Communication-Computation Efficient Gradient Coding

Min Ye [1] **Emmanuel Abbe** [2]

Abstract

This paper develops coding techniques to reduce the running time of distributed learning tasks. It characterizes the fundamental tradeoff to compute gradients in terms of three parameters: computation load, straggler tolerance and communication cost. It further gives an explicit coding scheme that achieves the optimal tradeoff based on recursive polynomial constructions, coding both across data subsets and vector components. As a result, the proposed scheme allows to minimize the running time for gradient computations. Implementations are made on Amazon EC2 clusters using Python with mpi4py package. Results show that the proposed scheme maintains the same generalization error while reducing the running time by 32% compared to uncoded schemes and 23% compared to prior coded schemes focusing only on stragglers (Tandon et al., ICML 2017).

1. Introduction

Distributed computation plays a key role in the computational challenges faced by machine learning for large data sets (Dean et al., 2012; Abadi et al., 2016). This requires overcoming a few obstacles: First the straggler effect, i.e., slow workers that hamper the computation time. Second, the communication cost; gradients in deep learning typically consist nowadays in millions of real-valued components, and the transmission of these high-dimensional vectors can amortize the savings of the computation time in large-scale distributed systems (Recht et al., 2011; Li et al., 2014a;b). This has driven researchers to use in particular gradient sparsification and gradient quantization to reduce communication cost (Gupta et al., 2015; Alistarh et al., 2017; Wen et al., 2017).

More recently, coding theory has found its way into distributed computing (Li et al., 2015; Lee et al., 2016; Dutta et al., 2016; Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017; Dutta et al., 2017; Yu et al., 2017a;b; Yang et al., 2017; Li et al., 2017a; Karakus et al., 2017; Charles et al., 2017; Zhu et al., 2017; Li et al., 2017b; Yu et al., 2018), following the path of exporting coding techniques to distributed storage (Dimakis et al., 2010; Ye & Barg, 2017), caching (Maddah-Ali & Niesen, 2015) and queuing (Joshi et al., 2015). A few works have also initiated the use of coding techniques in distributed learning (Li et al., 2015; Lee et al., 2016; Tandon et al., 2017). Of particular interest to us is (Tandon et al., 2017), which introduces coding techniques to mitigate the effect of stragglers in gradient computation. While this is a central task in machine learning, (Tandon et al., 2017) does not take into account the communication cost which is important in such applications as mentioned above.

This paper takes a global view on the running time of distributed learning tasks by considering the three parameters, namely, computation load, straggler tolerance and communication cost. We identify a three-fold fundamental tradeoff between these parameters in order to efficiently compute gradients (and more generally summations of vectors), exploiting distributivity both across data subsets and vector components. The tradeoff reads

$$\frac{d}{k} \geq \frac{s+m}{n}, \tag{1}$$

where n is the number of workers, k is the number of data subsets, d is the number of data subsets assigned to each worker, s is the number of stragglers, and m is the communication reduction factor. This generalizes the results in (Tandon et al., 2017) that correspond to $m = 1$. Note that one cannot derive (1) from the results of (Tandon et al., 2017), and we will explain this in more detail below.

We further give an explicit code construction based on recursive polynomials that achieves the derived tradeoff. The key steps in our coding scheme are as follows: In order to reduce the dimension of transmitted vector for each worker, we first partition the coordinates of the gradient vector into m groups of equal size. Then we design two matrices B

[1]Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA [2]Program in Applied and Computational Mathematics and Department of Electrical Engineering, Princeton University, and the School of Mathematics, Institute for Advanced Study, Princeton, NJ 08544, USA. Correspondence to: Min Ye <yeemmi@gmail.com>, Emmanuel Abbe <eabbe@princeton.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

and V, where the $(n-s) \times n$ matrix V has the property that any $(n-s) \times (n-s)$ submatrix is invertible. This property corresponds to the requirement that our coding scheme can tolerate *any* s stragglers, and it can be easily satisfied by setting V to be a (non-square) Vandermonde matrix. Furthermore, the $(mn) \times (n-s)$ matrix B satisfies the following two property: (1) the last m columns of B consisting of n identity matrices of size $m \times m$; (2) for every $j \in [n]$, the product of the ith row of B and the jth column of V must be 0 for a specific set of values of i, and the cardinality of this set is $(n-d)m$. The first property of B guarantees the recovery of the sum gradient vector, and the second property ensures that each worker is assigned at most d data subsets. We make use of the natural connection between the Vandermonde structure and polynomials to construct our matrix B recursively: More precisely, we can view each row of B as coefficients of some polynomial, and the product of B and V simply consists of the evaluations of these polynomials at certain points. We can then define these polynomials by specifying their roots so that the two properties of B are satisfied. We also mention that the conditions in our construction are more restrictive than those in (Dutta et al., 2016) and (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017): In our setting, the conditions in (Dutta et al., 2016) only require that the last m columns of B contain at most n nonzero entries, and no requirements are imposed on the positions of these nonzero entries; as mentioned above, (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) only deal with the special case of $m = 1$ and do not allow for dimensionality reduction of the gradient vectors. Due to these more relaxed conditions, the constructions in (Dutta et al., 2016) and (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) do not have the recursive polynomial structure, which is the main technical novelty in our paper.

We support our theoretical findings by implementing our scheme on Amazon EC2 clusters using Python with mpi4py package. Experimental results show that the proposed scheme reduces the running time by 32% compared to uncoded schemes and by 23% compared to prior work (Tandon et al., 2017), while maintaining the same generalization error on the Amazon Employee Access dataset from Kaggle, which was also used in (Tandon et al., 2017) for state-of-the-art experiments.

