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

Amirhossein Reisizadeh*, Saurav Prakash†, Ramtin Pedarsani*, Amir Salman Avestimehr†
* University of California, Santa Barbara † University of Southern California

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 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. Furthermore, we empirically evaluate the performance of our proposed CR design over Amazon EC2 and demonstrate that it achieves speedups of up to $18.9\times$ and $7.9\times$, respectively over the benchmarks GC and RAR.

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., [1]–[7]).

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 [1]–[6]. 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 [3], [8], they face major system bottlenecks due to (1) bandwidth congestion at the master due to concurrent communications from the workers to the master [9]; and (2) the delays caused by slow workers or stragglers that significantly increase the run-time [4].

To alleviate the communication bottleneck in distributed learning, various bandwidth efficient strategies have been proposed [10]–[12]. Particularly, Ring-AllReduce (RAR) [9] 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 [10], 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 [13]–[15]. More recent approaches to mitigate bandwidth bottleneck in distributed gradient aggregation include compression and quantization of the gradients [16], [17].

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. Straggler bottleneck becomes even more significant as the cluster size increases [18], [19].

One approach to mitigate stragglers in distributed computation is to introduce computational redundancy via replication. [20] proposes to replicate the straggling task on other available nodes. In [21], the authors propose a partial data replication for robustness. Other relevant replication based strategies have been proposed in [22]–[24]. Recently, coding theoretic approaches have also been proposed for straggler mitigation [25]–[31]. Specifically, Gradient Coding (GC) [32] 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

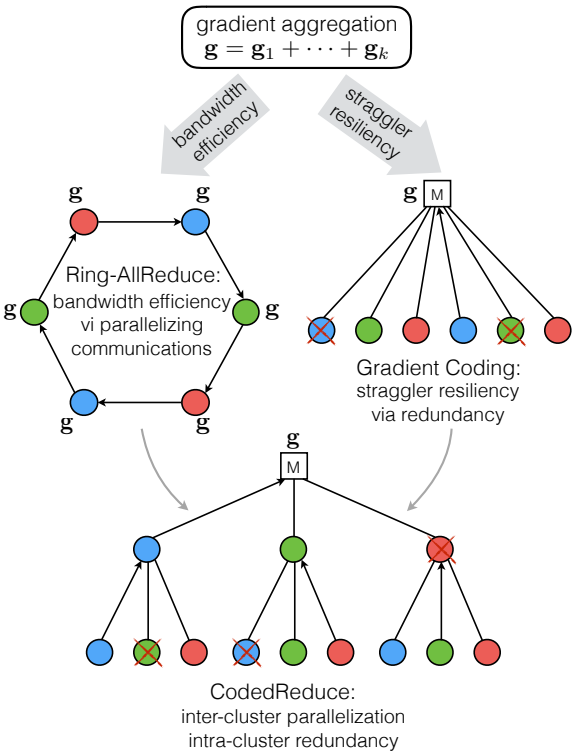


Fig. 1: Illustration of RAR, GC and CR: In RAR, workers communicate only with their neighbors on a ring, which results in high bandwidth utilization; however, RAR is prone to stragglers. GC is robust to stragglers by doing redundant computations at workers; however, GC imposes bandwidth bottleneck at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

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, where it was shown to achieve significant speedups over the classical uncoded master-worker setup [32].

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 introduces 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 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. Secondly, 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.

Additional 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.

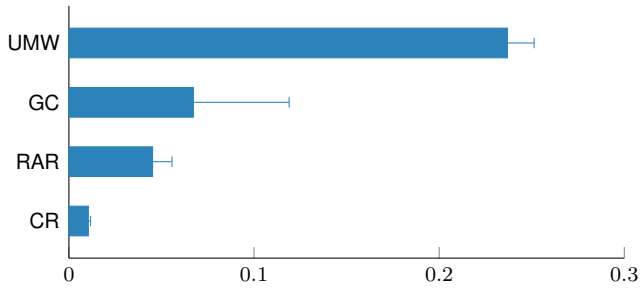


Fig. 2: Average iteration time for gradient aggregation in different schemes CR, RAR, GC and UMW: Training a linear model is implemented on a cluster of $N = 84$ `t2.micro` instances using `Python` with `mpi4py`.

