Byzantine-Tolerant Distributed Coordinate Descent

Deepesh Data
University of California, Los Angeles, USA
Email: deepeshdata@ucla.edu

Suhas Diggavi
University of California, Los Angeles, USA
Email: suhasdiggavi@ucla.edu

***Abstract*—We study distributed coordinate descent (CD) in the master-worker architecture under adversarial attacks, where the data is partitioned (across the parameter space) and distributed among m worker nodes (t of which can be maliciously corrupt), which update some coordinates of their part of the parameter vector, in parallel and iteratively, using CD updates, with the help of the master. We propose a method based on data encoding and real error correction to combat the adversary. Our method can tolerate up to $\lceil \frac{m-1}{2} \rceil$ corrupt nodes, which is information-theoretically optimal. Our design gives a trade-off between the resiliency t, the required redundancy, and the computation at master and worker nodes. For example, with *constant* overhead in the storage and computational complexity over that required by the plain distributed CD, we can tolerate up to $m/3$ corrupt nodes. We design a sparse encoding scheme, which yields low encoding complexity.**

I. Introduction

Distributed implementations for large-scale computation are becoming increasingly common, especially with the advent of large-scale optimization and learning [1]. With growing interest in federated learning [2], harnessing of non-enterprise nodes in the computation is being envisaged. This is accentuated by increasing malicious software, and hardware attacks on computation nodes [3]. It is well-known that distributed computation is vulnerable to even a single (Byzantine) adversary [4]. This relates to well-studied problems in Byzantine fault tolerance [5], though its application to large-scale distributed optimization and learning is more recent [4], [6].

In this paper, we study Byzantine-tolerant distributed optimization to learn a *generalized linear model* (e.g., linear regression, logistic regression, etc.). Our framework for distributed computation is a *model-parallelism* architecture [7], where several worker nodes work in parallel on optimizing different subsets of coordinates of the parameter vector. These parallel computations are combined together for optimization through *coordinate descent* (CD) [7]–[9], which has been shown to be very effective for solving generalized linear models, and is particularly widely used for sparse logistic regression, SVM, and LASSO [7]. Given its simplicity and effectiveness, it is chosen over *gradient descent* (GD) in such applications. This motivates us to explore how *to make coordinate descent robust to Byzantine adversaries.*

We propose a data encoding method to deal with Byzantine attacks, inspired by real-error correction [10]; see Figure 1. Our encoding adds redundancy and enlarges the parameter space, such that even if any $t \leq \lceil \frac{m-1}{2} \rceil$ worker nodes maliciously deviate from their pre-specified programs and collude, we can still execute distributed CD (on encoded data together with decoding at the master node) correctly, with $O(m/(m-2t))$ multiplicative overhead (which is *constant*, even if $t < m/2$ is a constant fraction of m) on the computational complexity and space requirement than what is required by the plain distributed CD, that does not provide any adversarial protection.. Our design enables a trade-off between the Byzantine resilience (in terms of the number of adversarial nodes) and the redundancy of encoding, computation at master and worker nodes (as stated in our main result Theorem 1). Though encoding is a one-time process, it has to be efficient to exploit the power of distributed computation. We design a sparse encoding matrix, based on real error-correction [10], for which the encoding process incurs a factor of $(2t+1)$ one-time computation cost over conventional data distribution. The redundancy introduced by the encoding process is $O(2m/(m-2t))$, which is *constant*, even if t is constant fraction of m.

Related work. Distributed computing with Byzantine adversaries has a rich history since the work of [5], and has received recent attention in the context of large-scale distributed optimization and learning [3], [4], [6]. Coding-theoretic techniques have recently emerged and been used to mitigate the effect of *stragglers*, i.e., slow machines, for computing the full gradients (in distributed GD) [11]–[14]. For the Byzantine adversaries, [3] proposed a method based on *data replication* with non-trivial decoding at the master node for gradient computation. Data encoding was used in [15] both for distributed GD and CD, but for straggler mitigation; the encoding in [15] has low-redundancy and is efficient. Data encoding for combating the Byzantine attacks was used in [16] for computing the gradients exactly in distributed GD. As far as we know, making distributed CD resilient against Byzantine attacks has not seen much attention.

Paper organization. In Section II, we setup the problem and describe the distributed CD algorithm. In Section III, we explain our approach to make the distributed CD Byzantine-tolerant, using data encoding and real error correction, and state our main theorem. We give our encoding and decoding methods in Section IV. Omitted details from this paper can be found in the full version [17].