1.1. Related Literature

Slow workers (processors) called "stragglers" can hamper the computation time as the taskmaster needs to wait for all workers to complete their processing. Recent literature proposes adding redundancy in computation tasks of each worker so that the taskmaster can compute the final result using outputs from only a subset of workers and ignore the stragglers. The most popular ways to introduce redundancy in computation are based on either replication schemes or coding theoretic techniques (Ananthanarayanan et al., 2013; Wang et al., 2014; Shah et al., 2016; Lee et al., 2016). Lee et al. (Lee et al., 2016) initialized the study of using erasure-correcting codes to mitigate straggler effects for linear machine learning tasks such as linear regression and matrix multiplication. Subsequently, Dutta et al. proposed new efficient coding schemes to calculate convolutions (Dutta et al., 2017) and the product of a matrix and a long vector (Dutta et al., 2016), Yu et al. introduced optimal coding schemes to compute high-dimensional matrix multiplication (Yu et al., 2017a; 2018) and Fourier Transform (Yu et al., 2017b), and Yang et al. developed coding methods for parallel iterative linear solver (Yang et al., 2017). Tandon et al. (Tandon et al., 2017) further used coding theoretic methods to avoid stragglers in nonlinear learning tasks. More specifically, (Tandon et al., 2017) presented an optimal trade-off between the computation load and *straggler tolerance* (the number of tolerable stragglers) in synchronous gradient descent for *any* loss function. Several code constructions achieving this trade-off were given in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017). Li et al. (Li et al., 2017a) considered distributed gradient descent under a probabilistic model and proposed the Batched Coupon's Collector scheme to alleviate straggler effect under this model. At the same time, the schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) are designed to combat stragglers for the worst-case scenario. While most research focused on recovering the exact results in the presence of stragglers, (Karakus et al., 2017; Raviv et al., 2017; Charles et al., 2017) suggested allowing some small deviations from the exact gradient in each iteration of the gradient descent and showed that one can obtain a good approximation of the original solution by using coding theoretic methods.

As mentioned above, high network communication cost for synchronizing gradients and parameters is also a well-known bottleneck of distributed learning. In particular for deep learning, gradient vectors typically consist of millions of real numbers, and for large-scale distributed systems, transmissions of high-dimensional gradient vectors might even amortize the savings of computation time (Recht et al., 2011; Li et al., 2014a;b). The most widely used methods to reduce communication cost in the literature are based on gradient sparsification and gradient quantization (Gupta et al., 2015; Alistarh et al., 2017; Wen et al., 2017).

In this paper we directly incorporate the communication cost into the framework of reducing running time for gradient computation, in addition to computation load and straggler tolerance. In particular, we take advantage of distributing the computations over subsets of vector components in addition to subsets of data samples. The advantages of our coding scheme over the uncoded schemes and the schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al.,

Table 1: Main notation

n	the number of workers
k	the number of data subsets in total; in most part of the paper we assume $n = k$
d	the number of data subsets assigned to each worker
s	the number of stragglers
m	the communication cost reduction factor
l	the dimension of gradient vectors
$g_i, i \in [k]$	the partial gradient vector of data subset D_i; $g_i = (g_i(0), g_i(1), \dots, g_i(l-1))$
$f_i(g_{i_1}, g_{i_2}, \dots, g_{i_d})$	the transmitted vector of worker W_i, sometimes abbreviated as f_i

2017) are demonstrated by both experimental results and numerical analysis in Sections 4 and 5. We also strengthen the numerical analyses by studying the behavior of the running time using probabilistic models for the computation and communication times, obtaining improvements that are consistent with the outcome of the Amazon experiments (see Section 5). Our results apply to both batch gradient descent and mini-batch stochastic gradient descent (SGD), which is the most popular algorithm in large-scale distributed learning.

As a final remark, (Li et al., 2015; 2017b) also studied the trade-off between computation and communication in distributed learning, but the problem setup in (Li et al., 2015; 2017b) is different from our work in nature. We study distributed gradient descent while (Li et al., 2015; 2017b) focused on MapReduce framework. The communication in our problem is from all the worker nodes to one master node, while the communication in (Li et al., 2015; 2017b) is from all workers to all workers, and there is no master node in (Li et al., 2015; 2017b). This difference in problem setup leads to completely different results and techniques.

2. Problem Formulation and Main Results

We begin with a brief introduction on distributed gradient descent. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^l$ and $y \in \mathbb{R}$, we want to learn parameters $\beta \in \mathbb{R}^l$ by minimizing a generic loss function $L(D; \beta) := \sum_{i=1}^N L(x_i, y_i; \beta)$, for which gradient descent is commonly used. More specifically, we begin with some initial guess of β as $\beta^{(0)}$, and then update the parameters according to the following rule:

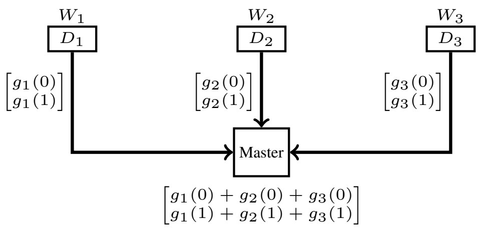
$$\beta^{(t+1)} = h(\beta^{(t)}, g^{(t)}), \tag{2}$$

where $g^{(t)} := \nabla L(D; \beta^{(t)}) = \sum_{i=1}^N \nabla L(x_i, y_i; \beta^{(t)})$ is the gradient of the loss at the current estimate of the parameters and h is a gradient-based optimizer. As in (Tandon et al., 2017), we assume that there are n workers $W_1, W_2, \dots, W_n$, and that the original dataset D is partitioned into k subsets of equal size, denoted as $D_1, D_2, \dots, D_k$. Define the partial gradient vector of D_i as $g_i^{(t)} := \sum_{(x,y)\in D_i} \nabla L(x, y; \beta^{(t)})$. Clearly $g^{(t)} = g_1^{(t)} + g_2^{(t)} + \cdots + g_k^{(t)}$. Suppose that each worker is assigned d data subsets, and there are s stragglers, i.e., we only wait for the results from the first $n-s$ workers. For $i = 1, 2, \dots, n$, we write the datasets assigned to worker W_i as $\{D_{i_1}, D_{i_2}, \dots, D_{i_d}\}$. Each worker computes its partial gradient vectors $g_{i_1}^{(t)}, g_{i_2}^{(t)}, \dots, g_{i_d}^{(t)}$ and returns $f_i(g_{i_1}^{(t)}, g_{i_2}^{(t)}, \dots, g_{i_d}^{(t)})$, a prespecified function of these partial gradients. In order to update the parameters according to (2), we require that the sum gradient vector $g^{(t)}$ can be recovered from the results of the first $n-s$ workers no matter who the s stragglers will be. Due to complexity consideration, we would further like $f_i, i \in [n]$ to be linear functions. Lee et al. (Tandon et al., 2017) showed that this is possible if and only if $\frac{d}{k} \ge \frac{s+1}{n}$. Since the functions $f_i, i \in [n]$ are time invariant, in the rest of this paper we will omit the superscript (t) for simplicity of notation. Recall that in batch gradient descent, we use all the samples to update parameters in each iteration, and in mini-batch SGD we use a small portion of the whole dataset in each iteration. Since we only focus on each iteration of the gradient descent algorithm, our results apply to both batch gradient descent and mini-batch SGD. For readers' convenience, we list the main notation in Table 1.