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.

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 $\mathcal{D} = \{\mathbf{x}_j \in \mathbb{R}^{p+1} : j = 1, \cdots, d\}$, the goal is to solve the following optimization problem:

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

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. 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 one only 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.

B. Ring-AllReduce

In AllReduce-type aggregation schemes, the data set is uniformly distributed over N worker nodes $\{W_1, \cdots, W_N\}$ 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 $\mathbf{v}_{1,i}, \cdots, \mathbf{v}_{N,i}$. 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 [9].

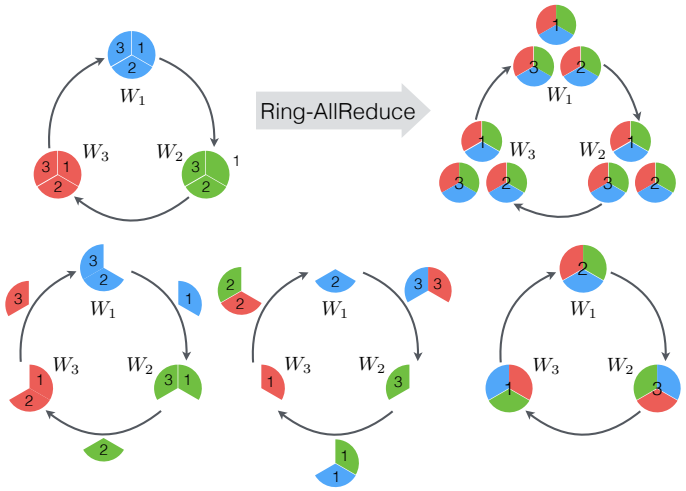


Fig. 3: Illustration of communication strategy in RAR for $N = 3$ workers.

C. Gradient Coding

Gradient Coding (GC) [32] was recently proposed to to provide straggler resiliency in a master-worker topology with one master node and N distributed worker nodes $\{W_1, \cdots, W_N\}$ 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 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 $\mathbf{g} = 2\left(\frac{1}{2}\mathbf{g}_1 + \mathbf{g}_2\right) - (\mathbf{g}_2 - \mathbf{g}_3)$.

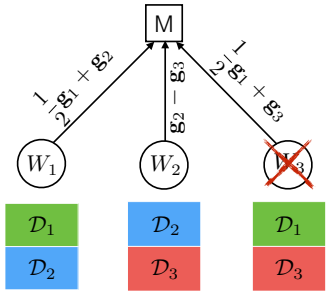


Fig. 4: Illustration of data allocation and communication strategy in GC for $N = 3$ workers.

In general, to be robust to *any* $S \in [N] = \{1, \cdots, N\}$ stragglers, GC uniformly partitions the data set $\mathcal{D}$ to $\{\mathcal{D}_1, \cdots, \mathcal{D}_k\}$ (e.g. $k = N$) with corresponding partial gradients $\mathbf{g}_1, \cdots, \mathbf{g}_k$ and distributes them redundantly among the workers such that each

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	$r^{1/L}$	$\Theta\left(N^{1-1/L}\right)$

partition is placed in $S+1$ workers, thus achieving a computation load of $r_{\mathsf{GC}} = \frac{S+1}{N}$. Let matrix $\mathbf{G} = [\mathbf{g}_1, \cdots, \mathbf{g}_k]^\top \in \mathbb{R}^{k\times p}$ 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 $\mathbf{B} = [\mathbf{b}_1; \cdots; \mathbf{b}_n] \in \mathbb{R}^{N\times k}$ 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 $\mathbf{g} = \mathbf{a}_f\mathbf{B}\mathbf{G}$ where the row vector $\mathbf{a}_f \in \mathbb{R}^{1\times N}$ corresponds to a particular set of S stragglers and $\mathbf{A} = [\mathbf{a}_1; \cdots; \mathbf{a}_F]$ 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 $0 \le r \le 1$, 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.

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).