Notations. We denote vectors by bold small letters (e.g., $\mathbf{x}, \mathbf{y}$, etc.) and matrices by bold capital letters (e.g., $\mathbf{R}, \mathbf{X}$, etc.). For any $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. For $\mathbf{u} \in \mathbb{R}^n$, we define $\mathsf{supp}(\mathbf{u}) := \{i \in [n] : u_i \neq 0\}$.

II. Problem Setup and Preliminaries

We are given a data matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, and we want to find a *generalized linear model* $\mathbf{w} \in \mathbb{R}^d$ that minimizes

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \phi(\mathbf{X}\mathbf{w}) := \sum_{i=1}^{n} \ell(\langle \mathbf{x}_i, \mathbf{w}\rangle). \tag{1}$$

Here the loss function $\ell : \mathbb{R} \to \mathbb{R}$ is a convex and differentiable, and $\langle \mathbf{x}_i, \mathbf{w}\rangle$ is the dot product between the i'th data point and the parameter vector $\mathbf{w}$. This encompasses popular machine learning problems, such as *linear regression*, *logistic regression*, etc. For $\mathcal{U} \subseteq [d]$, we write $\nabla_{\mathcal{U}}\phi(\mathbf{X}\mathbf{w})$ to denote the gradient of $\phi(\mathbf{X}\mathbf{w})$ with respect to $\mathbf{w}_{\mathcal{U}}$, where $\mathbf{w}_{\mathcal{U}}$ denotes the $|\mathcal{U}|$-length vector obtained by restricting $\mathbf{w}$ to the coordinates in $\mathcal{U}$. To make the notation less cluttered, let $\phi'(\mathbf{X}\mathbf{w})$ denote the n-length vector, whose i'th entry is equal to $\ell'(\langle \mathbf{x}_i, \mathbf{w}\rangle) := \frac{d}{du}\ell(u)|_{u=\langle \mathbf{x}_i,\mathbf{w}\rangle}$, the differentiation of ℓ at $\langle \mathbf{x}_i, \mathbf{w}\rangle$. Note that $\nabla\phi(\mathbf{X}\mathbf{w}) = \mathbf{X}^T\phi'(\mathbf{X}\mathbf{w})$ and that $\nabla_{\mathcal{U}}\phi(\mathbf{X}\mathbf{w}) = \mathbf{X}_{\mathcal{U}}^T\phi'(\mathbf{X}\mathbf{w})$, where $\mathbf{X}_{\mathcal{U}}$ denotes the $n \times |\mathcal{U}|$ matrix obtained by restricting the column indices of $\mathbf{X}$ to the elements in $\mathcal{U}$.

A. Distributed Coordinate Descent

Coordinate descent (CD) is an iterative algorithm, where, in each iteration, we choose a set of coordinates and update only those coordinates (while keeping the other coordinates fixed). In distributed CD, we take advantage of the parallel architecture to improve the running time of (centralized) CD. In the distributed setting, we divide the data matrix vertically into m parts and store the i'th part at the i'th worker node. Concretely, assume, for simplicity, that m divides d. Let $\mathbf{X} = [\mathbf{X}_1\ \mathbf{X}_2\ \dots\ \mathbf{X}_m]$ and $\mathbf{w} = [\mathbf{w}_1^T\ \mathbf{w}_2^T\ \dots\ \mathbf{w}_m^T]^T$, where each $\mathbf{X}_i$ is an $n\times(d/m)$ matrix and each $\mathbf{w}_i$ is length d/m vector. Worker i stores $\mathbf{X}_i$ and is responsible for updating $\mathbf{w}_i$. In coordinate descent, since we update only a few coordinates in each round, there are a few options on how to update these coordinates in a distributed manner:

Subset of workers: Master picks a subset $\mathcal{S} \subset [m]$ of workers and asks them to update their $\mathbf{w}_i$'s [9]. This may not be good in the adversarial setting, because if only a small subset of workers are updating their parameters, the adversary can corrupt those workers and disrupt the computation.

Subset of coordinates for all workers: All the worker nodes update only a subset of the coordinates of their local parameter vector $\mathbf{w}_i$'s. Master can (deterministically or randomly) pick a subset $\mathcal{U}$ (which may or may not be different for all workers) of $f \leq d/m$ coordinates and asks each worker to updates only those coordinates. If master picks $\mathcal{U}$ deterministically, it can cycle through and update all coordinates of the parameter vector in $\lceil d/mf\rceil$ iterations.

