Algorithm 2 in [18]).
2) The data set $\mathcal{D}_{T(0,1)} := \mathcal{D}$ is partitioned to k batches and redundantly assigned to sub-trees $T(1,1), \ldots, T(1,n)$ such that each batch is placed in $s+1$ sub-trees (for $k = n$). We let $\mathcal{D}^{T(l,j)}$ denote the collective data set assigned to sub-tree $T(l,j)$. This task allocation algorithm is formally defined as sub-routine TaskAlloc (Algorithm 1) in [22].
3) At each sub-tree $T(1,j)$, the root node $(1,j)$ picks $r_{\mathsf{TGC}} d$ data points from $\mathcal{D}^{T(1,j)}$ as its local data set $\mathcal{D}(1,j)$ and assigns the remaining data $\mathcal{D}_{T(1,j)} := \mathcal{D}^{T(1,j)} \setminus \mathcal{D}(1,j)$ to its children similar to GC by calling the subroutine TaskAlloc.
4) The previous step is carried out at each node layer-wise till reaching the leaf nodes in layer L where $\mathcal{D}^{T(L,j)} = \mathcal{D}(L,j)$.

After the local computations are assigned to each node, the distributed gradient aggregation is carried out as follows:

Distributed Gradient Aggregation: The following tasks are executed at the workers:

1) The decoding matrix $\mathbf{A}$ corresponding to $\mathbf{B}$ is transmitted to each parent node (Algorithm 1 in [18]).
2) Each worker (l,j) computes the partial gradient $\mathbf{g}_{\mathcal{D}(l,j)}$ from its local data set $\mathcal{D}(l,j)$.
3) Each worker (L,j) in the last layer sends $\mathbf{g}_{\mathcal{D}(L,j)}$ to its parent.
4) Every node (l,j) in layer $l \in \{1, \cdots, L-1\}$ waits for results from any $n-s$ of its children, and then recovers the partial gradient corresponding to $\mathcal{D}^{T(l,j)}$. This is formally explained in Algorithm 2 in [22] as the communication module. Specifically, depending on which children nodes are the stragglers, node (l,j) picks the corresponding row in decoding matrix $\mathbf{A}$ so that together with $\mathbf{g}_{\mathcal{D}(l,j)}$ it can recover the required partial gradient and send it to its parent.
5) Previous step is repeated till reaching the master, where it waits for results from any $n-s$ children and recovers the full gradient $\mathbf{g}_{\mathcal{D}}$.

Next, we derive the computation load of the proposed TGC scheme which corresponds to the achievability part of Theorem 1 (Proof is provided in [22]).

Lemma 1 (Achievability)**.** *For an (n, L)–regular tree topology and resiliency $0 \leq \alpha < 1$, the computation load of the proposed TGC scheme is as follows:*

$$r_{TGC} = \frac{1}{\binom{n}{s+1} + \cdots + \binom{n}{s+1}^{L}}. \tag{11}$$

We conclude this section by proving the optimality of the proposed TGC scheme by providing the converse of Theorem 1 (Proof is provided in [22]).

Lemma 2 (Converse)**.** *For any distributed gradient aggregation strategy over a regular (n, L)–regular tree topology with resiliency $0 \leq \alpha < 1$, the computation load is lower bounded as follow:*

$$r \geq \frac{1}{\binom{n}{s+1} + \cdots + \binom{n}{s+1}^{L}} = r_{TGC}. \tag{12}$$

V. Experiments over Amazon Cluster

In this section, we present the results from our experiments conducted over Amazon EC2, demonstrating the practical gains of the proposed TGC scheme over two baseline approaches: (1) Uncoded distributed gradient aggregation, in which the machines are arranged in the master-worker setup and the data set is uniformly partitioned among the workers, and (2) GC scheme. We consider the problem of training a logistic regression model via gradient descent on GISETTE data set, where the task is to differentiate between the two often confused digits of '4' and '9'. For training, we chose $d = 6552$ samples, while the model size is $p = 5001$. Next we evaluate the convergence speeds of the three schemes by plotting the relative error rate as a function of the wall-clock time for each algorithm:

$$\text{Relative Error Rate} = \frac{\left\|\theta^{(t)} - \theta^{(t-1)}\right\|^2}{\left\|\theta^{(t-1)}\right\|^2},$$

where $\theta^{(t)}$ represents the current model at iteration t.

