CodedReduce: A Fast and Robust Framework for Gradient Aggregation in Distributed Learning

Amirhossein Reisizadeh, Saurav Prakash, *Graduate Student Member, IEEE*, Ramtin Pedarsani, *Senior Member, IEEE*, and Amir Salman Avestimehr, *Fellow, IEEE*

Abstract*— We focus on the commonly used synchronous Gradient Descent paradigm for large-scale distributed learning, for which there has been a growing interest to develop efficient and robust gradient aggregation strategies that overcome two key system bottlenecks: communication bandwidth and stragglers' delays. In particular, Ring-AllReduce (RAR) design has been proposed to avoid bandwidth bottleneck at any particular node by allowing each worker to only communicate with its neighbors that are arranged in a logical ring. On the other hand, Gradient Coding (GC) has been recently proposed to mitigate stragglers in a master-worker topology by allowing carefully designed redundant allocation of the data set to the workers. We propose a joint communication topology design and data set allocation strategy, named CodedReduce (CR), that combines the best of both RAR and GC. That is, it parallelizes the communications over a tree topology leading to efficient bandwidth utilization, and carefully designs a redundant data set allocation and coding strategy at the nodes to make the proposed gradient aggregation scheme robust to stragglers. In particular, we quantify the communication parallelization gain and resiliency of the proposed CR scheme, and prove its optimality when the communication topology is a regular tree. Moreover, we characterize the expected run-time of CR and show order-wise speedups compared to the benchmark schemes. Finally, we empirically evaluate the performance of our proposed CR design over Amazon EC2 and demonstrate that it achieves speedups of up to $27.2\times$ and $7.0\times$, respectively over the benchmarks GC and RAR.

***Index Terms*— Distributed learning, communication topology, gradient aggregation.**

I. Introduction

MODERN machine learning algorithms are now used in a wide variety of domains. However, training a large-scale model over a massive data set is an extremely computation and storage intensive task, e.g. training ResNet with more than 150 layers and hundreds of millions of parameters over the data set ImageNet with more than 14 million images. As a result, there has been significant interest in developing distributed learning strategies that speed up the training of learning models (e.g., [2]–[8]).

In the commonly used Gradient Descent (GD) paradigm for learning, parallelization can be achieved by arranging the machines in a master-worker setup. Through a series of iterations, the master is responsible for updating the underlying model from the results received from the workers, where they compute the partial gradients using their local data batches and upload to the master at each iteration. For the master-worker setup, both synchronous and asynchronous methods have been developed [2]–[7]. In synchronous settings, all the workers wait for each other to complete the gradient computations, while in asynchronous methods, the workers continue the training process after their local gradient is computed. While synchronous approaches provide better generalization behaviors than the asynchronous ones [4], [9], they face major system bottlenecks due to (1) bandwidth congestion at the master due to concurrent communications from the workers to the master [10]; and (2) the delays caused by slow workers or stragglers that significantly increase the run-time [5].

To alleviate the communication bottleneck in distributed learning, various bandwidth efficient strategies have been proposed [11]–[13]. Particularly, Ring-AllReduce (RAR) [10] strategy has been proposed by allowing each worker to only communicate with its neighbors that are arranged in a logical ring. More precisely, the data set, $\mathcal{D}$, is uniformly distributed among N workers and each node combines and passes its partial gradient along the ring such that at the end of the collective operation, each worker has a copy of the full gradient $\mathbf{g}$ (Figure 1). Due to the master-less topology of RAR, it avoids bandwidth bottleneck at any particular node. Furthermore, as shown in [11], RAR is provably bandwidth optimal and induces $\mathcal{O}(1)$ communication overhead that does not depend on the number of distributed workers. As a result, RAR has recently become a central component in distributed deep learning for model updating [14]–[16]. More recent approaches to mitigate bandwidth bottleneck in distributed gradient aggregation include compression and quantization of the gradients [17]–[19].

Despite being bandwidth efficient, AllReduce-type algorithms are inherently sensitive to stragglers, which makes them prone to significant performance degradation and even complete failure if *any* of the workers slows down.

Manuscript received August 25, 2020; revised April 11, 2021 and August 3, 2021; accepted August 8, 2021; approved by IEEE/ACM Transactions on Networking Editor R. La. This work was supported in part by the NSF under Grant CNS-2003035, Grant CCF-1408639, Grant CCF-1755808, and Grant NETS-1419632; in part by the Office of Naval Research (ONR) under Award N000141612189; and in part by the National Security Agency (NSA) Grant, a Research Gift from Intel and by Defense Advanced Research Projects Agency (DARPA) under Contract HR001117C0053. A part of this work was presented in IEEE International Symposium on Information Theory, 2019 [DOI: 10.1109/ISIT.2019.8849431]. *(Amirhossein Reisizadeh and Saurav Prakash contributed equally to this work.) (Corresponding author: Amirhossein Reisizadeh.)*

Amirhossein Reisizadeh and Ramtin Pedarsani are with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: reisizadeh@ucsb.edu; ramtin@ece.ucsb.edu).

Saurav Prakash and Amir Salman Avestimehr are with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: sauravpr@usc.edu; avestimehr@ee.usc.edu).

Digital Object Identifier 10.1109/TNET.2021.3109097

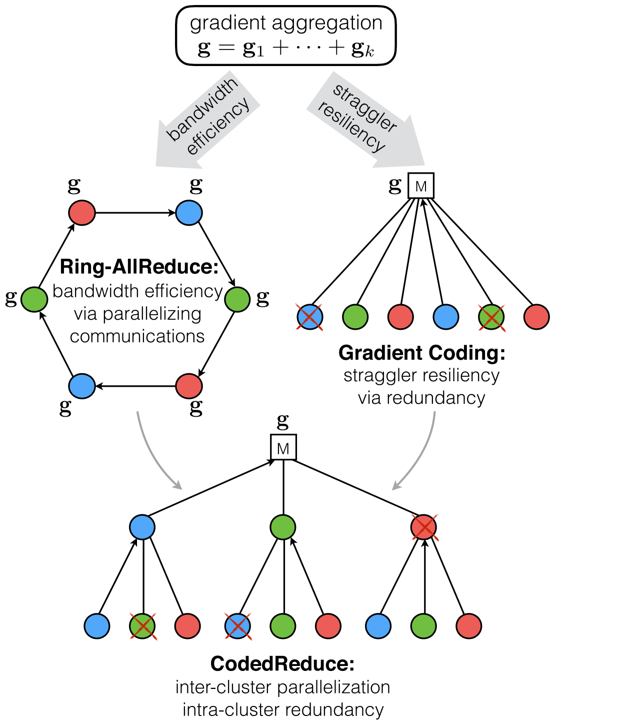

gradient aggregation
$\mathbf{g} = \mathbf{g}_1 + \cdots + \mathbf{g}_k$
bandwidth efficiency
straggler resiliency
Ring-AllReduce:
bandwidth efficiency via parallelizing communications
Gradient Coding:
straggler resiliency via redundancy
M
g
CodedReduce:
inter-cluster parallelization
intra-cluster redundancy

Fig. 1. Illustration of RAR, GC and CR: In RAR, workers communicate only with their neighbors on a ring, which results in high bandwidth utilization; however, RAR is prone to stragglers. GC is robust to stragglers by doing redundant computations at workers; however, GC imposes bandwidth bottleneck at the master. CR achieves the benefits of both worlds, providing high bandwidth efficiency along with straggler resiliency.

Straggler bottleneck becomes even more significant as the cluster size increases [20], [21].

One approach to mitigate stragglers in distributed computation is to introduce computational redundancy via replication. [22] proposes to replicate the straggling task on other available nodes. In [23], the authors propose a partial data replication for robustness. Other relevant replication based strategies have been proposed in [24]–[26]. Recently, coding theoretic approaches have also been proposed for straggler mitigation [27]–[37]. Specifically, Gradient Coding (GC) [38] has been proposed to alleviate stragglers in distributed gradient aggregation in a master-worker topology (Figure 1). In GC, the data set $\mathcal{D}$ is carefully and redundantly distributed among the N workers where each worker computes a *coded* gradient from its local batch. The master node waits for the results of *any* $N-S$ workers and recovers the total gradient $\mathbf{g}$, where the design parameter S denotes the maximum number of stragglers that can be tolerated. Therefore, GC prevents the master from waiting for *all* the workers to finish their computations, and it was shown to achieve significant speedups over the classical uncoded master-worker setup [38].

However, as the cluser size gets large, GC suffers from significant network congestion at the master. In particular, the communication overhead increases to $\mathcal{O}(N)$, as the master needs to receive messages from $\mathcal{O}(N)$ workers. Thus, it is essential to design distributed learning strategies that alleviate stragglers while imposing low communication overhead across the cluster. Consequently, our goal in this paper is to answer the following fundamental question:

Can we achieve the communication parallelization of RAR and the straggler toleration of GC simultaneously in distributed gradient aggregation?

We answer this question in the affirmative. As the main contribution of this paper, we propose a joint design of data allocation and communication strategy that is robust to stragglers, alongside being bandwidth efficient. Specifically, we propose a scalable and robust scheme for synchronous distributed gradient aggregation, called CodedReduce (CR).

There are two key ideas behind CR. Firstly, we use a logical tree topology for communication consisting of a master node, L layers of workers, where each *parent* node has n *children* nodes (Figure 1). In the proposed configuration, each node communicates only with its parent node for *downloading* the updated model and *uploading* partial gradients. As in the classical master-worker setup, the root node (master) recovers the full gradient and updates the model. Except for the leaf nodes, each node receives *enough* number of *coded* partial gradients from its children, combines them with its local and partial gradient and uploads the result to its parent. This distributed communication strategy alleviates the communication bottleneck at the nodes, as multiple parents can concurrently receive from their children. Secondly, the coding strategy utilized in CR provides robustness to stragglers. Towards this end, we exploit ideas from GC and propose a data allocation and communication strategy such that *each* node needs to only wait for *any* $n-s$ of its children to return their results.

The theoretical guarantees of the proposed CR scheme are two-fold. First, we characterize the computation load introduced by the proposed CR and prove that for a fixed straggler resiliency, CR achieves the optimal *computation load* (relative size of the assigned local data set to the total data set) among all the robust gradient aggregation schemes over a fixed tree topology. Moreover, CR significantly improves upon GC in the computation load of the workers. More precisely, to be robust to straggling/failure of α fraction of the children, GC loads each worker with $\approx\alpha$ fraction of the total data set, while CR assigns only $\approx\alpha^L$ fraction of the total data set, which is a major improvement. Secondly, we model the workers' computation times as shifted exponential random variables and asymptotically characterize the average latency of CR, that is the expected time to aggregate the gradient at the master node as the number of workers tends to infinity. This analysis further demonstrates how CR alleviates the bandwidth efficiency and speeds up the training process by parallelizing the communications via a tree.

In addition to provable theoretical guarantees, the proposed CR scheme offers substantial improvements in practice. As a representative case, Figure 2 provides the gradient aggregation time averaged over many gradient descent iterations implemented over Amazon EC2 clusters. Compared to three benchmarks – classical Uncoded Master-Worker (UMW), GC, RAR – the proposed CR scheme attains speedups of $22.5\times$, $6.4\times$ and $4.3\times$, respectively.