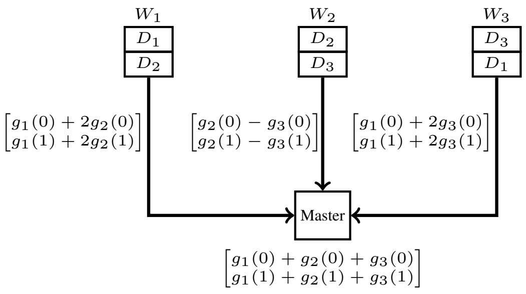
Let us write each partial gradient vector as $g_i = (g_i(0), g_i(1), \dots, g_i(l-1))$ for $i = 1, 2, \dots, k$. We will show that when $s \le \frac{d}{k}n - 1$, each worker only needs to transmit a vector[1] of dimension $l/(\frac{d}{k}n - s)$. In other words, we can reduce the communication cost by a factor of $\frac{d}{k}n - s$.

Roughly speaking, (Tandon et al., 2017) showed the following two-dimensional tradeoff: if we assign more computation load at each worker, then we can tolerate more stragglers. In this paper we will show a three-dimensional tradeoff between computation load at each worker, straggler tolerance and the communication cost: for a fixed computation load, we can reduce the communication cost by waiting for results from more workers. Fig. 1 uses a toy example to illustrate this tradeoff as well as the basic idea of how to reduce the communication cost. In Fig. 1 the gradient vector has dimension $l = 2$, and it is clear that this idea

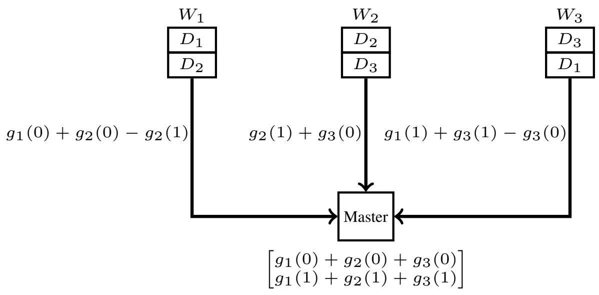
[1]Assume that $k|(dn)$ and $(\frac{d}{k}n - s)|l$.


W_1
D_1
W_2
D_2
W_3
D_3
$\begin{bmatrix} g_1(0) \\ g_1(1) \end{bmatrix}$
$\begin{bmatrix} g_2(0) \\ g_2(1) \end{bmatrix}$
$\begin{bmatrix} g_3(0) \\ g_3(1) \end{bmatrix}$
Master
$\begin{bmatrix} g_1(0)+g_2(0)+g_3(0) \\ g_1(1)+g_2(1)+g_3(1) \end{bmatrix}$

(a) Naive Synchronous Gradient Descent: each worker transmits two scalars, and Master needs to wait for the results from all three workers.


W_1
D_1
D_2
W_2
D_2
D_3
W_3
D_3
D_1
$\begin{bmatrix} g_1(0)+2g_2(0) \\ g_1(1)+2g_2(1) \end{bmatrix}$
$\begin{bmatrix} g_2(0)-g_3(0) \\ g_2(1)-g_3(1) \end{bmatrix}$
$\begin{bmatrix} g_1(0)+2g_3(0) \\ g_1(1)+2g_3(1) \end{bmatrix}$
Master
$\begin{bmatrix} g_1(0)+g_2(0)+g_3(0) \\ g_1(1)+g_2(1)+g_3(1) \end{bmatrix}$

(b) Straggler-efficient gradient coding (Tandon et al., 2017): each worker transmits two scalars, and Master can calculate the sum vector from the results of any two workers.



(c) Communication-efficient gradient coding (this paper): each worker only transmits one scalar, and Master needs to wait for the results from all three workers.

(d) Both straggler- and communication- efficient gradient coding (this paper): each worker transmits a lower dimensional vector, and Master only needs to wait for the results from a subset of workers. See Fig. 2 for an example where we can simultaneously mitigate straggler effects and reduce communication cost.

Figure 1: The idea of communication efficient gradient coding.

extends to gradient vectors of any dimension (by padding a zero when l is odd). To quantify the tradeoff, we introduce the following definition.

Definition 1. *Given n and k, we say that a triple of nonnegative integers (d, s, m) satisfying that $1 \le d \le k$ and $m \ge 1$ is achievable*[2] *if there is a distributed synchronous gradient descent scheme such that: 1) Each worker is assigned d data subsets; 2) There are n functions $f_1, f_2, \dots, f_n$ from $\mathbb{R}^{dl}$ to $\mathbb{R}^{l/m}$ such that the gradient vector $g_1 + g_2 + \dots + g_k$ can be recovered from any $n - s$ out of the following n vectors*

$$f_i(g_{i_1}, g_{i_2}, \dots, g_{i_d}), i = 1, 2, \dots, n, \tag{3}$$

where $i_1, i_2, \dots, i_d$ are the indices of datasets assigned to worker W_i; 3) $f_1, f_2, \dots, f_n$ are linear functions. In other words, $f_i(g_{i_1}, g_{i_2}, \dots, g_{i_d})$ is a linear combination of the coordinates of the partial gradient vectors $g_{i_1}, g_{i_2}, \dots, g_{i_d}$.

