Stochastic Gradient Coding for Straggler Mitigation in Distributed Learning

Rawad Bitar, *Member, IEEE*, Mary Wootters, *Member, IEEE*, and Salim El Rouayheb, *Member, IEEE*

***Abstract*—We consider distributed gradient descent in the presence of stragglers. Recent work on *gradient coding* and *approximate gradient coding* have shown how to add redundancy in distributed gradient descent to guarantee convergence even if some workers are *stragglers*—that is, slow or non-responsive. In this work we propose an approximate gradient coding scheme called *Stochastic Gradient Coding* (SGC), which works when the stragglers are random. SGC distributes data points redundantly to workers according to a pair-wise balanced design, and then simply ignores the stragglers. We prove that the convergence rate of SGC mirrors that of batched Stochastic Gradient Descent (SGD) for the ℓ_2 loss function, and show how the convergence rate can improve with the redundancy. We also provide bounds for more general convex loss functions. We show empirically that SGC requires a small amount of redundancy to handle a large number of stragglers and that it can outperform existing approximate gradient codes when the number of stragglers is large.**

***Index Terms*—Distributed computing, straggler mitigation, stochastic gradient descent, machine learning algorithms, convergence analysis.**

I. Introduction

We consider a distributed setting where a master wants to run a gradient-descent-like algorithm to solve an optimization problem distributed across several workers. Let $X \in \mathbb{R}^{m\times \ell}$ be a data matrix and let $\mathbf{x}_i \in \mathbb{R}^{\ell}$ denote the i'th row of X. Let $\mathbf{y} \in \mathbb{R}^m$ be a vector of labels, so $\mathbf{x}_i$ has label y_i. Define $A \triangleq [X|\mathbf{y}]$ to be the concatenation of X and $\mathbf{y}$. The master wants to find a vector $\boldsymbol{\beta}^* \in \mathbb{R}^{\ell}$ that best represents the data X as a function of the labels $\mathbf{y}$. That is, the goal is to iteratively solve an optimization problem

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} \mathcal{L}(A, \boldsymbol{\beta}), \tag{1}$$

Manuscript received October 14, 2019; revised March 30, 2020; accepted April 23, 2020. Date of publication April 29, 2020; date of current version June 8, 2020. The work of Rawad Bitar and Salim El Rouayheb was supported by NSF under Grant CNS-1801708. The work of Mary Wootters was supported in part by NSF under Grant CCF-1657049, and in part by NSF CAREER under Grant CCF-1844628. This work was presented in part at IEEE Information Theory Workshop 2019 [1]. *(Corresponding author: Rawad Bitar.)*

Rawad Bitar was with the ECE Department, Rutgers University, New Brunswick, NJ 08901 USA. He is now with the ECE Department, Technical University of Munich, 80802 Munich, Germany (e-mail: rawad.bitar@tum.de).

Mary Wootters is with the Computer Science and Electrical Engineering Department, Stanford University, Stanford, CA 94305 USA (e-mail: marykw@stanford.edu).

Salim El Rouayheb is with the ECE Department, Rutgers University, New Brunswick, NJ 08901 USA (e-mail: salim.elrouayheb@rutgers.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/JSAIT.2020.2991361

for a given loss function $\mathcal{L}$, by simulating or approximating an update rule of the form

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \gamma_t \nabla \mathcal{L}\big(A, \boldsymbol{\beta}_t\big). \tag{2}$$

Many natural loss functions $\mathcal{L}(A, \boldsymbol{\beta})$ can be written as the sum over individual rows $\mathbf{a}_i$ of A, i.e.,

$$\mathcal{L}(A, \boldsymbol{\beta}) = \sum_{i=1}^{m} \mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}), \tag{3}$$

such loss functions lend themselves naturally to distributed algorithms. In a distributed setting, the master partitions the data matrix A into rows $\mathbf{a}_i$ which are distributed between the workers. Each worker returns some linear combination(s) of the gradients $\nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t)$ that it can compute, and the master aggregates these together to compute or approximate the update step (2).

We focus on the setting where some of the workers may be *stragglers*, i.e., slow or unresponsive. This setting has been studied before in the systems community [2]–[5], and recently in the coding theory community [6]–[8]. A typical approach is to introduce some redundancy: for example, the same piece of data $\mathbf{a}_i$ might be held by several workers. There are several things that one might care about in such a scheme and in this paper we focus on the following four desiderata:

(A) *Convergence Speed:* We would like the error $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2$ to shrink as quickly as possible with the iterations t.

(B) *Redundancy:* We would like to minimize the amount of storage and computation overhead needed between the workers.

(C) *Communication:* We would like to minimize the amount of communication between the master and the workers.

(D) *Flexibility:* In practice, there is a great deal of variability in the number of stragglers over time. We would like an algorithm that degrades gracefully if more stragglers than expected occur.

Exact Gradient Coding for Worst-Case Stragglers: Much existing work has focused on simulating gradient descent *exactly*, even in the presence of worst-case stragglers, for example [6]–[9]. In that model, at each round an arbitrary set of s workers (for a fixed s) may not respond to the master. The goal is for the master to obtain the same update $\boldsymbol{\beta}_t$ at round t that gradient descent would obtain. For this to happen, the master should be able to obtain an exact value of the gradient $\nabla\mathcal{L}(A, \boldsymbol{\beta}_t)$. This has given rise to (exact) *gradient coding* [6], which focuses on optimizing desiderata (A) and (C) above. However, these schemes (and necessarily, any scheme in this

model) do not do so well on (B) and (D). First, it is not hard to see that in the presence of s worst-case stragglers, it is necessary for any $n - s$ workers to be able to recover all of the data, which necessitates a certain amount of overhead. Namely, every data vector should be replicated on $s+1$ different workers. Second, the gradient coding schemes for example in [6], [7] are brittle in the sense that they work perfectly for s failures, but cannot handle more than s stragglers.

No Coding at all for Random Stragglers: On the other hand, there has also been work on *approximately* simulating gradient descent. One approach (similar to the one in [3]) is to assume that the stragglers are random, rather than worst-case, and not employ any redundancy at all. Thus, the master obtains an approximate update (2) instead of an exact one by computing the sum in (3) without the responses of the stragglers. (We will later refer to this algorithm as "Ignore–Stragglers–SGD.") If the stragglers are independent at each round, this algorithm is a close approximation to Batch–SGD, see, e.g., [10]–[13], and performs in about the same way. However, for convex loss functions it is well known that, while Batch–SGD does converge to $\boldsymbol{\beta}^*$, the convergence is not as fast as that of classical gradient descent [14]–[16]. Thus, this approach maintains the good communication cost (C) of the coded approaches by requiring each worker to send one linear combination of the gradients to the master, and improves on (B) and (D), but sacrifices (A), the convergence rate.

Approximate Gradient Coding: Adding Redundancy to Approximate the Gradient: A line of work known as *approximate gradient coding* [9], [17]–[22] introduces redundancy in order to speed up the convergence rate of such an approximate scheme. This line of work studies the data redundancy d (that is, the number of times each row $\mathbf{a}_i$ of the data matrix A is replicated) needed to tolerate s stragglers and allow the master to compute an approximation of the gradient if more than s workers are stragglers [9], [18]–[20]. In [17] a variant of this idea is studied; in that work the data is encoded using LDPC code rather than being duplicated. In approximate gradient coding, the master is required to compute the exact gradient with high probability if fewer than s workers are stragglers. If more than s workers are stragglers, the distance between the computed gradient at the master and the true gradient can be made small if the redundancy factor is poly-logarithmic in the number of workers. So far, this line of work has mostly focused on desiderata (B), (C) and (D), and most works have not directly analyzed the convergence time (A). Two exceptions are [17] and [19], which we discuss more below. In this work, we introduce an approximate gradient coding scheme called *Stochastic Gradient Coding* (SGC) which works in the random straggler model and which does well simultaneously on desiderata (A)-(D). We analyze the convergence rate of SGC, and we present experimental work which demonstrates that SGC outperforms the most recently proposed schemes [18], [19] when p (the fraction of workers that the master will ignore in each iteration) is relatively large.

Remark 1 (Motivation for the Random Straggler Model, and for Large p): A model of random stragglers has been studied before (e.g., [9], [17]–[21], [23]), and is motivated as an easy-to-study model that captures non-persistent stragglers. However, one might also work in a "random stragglers" model even if all of the workers are fast: just like how SGD samples fewer points to save time and computation, so the master might sample random workers to save bandwidth and computation.

This second setting further motivates the case when p might be relatively large, which is the setting that we focus on in this work.

A. Contributions

We consider an approach that we call *Stochastic Gradient Coding* (SGC). The coding idea—which is similar to previous approaches in approximate gradient coding [18]—is simple: the master distributes data to the workers with a small amount of repetition according to a *pair-wise balanced scheme* (which we will define below); a data point $\mathbf{a}_i$ is replicated d_i times, and d_i can vary from data point to data point. Below, the redundancy parameter d refers to the average of the d_i's. Once the data is distributed, the algorithm proceeds similarly to the Ignore–Stragglers–SGD algorithm described above: workers compute gradients on their data and return a linear combination, and the master aggregates all of the linear combinations it receives to do an update step.

Remark 2 (The Role of Redundancy in SGC and in Approximate Gradient Coding With Random Stragglers): Since our scheme is replication-based, the reader may wonder why we use the word "coding." Here, we are using it in the same way as is standard to describe the many replication-based schemes in the gradient coding literature, for example [6], [9], [18], [19].[1] Since each worker responds with only a single vector (rather than all of the partial gradients it can compute), and a worker (with all its data) straggles as a unit, it matters how the data is distributed between the workers. We will see that this matters in our experiments in Section VI, where we compare the SGC method of distributing data to the different data distribution methods of [19].

One point of our work is to understand to what extent adding redundancy can speed up the convergence of SGD. To this end, we will also compare SGC with the "Ignore–Stragglers–SGD" algorithm alluded to above, where there is no replication of the data and the master simply ignores slow workers when estimating the gradient.

One contribution of this work is to provide a rigorous convergence analysis of SGC. We show that SGC with only a small amount of redundancy d is able to regain the benefit of (A) from the (exact) coded approaches, while still preserving the benefits of (B), (C), (D) that the "Ignore–Stragglers–SGD" approach sketched above does. A second contribution is extensive experimental evidence which suggests that for the same small redundancy factor d SGC outperforms other schemes when there are many stragglers.

More precisely, our contributions are as follows (all in the stochastic straggler model):

- In the special case of the ℓ_2 loss function, we show that SGC with redundancy factor $d > 1$, can obtain error bounds where $\|\boldsymbol{\beta}^* - \boldsymbol{\beta}_t\|_2$ decreases at first exponentially

[1] We note also that our replication-based data distribution is similar to a *Fractional Repetition* (FR) code [24].

and then proportionally to $\frac{1}{td}$. This mirrors existing results on SGD (which corresponds to the case $d = 1$), and quantifies the trade-off between replication and error. This is made formal in Theorem 3.