COMPALLOC: For specified n and s, GC (Algorithm 2 in [32]) constructs the encoding matrix $\mathbf{B} = [\mathbf{b}_1; \cdots; \mathbf{b}_n] = [b_{i\kappa}]$. 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 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 [32]).

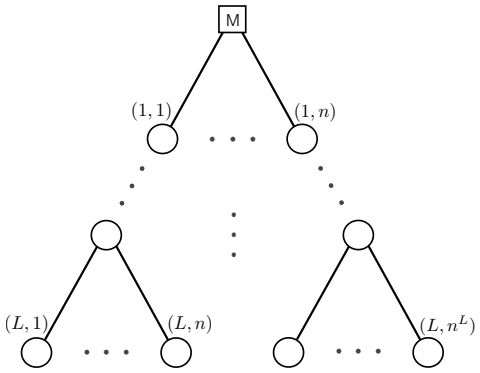


Fig. 5: (n, L)–regular tree topology.

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.

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).

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 form 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 via proper row in the decoding matrix $\mathbf{A}$, e.g., parent node $(L-1,1)$ recovers from its children's messages $[\mathbf{m}_{(L,1)}; \cdots ; \mathbf{m}_{(L,n)}]$ 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
$$\mathbf{m}_{(L-1,1)} = \mathbf{a}_{f(L-1,1)}[\mathbf{m}_{(L,1)}; \cdots ; \mathbf{m}_{(L,n)}] + \mathbf{g}_{\mathcal{D}(L-1,1)}.$$
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 toy example to better illustrate the proposed CR scheme.

Example 2 (CodedReduce)**.** Consider a $(3,2)$–regular tree with $N = 12$ nodes and $s = 1$ stragglers 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. $\mathcal{D}(1,1) = \mathcal{D}_1^{T(1,1)} \cup \mathcal{D}_2^{T(1,1)}$

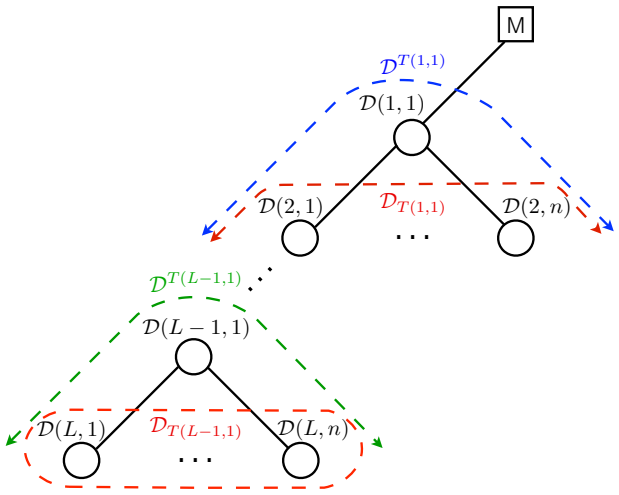


Fig. 6: Illustration of task allocation in CR.

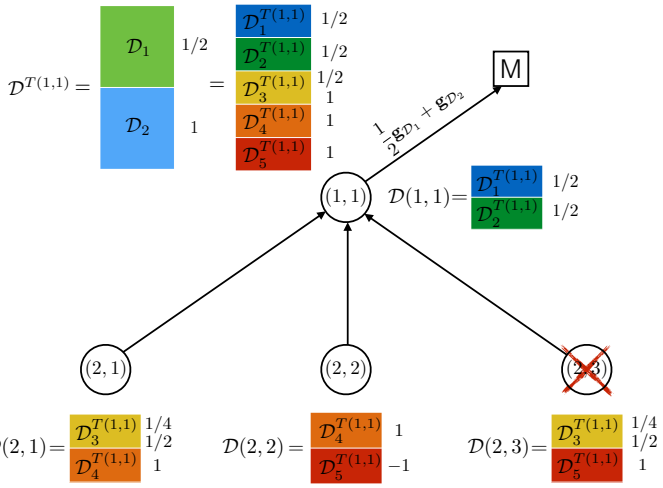


Fig. 7: Illustration of data allocation and communication strategy in CR for a $(3,2)$–regular tree.