II. Problem Setup and Background

In this section, we provide the problem setup followed by a brief background on RAR and GC and their corresponding straggler resiliency and communication parallelization.

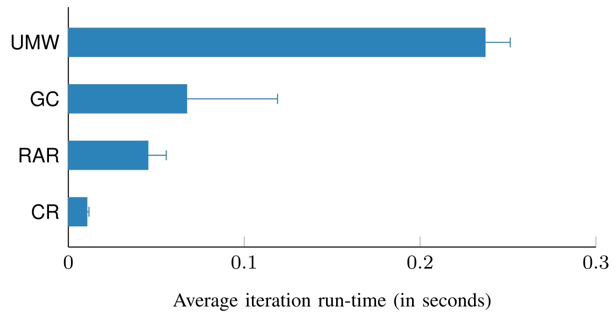


Fig. 2. Average iteration time for gradient aggregation in different schemes CR, RAR, GC and UMW: Training a linear model is implemented on a cluster of $N = 84$ t2.micro instances.

A. Problem Setting

Many machine learning tasks involve fitting a model over a training data set by minimizing a loss function. For a given labeled data set $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^{p+1} : j = 1, \ldots, d\}$, the goal is to solve the following optimization problem:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^p} \sum_{\mathbf{x} \in \mathcal{D}} \ell(\theta; \mathbf{x}) + \lambda R(\theta), \tag{1}$$

where $\ell(\cdot)$ and $R(\cdot)$ respectively denote the loss and regularization functions, and the optimization problem is parameterized by λ. One of the most popular ways of solving (1) in distributed learning is to use the Gradient Descent (GD) algorithm. More specifically, under standard convexity assumptions, the following sequence of model updates $\{\theta^{(t)}\}_{t=0}^{\infty}$ converges to the optimal solution θ^*:

$$\theta^{(t+1)} = h_R\left(\theta^{(t)}, \mathbf{g}\right), \tag{2}$$

where $h_R(\cdot)$ is a gradient-based optimizer depending on the regularizer $R(\cdot)$ and

$$\mathbf{g} = \sum_{\mathbf{x} \in \mathcal{D}} \nabla \ell\left(\theta^{(t)}; \mathbf{x}\right), \tag{3}$$

denotes the gradient of the loss function evaluated at the model at iteration t over the data set $\mathcal{D}$. Under certain assumptions, the iterations in (2) converge to a local optimum in the non-convex case, as well. For instance, if all the saddle points of a smooth non-convex objective are strict-saddle, then the iterations in (2) converge to a local minimum [39]. The core component of the iterations defined in (2) is the computation of the gradient vector $\mathbf{g}$ at each iteration. At scale, due to limited storage and computation capacity of the computing nodes, gradient aggregation task (3) has to be carried out over distributed nodes. This parallelization, as we discussed earlier, introduces two major bottlenecks: stragglers and bandwidth contention. The goal of the distributed gradient aggregation scheme is to provide straggler resiliency as well as communication parallelization. At a high level, straggler resiliency, α, refers to the fraction of the straggling workers that the distributed aggregation scheme is robust to, and communication parallelization gain, β, quantifies the number of simultaneous communications in the network by distributed nodes compared to only one simultaneous communication in a single-node (master-worker) aggregation scheme.

Next, we discuss the data allocation and communication strategy of two synchronous gradient aggregation schemes in distributed learning and their corresponding straggler resiliency and communication parallelization gain.

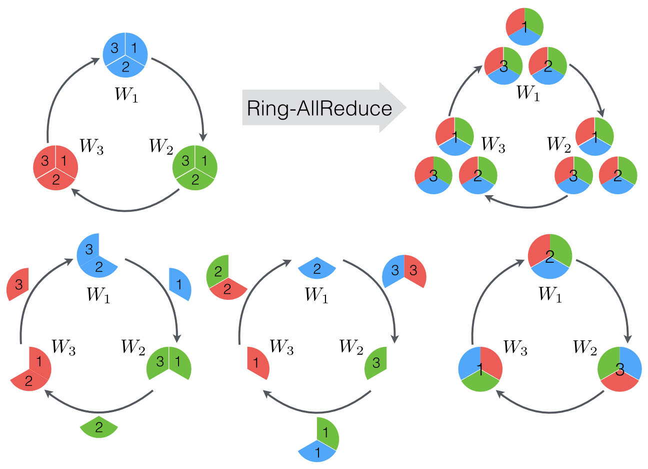


Fig. 3. Illustration of communication strategy in RAR for $N = 3$ workers.

B. Ring-AllReduce

In AllReduce-type aggregation schemes, the data set is uniformly distributed over N worker nodes $\{W_1, \ldots, W_N\}$ which coordinate among themselves in a master-less setting to aggregate their partial gradients and compute the aggregate gradient $\mathbf{g}$ at each worker. Particularly in RAR, each worker W_i partitions its local partial gradient into N segments $\mathbf{v}_{1,i}, \ldots, \mathbf{v}_{N,i}$. In the first round, W_i transmits $\mathbf{v}_{i,i}$ to W_{i+1}. Each worker then adds up the received segment to the corresponding segment of its local gradient, i.e., W_i obtains $\mathbf{v}_{i-1,i-1} + \mathbf{v}_{i-1,i}$. In the second round, the reduced segment is forwarded to the neighbor and added up to the corresponding segment. Proceeding similarly, at the end of $N-1$ rounds, each worker has a unique segment of the full gradient, i.e., W_i has $\mathbf{v}_{i+1,1} + \ldots + \mathbf{v}_{i+1,N}$. After the reduce-scatter phase, the workers execute the collective operation of AllGather where the full gradient $\mathbf{g}$ becomes available at each node. The RAR operation for a cluster of three workers is illustrated in Figure 3.

It is clear that RAR cannot tolerate *any* straggling nodes since the communications are carried out over a ring and each node requires its neighbor's result to proceed in the ring, i.e., the straggler resiliency for RAR is $\alpha_{\mathsf{RAR}} = 0$. However, the ring communication design in RAR alleviates the communication congestion at busy nodes, and achieves communication parallelization gain $\beta_{\mathsf{RAR}} = \Theta(N)$ which is optimal [10].

C. Gradient Coding

Gradient Coding (GC) [38] was recently proposed to provide straggler resiliency in a master-worker topology with one master node and N distributed worker nodes $\{W_1, \ldots, W_N\}$ as depicted in Figure 1. We start the description of GC with an illustrative example.

Example 1 (Gradient Coding): To make gradient aggregation over $N = 3$ workers robust to any $S = 1$ straggler, GC partitions the data set to $\{\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3\}$ and assigns 2

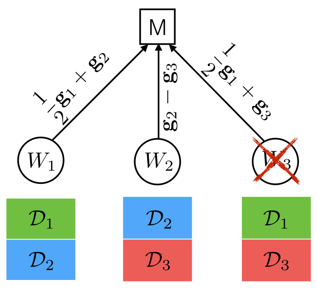


Fig. 4. Illustration of data allocation and communication strategy in GC for $N = 3$ workers.

partitions to each worker as depicted in Figure 4. Full gradient $\mathbf{g} = \mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$ can be recovered from any $N - S = 2$ workers, e.g., the master recovers $\mathbf{g}$ from W_1 and W_2 by combining their results as $\mathbf{g} = 2\left(\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2\right) - (\mathbf{g}_2 - \mathbf{g}_3)$.

In general, to be robust to *any* $S \in [N] = \{1, \ldots, N\}$ stragglers, GC uniformly partitions the data set $\mathcal{D}$ to $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$ (e.g. $k = N$) with corresponding partial gradients $\mathbf{g}_1, \ldots, \mathbf{g}_k$ and distributes them redundantly among the workers such that each partition is placed in $S+1$ workers, thus achieving a computation load of $r_{\mathsf{GC}} = \frac{S+1}{N}$. Let matrix $\mathbf{G} = [\mathbf{g}_1, \ldots, \mathbf{g}_k]^\top \in \mathbb{R}^{k\times p}$ denote the collection of partial gradients. Each worker W_i then computes its local partial gradients and sends $\mathbf{b}_i\mathbf{G}$ to the master, where $\mathbf{B} = [\mathbf{b}_1; \ldots; \mathbf{b}_N] \in \mathbb{R}^{N\times k}$ denotes the encoding matrix, i.e. non-zero elements in $\mathbf{b}_i$ specifies the partitions stored in worker W_i. Upon receiving the results of any $N - S$ workers, the master recovers the total gradient $\mathbf{g}$ by linearly combining the received results, that is $\mathbf{g} = \mathbf{a}_f\mathbf{BG}$ where the row vector $\mathbf{a}_f \in \mathbb{R}^{1\times N}$ corresponds to a particular set of S stragglers and $\mathbf{A} = [\mathbf{a}_1; \ldots; \mathbf{a}_F]$ denotes the decoding matrix with $F = \binom{N}{S}$ distinct straggling scenarios. The GC algorithm designs encoding and decoding matrices $(\mathbf{B}, \mathbf{A})$ such that, in the worst case, the full gradient $\mathbf{g}$ is recoverable from the results of *any* $N - S$ out of N workers, i.e. straggler resiliency $\alpha_{\mathsf{GC}} = S/N$ is attained. Although GC prevents the master to wait for *all* the workers to finish their computations, it requires simultaneous communications from the workers that will cause congestion at the master node, and lead to parallelization gain $\beta_{\mathsf{GC}} = \Theta(1)$ for a constant resiliency.

Having reviewed RAR and GC strategies and their resiliency and parallelization properties, we now informally provide the guarantees of our proposed CR scheme in the following remark.

Remark 1: CR arranges the available N workers via a tree configuration with L layers of nodes and each parent having n children, i.e. $N = n + \cdots + n^L$. The proposed data allocation and communication strategy in CR results in communication parallelization gain $\beta_{\mathsf{CR}} = \Theta(N^{1-1/L})$ which approaches $\beta_{\mathsf{RAR}} = \Theta(N)$ for large L. Moreover, given a computation load $0 \leq r \leq 1$, CR is robust to straggling of $\alpha_{\mathsf{CR}} \approx r^{1/L}$ fraction of the children per any parent in the tree, while GC is robust to only $\alpha_{\mathsf{GC}} \approx r$ fraction of nodes and RAR has no straggler resiliency. Therefore, CR achieves the best of RAR and GC, simultaneously. Table I summarizes these results and Theorems 1 and 2 formally characterize such guarantees.

III. Proposed CodedReduce Scheme

In this section, we first present our proposed CodedReduce (CR) scheme by describing data set allocation and communication strategy at the nodes followed by an illustrative example. Then, we provide theoretical guarantees of CR and conclude the section with optimality of CR.

TABLE I

Communication Parallelization Gain and Straggler Resiliency of Three Designs RAR, GC, and CR in a System With N Nodes With Computation Load r, Where CR Has a Tree Communication Topology of L Layers

Scheme	Straggler Resiliency (α)	Communication Parallelization Gain (β)
RAR	0	$\Theta(N)$
GC	r	$\Theta(1)$
CR	$r^{1/L}$	$\Theta\left(N^{1-1/L}\right)$

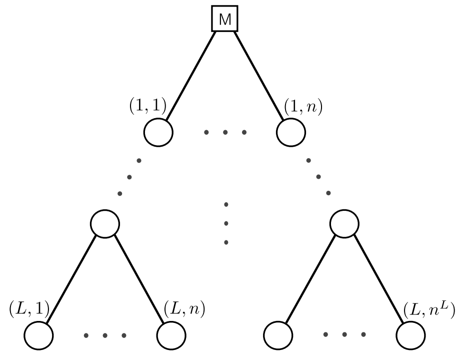


Fig. 5. (n, L)–regular tree topology.