Theorem 1. *Let k, n be positive integers. A triple (d, s, m) is achievable if and only if*

$$\frac{d}{k} \ge \frac{s+m}{n}. \tag{4}$$

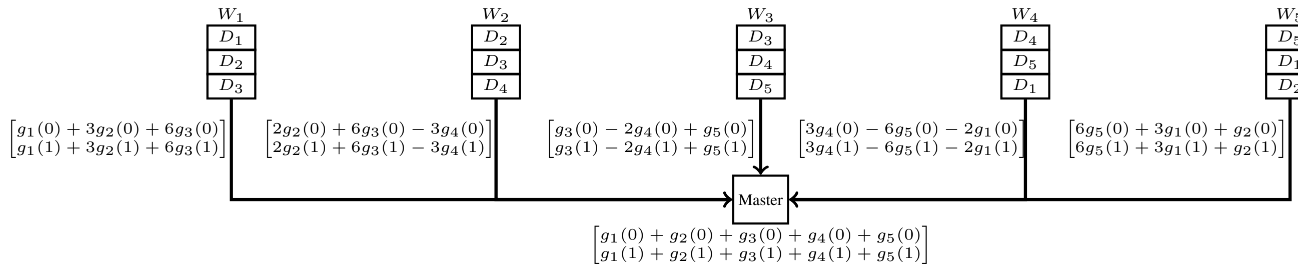
[2]Throughout we assume that $m|l$. Since l is typically very large and m is relatively small, the condition $m|l$ can always be satisfied by padding a few zeroes at the end of the gradient vectors.

The converse proof can be found in the full version (Ye & Abbe, 2018), and the achievability scheme is given in Section 3. Note that the special case $m = 1$ in Theorem 1 is the same as the case considered in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017). We also remark that although (4) looks very similar to Theorem 1 in (Dutta et al., 2016), their coding scheme can not be used to achieve (4) with equality when $m > 1$. In (Ye & Abbe, 2018), we discuss the differences between our work and (Dutta et al., 2016) in detail. In particular, we show that the constraint in our problem is stronger than that in (Dutta et al., 2016).

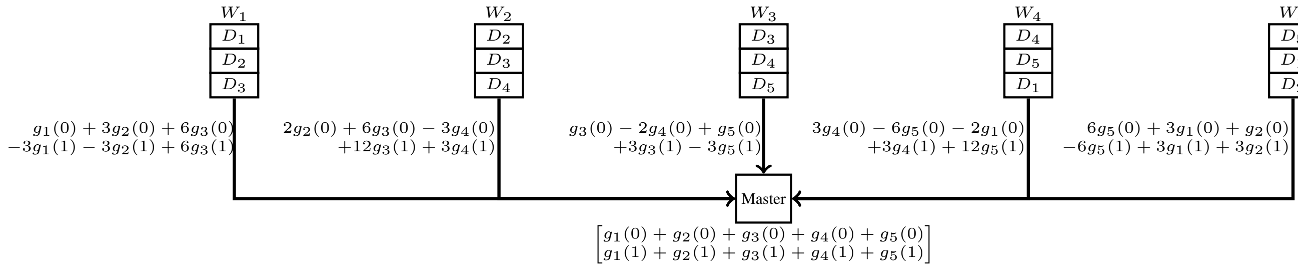
Notice that the computation load at each worker is known by $\frac{d}{k}$, not the value of k itself. We are interested in achieving the optimal computation load in (4), and the value of k does not matter. Therefore we will assume that $k = n$ for the remainder of this paper. Under this assumption, (4) has the following simple form

$$d \ge s + m. \tag{5}$$

In Fig. 2 we take $n = k = 5, d = 3, l = 2$, and show the implementation for two different choices of the pair (s, m). The communication cost of Fig. 2b is half of that of Fig. 2a, but the system in Fig. 2b can only tolerate one straggler while the system in Fig. 2a can tolerate two stragglers. Table


W_1
D_1 D_2 D_3
W_2
D_2 D_3 D_4
W_3
D_3 D_4 D_5
W_4
D_4 D_5 D_1
W_5
D_5 D_1 D_2
$\begin{bmatrix} g_1(0)+3g_2(0)+6g_3(0) \\ g_1(1)+3g_2(1)+6g_3(1) \end{bmatrix}$
$\begin{bmatrix} 2g_2(0)+6g_3(0)-3g_4(0) \\ 2g_2(1)+6g_3(1)-3g_4(1) \end{bmatrix}$
$\begin{bmatrix} g_3(0)-2g_4(0)+g_5(0) \\ g_3(1)-2g_4(1)+g_5(1) \end{bmatrix}$
$\begin{bmatrix} 3g_4(0)-6g_5(0)-2g_1(0) \\ 3g_4(1)-6g_5(1)-2g_1(1) \end{bmatrix}$
$\begin{bmatrix} 6g_5(0)+3g_1(0)+g_2(0) \\ 6g_5(1)+3g_1(1)+g_2(1) \end{bmatrix}$
Master
$\begin{bmatrix} g_1(0)+g_2(0)+g_3(0)+g_4(0)+g_5(0) \\ g_1(1)+g_2(1)+g_3(1)+g_4(1)+g_5(1) \end{bmatrix}$

(a) $s = 2, m = 1$: Each worker transmits two scalars, and Master can calculate the sum vector from the results of any 3 workers.



(b) $s = 1, m = 2$: Each worker only transmits one scalar, and Master can calculate the sum vector from the results of any 4 workers. Table 2 shows how to do this calculation.

Figure 2: Tradeoff between communication cost and straggler tolerance

Table 2: Calculate the sum gradient vector in Fig. 2b when there is one straggler.