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 $(1,3)$:

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

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.

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 establishes 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 $r_{\mathsf{GC}} = \frac{S+1}{N} = \frac{\alpha N+1}{N} \approx \alpha$ fraction of the data set, while CR considerably reduces it to $r_{\mathsf{CR}} = 1/\sum_{l=1}^{L}\left(\frac{n}{\alpha n+1}\right)^l \approx \alpha^L$. For $\alpha = 0.5$ as an instance, GC reduces the computation load $7\times$ by rearranging the nodes in 1 layer to 3 layers.

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. Motivated by this, we propose a random computation time model to analyze the latency performance of CR. We consider shifted exponential distribution for workers' computation time which is used in several prior works [33]–[35]. 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:

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

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.

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 α satisfies the followings:*

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

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

Remark 3. Theorem 2 implies that the expected run-time of the proposed CR algorithm breaks down to two terms: $\mathbb{E}\left[T_{\mathsf{CR}}\right] = \Theta(1) + \Theta(n)$, where the constant and scaling terms 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 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 experimental results conducted over Amazon EC2, demonstrating the gains of CR over baseline approaches.

We conduct two experiments as follows. Firstly, we train the real data set GISETTE [36] by solving a logistic regression problem using GD over a cluster of $N = 156$ workers. 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.

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

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

where the true model θ_* and features $\mathbf{x}_j(1:p) = [\mathbf{x}_j(1); \cdots; \mathbf{x}_j(p)]$ are drawn randomly from $\mathbb{R}^p$ and z_j is a standard Gaussian noise. We consider the following normalized error rate:

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

Next, we plot the rates defined in (9) and (11) for our logistic regression and linear regression experiments respectively, as functions of wall-clock training time. For both experiments, we used `t2.micro` instances for master and all workers. For the distributed implementation, we used `Python` with `mpi4py` package. The results from our experiments are presented in Figures 8 and 9, where we CR is compared with three benchmarks:

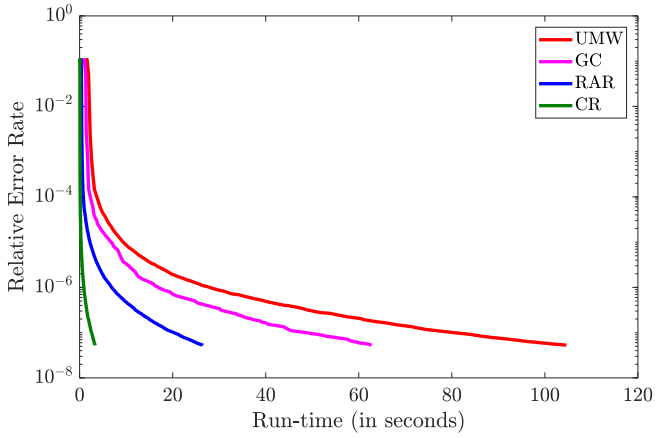


Fig. 8: Convergence plots for relative error rate for logistic regression. CR achieves a speedup of up to $31.4\times$, $18.8\times$ and $7.9\times$ respectively over UMW, GC, and RAR.

(1) 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 is optimized empirically to obtain the best configuration.

(2) Ring-AllReduce (RAR): The data set is uniformly partitioned over the workers and the MPI function `MPI_Allreduce()` is used for gradient aggregation.

(3) Gradient Coding (GC): We implement GC as described in Section II-C. The straggler parameter S is empirically optimized to obtain the configuration with best performance.

(4) 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.

We summarize our observations and conclusions from the experiments in the following remarks:

• CR achieves significant speed ups over the benchmark schemes. As demonstrated in Figure 8, CR achieves a speedup of up to $31.4\times$, $18.9\times$ and $7.9\times$ respectively over UMW, GC, and RAR for logistics regression. In the linear regression problem, the corresponding speedups are $28.9\times$, $17.0\times$ and $6.8\times$, respectively as demonstrated in Figure 9.