- For more general loss functions, we show that SGC has at least the same convergence rate as Ignore–Stragglers–SGD, and we give some theoretical evidence that the error $\|\boldsymbol{\beta}^* - \boldsymbol{\beta}_t\|$ may decrease as d increases. This is made formal in Theorem 4.
- We provide numerical simulations comparing SGC to gradient descent, Ignore–Stragglers–SGD and a few other versions of SGD, and other approximate gradient coding methods. Our simulations show that indeed SGC improves the accuracy of Ignore–Stragglers–SGD, with far less redundancy than would be required to implement exact gradient descent using coding. In addition, we compare SGC to other approximate gradient methods existing in the literature and show that SGC outperforms the existing methods when the probability of workers being stragglers is high.

Remark 3: Like previous works [17], [19], we measure the error as a function of the number of iterations. We note that an important metric in distributed computing is the error as a function of the total running time. In particular, it could be possible that the coding—which introduces computational overhead at the workers—can slow down the process more than the gains in convergence help. As with previous work, we envision settings where the bottleneck is the delay experienced by the master per iteration, for example as caused by network and system management delays. In such settings it is meaningful to consider the number of iterations.

B. Relationship to Previous Work on Approximate Gradient Coding

We provide a more detailed description of previous work in Section VII, but first we briefly mention some of the main differences between our work and existing work on approximate gradient coding [9], [17]–[21].

First, we note that our SGC scheme is quite similar to Bernoulli Gradient Coding (BGC) studied in [18], where the data is distributed uniformly at random to d workers. One difference between our work and that work is that we allow for the redundancy of different data points $\mathbf{a}_i$ to vary for different i; we will see that for the ℓ_2 loss function it makes sense to choose d_i based on $\|\mathbf{x}_i\|_2$. A second difference between our work and [18] is that [18] does not provide a complete convergence analysis. The works [9], [20], [21] also study schemes similar in flavor to SGC, but these works also do not provide complete convergence analyses.

The works of [17], [19] do provide convergence analyses, although for schemes that are quite different from SGC. More precisely, [17] studies a scheme with LDPC coding, rather than repetition. The work of [19] studies a scheme based on Fractional Repetition (FR) codes, which was proposed in [18]. However, the FR codes studied by [19] results in a very different data distribution scheme than the one we study. Their scheme partitions the data and the workers into different blocks and every worker in a block receives all of the data from the corresponding block.

Additionally, we obtain slightly different error guarantees than the analyses of [17], [19]. More precisely, the analysis of [19] proves a bound where the error decreases exponentially in T (the number of iterations of the algorithm) until some noise floor is hit. The analysis of [17] studies the special case of the ℓ_2 loss function, and shows that the error decays like $\mathcal{O}(1/\sqrt{T})$. In contrast, for SGC and for the special case of the ℓ_2 loss function, we show that the error decays exponentially in T at first and then switches to dacaying like $\mathcal{O}(1/T)$; this mirrors existing results for SGD for the ℓ_2 loss function. We give a more general result that holds for general convex loss functions and show that the error decays as $\mathcal{O}(1/T)$.

Finally, we provide empirical results which suggest that our scheme can outperform existing gradient coding schemes (in particular, the FR-based approach of [18], [19] and BGC [18]) in some parameter regimes. We do not compare our scheme empirically to that of [17], [21], [22] because they requires more work on the master's end (to encode and decode) and are thus not directly comparable to our work.

C. Organization

We give a more precise definition of our set-up in Section II. We describe the SGC algorithm in Section III. In Section IV, we give a more detailed overview of both our theoretical and empirical results, which are fleshed out in Sections V and VI respectively. The proofs of our results can be found in the Appendix. We provide more detail on related work in Section VII.

II. Setup

A. Probabilistic Model of Stragglers

In this paper, we adopt a probabilistic model of stragglers. More precisely, we assume that at every iteration each worker may be a straggler with some probability p, and this is independent between workers and between iterations. Our probabilistic model is similar to the model in [9], [17]–[21] and is in contrast to the worst-case model assumed by much of the literature on coded computation. (See Remark 1). In our numerical simulations, we relax the assumption of independence and show that similar results hold when the identities of the stragglers are somewhat persistent from round to round and change only after a fixed number of iterations.

B. Computational Model

Our computational model has two stages, a distribution stage and a computation stage.

In the *distribution stage*, the master encodes the data using unequal data repetition code. More precisely, the master can decide to send each row $\mathbf{a}_i$ of A to d_i different workers. We refer to the parameter $d = \frac{1}{m}\sum_{i=1}^{m} d_i$ as the *average redundancy* of the scheme.

The *computation stage* is made up of rounds, each of which contains two repeating steps. In the first step, the master does some local computation and then sends a message to each

worker. In the second step, each worker does some local computation and tries to send a message back to the master; however, with probability p the message may not reach the master. Then the round is over and the master repeats the first step to begin the next round. We refer to the total amount of communication per round as the *communication* of the scheme. We allow each worker to send only one message to the master to reduce the communication.

III. Stochastic Gradient Coding

In this section, we describe our solution, which we call *Stochastic Gradient Coding* (SGC). The idea behind SGC is extremely simple. It is very much like the Ignore–Stragglers–SGD algorithm described above, except we introduce a small amount of redundancy. We describe the distribution stage and the computation stage of our algorithm below. Our scheme has parameters $d_1, \ldots, d_m$, which control the redundancy of each row, and a parameter γ_t which controls the step size. We will see in the theoretical and numerical analyses how to set these parameters.

In our analysis, we focus on *pair-wise balanced schemes.*

Definition 1: We say that a distribution scheme that sends $\mathbf{a}_i$ to d_i different workers is *pair-wise balanced* if for all $i \neq i'$, the number of workers that receives $\mathbf{a}_i$ and $\mathbf{a}_{i'}$ is $\frac{d_i d_{i'}}{n}$.

Notice that with a completely random distribution scheme, the expected number of workers who receive both $\mathbf{a}_i$ and $\mathbf{a}_{i'}$ for $i \neq i'$ is equal to $\frac{d_i d_{i'}}{n}$. In our analysis, it is convenient to deal with schemes that are exactly pair-wise balanced. However, for small d_i it is clear that no such schemes exist (indeed, we may have $\frac{d_i d_{i'}}{n} < 1$). In our simulations, we choose a uniformly random scheme[2] which seems to work well (see Section VI). We believe that our analysis should extend to a random assignment as well, although for simplicity we focus on pair-wise balanced schemes in our theoretical results.

The way SGC works is as follows:

- *Distribution Stage:* The master creates d_i copies of each row $\mathbf{a}_i$, $i = 1, \ldots, m$, and sends them to d_i distinct workers according to a pair-wise balanced scheme. We denote by S_j, $j = 1, \ldots, n$, the set of indices of the data vectors given to worker $\mathtt{W}_j$, i.e., $S_j = \{i; \mathbf{a}_i \text{ is given to } \mathtt{W}_j\}$.
- *Computation Stage:* At each iteration t, the master sends $\boldsymbol{\beta}_t$ to all the workers. Each worker $\mathtt{W}_j$ computes

$$f_j(\boldsymbol{\beta}_t) \triangleq \gamma_t \sum_{i \in S_j} \frac{1}{d_i(1-p)} \nabla \mathcal{L}\big(\mathbf{a}_i, \boldsymbol{\beta}_t\big) \tag{4}$$

and sends the result to the master. The master aggregates all the received answers from non straggler workers, sums them and updates $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \gamma_t \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{\mathcal{I}_i^j}{d_i(1-p)} \nabla \mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t),$$

where $\mathcal{I}_i^j$ is the indicator function for worker j being non straggler and having obtained point $\mathbf{a}_i$ during the data distribution, i.e.,

$$\mathcal{I}_i^j = \begin{cases} 1 & \text{if worker } j \text{ is non straggler} \\ & \text{and has point } \mathbf{a}_i, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathcal{I}_i^j$ depends on the iteration t, however we drop t from the notation for notational convenience since the value of t will be clear from the context.

For use below, we define

$$\hat{\mathbf{g}}_t \triangleq \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{\mathcal{I}_i^j}{d_i(1-p)} \nabla \mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t). \tag{5}$$

We call $\hat{\mathbf{g}}_t$ the estimate of the gradient at iteration t which estimates the exact gradient of the loss function in (3),

$$\mathbf{g}_t \triangleq \sum_{i=1}^{m} \nabla \mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t).$$

IV. Summary of Our Main Results

In this section, we summarize both our theoretical and numerical results.

A. Theoretical Results

Our main theoretical contributions are to derive results for SGC that mirror known results for SGD and Batch–SGD. There are two important differences between our results and those for Batch–SGD.

1) First, one of our goals is to show how the error $\|\boldsymbol{\beta}^* - \boldsymbol{\beta}_t\|_2^2$ depends on the redundancy parameter d; we show that it is roughly like $1/d$. This explains why SGC can work much better than Ignore–Stragglers–SGD (say, so that $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|_2^2$ is half as large), even with relatively low redundancy (say, $d = 2$). In Batch–SGD we always have $d = 1$.
2) Second, it is nontrivial to adapt existing results for Batch–SGD to our setting. The reason is that the batches are not uniform in our setting; rather, they depend on the way that the data is distributed. We note that this is true even if the data is distributed randomly to begin with: in that case it is true that the marginals of the batches are uniformly random (that is, in each round the set of gradients that the master receives is a uniformly random subset of all of them) but because the randomness from the initial distribution is fixed throughout the computation, if we view it this way then the batches are no longer independent. The main technical challenge in our analysis (in particular, the proof of Theorem 1 below) is to deal with this issue.[3]

We adapt existing result from the SGD literature to prove a tighter bound that holds for arbitrary convex loss functions. And we derive a stronger convergence guarantee for the ℓ_2 loss function.

[2]In our simulations, we assign rows to d_i workers uniformly at random, which approximates a pair-wise balanced scheme. Similarly, the BGC construction of [18] approximates a pair-wise balanced scheme where each row is assigned to d workers uniformly at random, i.e., $d_i = d$ for all $i \in [m]$.

[3]We note that this is not an issue for our proof of Theorem 2, since we are able to adapt existing results that depend only on the mean and variance of the gradient estimates.

Special Case: ℓ_2 Loss Function: We begin with a result which is specialized for the ℓ_2 loss function. This result is of a similar flavor as the results of [25]–[27] on SGD and the randomized Kaczmarz algorithm.[4] Those works show that the speed of convergence is exponential to begin with, and then begins to decay polynomially like $1/t$ once an unavoidable limit is reached. In this work, we show an analogous result for the ℓ_2 loss function. In this case we show that the convergence is exponential to begin with, until the noise is on the order of ℓ_2 normal of the residual $\mathbf{r} \triangleq X\boldsymbol{\beta}^* - \mathbf{y}$, and then it begins to decay polynomially like $1/(dt)$.

Thus, our analysis generalizes the case when $d = 1$ (aka, Ignore–Stragglers–SGD), and we see that as the repetition factor d increases, the error of SGC decreases. We state our main theorem informally below, and we state the formal version in Section V. Throughout the paper we use the superscript T to denote the transpose of a matrix.

Theorem 1 (Informal; see Theorem 3 for a Formal Version): Consider an SGC algorithm run on a matrix $A \triangleq [X|\mathbf{y}]$ of dimension $m \times (\ell + 1)$ distributed to n workers. Suppose that the distribution scheme is pairwise balanced, and that each row $\mathbf{a}_i$ of $A = [X|\mathbf{y}]$ is sent to d_i different workers, where d_i is chosen proportional to $\|\mathbf{x}_i\|_2^2$.