In Algorithm 1, we give the distributed CD algorithm with the second approach, where all the worker nodes update the coordinates of their local parameter vectors for a single subset $\mathcal{U}$. We will adopt this approach in our method to make the distributed CD Byzantine-resilient. Let $r = d/m$. For any $i \in [m]$, let $\mathbf{w}_i = [w_{i1}\ w_{i2} \dots w_{ir}]^T$ and $\mathbf{X}_i = [\mathbf{X}_{i1}\ \mathbf{X}_{i2} \dots \mathbf{X}_{ir}]$, where $\mathbf{X}_{ij}$ is the j'th column of $\mathbf{X}_i$. For any $i \in [m]$ and $\mathcal{U} \subseteq [r]$, let $\mathbf{w}_{i\mathcal{U}}$ denote the $|\mathcal{U}|$-length vector that is obtained from $\mathbf{w}_i$ by restricting its entries to the coordinates in $\mathcal{U}$; similarly, let $\mathbf{X}_{i\mathcal{U}}$ denote the $n \times |\mathcal{U}|$ matrix obtained by restricting the column indices of $\mathbf{X}_i$ to the elements in $\mathcal{U}$.

Adversary model. The adversary can corrupt *any* t of the worker nodes and the compromised nodes can send local outcomes that are arbitrarily far away from the actual local outcomes.[1] At some iteration, suppose worker i's true outcome is $\mathbf{u}_i$, then we can assume, without loss of generality, that master receives $\mathbf{u}_i + \mathbf{e}_i$ from worker i, where $\mathbf{e}_i = \mathbf{0}$ if the i'th worker is honest, otherwise can be arbitrary. The adversarial nodes can collude, and can even know the data of other workers. The master node does not know which t worker nodes are corrupted, but knows t, the maximum possible number of adversarial nodes.

Algorithm 1: Distributed Coordinate Descent

```
Each worker $i$ starts with an arbitrary/random $\mathbf{w}_i$.
Repeat (until the stopping criteria is not satisfied)
  1) Each worker $i \in [m]$ computes $\mathbf{X}_i\mathbf{w}_i$ and sends it to
     the master node. Note that $\mathbf{X}\mathbf{w} = \sum_{i=1}^m \mathbf{X}_i\mathbf{w}_i$.[2]
  2) Master computes $\phi'(\mathbf{X}\mathbf{w})$ and broadcasts it.
  3) For some $\mathcal{U} \subseteq [r]$ (where $\mathcal{U}$ can be picked in a round-
     robin fashion), each worker $i \in [m]$ updates its local
     parameter vector as
         $\mathbf{w}_{i\mathcal{U}} \leftarrow \mathbf{w}_{i\mathcal{U}} - \alpha\nabla_{i\mathcal{U}}\phi(\mathbf{X}\mathbf{w})$        (2)
     while keeping the other coordinates of $\mathbf{w}_i$ unchanged,
     and sends the updated $\mathbf{w}_i$ to the master, which can check
     for the stopping criteria.
```

In Algorithm 1, for each worker i to update $\mathbf{w}_i$ according to (2), where $\nabla_{i\mathcal{U}}\phi(\mathbf{X}\mathbf{w}) = \mathbf{X}_{i\mathcal{U}}^T\phi'(\sum_{j=1}^m \mathbf{X}_j\mathbf{w}_j)$ and worker i has only $(\mathbf{X}_i, \mathbf{w}_i)$, every other worker j sends $\mathbf{X}_j\mathbf{w}_j$ to the master, who computes $\phi'(\sum_{j=1}^m \mathbf{X}_j\mathbf{w}_j)$ and sends it back to all the workers. Observe that, even if one worker is corrupt, it can send an adversarially chosen vector to make the computation at the master deviate arbitrarily from the desired computation, which may adversely affect the update at all the worker nodes subsequently.[3] Similarly, corrupt workers can send adversarially chosen information to affect the stopping criterion.

III. Our Approach

We solve this problem using data encoding and error correction over real numbers. To combat the effect of adversary, we add redundancy to enlarge the parameter space.

[1]Our results are also applicable against a stronger adversary, which can adaptively *choose* which of the t worker nodes to attack in each iteration. However, in this model, we do not allow the adversary to modify the stored data at the attacked nodes; otherwise, over time, such an adversary can corrupt all the data, rendering any defense impossible.

[2]After the 1st iteration, worker i need not compute $\mathbf{X}_i\mathbf{w}_i$ in every iteration, as only a few coordinates of $\mathbf{w}_i$ are updated, it only needs to compute it for the updated coordinates.