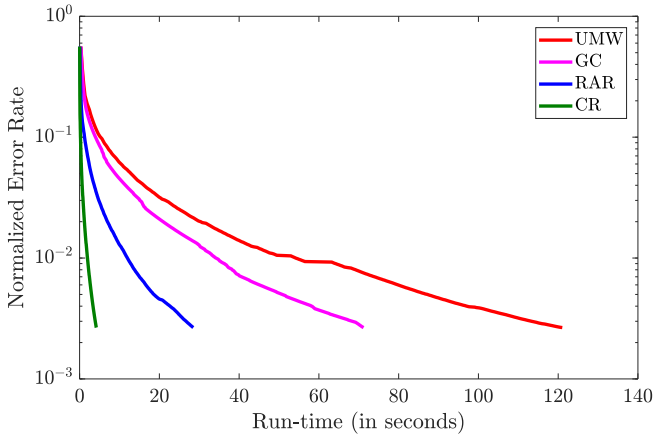


Fig. 9: Convergence plots for normalized error rate for linear regression. CR achieves a speedup of up to $28.9\times$, $17.0\times$ and $6.8\times$ respectively over UMW, GC, and RAR.

• These experiments complement the theoretical gains of CR that have been established earlier. As demonstrated in the figures, although GC improves the overall time in comparison to UMW, it still suffers from bandwidth bottleneck at the master. On the other hand, RAR is bottlenecked by straggling nodes which reduce its performance. CR however, jointly improves bandwidth utilization and straggler toleration, leading to significant speedups in distributed machine learning problems while maintaining the same generalization error.

Lastly, we provide similar experimental results for a smaller cluster with $N = 84$ workers in Appendix E.

REFERENCES

[1] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal Distributed Online Prediction Using Mini-Batches," *Journal of Machine Learning Research*, vol. 13, no. Jan, pp. 165–202, 2012.

[2] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized Stochastic Gradient Descent," in *Advances in Neural Information Processing Systems*, pp. 2595–2603, 2010.

[3] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," *arXiv preprint arXiv:1604.00981*, 2016.

[4] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent," in *Advances in Neural Information Processing Systems*, pp. 693–701, 2011.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large Scale Distributed Deep Networks," in *Advances in neural information processing systems*, pp. 1223–1231, 2012.

[6] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System.," in *OSDI*, vol. 14, pp. 571–582, 2014.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: A System for Large-Scale Machine Learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[8] G. Cong, O. Bhardwaj, and M. Feng, "An efficient, distributed stochastic gradient descent algorithm for deep-learning applications," in *Parallel Processing (ICPP), 2017 46th International Conference on*, pp. 11–20, IEEE, 2017.

[9] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[10] P. Patarasuk and X. Yuan, "Bandwidth Efficient All-reduce Operation on Tree Topologies," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8, IEEE, 2007.

[11] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[12] 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 *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–8, IEEE, 2010.

[13] A. Gibiansky, "Bringing HPC Techniques to Deep Learning," tech. rep., Baidu Research, Tech. Rep., 2017, http://research. baidu. com/bringing-hpc-techniques-deep-learning/. Bingjing Zhang TESTS & CERTIFICATIONS IBM Certified Database Associate-DB2 Universal Database, 2017.

[14] A. Sergeev and M. Del Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[15] P. H. Jin, Q. Yuan, F. Iandola, and K. Keutzer, "How to scale distributed deep learning?," *ML Systems Workshop, NIPS*, 2016.

[16] 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 *Advances in Neural Information Processing Systems*, pp. 8056–8067, 2018.

[17] M. Yu, Z. Lin, K. Narra, S. Li, Y. Li, N. S. Kim, A. Schwing, M. Annavaram, and S. Avestimehr, "GradiVeQ: Vector Quantization for Bandwidth-Efficient Gradient Aggregation in Distributed CNN Training," in *Advances in Neural Information Processing Systems*, pp. 5129–5139, 2018.

[18] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[19] Y. Zhao, L. Wang, W. Wu, G. Bosilca, R. Vuduc, J. Ye, W. Tang, and Z. Xu, "Efficient Communications in Training Large Scale Neural Networks," in *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pp. 110–116, ACM, 2017.