Straggler	Calculate $g_1(0)+g_2(0)+g_3(0)+g_4(0)+g_5(0)$	Calculate $g_1(1)+g_2(1)+g_3(1)+g_4(1)+g_5(1)$
W_1	$\frac{1}{2}f_2 - 2f_3 - \frac{1}{2}f_4$	$-\frac{1}{6}f_2 + f_3 + \frac{1}{2}f_4 + \frac{1}{3}f_5$
W_2	$\frac{1}{4}f_1 - \frac{1}{2}f_3 + \frac{1}{4}f_5$	$-\frac{1}{12}f_1 + \frac{1}{2}f_3 + \frac{1}{3}f_4 + \frac{1}{4}f_5$
W_3	$\frac{1}{3}f_1 - \frac{1}{6}f_2 + \frac{1}{6}f_4 + \frac{1}{3}f_5$	$-\frac{1}{6}f_1 + \frac{1}{6}f_2 + \frac{1}{6}f_4 + \frac{1}{6}f_5$
W_4	$\frac{1}{4}f_1 - \frac{1}{2}f_3 + \frac{1}{4}f_5$	$-\frac{1}{4}f_1 + \frac{1}{3}f_2 - \frac{1}{2}f_3 + \frac{1}{12}f_5$
W_5	$\frac{1}{2}f_2 - 2f_3 - \frac{1}{2}f_4$	$-\frac{1}{3}f_1 + \frac{1}{2}f_2 - f_3 - \frac{1}{6}f_4$

2 below shows how to calculate the sum gradient vector in Fig. 2b when there is one straggler. In the table we abbreviate $f_i(g_{i_1}, g_{i_2}, \dots, g_{i_d})$ in (3) as f_i, i.e., f_i is the transmitted vector of W_i.

Numerical Stability: In the proof of Theorem 1, we use Vandermonde matrices and assume that all the computations have infinite precision, which is not possible in real world applications. According to our experimental results, the stability issue of Vandermonde matrices can be ignored up to $n = 20$, which covers the regime considered in most related works (Dutta et al., 2016; Tandon et al., 2017). However, beyond that we need to design numerically stable coding schemes and give up the optimal trade-off (5) between d, s and m. In the full version (Ye & Abbe, 2018) we find an achievable region for which the condition numbers of all operations in the gradient reconstruction phase are upper bounded by a given value κ, so that the numerical stability can be guaranteed.

3. Coding Scheme

In this section, we present a coding scheme achieving (5) with equality, i.e., the parameters in our scheme satisfy $d = s + m$. First we introduce two binary operations $\oplus$ and $\ominus$ over the set $[n]$. For $a, b \in [n]$, define $a \oplus b := ((a+b-1) \bmod n)+1$ and $a \ominus b := ((a-b-1) \bmod n)+1$. In our scheme, each worker W_i is assigned with d data subsets $D_i, D_{i\oplus 1}, D_{i\oplus 2}, \dots, D_{i\oplus(d-1)}$. This is equivalent to say that each data subset D_i is assigned to d workers $W_i, W_{i\ominus 1}, W_{i\ominus 2}, \dots, W_{i\ominus(d-1)}$.

Let $\theta_1, \theta_2, \dots, \theta_n$ be n distinct real numbers. Define n polynomials $p_i, i \in [n]$,

$$p_i(x) = \prod_{j=1}^{n-d} (x - \theta_{i\oplus j}). \tag{6}$$

Before proceeding further, let us explain the meaning of θ_i and p_i. Each θ_i is associated with the worker W_i, and each p_i is associated with the dataset D_i. In our scheme,

$p_j(\theta_i) \neq 0$ means that worker W_i needs the value of g_j to calculate $f_i(g_{i_1}, g_{i_2}, \dots, g_{i_d})$, and therefore D_j is assigned to W_i. On the other hand, $p_j(\theta_i) = 0$ means that D_j is not assigned to W_i. By (6), we can see that each dataset D_i is NOT assigned to $W_{i\oplus 1}, W_{i\oplus 2}, \dots, W_{i\oplus(n-d)}$.

Next we construct an $(mn) \times (n-s)$ matrix $B = (b_{ij})$ from the polynomials $p_i, i \in [n]$ defined in (6). Let $p_{i,j}, j = 0, 1, \dots, n-s-1$ be the coefficients of the polynomial p_i, i.e., $p_i(x) = \sum_{j=0}^{n-s-1} p_{i,j} x^j$. Since $\deg(p_i) = n-d$ and $d = s+m \geq s+1$, we have $p_{i,n-d} = 1$ and $p_{i,n-d+1} = p_{i,n-d+2} = \dots = p_{i,n-s-1} = 0$. For every $i \in [n]$, we define m polynomials $p_i^{(1)}, p_i^{(2)}, \dots, p_i^{(m)}$ recursively:

$$\begin{aligned} &p_i^{(1)}(x) := p_i(x), \\ &p_i^{(u)}(x) := x p_i^{(u-1)}(x) - p_{i,n-d-1}^{(u-1)} p_i^{(1)}(x), \\ &\qquad\qquad u = 2, 3, \dots, m, \end{aligned} \tag{7}$$

where $p_{i,j}^{(u)}, j = 0, 1, \dots, n-s-1$ are the coefficients of $p_i^{(u)}$, i.e., $p_i^{(u)}(x) = \sum_{j=0}^{n-s-1} p_{i,j}^{(u)} x^j$. Clearly, $p_i^{(u)}$ is a polynomial of degree $\deg(p_i^{(u)}) = n-d+u-1$, and its leading coefficient is 1, i.e.,

$$\begin{aligned} &p_{i,n-d+u-1}^{(u)} = 1 \text{ for } u = 1, 2, \dots, m, \\ &p_{i,n-d+u}^{(u)} = p_{i,n-d+u+1}^{(u)} = \dots = p_{i,n-s-1}^{(u)} = 0 \\ &\qquad\qquad \text{for } u = 1, 2, \dots, m-1. \end{aligned} \tag{8}$$

It is also clear that $p_i | p_i^{(u)}$ for all $u \in [m]$ and all $i \in [n]$, so for all $u \in [m]$ and all $i \in [n]$, we have $p_i^{(u)}(\theta_{i\oplus 1}) = p_i^{(u)}(\theta_{i\oplus 2}) = \dots = p_i^{(u)}(\theta_{i\oplus(n-d)}) = 0$, which is equivalent to

$$p_{i\ominus 1}^{(u)}(\theta_i) = p_{i\ominus 2}^{(u)}(\theta_i) = \dots = p_{i\ominus(n-d)}^{(u)}(\theta_i) = 0 \tag{9}$$

for all $u \in [m]$ and all $i \in [n]$. By a simple induction on u, one can further see that for $u = 2, 3, \dots, m$,

$$p_{i,n-d}^{(u)} = p_{i,n-d+1}^{(u)} = \dots = p_{i,n-d+u-2}^{(u)} = 0. \tag{10}$$