[3]Specifically, suppose the i'th worker is corrupt and the adversary wants master to compute $\phi'(\mathbf{X}\mathbf{w} + \mathbf{e})$ for any arbitrary vector $\mathbf{e} \in \mathbb{R}^n$ of its choice, then the i'th worker can send $\mathbf{X}_i\mathbf{w}_i + \mathbf{e}$ to the master.

Let $\widetilde{\mathbf{X}}^R = \mathbf{X}\mathbf{R}$, where $\mathbf{R} = [\mathbf{R}_1 \ \mathbf{R}_2 \ \ldots \ \mathbf{R}_m] \in \mathbb{R}^{d\times pm}$ with $pm \geq d$, and each $\mathbf{R}_i$ is a $p \times d$ matrix. We will determine $\mathbf{R}$ and the value of p later. We consider $\mathbf{R}$'s which are of full row-rank. Let $\mathbf{R}^+ := \mathbf{R}^T(\mathbf{R}\mathbf{R}^T)^{-1}$ denote its Moore-Penrose inverse such that $\mathbf{R}\mathbf{R}^+ = I_d$, where I_d is the $d \times d$ identity matrix. Note that $\mathbf{R}^+$ is of full column-rank. Let $\mathbf{v} = \mathbf{R}^+\mathbf{w}$ be the transformed vector, which lies in a larger (than d) dimensional space. Let $\mathbf{R}^+ = [(\mathbf{R}_1^+)^T \ (\mathbf{R}_2^+)^T \ \ldots \ (\mathbf{R}_m^+)^T]^T$, where each $\mathbf{R}_i^+ := (\mathbf{R}^+)_i$ is a $p \times d$ matrix. With this, by letting $\mathbf{v} = [\mathbf{v}_1^T \ \mathbf{v}_2^T \ \ldots \ \mathbf{v}_m^T]^T$, we have that $\mathbf{v}_i = \mathbf{R}_i^+\mathbf{w}$ for every $i \in [m]$. Now, consider the following modified problem over the encoded data.

$$\arg\min_{\mathbf{v}\in\mathbb{R}^{pm}} \phi(\widetilde{\mathbf{X}}^R\mathbf{v}). \tag{3}$$

Observe that, since $\mathbf{R}$ is of full row-rank, $\min_{\mathbf{w}\in\mathbb{R}^d} \phi(\mathbf{X}\mathbf{w})$ is equal to $\min_{\mathbf{v}\in\mathbb{R}^{pm}} \phi(\widetilde{\mathbf{X}}^R\mathbf{v})$; and from an optimal solution to one problem we can obtain an optimal solution to the other problem. We design an encoding/decoding scheme such that when we optimize the encoded problem (3) using Algorithm 1, the vector $\mathbf{v}$ that we get in each iteration is of the form $\mathbf{v} = \mathbf{R}^+\mathbf{w}$ for some vector $\mathbf{w} \in \mathbb{R}^d$.[4] It may not be clear why we need this, but as we will see later, this property will be crucial in our solution.

Now, instead of solving (1), we solve its encoded form (3) using Algorithm 1, where each worker i stores $\widetilde{\mathbf{X}}_i^R = \mathbf{X}\mathbf{R}_i$ and is responsible for updating (some coordinates of) $\mathbf{v}_i$. In the following, let $\mathcal{U} \subseteq [p]$ be a fixed arbitrary subset of $[p]$. Let $\mathbf{v}^0 := \mathbf{R}^+\mathbf{w}^0$ for some $\mathbf{w}^0$ at time $t = 0$. Suppose, at the beginning of the t'th iteration, we have $\mathbf{v}^t = \mathbf{R}^+\mathbf{w}^t$ for some $\mathbf{w}^t$, and each worker i updates $\mathbf{v}_{i\mathcal{U}}^t$ according to

$$\mathbf{v}_{i\mathcal{U}}^{t+1} = \mathbf{v}_{i\mathcal{U}}^t - \alpha\nabla_{i\mathcal{U}}\phi(\widetilde{\mathbf{X}}^R\mathbf{v}^t), \tag{4}$$