A. Description of CR Scheme

Let us start with the proposed network configuration. CR arranges the communication pattern among the nodes via a *regular* tree structure as defined below. An (n, L)–regular tree graph T consists of a master node and L layers of worker nodes. At any layer (except for the lowest), each *parent* node is connected to n *children* nodes in the lower layer, i.e. there is a total of $N = n + \cdots + n^L$ nodes (See Figure 5). Each node of the tree is identified with a pair (l, i), where $l \in [L]$ and $i \in [n^l]$ denote the corresponding layer and the node's index in that layer, respectively. Furthermore, $T(l, i)$ denotes the sub-tree with the root node (l, i).

We next introduce a notation that eases the algorithm description. 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 $\mathbf{g}_{b\mathcal{D}} = b\mathbf{g}_{\mathcal{D}} = b\sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\theta^{(t)}; \mathbf{x})$. As a building block of CR, we define the sub-routine CompAlloc in which given a generic data set $\mathcal{D}$, n workers are carefully assigned with data partitions and combining coefficients such that the full gradient over $\mathcal{D}$ is retrievable from the computation results of any $n - s$ workers (Pseudo-code in Appendix A).

1) CompAlloc: For specified n and s, GC (Algorithm 2 in [38]) constructs the encoding matrix $\mathbf{B} = [\mathbf{b}_1; \ldots; \mathbf{b}_n] = [b_{i\kappa}]$. In CompAlloc, the input data set $\mathcal{D}$ is partitioned to $\mathcal{D} = \cup_{\kappa=1}^k \mathcal{D}_\kappa$ and distributed among the n workers along with the corresponding coefficients. That is, each worker $i \in [n]$ is assigned with $\mathcal{D}(i) = \cup_{\kappa=1}^k b_{i\kappa}\mathcal{D}_\kappa$ which specifies its local data set and corresponding combining coefficients. The parent of the n workers is then able to recover

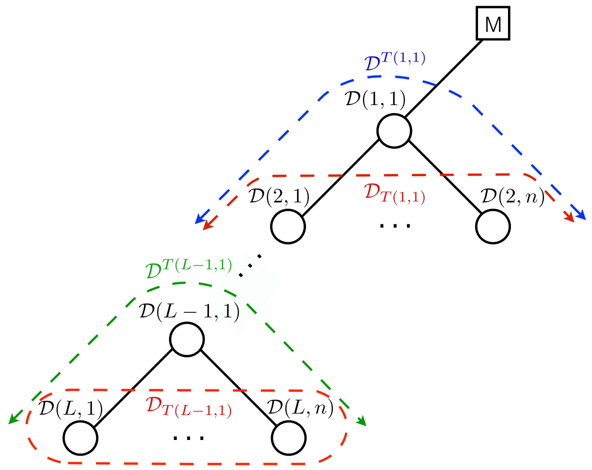


Fig. 6. Illustration of task allocation in CR.

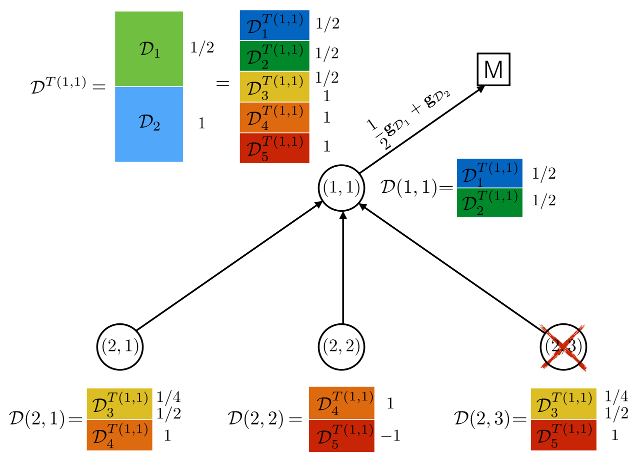

$\mathcal{D}^{T(1,1)}=$
$\mathcal{D}_1$ 1/2
$\mathcal{D}_2$ 1
$=$
$\mathcal{D}_1^{T(1,1)}$ 1/2
$\mathcal{D}_2^{T(1,1)}$ 1/2
$\mathcal{D}_3^{T(1,1)}$ 1/2 1
$\mathcal{D}_4^{T(1,1)}$ 1
$\mathcal{D}_5^{T(1,1)}$ 1
M
$\frac{1}{2}\mathbf{g}_{\mathcal{D}_1}+\mathbf{g}_{\mathcal{D}_2}$
(1, 1)
$\mathcal{D}(1,1)=$
(2, 1)
(2, 2)
(2, 3)
$\mathcal{D}(2,1)=$ $\mathcal{D}_3^{T(1,1)}$ 1/4 1/2 $\mathcal{D}_4^{T(1,1)}$ 1
$\mathcal{D}(2,2)=$ $\mathcal{D}_4^{T(1,1)}$ 1 $\mathcal{D}_5^{T(1,1)}$ −1
$\mathcal{D}(2,3)=$ $\mathcal{D}_3^{T(1,1)}$ 1/4 1/2 $\mathcal{D}_5^{T(1,1)}$ 1

Fig. 7. Illustration of data allocation and communication strategy in CR for a $(3, 2)$–regular tree.

the gradient over $\mathcal{D}$, i.e. $\mathbf{g}_{\mathcal{D}}$ upon receiving the partial coded gradients of any $n-s$ workers and using the decoding matrix $\mathbf{A}$ designed by GC (Algorithm 1 in [38]).

2) CodedReduce: CR is implemented in two phases. It first allocates each worker with its local computation task via CR.ALLOCATE procedure. This specifies each worker with its local data set and combining coefficients. Then, the communication strategy is determined by CR.EXECUTE.

3) CR.Allocate:

1) Starting from the master, data set $\mathcal{D}^{T(1,i)}$ is assigned to sub-tree $T(1,i)$ for $i\in[n]$ via the allocation module COMPALLOC (Figure 6).
2) In layer $l=1$, each worker $(1,i)$, $i\in[n]$, picks $r_{\mathsf{CR}}d$ data points from the corresponding sub-tree's data set $\mathcal{D}^{T(1,i)}$ as its local data set $\mathcal{D}(1,i)$ and passes the rest $\mathcal{D}_{T(1,i)}=\mathcal{D}^{T(1,i)}\setminus\mathcal{D}(1,i)$ to its children and their sub-trees (Figure 6).
3) Step (1) is repeated by using the module COMPALLOC and treating $\mathcal{D}_{T(1,i)}$ as the input data set to distribute it among the children of node $(1,i)$.
4) Same procedure is applied till reaching the bottom layer (Figure 6). By doing so, the data set $\mathcal{D}$ is redundantly distributed across the tree while all the workers are equally loaded with $r_{\mathsf{CR}}d$ data points, where in Theorem 1 we will show that r_{CR} is a self-derived pick for CR given in (5).

4) CR.Execute:

1) All the N nodes start their local partial *coded* gradient computations on the current model $\theta^{(t)}$, i.e. $\mathbf{g}_{\mathcal{D}(l,i)}$ for all nodes (l,i). Note that $\mathbf{g}_{\mathcal{D}(l,i)}$ is a coded gradient (i.e. a linear combination of partial gradients) since $\mathcal{D}(l,i)$ carries combining coefficients along with its data points.
2) Starting from the leaf nodes, they send their partial coded gradient computation results (messages) $\mathbf{m}_{(L,i)}=\mathbf{g}_{\mathcal{D}(L,i)}$ up to their parents.
3) Upon receiving enough results from their children (any $n-s$ of them), workers in layer $L-1$ recover a linear combination of their children's messages via proper row in the decoding matrix $\mathbf{A}$, e.g., parent node $(L-1,1)$ recovers from its children's messages $[\mathbf{m}_{(L,1)};\ldots;\mathbf{m}_{(L,n)}]$ via the proper decoding row $\mathbf{a}_{f(L-1,1)}$.
4) Recovered partial gradient is added to the local partial coded gradient and is uploaded to the parent, e.g. node $(L-1,1)$ uploads $\mathbf{m}_{(L-1,1)}$ to its parent, where
$$\mathbf{m}_{(L-1,1)}=\mathbf{a}_{f(L-1,1)}[\mathbf{m}_{(L,1)};\ldots;\mathbf{m}_{(L,n)}]+\mathbf{g}_{\mathcal{D}(L-1,1)}.$$
5) The same procedure is repeated till reaching the master node which is able to aggregate the total gradient $\mathbf{g}_{\mathcal{D}}$.

The pseudo-code for CR is available in Appendix B.

B. An Example for CR

In this section, we provide a simple example to better illustrate the proposed CR scheme.

Example 2 (CodedReduce): Consider a $(3,2)$–regular tree with $N=12$ nodes and $s=1$ straggler per parent. From GC, we have the decoding and encoding matrices
$$\mathbf{A}=\begin{pmatrix}0&1&2\\1&0&1\\2&-1&0\end{pmatrix},\quad \mathbf{B}=\begin{pmatrix}1/2&1&0\\0&1&-1\\1/2&0&1\end{pmatrix}.\tag{4}$$

Following CR's description, we partition the data set of size d as $\mathcal{D}=\{\mathcal{D}_1,\mathcal{D}_2,\mathcal{D}_3\}$ and assign $\mathcal{D}^{T(1,1)}=\frac{1}{2}\mathcal{D}_1\cup\mathcal{D}_2$ to sub-tree $T(1,1)$. Node $(1,1)$ then picks $r_{\mathsf{CR}}d=\frac{4}{15}d$ 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 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 7 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:
$$\begin{aligned}\mathcal{D}(2,1)&=\frac{1}{2}\mathcal{D}_3^{T(1,1)}\cup\mathcal{D}_4^{T(1,1)},\\ \mathcal{D}(2,2)&=\mathcal{D}_4^{T(1,1)}\cup(-1)\mathcal{D}_5^{T(1,1)},\\ \mathcal{D}(2,3)&=\frac{1}{2}\mathcal{D}_3^{T(1,1)}\cup\mathcal{D}_5^{T(1,1)}.\end{aligned}$$

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

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

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 $(1,3)$:

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

C. Theoretical Guarantees of CR

In this section, we formally present the theoretical guarantees of CR. We first characterize the computation load induced by CR and demonstrate its significant improvement over GC. Then, we consider the commonly-used shifted exponential run-time computation distribution and a single-port communication model for workers and asymptotically characterize the expected run-time of CR and conclude with a discussion on its communication parallelization gain.

1) Computation Load Optimality: We show that for a fixed tree topology, the proposed CR is optimal in the sense that it achieves the minimum per-node computation load for a target resiliency. This optimality is established in two steps per Theorem 1: (i) we first show the achievability by characterizing the computation load of CR; and (ii) we establish a converse showing that CR's computation load is as small as possible. Proof is available in Appendix C.