Suppose that n is sufficiently large and that

$$d = \frac{1}{m}\sum_{i=1}^{m} d_i \geq 8\left(\frac{p}{1-p}\right).$$

Choose an error tolerance $\varepsilon > 0$. Then, it is possible to choose a step size γ_t at each step t so that the following guarantee holds on the iterates $\boldsymbol{\beta}_T$ of SGC, for $T \geq 2\log(1/\varepsilon^2)$:

$$\mathbb{E}\left[\left\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\right\|_2^2\right] \leq \varepsilon^2\left\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\right\|_2^2 + \frac{\|\tilde{\mathbf{r}}\|^2}{d \cdot T} \cdot \left(\log^2(1/\varepsilon)\frac{p}{1-p}\right),$$

where $\tilde{\mathbf{r}} = (X\boldsymbol{\beta}^* - \mathbf{y})/\|X^T X\|_2$.

That is, if the residual $\tilde{\mathbf{r}}$ is very tiny, so that the second term is smaller than the first, then the algorithm reaches accuracy ε in roughly $\log(1/\varepsilon)$ steps. However, if $\tilde{\mathbf{r}}$ is larger, then the convergence becomes polynomial, matching what we expect from SGD. In this second case, the difference is that the replication factor d appears in the denominator, so that when d is larger, the error is smaller, explaining why replication helps. Notice that if p is constant, we expect good performance when $d = O(1)$. In contrast, to exactly simulate gradient descent via coding would require $d = \Omega(n)$.

The main difficulty in proving Theorem 3 (the formal version of Theorem 1) is that because the data distribution is fixed ahead of time, the "batches" that the master acquires in each round are not uniformly random, but rather come from some distribution determined by the data distribution.

Beyond ℓ_2 Loss Function: Our result above is limited in that it only applies to the ℓ_2 loss function. We believe that the analysis of Theorem 3 should apply to general loss functions,

[4]We note that [26] also holds for more general loss functions.

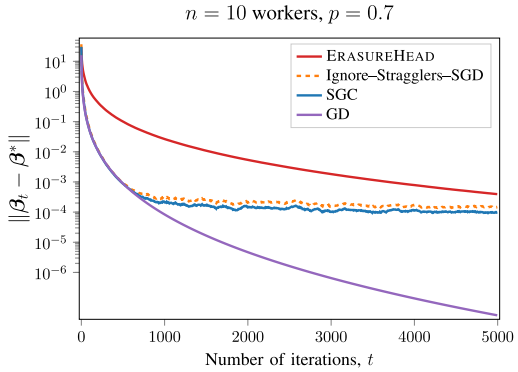


Fig. 1. Comparison between Ignore–Stragglers–SGD, SGC and ErasureHead in terms of the distance between $\boldsymbol{\beta}_t$ and $\boldsymbol{\beta}^*$ the value of $\boldsymbol{\beta}$ that minimizes the loss function. SGC outperforms Ignore–Straggler–SGD at the expense of adding small redundancy, $d = 2$ in this example.

but for now we observe that in fact a convergence rate of $1/t$ does follow for SGC from a result of [14].

In that work, the authors give a general analysis of stochastic gradient descent, which works as long as (in our language) the master is computing an unbiased estimator of the gradient. The convergence speed of the algorithm then depends on the variance of this estimate. This result applies in our setting.

Theorem 2 (Informal; see Theorem 4 for a Formal Version): Suppose that SGC is run on a matrix $A \triangleq [X|\mathbf{y}]$ of dimension $m \times (\ell + 1)$ distributed to n workers. Suppose that the distribution scheme is pairwise balanced, and that each row $\mathbf{a}_i$ of A is sent to d_i different workers, $d_i \leq n$. Consider a version of the optimization problem in (1) where $\boldsymbol{\beta}$ is constrained to a convex set $\mathcal{W}$. Under some mild assumptions on the loss function $\mathcal{L}$ and assuming there exists a constant C such that

$$\|\nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})\|_2^2 \leq C^2$$

for all $i \in [n]$ and for all $\boldsymbol{\beta} \in \mathcal{W}$, then there is a way to choose the step size γ_t at each step t so that the error after T iterations is bounded by

$$\mathbb{E}\left[\left\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\right\|_2^2\right] \leq \mathcal{O}(1/T).$$

The proof of Theorem 2 (given [14, Lemma 1]) boils down to showing that our gradient estimator $\hat{\mathbf{g}}_t$ is an unbiased estimator of the true gradient and that $\mathbb{E}[\|\hat{\mathbf{g}}_t\|_2^2]$ is bounded for all t, which we do in the Appendix.

We give more precise statements of these theorems in Section V, and prove them in the Appendix.

B. Numerical Simulations

We run extensive simulations on synthetic data A of dimension 1000×100 generated from a Gaussian distribution. We compare SGC to four other algorithms detailed in Section VI and show that SGC outperforms all other algorithms when there are many stragglers. A typical result is shown in Figure 1. In it, we observe that SGC and ErasureHead outperform Ignore–Stragglers–SGD at the expense of doubling the redundancy. In Figure 2 we plot the convergence of approximate

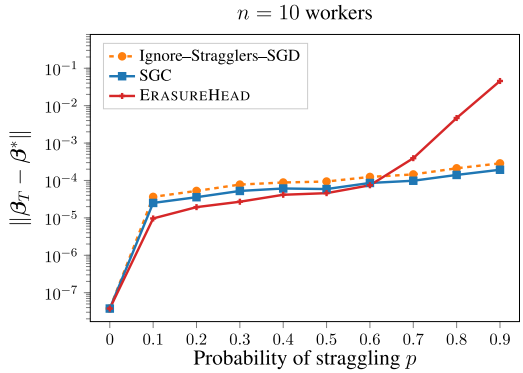


Fig. 2. Final convergence of all algorithms run for $T = 5000$ iterations as function of p the probability of workers being stragglers. We omit GD in this setting, because it has the same performance as all algorithms when $p = 0$.

gradient codes as function of p. We observe that SGC outperforms ErasureHead when the number of stragglers is large, $p > 0.6$. As expected, the approximate algorithms have worse accuracy than full-blown gradient descent, but we note that implementing exact gradient descent with a p fraction of stragglers would require redundancy $d \approx pn \gg 2$. Moreover, we observe the flexibility of the approximate algorithms in the number of stragglers, and we note that computing GD exactly would lack this flexibility. In Section VI, we comment on how the dependency between stragglers affect the convergence of SGC. Our implementation is publicly available [28].

V. Theoretical Results

In this section we precisely state our theoretical results. We begin with a specialized result for the ℓ_2 loss function, and then include a result for more general loss functions.

A. Special Case: ℓ_2 Loss Function

We begin with a result that holds for the special case of an ℓ_2 loss function, aka, regression. Inspired by the approach of [27] for SGD, our approach is to consider a *weighted* distribution scheme; that is, we choose d_i proportionally to $\|\mathbf{x}_i\|_2^2$. While the statement below is only for the ℓ_2 loss function, we conjecture that it holds for more general loss functions.

Define a parameter

$$\mu = \frac{\frac{1}{m}\|X\|_F^2}{\|X^T X\|}.$$

This parameter measures how incoherent X is. If X is orthogonal, $\mu = 1$, while if, for example, X is the all-ones matrix, then $\mu = 1/m$. It is not hard to check that $\mu \in [0, 1]$.

Suppose that $\mathcal{D}$ is a pair-wise balanced distribution scheme which sends $\mathbf{a}_i$ to d_i different workers, where

$$d_i = \sigma \cdot \|x_i\|_2^2, \tag{6}$$

$$\sigma = \frac{md}{\|X\|_F^2} = \frac{d}{\mu\|X^T X\|}, \tag{7}$$

$$d = \frac{1}{m}\sum_{i\in[m]} d_i. \tag{8}$$

The parameter d is the average redundancy of the scheme that will control σ and the d_i's. Notice that, as stated, it is possible that the d_i end up being non-integers; in the following, we will assume for simplicity below that $d_i \in \mathbb{Z}$ for all i. Notice that if $\|\mathbf{x}_i\|_2 = 1$ for all i, then this will be the case because we can choose $d_i = d$ to be any integer of our choice, and this defines σ.

Theorem 3: Consider an SGC algorithm run on a matrix $A \triangleq [X|\mathbf{y}]$ of dimension $m \times (\ell + 1)$ distributed to n workers according to a pairwise balanced distribution scheme with d_i as described above, with loss function

$$\mathcal{L}([X|\mathbf{y}], \boldsymbol{\beta}) = \|X\boldsymbol{\beta} - \mathbf{y}\|_2^2,$$

and assume that the degrees $d_i \leq n$ are all integers.

Suppose the stragglers follow the stochastic model of Section II, and that each worker is a straggler independently with probability p. Choose $\varepsilon > 0$ and choose $T \geq 2\log(1/\varepsilon^2)$.

Suppose that the number of workers n satisfies $n \geq 8\left(\frac{p}{1-p}\right)$, and that

$$8\mu\left(\frac{p}{1-p}\right) \leq d.$$

Choose a step size

$$\gamma_t = \frac{1}{\|X^T X\|} \cdot \min\left\{\frac{1}{2}, \frac{\log(1/\varepsilon^2)}{t}\right\}.$$

Then, after T iterations of SGC, we have

$$\mathbb{E}\left[\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\|_2^2\right] \leq \varepsilon^2\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|_2^2 + \frac{\|\tilde{\mathbf{r}}\|^2\mu}{dT}\left(\log^2(1/\varepsilon^2)\left(\frac{p}{1-p}\right)\right)$$

where the expectation is over the stragglers in each of the T iterations of SGC and where μ is as above, and where

$$\tilde{\mathbf{r}} = \frac{\|X\boldsymbol{\beta}^* - \mathbf{y}\|_2}{\|X^T X\|_2^{1/2}}.$$

Corollary 1: Suppose that $X\boldsymbol{\beta}^* = \mathbf{y}$ (that is, we are solving a system for which there is a solution) and that $n \geq 8p/(1-p)$. Then the algorithm described in Theorem 3 converges with

$$\mathbb{E}\left[\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\|_2^2\right] \leq \varepsilon^2\|\boldsymbol{\beta}_0 - \boldsymbol{\beta}^*\|^2$$

provided that $T \geq 2\log(1/\varepsilon^2)$ and $d \geq 8\mu p/(1-p)$.

In particular, since $\mu \leq 1$, this says that we need to take $d \gtrsim p/(1-p)$ and the algorithm converges extremely quickly.

B. Beyond ℓ_2 Loss Function

Now, we consider a constrained version of the problem given in (1), where $\boldsymbol{\beta}$ belongs to a bounded set $\mathcal{W}$. In this section, we state a result for general loss functions $\mathcal{L}$ which are λ-strongly convex.