[20] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," in *OSDI*, vol. 8, p. 7, 2008.

[21] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low Latency Geo-distributed Data Analytics," in *ACM SIGCOMM Computer Communication Review*, vol. 45, pp. 421–434, ACM, 2015.

[22] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Effective Straggler Mitigation: Attack of the Clones.," in *NSDI*, vol. 13, pp. 185–198, 2013.

[23] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, pp. 599–600, ACM, 2014.

[24] N. B. Shah, K. Lee, and K. Ramchandran, "When Do Redundant Requests Reduce Latency?," *IEEE Transactions on Communications*, vol. 64, no. 2, pp. 715–722, 2016.

[25] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[26] S. Dutta, V. Cadambe, and P. Grover, ""Short-Dot": Computing Large Linear Transforms Distributedly Using Coded Short Dot Products," in *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

[27] 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.

[28] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication," in *Advances in Neural Information Processing Systems*, pp. 4403–4413, 2017.

[29] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler Mitigation in Distributed Optimization Through Data Encoding," in *Advances in Neural Information Processing Systems*, pp. 5440–5448, 2017.

[30] M. Ye and E. Abbe, "Communication-Computation Efficient Gradient Coding," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 5610–5619, 10–15 Jul 2018.

[31] Q. Yu, S. Li, N. Raviv, M. Kalan, M. Soltanolkotabi, and A. S. Avestimehr, "Lagrange Coded Computing: Optimal Design for Resiliency, Security and Privacy," *To appear in Proceedings of 2019 AISTATS*, 2019.

[32] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning ," in *International Conference on Machine Learning*, pp. 3368–3376, 2017.

[33] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *INFOCOM, 2014 Proceedings IEEE*, pp. 826–834, IEEE, 2014.

[34] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded Computation over Heterogeneous Clusters," in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2408–2412, IEEE, 2017.

[35] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 857–866, IEEE, 2018.

[36] 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 recognition letters*, vol. 28, no. 12, pp. 1438–1444, 2007.

APPENDIX A
PSEUDO-CODE FOR COMPUTATION ALLOCATION SUB-ROUTINE

Algorithm 1 Computation Allocation

Input: dataset $\mathcal{D}$, n workers, straggler toleration s, computation matrix $\mathbf{B} = [\mathbf{b}_1; \cdots ; \mathbf{b}_n] \in \mathbb{R}^{n \times k}$
Output: data set allocation $\{\mathcal{D}_{(1)}, \cdots , \mathcal{D}_{(n)}\}$ for n workers

procedure COMPALLOC($\mathcal{D}, \mathbf{B}$)
  uniformly partition $\mathcal{D} = \cup_{\kappa=1}^{k} \mathcal{D}_\kappa$
  for worker $i \leftarrow 1$ to n **do**
    $\mathcal{D}(i) \leftarrow \cup_{\kappa=1}^{k} b_{i\kappa} \mathcal{D}_\kappa$ ▷ $\mathcal{D}_{(i)}$ is assigned to worker W_i
  end for
end procedure