Theorem 1: For a fixed (n,L)–regular tree, any gradient aggregation scheme robust to any s stragglers per any parent requires computation load r where

$$r \geq r_{CR} = \frac{1}{\left(\frac{n}{s+1}\right) + \cdots + \left(\frac{n}{s+1}\right)^L}. \tag{5}$$

Remark 2: While CR is α-resilient, i.e. robust to *any* $s = \alpha n$ stragglers per *any* parent node, it significantly improves the per-node computation (and storage) load compared to an equivalent GC scheme with the same resiliency. In particular, GC loads each worker with $r_{\text{GC}} = \frac{S+1}{N} = \frac{\alpha N+1}{N} \approx \alpha$ fraction of the data set, while CR considerably reduces it to $r_{\text{CR}} = 1/\sum_{l=1}^{L}\left(\frac{n}{\alpha n+1}\right)^l \approx \alpha^L$. For $\alpha = 0.5$ as an instance, CR reduces the computation load $7\times$ by rearranging the nodes from 1 layer to 3 layers.

Remark 3: CR makes the distributed GD strategy α-resilient, that is any $s = \alpha n$ stragglers per any parent node which sums up to a total of $S = \alpha N$ stragglers – the same as the worst case number of stragglers in GC. It is clear than if the stragglers are picked adversarially, for instance all the nodes in layer 1, then CR fails to recover the total gradient at the master. However, our experiments over Amazon EC2 confirm that stragglers are randomly distributed over the tree and not adversarially picked, which is aligned with the random stragglers pattern considered in this paper.

2) Total Gradient Computation Complexity: To better characterize the advantages of CR, we characterize its *total* gradient computation complexity in order to reach the final parameter model with predefined accuracy. More precisely, we focus on learning problems with strongly convex losses and let T_{CR} denote the total number of iterations to reach a final model θ such that $\|\theta-\theta^*\|^2 \leq \epsilon$. Since in each iteration of CR the *exact* gradient on all the d data samples is computed (same as in GD), therefore $T_{\text{CR}} = \mathcal{O}(\log(1/\epsilon))$. In each iteration, each of the N worker nodes compute $\alpha_{\text{CR}} \cdot d$ gradients, where according to Theorem 1, we have $\alpha_{\text{CR}} \approx \alpha^L$. All in all, in order to reach an ϵ-accurate model, the CR method requires $\mathcal{O}(\alpha^L \cdot N \cdot \log(1/\epsilon) \cdot d)$ gradient computations in total.

One simple and yet naive approach to mitigate stragglers is to update the model using the gradient computation results of *only* a fraction (α) of worker node (non-stragglers). This approach can be treated as standard Stochastic Gradient Descent (SGD) which requires $T_{\text{SGD}} = \mathcal{O}(1/\epsilon)$ iterations in total to reach an ϵ-accurate model. Since each of the N worker nodes store d/N samples (i.e. no redundant data allocation), therefore in each iteration, each node computes $\alpha d/N$ gradients. Putting all together, in order to reach ϵ-accurate model, SGD requires $\mathcal{O}(\alpha \cdot 1/\epsilon \cdot d)$ gradient computations in total. Comparing the two gradient computation complexities of CR and SGD, we observe that although SGD slashes the complexity by a *linear* factor N, however, it suffer from two *exponential* factors, that are growing α^L to α and $\log(1/\epsilon)$ to $1/\epsilon$ which significantly increase the total gradient computation complexities, as $\alpha^L \ll \alpha$ and $\log(1/\epsilon) \ll 1/\epsilon$.

3) Latency Performance: While we have derived the straggler resiliency of CR, the ultimate goal of a distributed gradient aggregation scheme is to have small latency which is partly attained by establishing higher communication parallelization.

a) Computation time model: We consider random computation time model for workers with shifted exponential distribution which is used in several prior works [40]–[42]. More precisely, for a worker W_i with assigned data set of size d_i, we model the computation time as a random variable with a shifted exponential distribution as follows:

$$\mathbb{P}[T_i \leq t] = 1 - e^{-\frac{\mu}{d_i}(t-ad_i)}, \quad \text{for } \; t \geq ad_i, \tag{6}$$

where system parameters $a = \Theta(1)$ and $\mu = \Theta(1)$ respectively denote the shift and the exponential rate. We assume that T_i's are independent.

b) Communication time model: To model the communication time and bandwidth bottleneck, we assume that each node is able to receive messages from only one other node at a time, and the total available bandwidth is dedicated to the communicating node. We also assume that communicating a partial gradient vector (of size p) from a child to its parent takes a constant time t_c.

The following theorem asymptotically characterizes the expected run-time of CR which we denote by T_{CR}

(Proof is available in Appendix D). More precisely, we consider the regime of interest where the data set size d and the number of layers L in the tree are fixed, while the number of children per parent, i.e. n is approaching infinity with a constant straggler ratio $\alpha = s/n = \Theta(1)$.

Theorem 2: Considering the computation time model in (6) for workers, the expected run-time of CR *on an* (n, L)*–regular tree with resiliency* $\alpha = \Theta(1)$ *satisfies the followings*:

$$\mathbb{E}\left[T_{\mathsf{CR}}\right] \geq \frac{r_{\mathsf{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathsf{CR}} d + \left(n(1-\alpha) - o(n) + L - 1\right)\left(1 - o(1)\right) t_c + o(1), \tag{7}$$

$$\mathbb{E}\left[T_{\mathsf{CR}}\right] \leq \frac{r_{\mathsf{CR}} d}{\mu} \log\left(\frac{1}{\alpha}\right) + a r_{\mathsf{CR}} d + n\left(1 - o(1)\right) L t_c + o(1). \tag{8}$$

Remark 4: Theorem 2 implies that the expected run-time of the proposed CR algorithm breaks down into two terms: $\mathbb{E}\left[T_{\mathsf{CR}}\right] = \Theta(1) + \Theta(n)$, where the two terms $\Theta(1)$ and $\Theta(n)$ correspond to computation and communication times, respectively. As a special case, it also implies that the average run-time for GC is $\mathbb{E}\left[T_{\mathsf{GC}}\right] = \Theta(1) + \Theta(N)$. This clearly demonstrates that CR is indeed alleviating the bandwidth bottleneck and it improves the communication parallelization gain from $\beta_{\mathsf{GC}} = \Theta(1)$ to $\beta_{\mathsf{CR}} = \Theta(N/n) = \Theta(N^{1-1/L})$ by parallelizing the communications over an L-layer tree structure.

IV. Empirical Evaluation of CR

In this section, we provide the results of our experiments conducted over Amazon EC2, for which we used `Python` with `mpi4py` package. Our results demonstrate significant speedups of CR over baseline approaches. We consider two sets of machine learning experiments, one with a real data set, and another with an artificial data set. For each machine learning setting, we consider two cluster configurations, one with $N = 84$ workers, and another with $N = 156$ workers, using `t2.micro` instance for master and all workers. Furthermore, each experiment is run for 300 rounds. Next, we describe the experiments in detail and provide the results.

A. Convex Optimization

1) Real Data Set: We consider the machine learning problem of logistic regression via gradient descent (GD) over the real data set GISETTE [43]. The problem is to separate the often confused digits '9' and '4'. We use $d = 6552$ training samples, with model size $p = 5001$. The following relative error rate is considered for model estimation:

$$\text{Relative Error Rate} = \frac{\left\|\theta^{(t)} - \theta^{(t-1)}\right\|^2}{\left\|\theta^{(t-1)}\right\|^2}, \tag{9}$$

where $\theta^{(t)}$ denotes the estimated model at iteration t. The following schemes are considered for data allocation and gradient aggregation:

1) Uncoded Master-worker (UMW): This is the naive scheme in which the data set is uniformly partitioned among the workers, and the master waits for results from all the workers to aggregate the gradient.

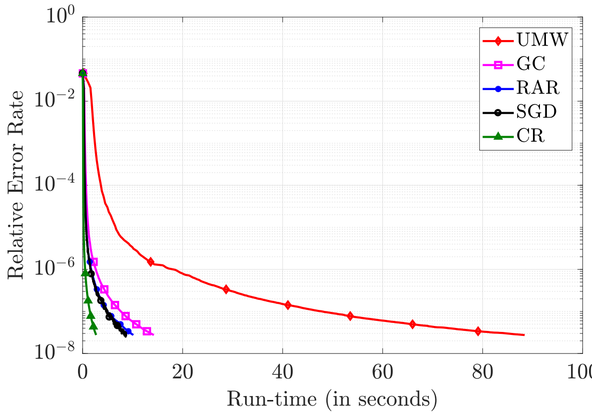


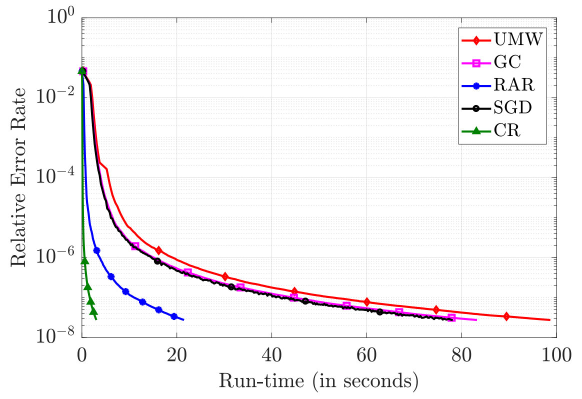
Fig. 8. Convergence curves for relative error rate vs wall-clock time for logistic regression over $N = 84$ workers. The straggler resiliency is $\alpha = 1/4$. CR achieves a speedup of up to 32.8×, 5.3×, 3.8× and 3.2× respectively over UMW, GC, RAR and SGD.

2) Gradient Coding (GC): We implement GC as described in Section II-C, with the straggler parameter $S = \alpha N$.
3) Ring-AllReduce (RAR): The data set is uniformly partitioned over the workers and the MPI function `MPI_Allreduce()` is used for gradient aggregation.
4) Stochastic Gradient Descent (SGD): The data allocation is the same as UMW. However, the master updates the model using the partial gradient obtained via aggregating the results from results of *only* the first $N - S$ children. Furthermore, as is typical in SGD experiments, we used a learning rate of $c_1/(t + c_2)$ where c_1 and c_2 were numerically optimized.
5) CodedReduce (CR): We implement our proposed scheme as presented in Section III on a tree with $(n, L) = (12, 2)$, while the straggler parameter $s = \alpha n$.

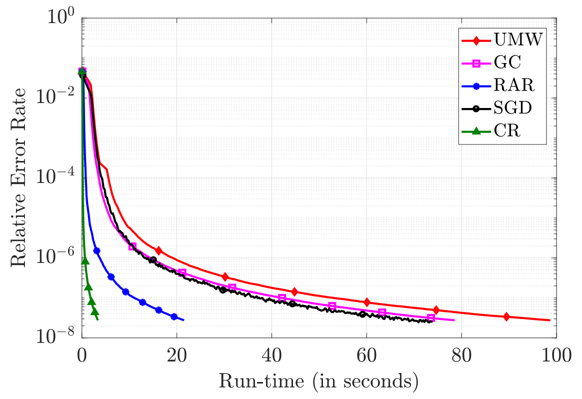
Next, we plot the relative error rate defined in (9) as a function of wall-clock time for our logistic regression experiments with $N = 84$ workers and $N = 156$ workers respectively in Fig. 8 and Fig. 9. For $N = 84$, we consider a straggler-resiliency of $\alpha = 1/4$, while for $N = 156$, we consider three different values of $\alpha : 1/12, 2/12$ and $3/12$.