Definition 2 (Strongly Convex Function): A function $\mathcal{L}$ is λ-strongly convex, if for all $\boldsymbol{\beta}, \boldsymbol{\beta}' \in \mathbb{R}^\ell$ and any subgradient $\mathbf{g}$ of $\mathcal{L}$ at $\boldsymbol{\beta}$,

$$\mathcal{L}(\boldsymbol{\beta}') \geq \mathcal{L}(\boldsymbol{\beta}) + \langle \mathbf{g}, \boldsymbol{\beta}' - \boldsymbol{\beta}\rangle + \frac{\lambda}{2}\|\boldsymbol{\beta}' - \boldsymbol{\beta}\|_2^2. \tag{9}$$

TABLE I
SUMMARY OF THE STOCHASTIC ALGORITHMS THAT WE IMPLEMENT IN OUR SIMULATIONS

Algorithm	Brief description
Stochastic Gradient Code (SGC)	The master sends each data vector $\mathbf{x}_i$ to d_i workers chosen at random, where d_i is proportional to the ℓ_2 norm of $\mathbf{x}_i$ and is computed as in (6) with $d = 2$. Each worker sends to the master the weighted sum of its partial gradients as in (4). The master computes the gradient estimate as the sum of the received results from non straggling workers.
Bernoulli Gradient Code (BGC) [18]	Similar to SGC but all data vectors are replicated d times, i.e., $d_i = 2$ for all $i \in [m]$.
ERASUREHEAD [18], [19]	Partitions the data set equally and sends each partition to d workers. Workers send the sum of the partial gradients to the master who computes the gradient estimate as the sum of distinct received partial gradients divided by total number of data vectors.
Ignore–Stragglers–SGD	Partitions the data among the workers with no redundancy. Workers send the sum of the partial gradients to the master who computes the gradient estimate as the sum of distinct partial gradients divided by the average number of data vectors received per iteration.
SGC–Send–All	Same as SGC with one difference: at each iteration the workers send all the partial gradients to the master. The master computes the gradient estimate as the sum of *distinct* partial gradients divided by the average number of data vectors received per iteration.

Theorem 4 below follows from the analysis in [14].

Theorem 4: Suppose that SGC is run on a matrix $A \triangleq [X|\mathbf{y}]$ of dimension $m \times (\ell + 1)$ distributed to n workers with each row of A sent to d_i different workers, $d_i \leq n$, according to a pairwise balanced distribution scheme. Consider a version of the optimization problem in (1) where $\boldsymbol{\beta}$ is constrained to a convex set $\mathcal{W}$, i.e.,

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta} \in \mathcal{W}} \mathcal{L}(A, \boldsymbol{\beta}),$$

and at each step of the algorithm $\boldsymbol{\beta}_{t+1} = \Pi_{\mathcal{W}}(\boldsymbol{\beta}_t - \gamma_t \hat{\mathbf{g}}_t)$, where Π is the projection operator. Let p denote the probability of a given worker being a straggler at a given iteration. Suppose that the loss function $\mathcal{L}$ is λ-strongly convex with respect to the optimal point $\boldsymbol{\beta}^* \in \mathcal{W}$, and that all of the partial gradients $\nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})$ are bounded for $i \in [n]$ and $\boldsymbol{\beta} \in \mathcal{W}$, i.e., there exists a constant C so that

$$\|\nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})\|_2^2 \leq C, \qquad \forall \boldsymbol{\beta} \in \mathcal{W}, i \in [n].$$

Suppose that the step size is set to be $\gamma_t = 1/(\lambda t)$. Then the error after T iterations is bounded by

$$\mathbb{E}\left\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\right\|_2^2 \leq \frac{4}{\lambda^2 T} \cdot \Phi \tag{10}$$

where $d_{\min} \triangleq \min_{i \in [m]} d_i$ and

$$\Phi \triangleq mC^2\left(\frac{p}{1-p} \cdot \frac{1}{d_{\min}} + \frac{(m-1)p}{n(1-p)} + m\right).$$

This shows that SGC does have a convergence rate of $O(1/T)$, matching regular SGD [14, Lemma 1]. This means that at least the convergence rate is not hurt by the fact that the data assignment is fixed. However, unlike Theorem 3, this result does not always significantly improve as d increases (although we note that the bound above *is* decreasing in $d_{\min}$, so in some parameter regimes—when $n \gg m$ and p is close to 1—this does indicate some improvement). We leave it as an interesting open problem to fully generalize our result of Theorem 3 to general loss functions.

VI. SIMULATION RESULTS

A. Simulation Setup

We simulated the performance of SGC on synthetic data X of dimension 1000×100. The data is generated as follows: each row vector $\mathbf{x}_i$ is generated using a Gaussian distribution $\mathcal{N}(0, 100)$. We pick a random vector $\bar{\boldsymbol{\beta}}$ with components being integers between 1 and 10 and generate $y_i \sim \mathcal{N}(\langle \mathbf{x}_i, \bar{\boldsymbol{\beta}} \rangle, 1)$. Our code and the generated data set can be found in [28].

We run linear regression using the ℓ_2 loss function, i.e.,

$$\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t) = \frac{1}{2}\left(\langle \mathbf{x}_i, \boldsymbol{\beta}_t \rangle - y_i\right)^2.$$

We show simulations for $n = 10$ workers. For each simulation we vary the probability of a worker being a straggler from $p = 0$ to $p = 0.9$ with a step of 0.1. We run the algorithm for 5000 iterations with a variable step size given[5] by

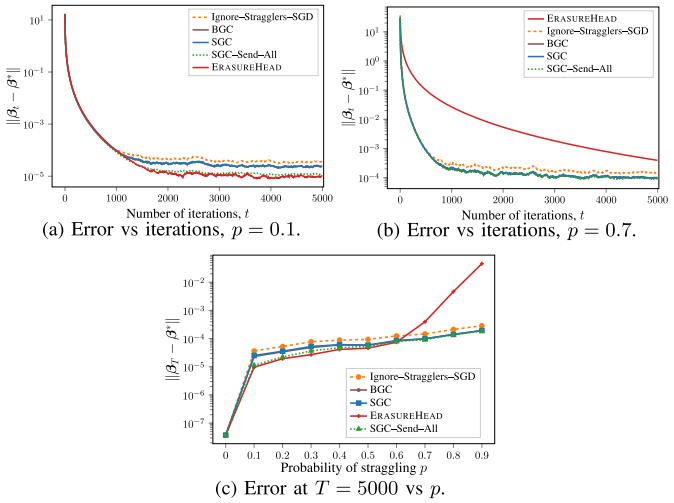
$$\gamma_t = 7\frac{\ln(10^{100})}{t^{0.7}}. \tag{11}$$

For all simulations, we run each experiment 10 times and average the results. For SGC, each data vector $\mathbf{x}_i$ is replicated d_i times, where the d_i's are computed as in (6) and (7) with $d = 2$. Then, each d_i is rounded to the nearest integer. Due to rounding, the actual value of d given in (8) will be close to 2. In our generated data set, the majority of the d_i's are equal to 2 while the others are either 1 or 3 resulting in average redundancy $d = 2.024$. For the other algorithms in Table I, the average redundancy d is chosen to be exactly equal to 2.

We omit comparing SGC to the gradient codes in [9] and [21] because they do not match our setting; the former requires a high redundancy factor d and the latters requires the master to run a decoding algorithm at each iteration.

Our metric used to compare the different algorithms is error versus number of iterations which does not capture the computation overhead induced by redundantly distributing the data to the workers. However, we envision SGC to be appealing to applications in which the computation overhead of a given

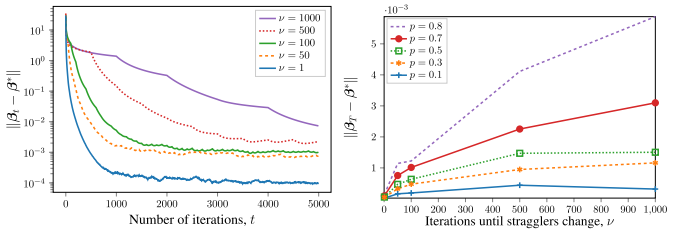
[5]In our theoretical analysis we assumed that the step size γ_t is proportional to $1/t$. In our numerical simulations, we tried different functions of γ_t and observed that the one in (11) gives better convergence rate for all the considered algorithms.



(a) Error vs iterations, $p = 0.1$. (b) Error vs iterations, $p = 0.7$.

(c) Error at $T = 5000$ vs p.

Fig. 3. Convergence as function of probability of workers being stragglers p is shown for small $p = 0.1$ in (a) and for large $p = 0.7$ in (b) for $n = 10$ workers. SGC convergence has two phases: an exponential decay in the beginning until it reaches an error floor. SGC has same performance as BGC, but outperforms ErasureHead for large values of p. In (c) the error floor at $T = 5000$ iterations is shown versus p.



(a) Error versus iterations, $p = 0.7$. (b) Error at iteration $T = 5000$ versus ν.

Fig. 4. The effect of the dependency of stragglers across iterations on the performance of SGC. We assume that the identity of the stragglers change every ν iterations. In (a) the convergence of the error as function of the number of iterations is shown for different values of ν and $p = 0.7$. SGC maintains an exponential decay in the error for the tested values of ν and p. However, the rate of the decay decreases and the error floor increases with the increase of ν. In (b), the error at iteration $T = 5000$ as function of ν is shown for different values of p.

worker is not the main bottleneck and the delay experienced by the master is dominated by other factors such as network and system management delays. This applies to all algorithms except Ignore–Stragglers–SGD.

B. Convergence

In Figures 3a and 5a, we plot the error $\|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|$ for up to 5000 iterations for small and large probability of workers being stragglers, namely for $p = 0.1$ and $p = 0.7$, respectively. Here, $\boldsymbol{\beta}^*$ given in (1) is computed using the pseudoinverse of X, i.e., $\boldsymbol{\beta}^* = (X^T X)^{-1} X^T Y$. We notice that for all p the convergence rate of SGC exhibits two phases: an exponential decay followed by an error floor. To see the benefit of replication, we compare SGC to Ignore–Stragglers–SGD. Both have the same performance in the exponential phase, but SGC has a lower error floor due to redundancy. The normalised difference between the error floor of SGC and Ignore–Stragglers–SGD is 0.3, i.e., SGC with $d = 2$ brings a 30% increase in performance over Ignore–Stragglers–SGD. A lower bound on the performance of SGC is SGC–Send–All which has a lower error floor because it computes a better estimate of the gradient at the expense of a higher communication cost. However, as p increases the gap between the two error floor of SGC and SGC–Send–All decreases. In our simulations, we notice the error floor of both algorithms almost match for $p \geq 0.6$ as can be seen in Figure 3c.

For our chosen data set, SGC and BGC have similar performance. This is mainly due to the fact that most of the data vectors have the same replication factor in BGC and SGC which is 2 times. For other data sets with more variance in the d_i's, we observe that SGC can have better performance. ErasureHead has better error floor than SGC for small values of p. However, for large p the rate of the exponential decay drastically decreases for ErasureHead.

C. Dependency Between Stragglers Across Iterations

Our theoretical analysis assumes that the stragglers are independent across iterations. We check the effect of this dependency on the numerical performance of SGC. We use a simple model to enforce dependency of stragglers across iterations by fixing the stragglers for ν iterations, after which the stragglers are chosen again randomly and iid with a probability p and this is repeated until the algorithm stops. The special value of $\nu = 1$ implies that the stragglers are independent. A large value of ν implies a longer dependency among the stragglers across iterations. We observe in Figure 4 that SGC still maintains the two phases behavior. However, as ν increases, the rate of convergence decreases and the error floor increases.