APPENDIX B
PSEUDO-CODE FOR CODEDREDUCE SCHEME

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

procedure CR.ALLOCATE
  GC generates $\mathbf{B}$ specified by n, s
  for $l \leftarrow 1$ to L **do**
    for $i \leftarrow 1$ to n^{l-1} **do**
      $\{\mathcal{D}^{T(l,n(i-1)+1)}, \cdots , \mathcal{D}^{T(l,ni)}\} = \text{COMPALLOC}(\mathcal{D}_{T(l-1,i)}, \mathbf{B})$
    end for
    for $i \leftarrow 1$ to n^l **do**
      pick $r_{\mathsf{CR}} \cdot d$ data points of $\mathcal{D}^{T(l,i)}$ as $\mathcal{D}(l,i)$
      $\mathcal{D}_{T(l,i)} \leftarrow \mathcal{D}^{T(l,i)} \setminus \mathcal{D}(l,i)$
    end for
  end for
end procedure
procedure CR.EXECUTE
  GC generates $\mathbf{A}$ from $\mathbf{B}$
  all the workers compute their local partial gradients $\mathbf{g}_{\mathcal{D}(l,i)}$
  for $l \leftarrow L-1$ to 1 **do**
    for $i \leftarrow 1$ to n^l **do**
      worker nodes (l,i):
        receives $[\mathbf{m}_{(l+1,n(j-1)+1)}; \cdots ; \mathbf{m}_{(l+1,nj)}]$ from its children
        receives $[\mathbf{m}_{(l+1,n(i-1)+1)}; \cdots ; \mathbf{m}_{(l+1,ni)}]$ from its children
        uploads $\mathbf{m}_{(l,i)} = \mathbf{a}_{f(l,i)}[\mathbf{m}_{(l+1,n(i-1)+1)}; \cdots ; \mathbf{m}_{(l+1,ni)}] + \mathbf{g}_{\mathcal{D}(l,i)}$ to its parent
    end for
  end for
  master node:
    receives $[\mathbf{m}_{(1,1)}; \cdots ; \mathbf{m}_{(l,n)}]$ from its children
    recovers $\mathbf{g} = \mathbf{a}_{f(0,1)}[\mathbf{m}_{(1,1)}; \cdots ; \mathbf{m}_{(1,n)}]$
end procedure

APPENDIX C
PROOF OF THEOREM 1

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

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

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

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

By the same argument for each layer, we have

$$|\mathcal{D}^{T(L,i)}| = \left(\frac{s+1}{n}\right)\left(\left(\frac{s+1}{n}\right)^{L-1} - \left(\frac{s+1}{n}\right)^{L-2} r_{\mathsf{CR}} - \cdots - \left(\frac{s+1}{n}\right) r_{\mathsf{CR}} - r_{\mathsf{CR}}\right)d, \quad \text{for all } i \in [n^L]. \tag{13}$$

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

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

Optimality: In an α–resilient scheme, the master node is able to recover from any $s = \alpha n$ straggling sub-trees $T(1,1), \cdots, T(1,n)$. 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 $T(2,1), \cdots, T(2,n)$. 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. $T(2,1), \cdots, T(2,s)$. Note that besides sub-tree $T(1,1)$, $\mathbf{x}$ is placed in only s more sub-trees, e.g. $T(1,2), \cdots, T(1,s+1)$. Now consider a straggling pattern where $T(1,2), \cdots, T(1,s+1)$ and $T(2,1), \cdots, T(2,s)$ 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 $i \in [n^{l-1}]$,

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

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

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

Consider the group of siblings[1] placed in layer L whose result reaches the master 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 $T_1, \cdots, T_n$ 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 $ad_i = a r_{\mathsf{CR}} d$ 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 $\{T_1, \cdots, T_n\}$. From the latency analysis result for coded computing systems in [27], we have the following.

Lemma 1 (Theorem 2, [27])**.** *With probability* $1 - o(1)$*, we have*

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

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

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

where inequality (a) uses the fact that $\mathbb{E}\left[T_{(n-s)}\right] = \frac{r_{\mathsf{CR}} d}{\mu}\left(H_n - H_s\right) + a r_{\mathsf{CR}} d$ 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

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

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

Lemma 2. *Let* $T_1, \cdots, T_n$ *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* $T_{(n-s)}$*:*

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

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

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

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

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

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

□

As described in Section III-A, in the proposed CR 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 are 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 $\mathbb{E}\left[T_{\mathsf{CR}}\right] \leq \mathbb{E}\left[T_{\text{comp}}\right] + L(n-s)t_c$. 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 $\varepsilon = \Theta\left(\frac{1}{n^{1/4}}\right)$:

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

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, $\{T_1^{gr}, \cdots, T_n^{gr}\}$ denote the random run-times of the nodes in the group and $T^{gr}_{(n-s)}$ represents its $(n-s)$'th order statistics. Clearly,

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

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

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

Consider the following event:

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

We can write

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

and

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

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

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

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

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

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

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

Therefore,

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

which completes the proof.