We make the following observations from the plots:

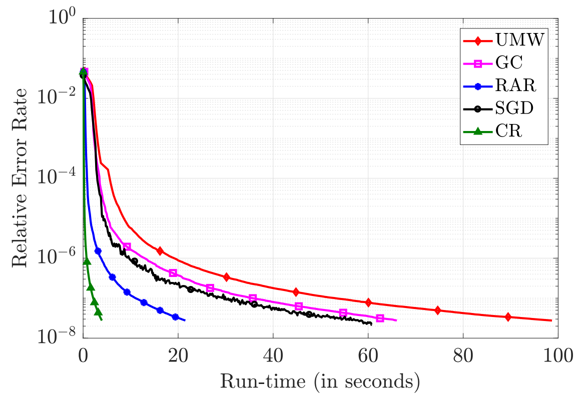
- As demonstrated by Fig. 8 and 9, CR achieves significant speedups over the baseline approaches. Specifically, for $(N, \alpha) = (84, 1/4)$, CR is faster than UMW, GC, RAR and SGD by 32.8×, 5.3×, 3.8× and 3.2× respectively. For $(N, \alpha) = (156, 1/12)$, CR achieves speedups of 32.3×, 27.2×, 7.0× and 25.4× respectively over UMW, GC, RAR and SGD. Similar speedups are obtained with $(N, \alpha) = (156, 2/12)$ and $(N, \alpha) = (156, 3/12)$, as demonstrated by Fig. 9(b) and Fig. 9(c) respectively.
- Although GC gains over UMW by avoiding stragglers, its performance is still bottlenecked by bandwidth congestion, and the increase in computation load at each worker by a factor of $(S + 1)$ in comparison to UMW. The bottlenecks are reflected in comparison with SGD, which has similar or better performance in comparison to GC due to much less computation load per worker.
- RAR significantly outperforms UMW as well as GC for $N = 84$ as well as $N = 156$ worker settings. Although RAR achieves similar performance in comparison to



(a) Convergence curves for $\alpha = 1/12$. CR achieves a speed up of up to $32.3\times$, $27.2\times$, $7.0\times$ and $25.4\times$ respectively over UMW, GC, RAR and SGD.



(b) Convergence curves for $\alpha = 2/12$. CR achieves a speed up of up to $29.3\times$, $23.3\times$, $6.4\times$ and $21.9\times$ respectively over UMW, GC, RAR and SGD.



(c) Convergence curves for $\alpha = 3/12$. CR achieves a speed up of up to $25.0\times$, $16.8\times$, $5.4\times$ and $15.4\times$ respectively over UMW, GC, RAR and SGD.

Fig. 9. Convergence results for relative error rate vs wall-clock time for logistic regression over $N = 156$ workers with different straggler resiliency α.

SGD for $N = 84$ workers scenario, it ultimately beats all the schemes with the generic master-worker topology when the cluster size is increased to $N = 156$. Our proposed CR algorithm combines the best of GC and RAR by providing straggler robustness via coding and alleviating bandwidth bottleneck via a tree topology.

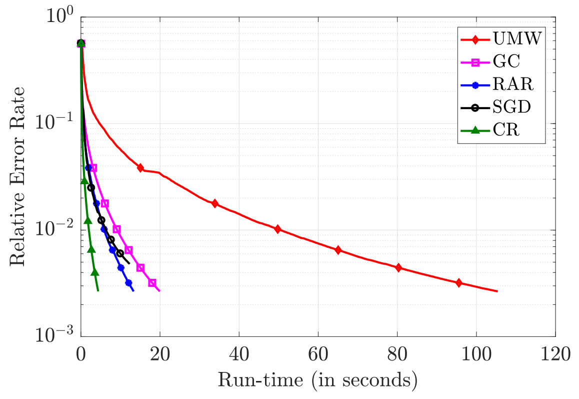


Fig. 10. Convergence curves for normalized error rate vs wall-clock time for linear regression over $N = 84$ workers. The straggler resiliency is $\alpha = 1/4$. CR achieves a speedup of up to $24.1\times$, $4.6\times$, $3.0\times$ and $2.8\times$ respectively over UMW, GC, RAR and SGD.

2) Artificial Data Set: Next we solve a linear regression problem via GD over a synthetic data set with parameters $(d, p) = (7644, 6500)$. We generate the data set using the following model:

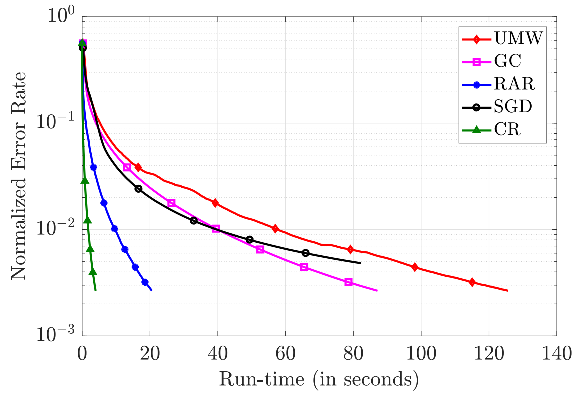
$$\mathbf{x}_j(p+1) = \mathbf{x}_j(1:p)^\top \theta_* + z_j, \quad \text{for } j \in [d], \tag{10}$$

where the true model θ_* and features $\mathbf{x}_j(1:p) = [\mathbf{x}_j(1); \ldots; \mathbf{x}_j(p)]$ are drawn randomly from $\mathcal{N}(0, I_p)$ distribution and z_j is a standard Gaussian noise. We consider the following normalized error rate:

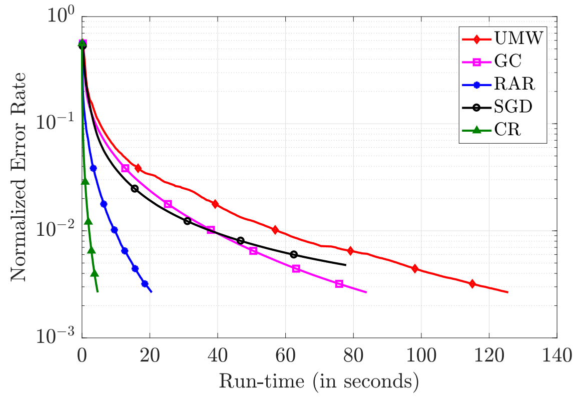
$$\text{Normalized Error Rate} = \frac{\left\|\theta^{(t)} - \theta_*\right\|^2}{\|\theta_*\|^2}. \tag{11}$$

In Fig. 10 and 11, we plot the normalized error rate defined in (11) as a function of wall-clock time for $N = 84$ and $N = 156$ respectively. We consider similar configuration and schemes as for the experiments with real data set. The following observations are made with regard to the experiments:

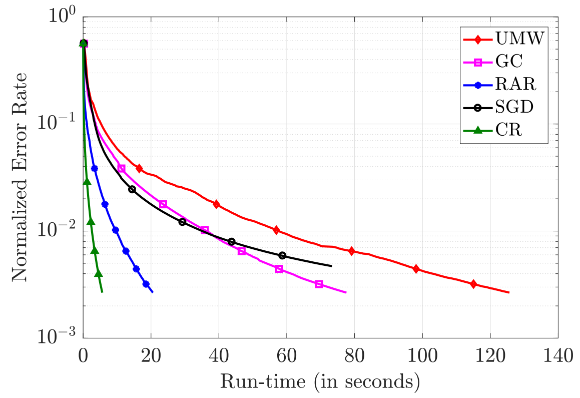
- As in the previous case of logistic regression with real data set, CR achieves significant speedups over baseline approaches for linear regression as well. Particularly, for $(N, \alpha) = (84, 1/4)$, CR achieves speedups of $24.1\times$, $4.6\times$, $3.0\times$ and $2.8\times$ over UMW, GC, RAR and SGD respectively. When $(N, \alpha) = (156, 1/12)$, CR achieves speedups of $31.7\times$, $22.0\times$, $5.2\times$ and $20.7\times$ in comparison to UMW, GC, RAR and SGD respectively. Similar speedups are obtained for $(N, \alpha) = (156, 2/12)$ and $(N, \alpha) = (156, 3/12)$.
- GC performs better than UMW by avoiding stragglers. However, its performance is still bottlenecked by bandwidth congestion and the increase in computation load at each worker by a factor of $(S + 1)$ in comparison to UMW.
- SGD achieves a gain in per iteration time over UMW and GC. However, it has higher normalized error with respect to the true model.
- Combined with the results of logistic regression, our experiments complement the theoretical gains of CR that have been established earlier. As demonstrated by the results, a tree-based topology is well-suited for bandwidth



(a) Convergence curves for $\alpha = 1/12$. CR achieves a speed up of up to $31.7\times$, $22.0\times$, $5.2\times$ and $20.7\times$ respectively over UMW, GC, RAR and SGD.



(b) Convergence curves for $\alpha = 2/12$. CR achieves a speed up of up to $27.1\times$, $18.1\times$, $4.4\times$ and $16.8\times$ respectively over UMW, GC, RAR and SGD.



(c) Convergence curves for $\alpha = 3/12$. CR achieves a speed up of up to $22.2\times$, $13.7\times$, $3.6\times$ and $13.0\times$ respectively over UMW, GC, RAR and SGD.

Fig. 11. Convergence results for normalized error rate vs wall-clock time for linear regression over $N = 156$ workers with different straggler resiliency α.

bottleneck alleviation in large-scale commodity clusters. Furthermore, the data allocation and coding strategy provide resiliency to stragglers.

Remark 5: Till now, we have considered small-scale datasets in our experiments, which is motivated by the fact that in edge based devices with non-dedicated resources, the amount of memory available for computation shall be low.

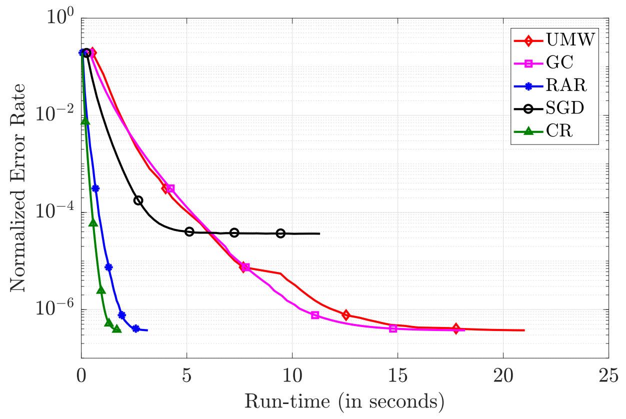


Fig. 12. Convergence curves for normalized error rate vs wall-clock time for linear regression over $N = 156$ workers and $(d, p) = (32760, 5000)$. The straggler resiliency is $\alpha = 1/4$ and the number of rounds is 50. CR achieves a speedup of up to $11.3\times$, $9.7\times$, $1.69\times$ and $6.1\times$ respectively over UMW, GC, RAR and SGD.