D. Effect of Increasing Redundancy

We present in this section the convergence results for the same experiments as before for $n = 100$ workers and average redundancy $d = 5$ (see Figure 5b). The goal is to check the effect of increasing d on the error floor of SGC. In particular, we are interested in the difference between SGC and Ignore–Stragglers–SGD. The normalised difference between the error floor of SGC and Ignore–stragglers–SGD is 0.5, i.e., the error floor of SGC with $d = 5$ is half the error floor of Ignore–Stragglers–SGD.

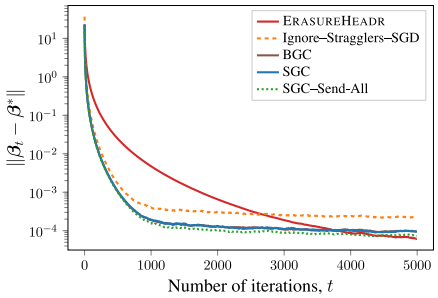
VII. Related Work

In this section we survey the related work more broadly than in the introduction.

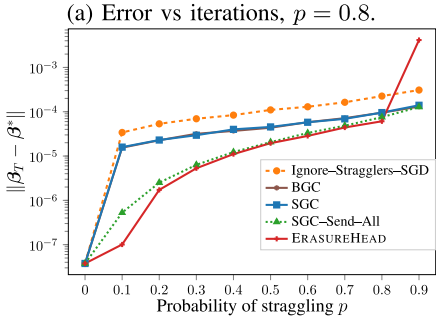
A. Coding Techniques for Straggler Mitigation

Straggler workers are the bottleneck of distributed systems and mitigation of stragglers is a must [2]. Amongst popular techniques, coding theoretic techniques are being used for straggler mitigation in different applications such as machine learning, see, e.g., [6], [7], [9], [17], [29]–[31], [31]–[34], [34]–[40], matrix multiplication, see, e.g., [8], [41]–[49], linear transforms, see, e.g., [50]–[53], and content download, see, e.g., [54]–[59].

There is a growing body of work on *gradient coding*, which focuses on the special and important case of gradient descent.



(a) Error vs iterations, $p = 0.8$.



(b) Error at $T = 5000$ vs p.

Fig. 5. Performance of SGC for $n = 100$ and $d = 5$. In (a) we show the convergence as function of probability of workers being stragglers $p = 0.8$. SGC with $d = 5$ brings a 50% decrease in the error floor compared to Ignore–Stragglers–SGD. In (b) the error floor at $T = 5000$ iterations is shown versus p.

For example, [6], [7], [29] present coding techniques to avoid stragglers and perform a gradient descent update at each iteration, i.e., at each iteration the master observes the gradient evaluated at the whole data matrix A. In this framework, the master distributes the data to workers with redundancy. In the works cited above, the goal is to *exactly* compute the gradient, even in the presence of stragglers, and the amount of redundancy depends on the number of stragglers to be tolerated. However, it is still useful (and often much more efficient) to approximate the gradient, rather than computing it exactly. This setting is what our work focuses on and we discuss it in more detail below.

B. Approximate Gradient Coding

In *approximate gradient coding*, the goal is not to compute the gradient exactly, but rather to compute an approximation to the gradient. This is the approach that we take, and there are several previous works which do this. The ones most relevant to our work are [3], [9], [17]–[23], which we discuss in more detail below.

In [3], the authors consider a setting where there are "extra" workers, and all the workers sample data randomly from X at each round. Again, the master waits for the fastest $n-s$ workers to complete. This is quite similar to the Ignore–Stragglers–SGD scheme; the difference is that in Ignore–Stragglers–SGD, the data is partitioned among the workers and the data held by the workers is fixed throughout the algorithm, while in [3] the workers sample a fresh subset of data at each iteration. The sampling of the data in [3] is done with replacement which may incur redundancy in the data held by the workers.

In [17], the authors focus on linear loss functions and use LDPC codes to encode the data sent to the workers. If fewer than s stragglers are present, then the master can compute the exact gradient. However, if more than s stragglers are present the master leverages the LDPC code to computes an estimate of the gradient. In [21] the authors propose a replication-based scheme to distribute the data to the workers. The data distribution scheme can be seen as a bipartite graph with the data vectors on one side and the workers on the other. An edge is drawn between a data vector i and a worker j if the vector $\mathbf{a}_i$ is given to worker j. The distribution of the degrees of the nodes corresponding to data vectors and to workers are drawn according to an LDGM scheme. The main drawback is that the master has to run a decoding algorithm to decode the sum of the partial gradients at each iteration.

In [9], [18], the authors present approximate gradient schemes. The main idea is to bound the distance between the computed approximate gradient and the actual gradient at each step. Both of these schemes have a similar framework to ours: the data is replicated among nodes according to an appropriate design, and the workers return a linear combination of the gradients that they can compute. In [9] the authors present a data replication scheme based on Ramanujan graphs. In [18] the authors present two constructions. The first is based on fractional repetition codes (FRC) and partitions the workers and data into blocks; within a block, each worker receives every data point from the corresponding block. The second construction called *Bernoulli Gradient Coding* (BGC) distributes each data point randomly to d different workers. We note that BGC is an approximation of the pairwise-balanced schemes we consider and can be seen as a case of SGC when all the data $\mathbf{a}_i$ have the same norm.

In [22] the authors present a data replication scheme based on balanced incomplete block designs. The scheme guarantees that the computed estimate of the gradient is close to the actual gradient even if the stragglers are not chosen randomly across iterations. This work is motivated by systems in which stragglers cannot be modeled statistically.

In [23], the authors also study a replication-based scheme, but they focus on a model where individual workers return many gradients asynchronously, rather than returning a linear combination of the gradients they can compute.

In [20] the authors present fundamental bounds on the error between the approximated gradient and true gradient at each round as function of the redundancy. In [19], the authors analyze the convergence rate of the fractional repetition scheme presented in [18] and show that under standard assumptions on the loss function, the algorithm maintains the convergence rate of centralized stochastic gradient descent.

C. Other Work on Stochastic Gradient Descent

Beginning with its introduction in [60], there has been a huge body of work on stochastic gradient descent (in a setting without stragglers), and we draw on this mathematical framework for our theoretical results. In the special case of ℓ_2 loss (which we focus on in this work), SGD coincides with the randomized Kaczmarz method [25], [26], and our proof of Theorem 3 is inspired by these analyses.

There has been a great deal of work on SGD and Batch–SGD in distributed settings where there is no redundancy of the data between workers and dealing with stragglers is not the primary concern, see, e.g., [27], [61]–[64]. In addition to the synchronous setting in which the master waits for all workers to make an update on $\boldsymbol{\beta}$, there has been work on the *asynchronous* setting in which the master makes an update on $\boldsymbol{\beta}$ every time a worker gets back, see, e.g., [12], [65]–[68]. For example, [68] shows that asynchronous SGD asymptotically behaves similarly to synchronous SGD in terms of convergence for convex optimization and under the similar assumptions on the loss function. In [67], the authors compare the convergence rate of synchronous and asynchronous SGD as a function of the wall clock time rather than number of iterations.

APPENDIX

Proof of Theorem 3

In this section, we prove Theorem 3. In the case when the loss function is

$$\mathcal{L}([X|y], \boldsymbol{\beta}) = \frac{1}{2}\|X\boldsymbol{\beta} - \mathbf{y}\|_2^2,$$

we have

$$\nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}) = (\langle \mathbf{x}_i, \boldsymbol{\beta}\rangle - y_i)\cdot \mathbf{x}_i,$$

so that

$$\sum_i^m \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}) = X^T(X\boldsymbol{\beta} - \mathbf{y}) = \nabla\mathcal{L}(A, \beta).$$

Fix an iteration t. Let Z_i (which depends on t; we suppress this dependence in the notation) be defined by

$$Z_i = \sum_{j=1}^n \mathcal{I}_i^j.$$

That is, Z_i is the number of workers who hold $\mathbf{a}_i$ who are not stragglers at round t. Thus, Z_i is a binomial random variable with mean $d_i(1-p)$ and variance $d_i p(1-p)$. Let

$$\tilde{Z}_i = Z_i - \mathbb{E}[Z_i],$$

so that

$$\mathbb{E}\Big[\tilde{Z}_i^2\Big] = d_i p(1-p)$$

and

$$\mathbb{E}\big[\tilde{Z}_i\tilde{Z}_j\big] = \frac{d_i d_j}{n} p(1-p).$$

From the definition of $\boldsymbol{\beta}_{t+1}$ and replacing $\hat{\mathbf{g}}_t$ by its value from (5), we have

$$\begin{aligned}
\boldsymbol{\beta}_{t+1} &= \boldsymbol{\beta}_t - \gamma_t\hat{\mathbf{g}}_t \\
&= \boldsymbol{\beta}_t - \gamma_t \sum_{j=1}^n \sum_{i=1}^m \frac{\mathcal{I}_i^j}{d_i(1-p)} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}) \\
&= \boldsymbol{\beta}_t - \gamma_t \sum_{i=1}^m \frac{Z_i}{d_i(1-p)} \big(\langle \mathbf{x}_i, \boldsymbol{\beta}_t\rangle - y_i\big)\mathbf{x}_i \\
&= \boldsymbol{\beta}_t - \sum_{i=1}^m Z_i\delta_i\big(\langle \mathbf{x}_i, \boldsymbol{\beta}_t\rangle - y_i\big)\mathbf{x}_i,
\end{aligned}$$

where we define

$$\delta_i = \frac{\gamma_t}{d_i(1-p)}.$$

(Notice that δ_i also depends on t; we suppress this for notational convenience).