Lemma 3 (Berstein's Inequality)**.** *Suppose $E_1, \cdots, E_m$ are independent random variables such that*

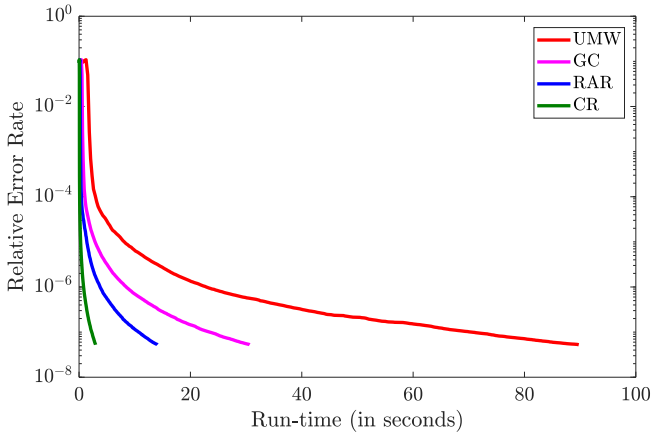
$$\mathbb{E}\left[|E_i|^k\right] \leq \frac{1}{2}\mathbb{E}\left[E_i^2\right]B^{k-2}k!,$$

for some $B > 0$ and every $i = 1, \cdots, m$, $k \geq 2$. Then, for $\varepsilon > 0$,

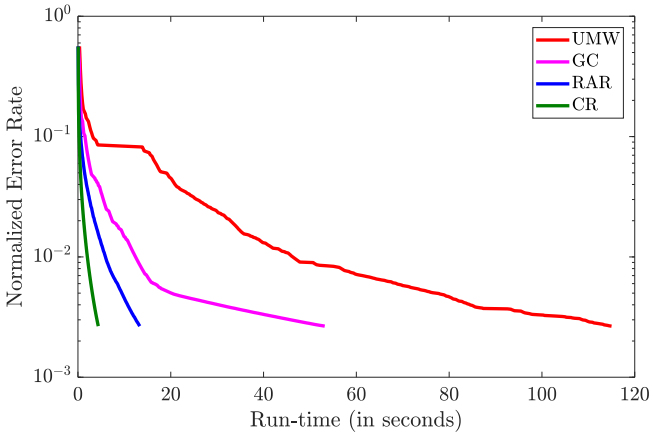
$$\mathbb{P}\left[\sum_{i=1}^{m} E_i - \sum_{i=1}^{m}\mathbb{E}\left[E_i\right] \geq \varepsilon\right] \leq \exp\left(-\frac{\varepsilon^2}{2\left(\sum_{i=1}^{m}\mathbb{E}\left[E_i^2\right] + \varepsilon B\right)}\right).$$

APPENDIX E
ADDITIONAL EXPERIMENTAL RESULTS

Figures 10(a) and 10(b) provide additional experimental results comparing the proposed CR with benchmarks UMW, GC, and RAR. The settings of these experiments are similar to ones described in Section 4 but for a cluster of $N = 84$ nodes. We have a tree topology with $L = 3$ layers and $n = 4$ children nodes per parent for CR.



(a) Convergence plots for relative error rate for logistic regression with real data set with $(d, p) = (6552, 5001)$. CR achieves a speed up of up to $30.3\times$, $10.3\times$ and $4.7\times$ respectively over UMW, GC, and RAR.



(b) Convergence plots for normalized error rate for linear regression with artificial data set with $(d, p) = (7644, 6500)$. CR achieves a speed up of up to $26.3\times$, $12.1\times$ and $3.0\times$ respectively over UMW, GC, and RAR.

Fig. 10: Convergence plots for logistic and linear regression over a cluster of size $N = 84$.

In this setting with 84 nodes, CR achieves a speedup of up to $30.3\times$, $10.3\times$ and $4.7\times$ respectively over UMW, GC, and RAR for logistics regression. In the linear regression problem, the corresponding speedups are $26.3\times$, $12.1\times$ and $3.0\times$, respectively.

Moreover, comparing the two settings of experiments, i.e. $N = 156$ versus $N = 84$ further demonstrates that the straggler and bandwidth contention bottlenecks are increasingly significant when the cluster size increases.