TABLE II
DETAILS OF THE NEURAL NETWORK ARCHITECTURE USED IN THE SIMULATIONS

Sl. No.	Parameter	Shape	Hyperparameters
1	Conv2d	$3\times16\times3\times3$	stride= 1, padding= $(1, 1)$
2	Conv2d	$16\times64\times4\times4$	stride= 1, padding= $(0, 0)$
3	Linear	64×384	-
4	Linear	384×192	-
5	Linear	192×10	-

Nevertheless, our proposed scheme CR can speedup general machine learning in cloud environments. To illustrate this point, we have carried out another experiment with a larger dataset $(d, p) = (32760, 500)$, with $(N, \alpha) = (156, 1/4)$. As illustrated by Fig. 12, CR outperforms the baseline approaches by considerable margins. Specifically, CR achieves a speedup of $11.3\times, 9.7\times, 1.69\times$ and $6.1\times$ over UMW, GC, RAR and SGD respectively.

B. Neural Networks

We carry out simulations for evaluating the benefits of CR in distributed training of neural networks with cross-entropy loss, which essentially involves non-convex and non-smooth loss functions due to variety of non-linearities such as ReLUs. For this, we consider the CIFAR10 dataset [44], which has 10 different categories of images. CIFAR10 has 50000 images while the test dataset has 10000 images. We provide the details of the neural network in Table II. We use an initial step size of 0.02, and a step decay of 0.7 at iterations 1300 and 2100. We use Glorot uniform initializer for initializing the convolutional layer weights, while for fully connected layers, we use the default initializer.

We consider a cluster of $N = 156$ servers, a resiliency of $5/12$, and $n = 12$ children per node for CR. We use a random subset of $d = 49920$ training images for training. Accuracy is reported on test dataset. We use the Pytorch library for neural network training. Furthermore, we use the computation and communication model as described earlier, where we assume $t_c = 0.05$ seconds, $a = 5\times10^{-5}$ seconds/data, and assume $a\mu = 1$.

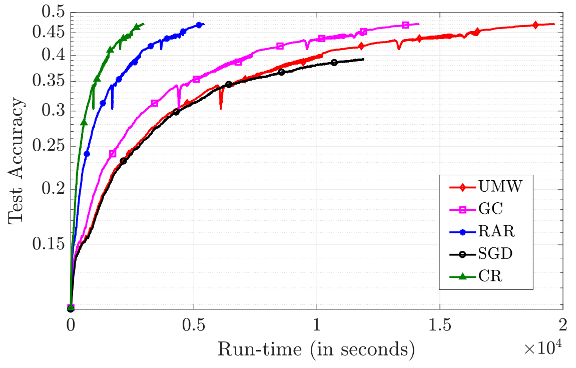


Fig. 13. Convergence curves for test accuracy vs wall-clock time for neural network training over $N = 156$ workers. The neural network model has $p \approx 120,000$ parameters. The straggler resiliency is $\alpha = 5/12$ and the number of rounds is 2500. CR achieves a speedup of up to $6.6\times$, $4.8\times$, $1.8\times$ and $4.0\times$ respectively over UMW, GC, RAR and SGD.

In Fig. 13, we plot the accuracy vs wall-clock time curves for the different approaches, where training is carried out for a total of 2500 iterations. Clearly, CR outperforms other approaches by significant margins. Particularly, CR achieves a speedup of up to $6.6\times$, $4.8\times$, $1.8\times$ and $4.0\times$ respectively over UMW, GC, RAR and SGD.

V. Conclusion

To conclude, we discussed two critical bottlenecks in scaling up Gradient Descent-based distributed learning frameworks: communication efficiency and stragglers' delays. We proposed CodedReduce (CR), that is a joint communication topology design and data set allocation strategy. CR combines the best of two existing approaches–Ring-AllReduce (RAR) and Gradient Coding (GC)–by leveraging communication parallelization of RAR and straggler resiliency of GC. Theoretically, we characterized the computation load and straggler resiliency of CR and its asymptotic expected run-time. Lastly, we empirically demonstrated that our proposed CR design achieves speedups of up to $27.2\times$ and $7.0\times$, respectively over the GC and RAR.

We also discussed that although the main goal in the proposed CR design is to recover the *exact* total gradient in each iteration of GD, one can relax this goal to inexact gradient aggregation leading to SGD-type optimization methods. We discussed how straggler resiliency and communication efficiency in GD-type methods can be improved by employing the CR design, while requiring lower computation complexity compared to naive SGD-type procedures. We note that although SGD has been widely considered for large-scale training, GD is still the prominent choice in many industry settings where one wants to make sure that the gradient computations are done completely so as not to lose even a little bit of performance. This is very critical since the model will be used by millions of people and even a slight improvement by GD would be useful. We note that CR may not be applicable in SGD settings in its current fashion. The reason is that the whole coded task allocation and execution described in the proposed CR algorithm is for the purpose of *exact* gradient recovery, i.e. GD. Such elaborate and extra gradient computation makes less sense if we relax our goal to inexact gradient recover, i.e. SGD. There are simple and complexity efficient approaches to deal with stragglers in SGD settings, such as wait for α fraction of nodes to respond, as explained in Sections III and IV. It is yet an interesting future direction to study potential coding opportunities for straggler mitigation in SGD scenarios.

Lastly, the tree structure proposed in this paper opens up new interesting directions in order to further improve the resiliency of distributed gradient aggregation schemes. For instance, given a fix set of available worker nodes, how can one find the optimal tree (i.e. optimal depth and width) in order to minimize the expected run-time.

Appendix A
Pseudo-Code for Computation Allocation Sub-Routine

Algorithm 1 Computation Allocation

```
    Input: dataset $\mathcal{D}$, $n$ workers, straggler toleration $s$,
    computation matrix $\mathbf{B} = [\mathbf{b}_1; \dots; \mathbf{b}_n] \in \mathbb{R}^{n\times k}$
    Output: data set allocation $\{\mathcal{D}_{(1)}, \dots, \mathcal{D}_{(n)}\}$ for $n$
    workers
1: procedure CompAlloc($\mathcal{D}, \mathbf{B}$)
2:     uniformly partition $\mathcal{D} = \cup_{\kappa=1}^{k} \mathcal{D}_\kappa$
3:     for worker $i \leftarrow 1$ to $n$ do
4:         $\mathcal{D}_{(i)} \leftarrow \cup_{\kappa=1}^{k} b_{i\kappa}\mathcal{D}_\kappa$ ▷ $\mathcal{D}_{(i)}$ is assigned to worker $W_i$
5:     end for
6: end procedure
```

Appendix B
Pseudo-Code for CodedReduce Scheme

See Algorithm 2.

Appendix C
Proof of Theorem 1

A. Achievability

According to the data allocation described in Algorithm 2, to be robust to any s straggling children of the master, the data set $\mathcal{D}$ is redundantly assigned to sub-trees $T(1,1), \dots, T(1,n)$ such that each data point is placed in $s+1$ sub-trees, which yields

$$|\mathcal{D}^{T(1,i)}| = \left(\frac{s+1}{n}\right) d, \quad \forall i \in [n]. \tag{12}$$

Then, nodes in layer $l = 1$ pick $r_{\mathsf{CR}} d$ data points as their corresponding data sets and similarly distribute the remaining among their children which together with (12) yields

$$\begin{aligned}|\mathcal{D}^{T(2,i)}| &= \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right) d - r_{\mathsf{CR}} d\right) \\ &= \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right) - r_{\mathsf{CR}}\right) d, \quad \forall i \in [n^2].\end{aligned}$$

By the same argument for each layer, we have

$$\begin{aligned}|\mathcal{D}^{T(L,i)}| = \left(\frac{s+1}{n}\right)\Bigg(\left(\frac{s+1}{n}\right)^{L-1} - \left(\frac{s+1}{n}\right)^{L-2} r_{\mathsf{CR}} \\ - \dots - \left(\frac{s+1}{n}\right) r_{\mathsf{CR}} - r_{\mathsf{CR}}\Bigg) d, \quad \forall i \in [n^L].\end{aligned} \tag{13}$$

Algorithm 2 CodedReduce

```
    Input: dataset $\mathcal{D}$, $(n,L)$–regular tree $T$, straggler toleration $s$ (per parent), model $\theta^{(t)}$
    Output: gradient $\mathbf{g}_{\mathcal{D}} = \sum_{\mathbf{x}\in\mathcal{D}} \nabla\ell(\theta^{(t)};\mathbf{x})$ aggregated at the master
 1: procedure CR.ALLOCATE
 2:     GC generates $\mathbf{B}$ specified by $n, s$
 3:     for $l \leftarrow 1$ to $L$ do
 4:         for $i \leftarrow 1$ to $n^{l-1}$ do
 5:             $\{\mathcal{D}^{T(l,n(i-1)+1)},\dots,\mathcal{D}^{T(l,ni)}\} = $ COMPALLOC$(\mathcal{D}_{T(l-1,i)},\mathbf{B})$
 6:         end for
 7:         for $i \leftarrow 1$ to $n^l$ do
 8:             pick $r_{\mathsf{CR}}\cdot d$ data points of $\mathcal{D}^{T(l,i)}$ as $\mathcal{D}(l,i)$
 9:             $\mathcal{D}_{T(l,i)} \leftarrow \mathcal{D}^{T(l,i)} \setminus \mathcal{D}(l,i)$
10:         end for
11:     end for
12: end procedure
13: procedure CR.EXECUTE
14:     GC generates $\mathbf{A}$ from $\mathbf{B}$
15:     all the workers compute their local partial gradients $\mathbf{g}_{\mathcal{D}(l,i)}$
16:     for $l \leftarrow L-1$ to 1 do
17:         for $i \leftarrow 1$ to $n^l$ do
18:             worker nodes $(l,i)$:
19:                 receives $[\mathbf{m}_{(l+1,n(i-1)+1)};\dots;\mathbf{m}_{(l+1,ni)}]$ from its children
20:                 uploads $\mathbf{m}_{(l,i)} = \mathbf{a}_{f(l,i)}[\mathbf{m}_{(l+1,n(i-1)+1)};\dots;\mathbf{m}_{(l+1,ni)}] + \mathbf{g}_{\mathcal{D}(l,i)}$ to its parent
21:         end for
22:     end for
23:     master node:
24:         receives $[\mathbf{m}_{(1,1)};\dots;\mathbf{m}_{(l,n)}]$ from its children
25:         recovers $\mathbf{g} = \mathbf{a}_{f(0,1)}[\mathbf{m}_{(1,1)};\dots;\mathbf{m}_{(1,n)}]$
26: end procedure
```

Putting (13) together with $|\mathcal{D}^{T(L,i)}| = r_{\mathsf{CR}}d$ yields

$$r_{\mathsf{CR}} = \frac{1}{\left(\frac{n}{s+1}\right) + \cdots + \left(\frac{n}{s+1}\right)^L}.$$

B. Optimality

In an α–resilient scheme, the master node is able to recover from any $s = \alpha n$ straggling sub-trees $T(1,1),\dots,T(1,n)$. Therefore, each data point has to be placed in at least $s+1$ of such sub-trees, which yields

$$|\mathcal{D}^{T(1,1)}| + \cdots + |\mathcal{D}^{T(1,n)}| \geq (s+1)d, \tag{14}$$

where the equality is achieved only if each data point is assigned to only $s+1$ sub-trees. Hence, we can assume the optimal scheme satisfies (14) with equality. Moving to the second layer, the following claim bounds the required redundancy assigned to sub-trees $T(2,1),\dots,T(2,n)$. Similar claim holds for any other group of the siblings in this layer.