We can now specify the entries of B as follows: for all $i \in [n], u \in [m], j \in \{1, 2, \dots, n-s\}$, let

$$b_{(i-1)m+u,j} = p_{i,j-1}^{(u)}. \tag{11}$$

By this definition, the following identity holds for every $x \in \mathbb{R}$:

$$\begin{aligned} &B[\ 1 \quad x \quad x^2 \quad \dots \quad x^{n-s-1}\ ]^T \\ =&[\ P_1(x) \quad P_2(x) \quad \dots \quad P_n(x)\ ]^T, \end{aligned} \tag{12}$$

where the m-dimensional row vectors $P_i(x) := [\ p_i^{(1)}(x) \quad p_i^{(2)}(x) \quad \dots \quad p_i^{(m)}(x)\ ]$ for all $i \in [n]$. Moreover, according to (8) and (10), the submatrix $B_{[(n-d+1):(n-s)]}$ consisting of the last m columns of B is

$$B_{[(n-d+1):(n-s)]} = [\ I_m \quad I_m \quad \dots \quad I_m\ ]^T, \tag{13}$$

where I_m is the $m \times m$ identity matrix, and there are n identity matrix on the right-hand side of (13).

Recall that we assume $m|l$ throughout the paper. For every $v = 0, 1, \dots, l/m - 1$ and $j \in [n]$, define an m-dimensional vector $y_v^{(j)} := [\ g_j(vm) \quad g_j(vm+1) \quad \dots \quad g_j(vm+m-1)\ ]$. For every $v = 0, 1, \dots, l/m-1$, define an (mn)-dimensional vector

$$z_v := [\ y_v^{(1)} \quad y_v^{(2)} \quad \dots \quad y_v^{(n)}\ ]. \tag{14}$$

According to (12),

$$\begin{aligned} &z_v B [\ 1 \quad \theta_i \quad \theta_i^2 \quad \dots \quad \theta_i^{n-s-1}\ ]^T \\ =&\sum_{j=1}^{n} \sum_{u=1}^{m} p_j^{(u)}(\theta_i) g_j(vm+u-1) \\ =&\sum_{j=0}^{d-1} \sum_{u=1}^{m} p_{i\oplus j}^{(u)}(\theta_i) g_{i\oplus j}(vm+u-1), \end{aligned} \tag{15}$$

where the second equality follows from (9).

Now we are ready to define the transmitted vector $f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)})$ for each worker $W_i, i \in [n]$:

$$\begin{aligned} &f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)}) := \\ &\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{l/m-1} \end{bmatrix} B [\ 1 \quad \theta_i \quad \theta_i^2 \quad \dots \quad \theta_i^{n-s-1}\ ]^T. \end{aligned} \tag{16}$$

By (15), the value of $f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)})$ indeed only depends on the values of $g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)}$.

To complete the description of our coding scheme, we only need to show that for any subset $\mathcal{F} \subseteq [n]$ with cardinality $|\mathcal{F}| = n-s$, we can calculate $g_1 + g_2 + \dots + g_n$ from $\{f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)}) : i \in \mathcal{F}\}$. Without loss of generality let us assume that $\mathcal{F} = \{1, 2, \dots, n-s\}$. Define the following $(n-s) \times (n-s)$ matrix

$$A := \begin{bmatrix} 1 & 1 & \dots & 1 \\ \theta_1 & \theta_2 & \dots & \theta_{n-s} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_1^{n-s-1} & \theta_2^{n-s-1} & \dots & \theta_{n-s}^{n-s-1} \end{bmatrix}. \tag{17}$$

Let the column vectors $\{e_1, e_2, \dots, e_{n-s}\}$ be the standard basis of $\mathbb{R}^{n-s}$, i.e., all coordinates of e_i are 0 except the ith coordinate which is 1. By (13), for all $0 \leq v \leq l/m - 1$ and all $u \in [m]$,

$$z_v B e_{n-d+u} = \sum_{j=1}^{n} g_j(vm+u-1). \tag{18}$$

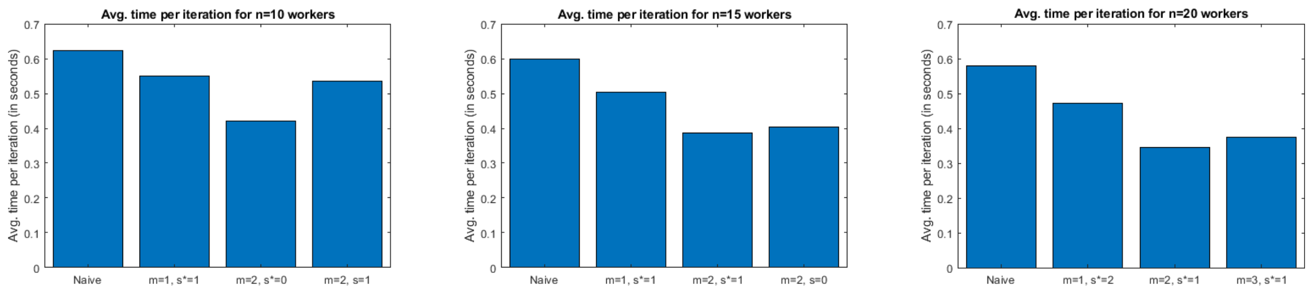


Figure 3: Avg. time per iteration for $n = 10, 15, 20$ workers, s^* means that it is the optimal value of s for that choice of m

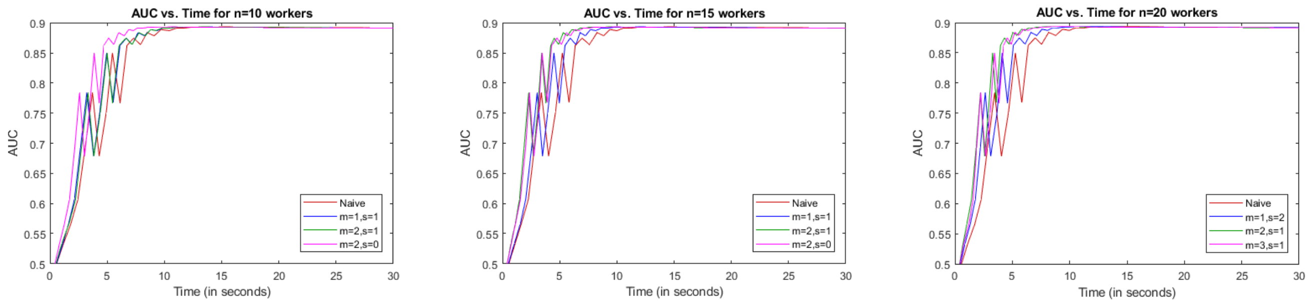


Figure 4: AUC vs. Time for $n = 10, 15, 20$ workers. The curves corresponding to $m > 1$ are always on the left side of the curves corresponding to $m = 1$ and the naive scheme, which means that our schemes achieve the target generalization error much faster than the other schemes.