Expanding out the terms, we have

$$\begin{aligned}
\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^* = \boldsymbol{\beta}_t - \boldsymbol{\beta}^* &- \sum_{i=1}^m \mathbb{E}[Z_i]\delta_i\mathbf{x}_i\mathbf{x}_i^T\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&- \sum_{i=1}^m \tilde{Z}_i\delta_i\mathbf{x}_i\mathbf{x}_i^T\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&- \sum_{i=1}^m \mathbb{E}[Z_i]\delta_i\big(\langle \mathbf{x}_i, \boldsymbol{\beta}^*\rangle - y_i\big)\mathbf{x}_i \\
&- \sum_{i=1}^m \tilde{Z}_i\delta_i\big(\langle \mathbf{x}_i, \boldsymbol{\beta}^*\rangle - y_i\big)\mathbf{x}_i
\end{aligned}$$

where we have split up $Z_i = \mathbb{E}[Z_i] + \tilde{Z}_i$ and

$$\big(\langle \mathbf{x}_i, \boldsymbol{\beta}_t\rangle - y_i\big)\mathbf{x}_i = \mathbf{x}_i\mathbf{x}_i^T\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) + \big(\langle \mathbf{x}_i, \boldsymbol{\beta}^*\rangle - y_i\big)\mathbf{x}_i.$$

Letting

$$\mathbf{r} \coloneqq X\boldsymbol{\beta}^* - \mathbf{y}$$

be the optimal residual and writing the above in matrix notation, we have

$$\begin{aligned}
\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^* = \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) &- (1-p)X^T D_d D_\delta X\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&- X^T D_{\tilde{Z}} D_\delta X\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&- (1-p)X^T D_d D_\delta \mathbf{r} \\
&- X^T D_{\tilde{Z}} D_\delta \mathbf{r},
\end{aligned}$$

where $D_{\tilde{Z}}$ is diagonal with entries $\tilde{Z}_i$, D_δ is diagonal with entries δ_i, and D_d is diagonal with entries d_i. Recalling that

$$\delta_i = \frac{\gamma_t}{(1-p)d_i}$$

we have

$$D_\delta \cdot D_d = \frac{\gamma_t}{1-p}.$$

Thus we can simplify the above as

$$\begin{aligned}
\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^* &= \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) - \gamma_t X^T X\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&\quad - X^T D_{\tilde{Z}} D_\delta X\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&\quad - \gamma_t X^T\mathbf{r} - X^T D_{\tilde{Z}} D_\delta \mathbf{r} \\
&= \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) - \gamma_t X^T X\big(\beta_t - \beta^*\big) \\
&\quad - X^T D_{\tilde{Z}} D_\delta X\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
&\quad - X^T D_{\tilde{Z}} D_\delta \mathbf{r}
\end{aligned}$$

using the fact that $X^T\mathbf{r} = 0$ since $\mathbf{r} = X\boldsymbol{\beta}^* - \mathbf{y}$ is the optimal residual. We simplify this further as:

$$\begin{aligned}
\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^* = \big(I - \gamma_t X^T X + X^T D_{\tilde{Z}} D_\delta X\big)\big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\
- X^T D_{\tilde{Z}} D_\delta \mathbf{r}.
\end{aligned}$$

Now we compute $\mathbb{E}[\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^*\|^2]$, where the expectation is over the choice of β_{t+1}, conditioned on $\boldsymbol{\beta}_t$. We let $E_1 \triangleq \mathbb{E}[\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}^*\|^2]$ and $\mathbf{b}_1 \triangleq (\boldsymbol{\beta}_t - \boldsymbol{\beta}^*)$, we have

$$\begin{aligned} E_1 = \mathbf{b}_1^T \Big[& \big(I - \gamma_t X^T X\big)^2 \\ & + X^T D_\delta \mathbb{E}[D_{\tilde{Z}} X X^T D_{\tilde{Z}}] D_\delta X \Big] \mathbf{b}_1 && (12) \\ & + \mathbf{r}^T D_\delta \mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] D_\delta \mathbf{r} && (13) \\ & + \mathbf{r}^T D_\delta \mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] D_\delta X \mathbf{b}_1 && (14) \\ & + \mathbf{r}^T D_\delta \mathbb{E}\big[D_{\tilde{Z}}\big] X X^T \big(I - \gamma_t X^T X\big) \mathbf{b}_1 && (15) \\ & + \mathbf{b}_1^T \big(I - \gamma_t X^T X\big) \big(X^T \mathbb{E}[D_{\tilde{Z}}] D_\delta X\big) \mathbf{b}_1. && (16) \end{aligned}$$

We handle each of these terms below. First, we observe that (15) and (16) are zero because $\mathbb{E}D_{\tilde{Z}} = 0$. In order to handle (12), (13), (14), we compute

$$\mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big].$$

The off-diagonal elements are given by

$$\mathbb{E}\big[\tilde{Z}_i \tilde{Z}_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle\big] = \frac{d_i d_j}{n} p(1-p) \langle \mathbf{x}_i, \mathbf{x}_j \rangle,$$

and the diagonal elements are given by

$$\mathbb{E}\Big[\tilde{Z}_i^2 \|\mathbf{x}_i\|^2\Big] = d_i p(1-p) \|\mathbf{x}_i\|^2 = d_i^2 p(1-p)/\sigma.$$

Thus,

$$\mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] = p(1-p)\bigg(\frac{1}{n} D_d X X^T D_d + \frac{1}{\sigma}\Big(I - \frac{D_d}{n}\Big) D_d^2\bigg).$$

Now we handle the terms (12) and (13). First, for (12), we have the equation shown on bottom of the page where in the last line we used the fact again that $D_\delta D_d = \gamma_t I/(1-p)$. Now we can bound this term by

$$\begin{aligned} (12) \le \bigg(& \big(1 - \gamma_t \|X^T X\|\big)^2 + \frac{p \gamma_t^2}{(1-p)n} \|X^T X\|^2 \\ & + \frac{\gamma_t^2 p}{(1-p)\sigma} \|X^T X\| \bigg) \|\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\|^2, \end{aligned}$$

where above we have used the fact that

$$\bigg\| X^T \Big(I - \frac{D_d}{n}\Big) X \bigg\| \le \big\| X^T X \big\|,$$

because $I - D_d/n$ is a diagonal matrix whose diagonal entries are all in [0, 1] (using the fact that $d_i \le n$ for all i). The second term (13) is bounded by

$$\begin{aligned} (13) &= \mathbf{r}^T D_\delta \mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] D_\delta \mathbf{r} \\ &= p(1-p) \mathbf{r}^T D_\delta \bigg(\frac{1}{n} D_d X X^T D_d \\ &\qquad + \Big(I - \frac{D_d}{n}\Big) \frac{1}{\sigma} D_d^2 \bigg) D_\delta \mathbf{r} \\ &= \frac{p(1-p)}{n} \mathbf{r}^T D_\delta D_d X X^T D_d D_\delta \mathbf{r} \\ &\quad + \frac{p(1-p)}{\sigma} \mathbf{r}^T \Big(I - \frac{D_d}{n}\Big) D_\delta D_d^2 D_\delta \mathbf{r} \\ &\le \frac{\gamma_t^2 p}{(1-p)n} \mathbf{r}^T X X^T \mathbf{r} + \frac{\gamma_t^2 p}{1-p} \cdot \frac{\mathbf{r}^T \Big(I - \frac{D_d}{n}\Big) \mathbf{r}}{\sigma} \\ &\le \gamma_t^2 \cdot \frac{p}{1-p} \cdot \frac{\|\mathbf{r}\|^2}{\sigma}, \end{aligned}$$

where we have used the fact that $X^T \mathbf{r} = 0$, and that $\mathbf{r}^T (I - D_d/n) \mathbf{r} \le \|\mathbf{r}\|^2$ because $d_i \le n$ for all i.

Finally we bound (14). We have, using our expression for $\mathbb{E}[D_{\tilde{Z}} X X^T D_{\tilde{Z}}]$ from above that

$$\begin{aligned} (14) &= \mathbf{r}^T D_\delta \mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] D_\delta X \mathbf{b}_1 \\ &= p(1-p) \mathbf{r}^T D_\delta \bigg(\frac{1}{n} D_d X X^T D_d \\ &\qquad + \frac{1}{\sigma}\Big(I - \frac{D_d}{n}\Big) D_d^2 \bigg) D_\delta X \mathbf{b}_1 \\ &= \frac{p(1-p)}{n} \mathbf{r}^T D_\delta D_d X X^T D_d D_\delta X \mathbf{b}_1 \\ &\quad + \frac{p(1-p)}{\sigma} \mathbf{r}^T D_\delta \Big(I - \frac{D_d}{n}\Big) D_d^2 D_\delta X \mathbf{b}_1. \end{aligned}$$

Using the fact that $D_\delta D_d = \gamma_t I/(1-p)$, we can write

$$\begin{aligned} (14) = & \frac{\gamma_t^2 p}{(1-p)n} \mathbf{r}^T X X^T X \mathbf{b}_1 \\ & + \frac{\gamma_t^2 p}{(1-p)\sigma} \mathbf{r}^T \Big(I - \frac{D_d}{n}\Big) X \mathbf{b}_1 \end{aligned}$$

Now, the first term is equal to zero because $\mathbf{r}^T X = 0$, and we have

$$\begin{aligned} (14) &= \frac{\gamma_t^2 p}{(1-p)\sigma} \mathbf{r}^T \Big(I - \frac{D_d}{n}\Big) X \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\ &= \frac{\gamma_t^2 p}{(1-p)\sigma n} \mathbf{r}^T D_d X \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \end{aligned}$$

again using the fact that $\mathbf{r}^T X = 0$. Finally, we can bound

$$\begin{aligned} (14) &= \frac{\gamma_t^2 p}{(1-p)\sigma n} \mathbf{r}^T D_d X \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \\ &\le \frac{\gamma_t^2 p}{(1-p)\sigma n} \|\mathbf{r}\| \big\| D_d X \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \big\| \\ &\le \frac{\gamma_t^2 p}{(1-p)\sigma} \|\mathbf{r}\| \big\| X \big(\boldsymbol{\beta}_t - \boldsymbol{\beta}^*\big) \big\| \end{aligned}$$

$$\begin{aligned} & \mathbf{b}_1^T \Big[\big(I - \gamma_t X^T X\big)^2 + X^T D_\delta \mathbb{E}\big[D_{\tilde{Z}} X X^T D_{\tilde{Z}}\big] D_\delta X \Big] \mathbf{b}_1 \\ &= \mathbf{b}_1^T \bigg[\big(I - \gamma_t X^T X\big)^2 + p(1-p) X^T D_\delta \Big(\frac{1}{n} D_d X X^T D_d + \frac{1}{\sigma}\Big(I - \frac{D_d}{n}\Big) D_d^2 \Big) D_\delta X \bigg] \mathbf{b}_1 \\ &= \mathbf{b}_1^T \bigg[\big(I - \gamma_t X^T X\big)^2 + \frac{p(1-p)}{n} X^T D_\delta D_d X X^T D_d D_\delta X + \frac{p(1-p)}{\sigma} X^T \Big(I - \frac{D_d}{n}\Big) D_\delta D_d^2 D_\delta X \bigg] \mathbf{b}_1 \\ &= \mathbf{b}_1^T \bigg[\big(I - \gamma_t X^T X\big)^2 + \frac{p(1-p)\gamma_t^2}{n} X^T X X^T X + \frac{\gamma_t^2 p}{(1-p)\sigma} X^T \Big(I - \frac{D_d}{n}\Big) X \bigg] \mathbf{b}_1 \end{aligned}$$

$$\leq \frac{\gamma_t^2 p}{(1-p)\sigma}\|\mathbf{r}\|\sqrt{\|X^TX\|}\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|$$
$$\leq \frac{\gamma_t^2 p}{(1-p)\sigma}\left(\frac{\|\mathbf{r}\|^2+\|X^TX\|\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2}{2}\right),$$

using the arithmetic-geometric-mean inequality in the final line. In particular, this term is similar to terms that appear in both (12) and (13), and (along with the observation that (15), (16) are zero) we have the equation shown on the bottom of the page.