Claim 1: The following inequality holds:

$$|\mathcal{D}^{T(2,1)}| + \cdots + |\mathcal{D}^{T(2,n)}| \geq (s+1)\left(|\mathcal{D}^{T(1,1)}| - rd\right).$$

Proof of Claim 1: First, note that $|\mathcal{D}^{T(1,1)} \setminus \mathcal{D}(1,1)| \geq |\mathcal{D}^{T(1,1)}| - rd$. If the claim does not hold, then there exists data point $\mathbf{x} \in \mathcal{D}^{T(1,1)} \setminus \mathcal{D}(1,1)$ such that $\mathbf{x}$ is placed in at most s sub-trees rooting in the node $(1,1)$, e.g. $T(2,1),\dots,T(2,s)$. Note that besides sub-tree $T(1,1)$, $\mathbf{x}$ is placed in only s more sub-trees, e.g. $T(1,2),\dots,T(1,s+1)$. Now consider a straggling pattern where $T(1,2),\dots,T(1,s+1)$ and $T(2,1),\dots,T(2,s)$ fail to return their results. Therefore, $\mathbf{x}$ is missed at the master and fails the aggregation recovery. □

By the same logic used in the above proof, Claim 1 holds for any parent node and its children, i.e. for any layer $l \in [L]$ and $i \in [n^{l-1}]$,

$$|\mathcal{D}^{T(l,n(i-1)+1)}| + \cdots + |\mathcal{D}^{T(l,ni)}| \geq (s+1) \times \left(|\mathcal{D}^{T(l-1,i)}| - rd\right). \tag{15}$$

Specifically applying (15) to layer L and noting that $|\mathcal{D}^{T(L,i)}| = |\mathcal{D}(L,i)| = rd$ for any i, we conclude that

$$rd\left(\left(\frac{n}{s+1}\right) + 1\right) \geq |\mathcal{D}^{T(L-1,1)}|.$$

We can then use the above inequality and furthermore write (15) for layer $L-1$ which results in

$$rd\left(\left(\frac{n}{s+1}\right)^2 + \left(\frac{n}{s+1}\right) + 1\right) \geq |\mathcal{D}^{T(L-2,1)}|.$$

By deriving the above inequality recursively up to the master node, we get

$$rd\left(\left(\frac{n}{s+1}\right)^{L-1} + \cdots + \left(\frac{n}{s+1}\right) + 1\right) \geq \frac{s+1}{n}d,$$

which concludes the optimality in Theorem 1.

Appendix D
Proof of Theorem 2

Let us begin with the lower bound

$$\mathbb{E}\left[T_{\mathsf{CR}}\right] \geq \frac{r_{\mathsf{CR}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\mathsf{CR}}d + \left(n(1-\alpha) - o(n) + L - 1\right)\left((1-o(1))\,t_c + o(1)\right).$$

Consider the group of siblings[1] placed in layer L whose result reaches their parent nodes first. Let $\widehat{T}$ denote the time at which the parent of such group is able to recover the partial gradient from its fastest children's computations, i.e. fastest $n-s$ of them. We also denote by $T_1,\dots,T_n$ the partial gradient computation times for the siblings. According to the random computation time model described in the paper and the computation load of CR, each T_i is shifted exponential with the shift parameter $ad_i = ar_{\mathsf{CR}}d$ and the rate parameter $\frac{\mu}{d_i} = \frac{\mu}{r_{\mathsf{CR}}d}$. Since CR is robust to any s stragglers per parent, the partial gradient computation time for any group of siblings is $T_{(n-s)}$, i.e. the $(n-s)$'th order statistics of $\{T_1,\dots,T_n\}$.

[1]A group of siblings refers to n nodes with the same parent.

In [29], authors consider coded computation scenarios in a master-worker topology where the master only needs to wait for results of the first α fraction of the workers. However, as in the scenario here, the limited bandwidth at the master only allows for one transmission at the time. From the latency analysis in [29], we have the following.

Lemma 1 (Theorem 2, [29]): With probability $1 - o(1)$, *we have*

$$\widehat{T} \geq T_{(n-s)} + (n(1-\alpha) - o(n))\, t_c. \tag{16}$$

Now, conditioned on the event in (16) we can write

$$\begin{aligned}\mathbb{E}[T_{\mathsf{CR}}] &\geq \left(\mathbb{E}\left[T_{(n-s)}\right] + (n(1-\alpha)-o(n))\,t_c\right)(1-o(1)) \\ &\quad + \left(\mathbb{E}\left[T_{(n-s)}\right] + Lt_c\right)o(1) \\ &\geq \mathbb{E}\left[T_{(n-s)}\right] \\ &\quad + (n(1-\alpha)-o(n)+L-1)(1-o(1))\,t_c \\ &\overset{(a)}{\geq} \frac{r_{\mathsf{CR}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\mathsf{CR}}d \\ &\quad + (n(1-\alpha)-o(n)+L-1)(1-o(1))\,t_c + o(1),\end{aligned}$$

where inequality (a) uses the fact that $\mathbb{E}\left[T_{(n-s)}\right] = \frac{r_{\mathsf{CR}}d}{\mu}(H_n - H_s) + ar_{\mathsf{CR}}d$ and $\log(i) < H_i = 1 + \frac{1}{2} + \cdots + \frac{1}{i} < \log(i+1)$ for any positive integer i.

To derive upper bound on $\mathbb{E}[T_{\mathsf{CR}}]$, that is

$$\mathbb{E}[T_{\mathsf{CR}}] \leq \frac{r_{\mathsf{CR}}d}{\mu}\log\left(\frac{1}{\alpha}\right) + ar_{\mathsf{CR}}d + n(1-o(1))\,Lt_c + o(1),$$

we prove the following concentration inequality on the computation time for any group of siblings.

Lemma 2: Let $T_1, \ldots, T_n$ *denote i.i.d. exponential random variables with constant rate* $\lambda = \Theta(1)$. *For* $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$ *and constant* $\alpha = \frac{s}{n}$, *we have the following concentration bound for the order statistics* $T_{(n-s)}$:

$$\mathbb{P}\left[T_{(n-s)} - \mathbb{E}\left[T_{(n-s)}\right] \geq \varepsilon\right] \leq e^{-\Theta(\sqrt{n})}. \tag{17}$$

Proof of Lemma 2: Given i.i.d. exponentials $T_1, \ldots, T_n \sim \exp(\lambda)$, we can write the successive differences of order statistics as independent exponentials. That is, we have

$$\begin{aligned}T_{(1)} &= E_1 \sim \exp\left(\frac{\lambda}{n}\right), \\ T_{(2)} - T_{(1)} &= E_2 \sim \exp\left(\frac{\lambda}{n-1}\right), \\ &\vdots \\ T_{(n-s)} - T_{(n-s-1)} &= E_{n-s} \sim \exp\left(\frac{\lambda}{s+1}\right), \\ &\vdots \\ T_{(n)} - T_{(n-1)} &= E_n \sim \exp(\lambda),\end{aligned}$$

where E_i's are independent. Thus, $T_{(n-s)} = \sum_{i=1}^{n-s} E_i$. We have the following for independent exponentials E_i's and $\lambda = \Theta(1)$:

$$\begin{aligned}\mathbb{E}\left[|E_i|^k\right] &= \mathbb{E}\left[E_i^k\right] \\ &= \left(\frac{\lambda}{n-i+1}\right)^k k! \\ &= \frac{1}{2}\mathbb{E}\left[E_i^2\right]\left(\frac{\lambda}{n-i+1}\right)^{k-2} k! \\ &\leq \frac{1}{2}\mathbb{E}\left[E_i^2\right] B^{k-2} k!,\end{aligned}$$

for $B = \frac{\lambda}{s} = \frac{\lambda}{\alpha n} = \Theta\left(\frac{1}{n}\right)$. Moreover,

$$\begin{aligned}\sum_{i=1}^{n-s}\mathbb{E}\left[E_i^2\right] &= 2\lambda^2\left(\frac{1}{n^2} + \cdots + \frac{1}{(s+1)^2}\right) \\ &\leq 2\lambda^2 \cdot \frac{n-s}{s^2} \\ &= \frac{2\lambda^2(1-\alpha)}{\alpha^2}\cdot\frac{1}{n} \\ &= \Theta\left(\frac{1}{n}\right).\end{aligned}$$

According to Bersterin's Lemma (See Lemma 3), for $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$ we have

$$\begin{aligned}&\mathbb{P}\left[T_{(n-s)} - \mathbb{E}\left[T_{(n-s)}\right] \geq \varepsilon\right] \\ &\leq \exp\left(-\frac{\varepsilon^2}{2\left(\sum_{i=1}^{n-s}\mathbb{E}\left[E_i^2\right] + \varepsilon B\right)}\right) \\ &\leq \exp\left(-\frac{\varepsilon^2}{2\left(\Theta\left(\frac{1}{n}\right) + \varepsilon\Theta\left(\frac{1}{n}\right)\right)}\right) \\ &= e^{-\Theta(\sqrt{n})}.\end{aligned}$$

□

As described in Section III-A, in the proposed CR scheme all the worker nodes start their assigned partial gradient computations simultaneously; each parent waits for enough number of children to receive their results; combines with its partial computation and sends the result up to its parent. To upper bound the total aggregation time T_{CR}, one can separate all the local computations from the communications. Let T_{comp} denote the time at which enough number of workers have executed their local gradient computations and no more local computation is needed for the final gradient recovery. Moreover, we assume that all the communications from children to parent are pipe-lined. Hence, we have $\mathbb{E}[T_{\mathsf{CR}}] \leq \mathbb{E}[T_{\text{comp}}] + L(n-s)t_c$. To bound the computation time T_{comp}, consider the following event which keeps the local computation times for *all* the N/n groups of siblings concentrated below their average deviated by $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$:

$$\mathcal{E}_1 := \left\{T_{(n-s)}^{gr} \leq \mathbb{E}\left[T_{(n-s)}^{gr}\right] + \varepsilon \text{ for all the } N/n \text{ groups } gr\right\},$$

where a group gr is a collection of n children with the same parent, i.e. there are N/n groups in the (n, L)–regular tree. For a group gr, $\{T_1^{gr}, \ldots, T_n^{gr}\}$ denote the random run-times of the nodes in the group and $T_{(n-s)}^{gr}$ represents its $(n-s)$'th order statistics. Clearly,

$$\mathbb{E}[T_{\text{comp}}|\mathcal{E}_1] \leq \mathbb{E}\left[T_{(n-s)}\right] + o(1). \tag{18}$$

Now let $\widetilde{T}$ denote the computation time corresponding to the slowest group of siblings, i.e.

$$\widetilde{T} := \max_{\text{over all } N/n \text{ groups } gr} T_{(n-s)}^{gr}.$$