According to (16), from $\{f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)}) : i \in \mathcal{F}\}$ we can obtain the values of

$$z_v BA \text{ for all } 0 \le v \le l/m - 1. \tag{19}$$

Since A is invertible, we can calculate $z_v B e_{n-d+u}$ for all $0 \le v \le l/m - 1$ and all $u \in [m]$ from the vectors in (19) by multiplying $A^{-1} e_{n-d+u}$ to the right. By (18),

$$\begin{aligned}&\{z_v B e_{n-d+u} : v \in \{0, 1, \dots, l/m - 1\}, u \in [m]\}\\ =&\{\sum_{j=1}^{n} g_j(vm + u - 1) : v \in \{0, 1, \dots, l/m - 1\}, u \in [m]\}\\ =&\{\sum_{j=1}^{n} g_j(i) : i \in \{0, 1, \dots, l - 1\}\}.\end{aligned}$$

Therefore we conclude that the sum vector $g_1 + g_2 + \cdots + g_n$ can be calculated from $\{f_i(g_i, g_{i\oplus 1}, \dots, g_{i\oplus(d-1)}) : i \in \mathcal{F}\}$ whenever $|\mathcal{F}| = n - s$. Thus we have shown that our coding scheme satisfies all three conditions in Definition 1, and this completes the proof of the achievability part of Theorem 1. Efficient implementation and numerical stability of this coding scheme are discussed in the full version (Ye & Abbe, 2018).

4. Experiments on Amazon EC2 Clusters

In this section, we use our proposed gradient coding scheme to train a logistic regression model on the Amazon Employee Access dataset from Kaggle[3], and we compare the running time and Generalization AUC[4] between our method and baseline approaches. More specifically, we compare our scheme against: (1) the *naive* scheme, where the data is uniformly divided among all workers without replication and the master node waits for all workers to send their results before updating model parameters in each iteration, and (2) the coding schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017), i.e., the special case of $m = 1$ in our scheme. Note that in (Tandon et al., 2017) the authors implemented their methods (which is the special case of $m = 1$ in this paper) to train the same model over the same dataset.

We used Python with `mpi4py` package to implement our gradient coding schemes proposed in Section 3, where we chose $\{\theta_1, \theta_2, \dots, \theta_n\}$ as $\{\pm(1 + i/2), i = 0, 1, 2, \dots, n/2 - 1\}$ for even n and $\{0, \pm(1 + i/2), i = 0, 1, 2, \dots, (n-1)/2 - 1\}$ for odd n. We used `t2.micro` instances on Amazon EC2 as worker nodes and a single `c3.8xlarge` instance as the master node.

As a common preprocessing step, we converted the categorical features in the Amazon Employee Access dataset to

[3]https://www.kaggle.com/c/amazon-employee-access-challenge

[4]AUC is short for area under the ROC-curve. The Generalization AUC can be efficiently calculated using the "sklearn.metrics.auc" function in Python.

Table 3: $\mathbb{E}[T_{\text{tot}}]$ for different choices of d and m

m \ d	1	2	3	4	5	6	7	8
1	36.1138	29.2288	27.3351	26.7469	26.4574	26.0891	25.4172	24.1063
2		23.1036	21.3994	21.5369	21.9114	22.2099	22.3189	22.1405
3			22.2604	21.3697	21.5749	21.9095	22.1707	22.2772
4				24.8036	23.2793	23.1114	23.1862	23.2611
5					28.5800	25.9827	25.2862	25.0141
6						32.8664	29.0745	27.7904
7							37.3977	32.3759
8								42.0638

binary features by one-hot encoding, which can be easily realized in Python. After one-hot encoding with interaction terms, the dimension of parameters in our model is $l = 343474$. For all three schemes (our proposed scheme, the schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) and the naive scheme), we used $N = 26220$ training samples and adopted Nesterov's Accelerated Gradient (NAG) descent (Bubeck, 2015) to train the model. These experiments were run on $n = 10, 15, 20$ worker nodes.

In Fig. 3, we compare average running time per iteration for different schemes. For coding schemes proposed in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017), i.e., coding schemes corresponding to $m = 1$ in our paper, we choose the optimal value of s such that it has the smallest running time among all possible choices of $(m = 1, s)$. For coding schemes proposed in this paper, i.e., schemes with $m > 1$, we choose two pairs of (m, s) with the smallest running time among all possible choices. We can see that for all three choices of n, our scheme outperforms the schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) by at least 23% and outperforms the naive scheme by at least 32%. We then plot generalization AUC vs. running time for these choices of (m, s) in Fig. 4. The curves corresponding to $m > 1$ are always on the left side of the curves corresponding to $m = 1$ and the naive scheme, which means that our schemes achieve the target generalization error much faster than the other two schemes.