Now we recall our choice of

$$\gamma_t = \frac{1}{\|X^TX\|}\cdot\min\left\{\frac{1}{2},\frac{\log(1/\varepsilon^2)}{t}\right\},$$

and the definition of

$$\sigma = \frac{d}{\mu}\frac{1}{\|X^TX\|}.$$

Let

$$\tilde{\gamma}_t \coloneqq \|X^TX\|\gamma_t = \min\left\{\frac{1}{2},\frac{\log(1/\varepsilon^2)}{t}\right\}.$$

Now we can simplify our bounds on (17), and (18) as:

$$(17) \leq \left(\left(1-\gamma_t\|X^TX\|\right)^2+\frac{p\gamma_t^2}{(1-p)n}\|X^TX\|^2 + 2\frac{\gamma_t^2 p}{(1-p)\sigma}\|X^TX\|\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2$$
$$\leq \left((1-\tilde{\gamma}_t)^2+\left(\frac{p}{1-p}\right)(\tilde{\gamma}_t)^2\cdot\frac{1}{n} + \left(\frac{2p}{1-p}\right)(\tilde{\gamma}_t)^2\left(\frac{\mu}{d}\right)\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2$$
$$\leq \left((1-\tilde{\gamma}_t)^2+\frac{1}{2}(\tilde{\gamma}_t)^2\right)\left\|\beta_t-\beta^*\right\|^2,$$

using the assumptions that $n \geq 4p/(1-p)$ and $d \geq 8\mu p/(1-p)$. Now we have:

$$(17) \leq \left((1-\tilde{\gamma}_t)^2+\frac{1}{2}(\tilde{\gamma}_t)^2\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2$$
$$= \left(1-2\tilde{\gamma}_t+\frac{3}{2}\tilde{\gamma}_t\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2$$
$$\leq (1-\tilde{\gamma}_t)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2,$$

using from the definition of $\tilde{\gamma}_t$ that $\tilde{\gamma}_t \leq 1/2$ and hence $\tilde{\gamma}_t^2 \leq \frac{1}{2}\tilde{\gamma}_t$.

Meanwhile,

$$(18) \leq 2\gamma_t^2\cdot\frac{p}{1-p}\frac{\|\mathbf{r}\|^2}{\sigma}$$
$$\leq 2(\tilde{\gamma}_t)^2\left(\frac{p}{1-p}\right)\frac{\|\mathbf{r}\|^2}{\sigma\|X^TX\|^2}$$
$$\leq 2(\tilde{\gamma}_t)^2\left(\frac{p}{1-p}\right)\frac{\|\tilde{\mathbf{r}}\|^2}{\sigma\|X^TX\|},$$

recalling that $\tilde{\mathbf{r}} = \mathbf{r}/\|X^TX\|^{1/2}$. Thus

$$(18) \leq 2(\tilde{\gamma}_t)^2\left(\frac{p}{1-p}\right)\frac{\|\tilde{\mathbf{r}}\|^2}{\sigma\|X^TX\|}$$
$$= 2(\tilde{\gamma}_t)^2\left(\frac{p}{1-p}\right)\left(\frac{\mu}{d}\right)\|\tilde{\mathbf{r}}\|^2.$$

Putting the two terms together, we conclude that for fixed t,

$$\mathbb{E}\left[\left\|\boldsymbol{\beta}_{t+1}-\boldsymbol{\beta}^*\right\|_2^2\right] \leq (1-\tilde{\gamma}_t)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|_2^2 + 2(\tilde{\gamma}_t)^2\left(\frac{p}{1-p}\right)\left(\frac{\mu}{d}\right)\|\tilde{\mathbf{r}}\|^2.$$

Now, we proceed by induction, using the fact that the stragglers are independent between the different rounds and conclude with the equation shown on bottom of the page. This proves the theorem.

$$\mathbb{E}\left\|\boldsymbol{\beta}_{t+1}-\boldsymbol{\beta}^*\right\|^2 \leq (12)+\ (13)+\ (14)$$
$$\leq \left(\left(1-\gamma_t\left\|X^TX\right\|\right)^2+\frac{p\gamma_t^2}{(1-p)n}\left\|X^TX\right\|^2+\frac{\gamma_t^2 p}{(1-p)\sigma}\|X^TX\|\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2 + \gamma_t^2\cdot\frac{p}{1-p}\cdot\frac{\|\mathbf{r}\|^2}{\sigma} + \frac{\gamma_t^2 p}{(1-p)\sigma}\left(\frac{\|\mathbf{r}\|^2+\|X^TX\|\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2}{2}\right)$$
$$\leq \left(\left(1-\gamma_t\|X^TX\|\right)^2+\frac{p\gamma_t^2}{(1-p)n}\|X^TX\|^2+\frac{2\gamma_t^2 p}{(1-p)\sigma}\|X^TX\|\right)\left\|\boldsymbol{\beta}_t-\boldsymbol{\beta}^*\right\|^2 \quad (17)$$
$$+ 2\gamma_t^2\cdot\frac{p}{1-p}\cdot\frac{\|\mathbf{r}\|^2}{\sigma} \quad (18)$$

$$\mathbb{E}\left[\left\|\beta_T-\beta^*\right\|_2^2\right] \leq \left(\prod_{t=1}^{T}(1-\tilde{\gamma}_t)\right)\left\|\boldsymbol{\beta}_0-\boldsymbol{\beta}^*\right\|_2^2+2T\tilde{\gamma}_T^2\left(\frac{p}{1-p}\right)\left(\frac{\mu}{d}\right)\|\tilde{\mathbf{r}}\|_2^2$$
$$\leq \left(1-\frac{\log(1/\varepsilon^2)}{T}\right)^T\left\|\boldsymbol{\beta}_0-\boldsymbol{\beta}^*\right\|_2^2+2T\cdot\frac{\log^2(1/\varepsilon^2)}{T^2}\left(\frac{p}{1-p}\right)\left(\frac{\mu}{d}\right)\|\tilde{\mathbf{r}}\|_2^2$$
$$\leq \varepsilon^2\left\|\boldsymbol{\beta}_0-\boldsymbol{\beta}^*\right\|_2^2+\frac{2}{Td}\cdot\left(\log^2(1/\varepsilon^2)\left(\frac{p}{1-p}\right)\mu\|\tilde{\mathbf{r}}\|_2^2\right)$$

Proof of Theorem 4

In this section we prove Theorem 4. Our proof of Theorem 4 relies on the following result from [14] which shows that any stochastic algorithm with a "good" estimator of the true gradient converges with rate $\mathcal{O}(\frac{1}{T})$. We translate this result to our setting.

Lemma 1 (Lemma 1 in [14]): Suppose $\mathcal{L}$ is λ-strongly convex over a convex set $\mathcal{W}$, and that $\hat{\mathbf{g}}_t$ is an unbiased estimator of a subgradient $\mathbf{g}_t$ of the loss function $\mathcal{L}$ at $\boldsymbol{\beta}_t$, i.e., $\mathbb{E}\hat{\mathbf{g}}_t = \mathbf{g}_t$. Suppose also that for all t, $\mathbb{E}\|\hat{\mathbf{g}}_t\|_2^2 \leq G$.[6] Then if we pick $\gamma_t = 1/\lambda t$, it holds for any T that

$$\mathbb{E}\Big[\big\|\boldsymbol{\beta}_T - \boldsymbol{\beta}^*\big\|_2^2\Big] \leq \frac{4G}{\lambda^2 T}.$$

(Proof of Theorem 4): In order to apply Theorem 1, we need to show that the estimate of the gradient obtained by the master at each iteration is unbiased. To see this, recall that at each iteration t, the master computes the following estimate of the gradient:

$$\hat{\mathbf{g}}_t \triangleq \sum_{j=1}^{n}\sum_{i=1}^{m} \frac{\mathcal{I}_i^j}{d_i(1-p)} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t). \tag{19}$$

[6]Here, the randomness in the expectation is over the next round of stragglers, conditioned on the previous rounds.

Therefore,

$$\mathbb{E}\big[\hat{\mathbf{g}}_t\big] = \textstyle\sum_{j=1}^{n}\sum_{i=1}^{m} \frac{\mathbb{E}\mathcal{I}_i^j}{d_i(1-p)} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t). \tag{20}$$

Recall that $\mathcal{I}_i^j$ is an indicator function equal to 1 if worker j is non straggler and has data vector $\mathbf{a}_i$. Thus,

$$\begin{aligned}\mathbb{E}\big[\hat{\mathbf{g}}_t\big] &= \sum_{j=1}^{n}\sum_{i=1}^{m} \frac{\mathbf{1}_{\text{worker } j \text{ has data vector } \mathbf{a}_i}}{d_i} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t)\\ &= \sum_{i=1}^{m} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta}_t)\\ &= \nabla\mathcal{L}\big(A, \boldsymbol{\beta}_t\big).\end{aligned}$$

Now, we need to show that under the conditions of the theorem, the variance $\mathbb{E}\|\hat{\boldsymbol{g}}(A, \boldsymbol{\beta}_t)\|_2^2$ is bounded. (Here, the randomness is over the choice of the stragglers in round t). As in the proof of Theorem 3, let Z_i be the binomial random variable that counts the number of non-stragglers (in a given round t) who have block i. Thus we have

$$\begin{aligned}\mathbb{E}\Big[Z_i^2\Big] &= d_i p(1-p) + d_i^2(1-p)^2,\\ \mathbb{E}\big[Z_{i_1}Z_{i_2}\big] &= \frac{d_{i_1}d_{i_2}}{n} p(1-p) + d_{i_1}d_{i_2}(1-p)^2.\end{aligned}$$

We compute (21), as shown at the bottom of the previous page, where we use the fact that the terms $\mathbb{E}[Z_{i_1}Z_{i_2}]/(d_{i_1}d_{i_2})$ are all positive to move the maximum inside the sum.

$$\begin{aligned}\mathbb{E}\Big[\big\|\hat{\boldsymbol{g}}\big(A, \boldsymbol{\beta}_t\big)\big\|_2^2\Big] &\leq \mathbb{E}\Big[\max_{\boldsymbol{\beta}\in\mathcal{W}}\big\|\hat{\boldsymbol{g}}(A, \boldsymbol{\beta})\big\|_2^2\Big]\\ &= \mathbb{E}\left[\max_{\boldsymbol{\beta}\in\mathcal{W}}\left\|\sum_{j=1}^{n}\sum_{i=1}^{m} \frac{\mathcal{I}_i^j}{d_i(1-p)} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})\right\|_2^2\right]\\ &= \frac{1}{(1-p)^2}\mathbb{E}\left[\max_{\boldsymbol{\beta}\in\mathcal{W}}\left\|\sum_{i=1}^{m} \frac{Z_i}{d_i} \nabla\mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})\right\|_2^2\right]\\ &= \frac{1}{(1-p)^2}\mathbb{E}\left[\max_{\boldsymbol{\beta}\in\mathcal{W}}\sum_{i_1=1}^{m}\sum_{i_2=1}^{m} \frac{Z_{i_1}Z_{i_2}}{d_{i_1}d_{i_2}} \big\langle\nabla\mathcal{L}(\mathbf{a}_{i_1}, \boldsymbol{\beta}), \nabla\mathcal{L}\big(\mathbf{a}_{i_2}, \boldsymbol{\beta}\big)\big\rangle\right]\\ &\leq \frac{1}{(1-p)^2}\sum_{i_1=1}^{m}\sum_{i_2=1}^{m} \frac{\mathbb{E}\big[Z_{i_1}Z_{i_2}\big]}{d_{i_1}d_{i_2}} \max_{\boldsymbol{\beta}\in\mathcal{W}}\big\langle\nabla\mathcal{L}\big(\mathbf{a}_{i_1}, \boldsymbol{\beta}\big), \nabla\mathcal{L}\big(\mathbf{a}_{i_2}, \boldsymbol{\beta}\big)\big\rangle \qquad (21)\\ \mathbb{E}\Big[\big\|\hat{\boldsymbol{g}}\big(A, \boldsymbol{\beta}_t\big)\big\|_2^2\Big] &\leq \frac{C^2}{(1-p)^2}\sum_{i_1=1}^{m}\sum_{i_2=1}^{m} \frac{\mathbb{E}\big[Z_{i_1}Z_{i_2}\big]}{d_{i_1}d_{i_2}}\\ &= \frac{C^2}{(1-p)^2}\left(\sum_{i=1}^{m}\left(\frac{d_i p(1-p) + d_i^2(1-p)^2}{d_i^2}\right) + \sum_{i_1\neq i_2}\left(\frac{d_{i_1}d_{i_2}p(1-p)}{n d_{i_1}d_{i_2}} + \frac{d_{i_1}d_{i_2}(1-p)^2}{d_{i_1}d_{i_2}}\right)\right)\\ &= C^2\left(\sum_{i=1}^{m}\left(\frac{p}{(1-p)d_i} + 1\right) + \sum_{i_1\neq i_2}\left(\frac{p}{n(1-p)} + 1\right)\right)\\ &\leq mC^2\left(\frac{p}{1-p}\cdot\frac{1}{d_{\min}} + \frac{(m-1)p}{n(1-p)} + m\right) \qquad (22)\end{aligned}$$