Consider the following event:

$$\mathcal{E}_2 := \left\{\widetilde{T} > \Theta(\log n)\right\}.$$

We can write

$$\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2^c\right] \le \Theta(\log n), \tag{19}$$

and

$$\begin{aligned}
&\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2\right]\mathbb{P}\left[\mathcal{E}_2\right] \\
&\le \mathbb{E}\left[\widetilde{T}|\mathcal{E}_1^c \cap \mathcal{E}_2\right]\mathbb{P}\left[\mathcal{E}_2\right] \\
&= \mathbb{E}\left[\widetilde{T}|\widetilde{T} \le \Theta(\log n)\right]\mathbb{P}\left[\widetilde{T} \le \Theta(\log n)\right] \\
&\le \mathbb{E}\left[\widetilde{T}\right] \\
&\le \mathbb{E}\left[T_{\max}\right] \\
&= \frac{r_{\mathsf{CR}} d}{\mu} H_N + a r_{\mathsf{CR}} d \\
&= \Theta\left(\log N\right) \\
&= L\Theta\left(\log n\right).
\end{aligned} \tag{20}$$

In the above derivation, $T_{\max}$ denotes the largest computation time over all the N nodes. Putting (19) and (20) together, we can write

$$\begin{aligned}
\mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c\right] &= \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2\right]\mathbb{P}\left[\mathcal{E}_2\right] \\
&\quad + \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c \cap \mathcal{E}_2^c\right]\mathbb{P}\left[\mathcal{E}_2^c\right] \\
&\le \Theta\left(\log n\right).
\end{aligned} \tag{21}$$

Moreover, using union bound on the N/n groups of workers, we derive the following inequality.

$$\begin{aligned}
\mathbb{P}\left[\mathcal{E}_1^c\right] &\le \frac{N}{n}\mathbb{P}\left[T_{(n-s)} \ge \mathbb{E}\left[T_{(n-s)}\right] + \varepsilon\right] \\
&\le \Theta\left(n^{L-1}\right) e^{-\Theta\left(\sqrt{n}\right)}.
\end{aligned} \tag{22}$$

Putting (18), (21) and (22) together, we have

$$\begin{aligned}
\mathbb{E}\left[T_{\text{comp}}\right] &= \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1\right]\mathbb{P}\left[\mathcal{E}_1\right] + \mathbb{E}\left[T_{\text{comp}}|\mathcal{E}_1^c\right]\mathbb{P}\left[\mathcal{E}_1^c\right] \\
&\le \mathbb{E}\left[T_{(n-s)}\right] + \varepsilon + \Theta\left(\log n\right)\Theta\left(n^{L-1}\right) e^{-\Theta\left(\sqrt{n}\right)} \\
&= \mathbb{E}\left[T_{(n-s)}\right] + o(1) \\
&= \frac{r_{\mathsf{CR}} d}{\mu}\left(H_n - H_s\right) + a r_{\mathsf{CR}} d + o(1).
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathbb{E}\left[T_{\mathsf{CR}}\right] &\le \mathbb{E}\left[T_{\text{comp}}\right] + Ln(1-\alpha)t_c \\
&= \frac{r_{\mathsf{CR}} d}{\mu}\left(H_n - H_s\right) + a r_{\mathsf{CR}} d + Ln(1-\alpha)t_c + o(1) \\
&\le \frac{r_{\mathsf{CR}} d}{\mu}\log\left(\frac{1}{\alpha}\right) + a r_{\mathsf{CR}} d + n\left(1 - o(1)\right)Lt_c + o(1),
\end{aligned}$$

which completes the proof.

Lemma 3 (Bernstein's Inequality): Suppose $E_1, \ldots, E_m$ *are independent random variables such that*

$$\mathbb{E}\left[|E_i|^k\right] \le \frac{1}{2}\mathbb{E}\left[E_i^2\right] B^{k-2} k!,$$

for some $B > 0$ *and every* $i = 1, \ldots, m,\ k \ge 2$. *Then, for* $\varepsilon > 0$,

$$\begin{aligned}
&\mathbb{P}\left[\sum_{i=1}^m E_i - \sum_{i=1}^m \mathbb{E}\left[E_i\right] \ge \varepsilon\right] \\
&\qquad \le \exp\left(-\frac{\varepsilon^2}{2\left(\sum_{i=1}^m \mathbb{E}\left[E_i^2\right] + \varepsilon B\right)}\right).
\end{aligned}$$

Acknowledgment

The authors sincerely thank the editor and all the reviewers for their valuable feedback and detailed comments. 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 U.S. Government.

References

[1] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Tree gradient coding," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2019, pp. 2808–2812.

[2] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, Jan. 2012.

[3] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2595–2603.

[4] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," 2016, *arXiv:1604.00981*. [Online]. Available: http://arxiv.org/abs/1604.00981

[5] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 693–701.

[6] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.

[7] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. OSDI*, vol. 14, 2014, pp. 571–582.

[8] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, 2016, pp. 265–283.

[9] G. Cong, O. Bhardwaj, and M. Feng, "An efficient, distributed stochastic gradient descent algorithm for deep-learning applications," in *Proc. 46th Int. Conf. Parallel Process. (ICPP)*, Aug. 2017, pp. 11–20.

[10] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, pp. 117–124, Feb. 2009.

[11] P. Patarasuk and X. Yuan, "Bandwidth efficient all-reduce operation on tree topologies," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Jun. 2007, pp. 1–8.

[12] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, 2005.

[13] K. Kandalla, H. Subramoni, A. Vishnu, and D. K. Panda, "Designing topology-aware collective communication algorithms for large scale infiniband clusters: Case studies with scatter and gather," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Forum (IPDPSW)*, Apr. 2010, pp. 1–8.

[14] *Bringing HPC Techniques to Deep Learning*. Accessed: Jan. 1, 2019. [Online]. Available: https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

[15] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," 2018, *arXiv:1802.05799*. [Online]. Available: http://arxiv.org/abs/1802.05799

[16] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, "How to scale distributed deep learning?" in *Proc. Syst. Workshop NIPS*, 2016, pp. 1–16.

[17] 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," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8056–8067.

[18] M. Yu *et al.*, "GradiVeQ: Vector quantization for bandwidth-efficient gradient aggregation in distributed CNN training," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5129–5139.

[19] J. Sun, T. Chen, G. B. Giannakis, Q. Yang, and Z. Yang, "Lazily aggregated quantized gradient innovation for communication-efficient federated learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Oct. 23, 2020, doi: 10.1109/TPAMI.2020.3033286.

[20] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[21] Y. Zhao *et al.*, "Efficient communications in training large scale neural networks," in *Proc. Thematic Workshops ACM Multimedia*, 2017, pp. 110–116.

[22] X. Zhang, Y. Wu, and C. Zhao, "MrHeter: Improving MapReduce performance in heterogeneous environments," *Cluster Comput.*, vol. 19, no. 4, pp. 1691–1701, Dec. 2016.

[23] Q. Pu *et al.*, "Low latency geo-distributed data analytics," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 421–434, 2015.

[24] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective straggler mitigation: Attack of the clones," in *Proc. NSDI*, vol. 13, 2013, pp. 185–198.

[25] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 599–600, Jun. 2014.

[26] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, Feb. 2016.

[27] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[28] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot' computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.

[29] A. Reisizadeh and R. Pedarsani, "Latency analysis of coded computation schemes over wireless networks," in *Proc. 55th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Oct. 2017, pp. 1256–1263.

[30] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4403–4413.

[31] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5440–5448.

[32] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 5610–5619.

[33] Q. Yu, S. Li, N. Raviv, M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proc. AISTATS*, 2019, pp. 1215–1225.

[34] K. G. Narra, Z. Lin, M. Kiamari, S. Avestimehr, and M. Annavaram, "Slack squeeze coded computing for adaptive straggler mitigation," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2019, pp. 1–16.

[35] H. V. K. G. Narra, Z. Lin, G. Ananthanarayanan, S. Avestimehr, and M. Annavaram, "Collage inference: Using coded redundancy for lowering latency variation in distributed image classification systems," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 453–463.

[36] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jul. 2019, pp. 301–310.

[37] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, "Coded computing for distributed machine learning in wireless edge network," in *Proc. IEEE 90th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2019, pp. 1–6.

[38] J. Xu, S.-L. Huang, L. Song, and T. Lan, "Live gradient compensation for evading stragglers in distributed learning," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 3368–3376.

[39] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht, "Gradient descent only converges to minimizers," in *Proc. Conf. Learn. Theory*, 2016, pp. 1246–1257.

[40] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 826–834.

[41] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 2408–2412.

[42] S. Li, S. M. Mousavi Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 857–866.

[43] I. Guyon, J. Li, T. Mader, P. A. Pletscher, G. Schneider, and M. Uhr, "Competitive baseline methods set new standards for the nips 2003 feature selection benchmark," *Pattern Recognit. Lett.*, vol. 28, no. 12, pp. 1438–1444, 2007.

[44] A. Krizhevsky, V. Nair, and G. Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

Amirhossein Reisizadeh received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2014, and the M.S. degree in electrical engineering from the University of California at Los Angeles (UCLA) in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of California at Santa Barbara (UCSB). He was a Finalist in Qualcomm Innovation Fellowship Program in 2019. He is interested in using information and coding-theoretic concepts to develop fast and efficient algorithms for large-scale machine learning, distributed computing, and optimization.

Saurav Prakash (Graduate Student Member, IEEE) received the Bachelor of Technology degree in electrical engineering from the Indian Institute of Technology (IIT), Kanpur, India, in 2016. He is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of Southern California (USC), Los Angeles. His research interests include information theory and data analytics with applications in large-scale machine learning and edge computing. He was one of Viterbi-India fellows in summer 2015. He is one of the recipients of Qualcomm Innovation Fellowship 2021. He also received Annenberg Graduate Fellowship in 2016.

Ramtin Pedarsani (Senior Member, IEEE) received the B.Sc. degree in electrical engineering from the University of Tehran, Tehran, Iran, in 2009, the M.Sc. degree in communication systems from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 2011, and the Ph.D. degree from the University of California, Berkeley, in 2015. He is currently an Assistant Professor with the ECE Department, University of California, Santa Barbara. His research interests include machine learning, information and coding theory, networks, and transportation systems. He was a recipient of the IEEE International Conference on Communications (ICC) Best Paper Award in 2014.

Amir Salman Avestimehr (Fellow, IEEE) received the B.S. degree in electrical engineering from Sharif University of Technology in 2003 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley.

He is currently a Professor and the Director of the Information Theory and Machine Learning (vITAL) Research Laboratory, Electrical and Computer Engineering Department, University of Southern California. His research interests include information theory, coding theory, and large-scale distributed computing and machine learning. He has received a number of awards for his research, including James L. Massey Research and Teaching Award from IEEE Information Theory Society, the Information Theory Society and Communication Society Joint Paper Award, the Presidential Early Career Award for Scientists and Engineers (PECASE) from the White House, the Young Investigator Program (YIP) Award from U.S. Air Force Office of Scientific Research, the National Science Foundation CAREER Award, David J. Sakrison Memorial Prize, and several best paper awards at conferences. He is the General Co-Chair of the 2020 International Symposium on Information Theory (ISIT). He has been an Associate Editor for IEEE TRANSACTIONS ON INFORMATION THEORY.