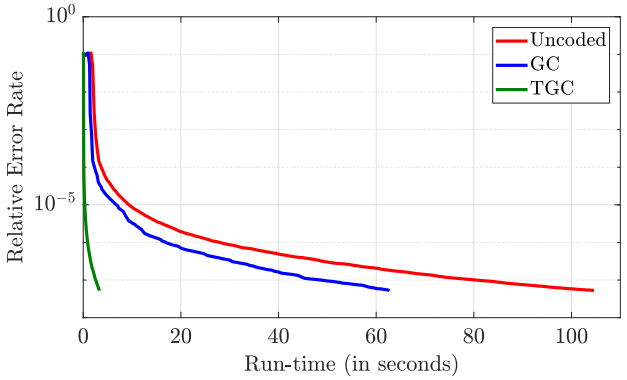


Fig. 3: Convergence plots for relative error rate for the three schemes – Uncoded, GC, TGC. As demonstrated here, TGC achieves a speedup of up to $31.4\times$ and $18.8\times$ over Uncoded and GC approaches.

We used a cluster of a master and $N = 156$ worker instances. For the proposed TGC scheme, we considered a regular tree with $L = 2$ layers and $n = 12$ children per parent. Furthermore, we empirically optimized the straggling parameters for both GC and TGC.

Remark 4. Compared to benchmarks Uncoded and GC, the proposed TGC achieves speedups of $31.4\times$ and $18.8\times$, respectively. Moreover, while GC improves upon Uncoded by resolving the straggler issue, TGC improves over GC via efficient bandwidth utilization as well as computation load reduction by a factor of $11\times$.

References

[1] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones.," *NSDI*, 2013.

[2] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, 2017.

[3] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Transactions on Information Theory*, 2017.

[4] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *NIPS*, 2016.

[5] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," *NIPS*, 2017.

[6] M. A. Attia and R. Tandon, "Information theoretic limits of data shuffling for distributed learning," *GLOBECOM*, 2016.

[7] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, 2019.

[8] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: an alternative trade-off curve," 2017.

[9] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," *ISIT*, 2017.

[10] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *NIPS*, 2017.

[11] J. Kosaian, K. Rashmi, and S. Venkataraman, "Learning a code: Machine learning for approximate non-linear coded computation," *arXiv preprint arXiv:1806.01259*, 2018.

[12] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," *ICML*, 2018.

[13] Q. Yu, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," in *AISTATS*, 2019.

[14] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *ISIT*, 2018.

[15] S. Prakash, A. Reisizadeh, R. Pedarsani, and S. Avestimehr, "Coded computing for distributed graph analytics," *ISIT*, 2018.

[16] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *ACM MobiHoc*, 2019.

[17] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *Communication, Control, and Computing (Allerton), 2017 55th Annual Allerton Conference on*, pp. 1256–1263, IEEE, 2017.

[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," *ICML*, 2017.

[19] T. Hoefler and D. Moor, "Energy, memory, and runtime tradeoffs for implementing collective communication operations," *Supercomputing Frontiers and Innovations*, 2014.

[20] H. Zhao and J. Canny, "Kylix: A sparse allreduce for commodity clusters," *ICPP*, 2014.

[21] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training," *NIPS 2018*, 2018.

[22] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded-reduce: A fast and robust framework for gradient aggregation in distributed learning," *arXiv preprint arXiv:1902.01981*, 2019.