We have $\langle \nabla \mathcal{L}(\mathbf{a}_{i_1}, \boldsymbol{\beta}), \nabla \mathcal{L}(\mathbf{a}_{i_2}, \boldsymbol{\beta}) \rangle$ less than or equal to $\|\nabla \mathcal{L}(\mathbf{a}_{i_1}, \boldsymbol{\beta})\|_2 \|\nabla \mathcal{L}(\mathbf{a}_{i_2}, \boldsymbol{\beta})\|_2$ by Cauchy-Shwarz, and thus $\max_{\boldsymbol{\beta} \in \mathcal{W}} \langle \nabla \mathcal{L}(\mathbf{a}_{i_1}, \boldsymbol{\beta}), \nabla \mathcal{L}(\mathbf{a}_{i_2}, \boldsymbol{\beta}) \rangle$ is less than or equal to $\max_{i \in [n]} \max_{\boldsymbol{\beta} \in \mathcal{W}} \|\nabla \mathcal{L}(\mathbf{a}_i, \boldsymbol{\beta})\|_2^2 \leq C^2$, by the assumptions of the theorem. Thus, we may continue the derivation as shown in (22), as shown at the bottom of the previous page. Plugging this estimate into Theorem 1 proves Theorem 4. ■

ACKNOWLEDGMENT

The authors thank Deanna Needell for helpful pointers to the literature.

REFERENCES

[1] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," in *Proc. IEEE Inf. Theory Workshop (ITW)*, 2019, pp. 1–5.

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

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

[4] G. Ananthanarayanan *et al.*, "Reining in the outliers in MapReduce clusters using Mantri," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, vol. 10, 2010, p. 24.

[5] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, vol. 8, 2008, p. 7.

[6] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2017, pp. 3368–3376.

[7] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 12, 2018, p. 9716.

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

[9] N. Raviv, I. Tamo, R. Tandon, and A. G. Dimakis, "Gradient coding from cyclic MDS codes and expander graphs," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 4302–4310.

[10] L. Bottou, *Online Learning and Stochastic Approximations*, vol. 17. Cambridge, MA, USA: MIT Press, 1998, p. 142.

[11] S. Shalev-Shwartz and A. Tewari, "Stochastic methods for ℓ_1-regularized loss minimization," *J. Mach. Learn. Res.*, vol. 12, pp. 1865–1892, Jun. 2011.

[12] K. Gimpel, D. Das, and N. A. Smith, "Distributed asynchronous online learning for natural language processing," in *Proc. 40th Conf. Comput. Nat. Lang. Learn.*, 2010, pp. 213–222.

[13] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "PEGASOS: Primal estimated sub-gradient solver for SVM," *Math. Program.*, vol. 127, no. 1, pp. 3–30, 2011.

[14] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization," 2011. arXiv:1109.5647.

[15] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.

[16] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[17] R. K. Maity, A. S. Rawat, and A. Mazumdar, "Robust gradient descent via moment encoding with LDPC codes," 2018. arXiv:1805.08327.

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

[19] H. Wang, Z. Charles, and D. Papailiopoulos, "ErasureHead: Distributed gradient descent without delays using approximate gradient coding," 2019. arXiv:1901.09671.

[20] S. Wang, J. Liu, and N. Shroff, "Fundamental limits of approximate gradient coding," 2019. arXiv:1901.08166.

[21] S. Horii, T. Yoshida, M. Kobayashi, and T. Matsushima, "Distributed stochastic gradient descent using LDGM codes," 2019. arXiv:1901.04668.

[22] S. Kadhe, O. O. Koyluoglu, and K. Ramchandran, "Gradient coding based on block designs for mitigating adversarial stragglers," 2019. arXiv:1904.13373.

[23] M. Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2019, pp. 8177–8181.

[24] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. IEEE 48th Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, 2010, pp. 1510–1517.

[25] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *J. Fourier Anal. Appl.*, vol. 15, no. 2, p. 262, 2009.

[26] D. Needell, R. Ward, and N. Srebro, "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1017–1025.

[27] D. Needell and R. Ward, "Batched stochastic gradient descent with weighted sampling," in *Proc. Int. Conf. Approx. Theory*, 2016, pp. 279–306.

[28] *SGC GitHub Repository*. Accessed: May 13, 2019. [Online]. Available: https://github.com/RawadB01/SGC

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

[30] N. Ferdinand and S. Draper, "Anytime stochastic gradient descent: A time to hear from all the workers," in *Proc. 56th Annu. Allerton Conf. Commun Control Comput.*, 2018, pp. 552–559.

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

[32] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," 2018. arXiv:1806.00939.

[33] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 5434–5442.

[34] E. Ozfaturay, D. Gunduz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," 2018. arXiv:1808.02240.

[35] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2018, pp. 1988–1992.

[36] L. Chen *et al.*, "DRACO: Robust distributed training via redundant gradients," 2018. arXiv:1803.09877.

[37] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, 2016, pp. 1–6.

[38] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.

[39] H. Yang and J. Lee, "Secure distributed computing with straggling servers using polynomial codes," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 1, pp. 141–150, Jan. 2019.

[40] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2019, pp. 3492–3496.

[41] Y. Keshtkarjahromi and H. Seferoglu, "Coded cooperative computation for Internet of Things," 2018. arXiv:1801.04357.

[42] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2017, pp. 2900–2904.

[43] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proc. 5th ACM Symp. Inf. Comput. Commun. Security (ASIACCS)*, 2010, pp. 48–59.

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

[45] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," 2018. arXiv:1802.03430.

[46] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2018, pp. 1993–1997.

[47] M. Fahim, H. Jeong, F. Haddadpour, S. Dutta, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," in *Proc. 55th Annu. Allerton Conf. Commun. Control Comput.*, 2017, pp. 1264–1270.

[48] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," 2018. arXiv:1801.07487.
[49] A. Mallick, M. Chaudhari, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," 2018. arXiv:1804.10331.
[50] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2017, pp. 2403–2407.
[51] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Trans. Inf. Theory*, vol. 63, no. 6, pp. 3729–3756, Jun. 2017.
[52] S. Dutta, V. Cadambe, and P. Grover, "Short-Dot: Computing large linear transforms distributedly using coded short dot products," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2016, pp. 2100–2108.
[53] S. Wang, J. Liu, N. Shroff, and P. Yang, "Fundamental limits of coded linear transform," 2018. arXiv:1804.09791.
[54] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 12–14, 2017.
[55] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Proc. 50th Annu. Allerton Conf. Commun. Control Comput.*, 2012, pp. 326–333.
[56] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2012, pp. 2766–2770.
[57] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, 2015.
[58] K. Lee, N. B. Shah, L. Huang, and K. Ramchandran, "The MDS queue: Analyzing the latency performance of erasure codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, pp. 2822–2842, May 2017.
[59] S. Kadhe, E. Soljanin, and A. Sprintson, "Analyzing the download time of availability codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, 2015, pp. 1467–1471.
[60] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Stat.*, vol. 22, no. 3, pp. 400–407, 1951.
[61] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2011, pp. 1647–1655.
[62] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2011, pp. 873–881.
[63] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, Jan. 2012.
[64] O. Shamir and N. Srebro, "Distributed stochastic optimization and learning," in *Proc. 52nd Annu. Allerton Conf. Commun. Control Comput.*, 2014, pp. 850–857.
[65] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, vol. 23. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.
[66] J. Dean *et al.*, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1223–1231.
[67] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," 2018. arXiv:1803.01113.
[68] S. Chaturapruek, J. C. Duchi, and C. Ré, "Asynchronous stochastic convex optimization: The noise is in the noise and SGD don't care," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1531–1539.

Rawad Bitar (Member, IEEE) received the Diploma degree in computer and communication engineering from the Faculty of Engineering, Lebanese University, Roumieh, Lebanon, in 2013, the M.S. degree from the Doctoral School, Lebanese University, Tripoli, Lebanon, in 2014, and the Ph.D. degree in electrical engineering from Rutgers University, New Brunswick, NJ, USA, in 2020. He is currently a Postdoctoral Researcher with the Technical University of Munich. His research interests are in the broad area of information theory and coding theory with a focus on coding for information theoretically secure distributed systems with application to machine learning.

Mary Wootters (Member, IEEE) received the B.A. degree in mathematics and computer science from Swarthmore College in 2008, and the Ph.D. degree in mathematics from the University of Michigan in 2014. She is an Assistant Professor of computer science and electrical engineering with Stanford University. She was an NSF Postdoctoral Fellow with Carnegie Mellon University from 2014 to 2016. She works in theoretical computer science, applied math, and information theory; her research interests include error correcting codes and randomized algorithms for dealing with high-dimensional data. She is a recipient of the NSF CAREER Award and was named a Sloan Research Fellow in 2019.

Salim El Rouayheb (Member, IEEE) received the Diploma degree in electrical engineering from the Faculty of Engineering, Lebanese University, Roumieh, Lebanon, in 2002, the M.S. degree from the American University of Beirut, Lebanon, in 2004, and the Ph.D. degree in electrical engineering from Texas A&M University, College Station, in 2009. He is currently an Assistant Professor with the ECE Department, Rutgers University, New Brunswick, NJ, USA. He was a Postdoctoral Research Fellow with UC Berkeley from 2010 to 2011, and a Research Scholar with Princeton University from 2012 to 2013. He was an Assistant Professor with the ECE Department, Illinois Institute of Technology from 2013 to 2017. His research interests are in the broad area of information theory and coding theory with a focus on network coding, coding for distributed storage and information theoretic security. He is a recipient of the NSF Career Award.