5. Analysis of the Total Computation and Communication Time

In this section we analyze the total runtime of our coding scheme for different choices of the design parameters (d, s, m) based on a probabilistic model. Following the probabilistic model of runtime in (Lee et al., 2016), we assume that both computation time and communication time have shifted exponential distribution, and the total runtime is the sum of the computation time and the communication time.[5] Formally speaking, for $i, j \in [n]$, let $T_{i,j}^{(1)}$ be the computation time of data subset D_j for worker W_i. Similarly, for $i \in [n]$, let $T_i^{(2)}$ be the communication time for worker W_i to send a vector of dimension l. We make the following assumption: 1) For $i \in [n]$, $T_{i,1}^{(1)} = T_{i,2}^{(1)} = \cdots = T_{i,n}^{(1)} = T_i^{(1)}$, where the random variables $T_i^{(1)}, i \in [n]$ are i.i.d. with distribution

$$\Pr(T_i^{(1)} \le t) = 1 - e^{-\lambda_1(t-t_1)}, \forall t \ge t_1;$$

2) The communication time for worker W_i to send a vector of dimension l' is $(l'/l)T_i^{(2)}$, where the random variables $T_i^{(2)}, i \in [n]$ are i.i.d. with distribution

$$\Pr(T_i^{(2)} \le t) = 1 - e^{-\lambda_2(t-t_2)}, \forall t \ge t_2;$$

3) The random variables $T_i^{(1)}, i \in [n]$ and $T_i^{(2)}, i \in [n]$ are mutually independent.

In the Table 3 we take $n = k = 8, \lambda_1 = 0.8, \lambda_2 = 0.1, t_1 = 1.6, t_2 = 6$, and we list $\mathbb{E}[T_{\text{tot}}]$ for all possible choices of d and m. Recall that we take $s = d - m$ to minimize T_{tot}. We can see that $d = 4, m = 3$ is the optimal choice, whose total runtime is 21.3697. The runtime for uncoded scheme ($d = m = 1$) is 36.1138, and the best achievable runtime for the coding schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) is 24.1063 (d=8,m=1). Therefore our coding scheme outperforms the uncoded scheme by 41% and outperforms the schemes in (Tandon et al., 2017; Halbawi et al., 2017; Raviv et al., 2017) by 11%.

Acknowledgements

This work was partly supported by NSF CAREER Award CCF-1552131, NSF Center for the Science of Information

[5]Since the total number of samples N in large-scale machine learning tasks is of order hundreds of millions, we have $N \gg n$ in our problem. The computation time is of order $\Theta(Nl)$ while the reconstruction time is of order $O(nl)$. Therefore we can ignore the reconstruction phase at the master node when estimating the total runtime.

CCF-0939370, and the Google Faculty Research Award.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. 2016. arXiv:1603.04467.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *Advances in Neural Information Processing Systems 30*, pp. 1707–1718. 2017.

Ananthanarayanan, G., Ghodsi, A., Shenker, S., and Stoica, I. Effective straggler mitigation: Attack of the clones. In *NSDI*, volume 13, pp. 185–198, 2013.

Bubeck, S. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8 (3-4):231–357, 2015.

Charles, Z., Papailiopoulos, D., and Ellenberg, J. Approximate gradient coding via sparse random graphs. 2017. arXiv:1711.06771.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pp. 1223–1231, 2012.

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J., and Ramchandran, K. Network coding for distributed storage systems. *IEEE Trans. Inform. Theory*, 56(9): 4539–4551, 2010.

Dutta, S., Cadambe, V., and Grover, P. Short-dot: Computing large linear transforms distributedly using coded short dot products. In *Advances In Neural Information Processing Systems*, pp. 2100–2108, 2016.

Dutta, S., Cadambe, V., and Grover, P. Coded convolution for parallel and distributed computing within a deadline. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2403–2407. IEEE, 2017.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1737–1746, 2015.

Halbawi, W., Azizan-Ruhi, N., Salehi, F., and Hassibi, B. Improving distributed gradient descent using Reed-Solomon codes, 2017. arXiv:1706.05436.

Joshi, G., Soljanin, E., and Wornell, G. Queues with redundancy: Latency-cost analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(2):54–56, 2015.

Karakus, C., Sun, Y., Diggavi, S., and Yin, W. Straggler mitigation in distributed optimization through data encoding. In *Advances in Neural Information Processing Systems*, pp. 5440–5448. 2017.

Lee, K., Lam, M., Pedarsani, R., Papailiopoulos, D., and Ramchandran, K. Speeding up distributed machine learning using codes. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 1143–1147. IEEE, 2016.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, pp. 3, 2014a.

Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pp. 19–27, 2014b.

Li, S., Maddah-Ali, M. A., and Avestimehr, A. S. Coded mapreduce. In *53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 964–971. IEEE, 2015.

Li, S., Kalan, S. M. M., Avestimehr, A. S., and Soltanolkotabi, M. Near-optimal straggler mitigation for distributed gradient methods. 2017a. arXiv:1710.09990.

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

Maddah-Ali, M. A. and Niesen, U. Decentralized coded caching attains order-optimal memory-rate tradeoff. *IEEE/ACM Transactions On Networking*, 23(4):1029–1040, 2015.

Raviv, N., Tamo, I., Tandon, R., and Dimakis, A. G. Gradient coding from cyclic MDS codes and expander graphs, 2017. arXiv:1707.03858.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pp. 693–701, 2011.

Shah, N. B., Lee, K., and Ramchandran, K. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2):715–722, 2016.

Tandon, R., Lei, Q., Dimakis, A. G., and Karampatziakis, N. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pp. 3368–3376, 2017.

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

Wen, W., Xu, C., Yan, F., Wu, C., Wang, Y., Chen, Y., and Li, H. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *Advances in Neural Information Processing Systems*, pp. 1508–1518. 2017.

Yang, Y., Grover, P., and Kar, S. Coding method for parallel iterative linear solver. 2017. arXiv:1706.00163.

Ye, M. and Abbe, E. Communication-computation efficient gradient coding. 2018. arXiv:1802.03475.

Ye, M. and Barg, A. Explicit constructions of high-rate MDS array codes with optimal repair bandwidth. *IEEE Trans. Inform. Theory*, 63(4):2001–2014, 2017.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Advances in Neural Information Processing Systems*, pp. 4406–4416. 2017a.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Coded fourier transform. 2017b. arXiv:1710.06471.

Yu, Q., Maddah-Ali, M. A., and Avestimehr, A. S. Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding. 2018. arXiv:1801.07487.

Zhu, J., Pu, Y., Gupta, V., Tomlin, C., and Ramchandran, K. A sequential approximation framework for coded distributed optimization. 2017. arXiv:1710.09001.