where $\nabla_{i\mathcal{U}}\phi(\widetilde{\mathbf{X}}^R\mathbf{v}^t) = (\widetilde{\mathbf{X}}_{i\mathcal{U}}^R)^T\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$. Recall that each $\mathbf{R}_i$ is a $d\times p$ matrix, and each $\mathbf{R}_i^+ := (\mathbf{R}^+)_i$ is a $p\times d$ matrix. We denote by $\mathbf{R}_{i\mathcal{U}}$ the $d \times |\mathcal{U}|$ matrix obtained by restricting the columns of $\mathbf{R}_i$ to the elements of $\mathcal{U}$. Analogously, we denote by $\mathbf{R}_{i\mathcal{U}}^+ := (\mathbf{R}^+)_{i\mathcal{U}}$ the $|\mathcal{U}| \times d$ matrix obtained by restricting the rows of $\mathbf{R}_i^+$ to the elements of $\mathcal{U}$. With this, we can write $\widetilde{\mathbf{X}}_{i\mathcal{U}}^R = \mathbf{X}\mathbf{R}_{i\mathcal{U}}$. Now, (4) can be equivalently written as

$$\mathbf{v}_{i\mathcal{U}}^{t+1} = \mathbf{v}_{i\mathcal{U}}^t - \alpha\mathbf{R}_{i\mathcal{U}}^T\mathbf{X}^T\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t). \tag{5}$$

In order to update $\mathbf{v}_{i\mathcal{U}}^t$, worker i requires $\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$, where $\widetilde{\mathbf{X}}^R\mathbf{v}^t = \sum_{j=1}^m \widetilde{\mathbf{X}}_j^R\mathbf{v}_j^t$ and worker i has only $(\widetilde{\mathbf{X}}_i^R, \mathbf{v}_i^t)$. In Section IV-A, we give a method which provides each worker access to $\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$ in every iteration t.

Our goal in each iteration of CD is to update some coordinates of the original parameter vector $\mathbf{w}$; instead, by solving the encoded problem, we are updating coordinates of the transformed vector $\mathbf{v}$. We would like to design an algorithm/encoding such that it has exactly the same convergence properties as if we are running the distributed CD on the original problem without any adversary! For this, naturally, we would like our algorithm to satisfy the following property:

Update on any (small) subset of coordinates of $\mathbf{w}$ *should be achieved by updating some (small) subset of coordinates of* $\mathbf{v}_i$*'s; and, by updating those coordinates of* $\mathbf{v}_i$*'s, we should be able to efficiently recover the correspondingly updated coordinates of* $\mathbf{w}$*. Furthermore, this should be doable despite the errors injected by the adversary in every iteration of the algorithm.*

Note that if each coordinate of $\mathbf{v}$ depends on too many coordinates of $\mathbf{w}$, then updating a few coordinates of $\mathbf{v}$ may affect many coordinates of $\mathbf{w}$, and it becomes information-theoretically infeasible to satisfy the above property (even without the presence of an adversary). This imposes a restriction that each row of $\mathbf{R}^+$ must have few non-zero entries, in such a way that updating $\mathbf{v}_{i\mathcal{U}}^t$'s, for any choice of $\mathcal{U} \subseteq [p]$, will collectively update only a subset (which may potentially depend on $\mathcal{U}$) of coordinates of the original parameter vector $\mathbf{w}^t$, and we can uniquely and efficiently recover those updated coordinates of $\mathbf{w}^t$, even from the *erroneous* vectors $\{\mathbf{v}_{i\mathcal{U}}^{t+1} + \mathbf{e}_{i\mathcal{U}}\}_{i=1}^m$, where at most t out of m error vectors $\{\mathbf{e}_{i\mathcal{U}}\}_{i=1}^m$ are non-zero and may have arbitrary entries. In order to achieve this, we will design a sparse encoding matrix $\mathbf{R}^+$ (which in turn determines $\mathbf{R}$), that satisfies the following properties:

P.1 $\mathbf{R}^+$ has structured sparsity, which induces a map $f : [p] \to \mathcal{P}([d])$ (where $\mathcal{P}([d])$ denotes the power set of $[d]$) such that
   a) $\{f(i) : i \in [p]\}$ partitions $\{1, 2, \ldots, d\}$, i.e., for every $i, j \in [p]$, such that $i \neq j$, we have $f(i) \cap f(j) = \emptyset$ and that $\bigcup_{i=1}^p f(i) = [d]$.
   b) $|f(i)| = |f(j)|$ for every $i, j \in [p-1]$, and $|f(p)| \leq |f(i)|$, for any $i \in [p-1]$.
   c) For any $\mathcal{U} \subseteq [p]$, define $f(\mathcal{U}) := \cup_{j\in\mathcal{U}} f(j)$. If we update $\mathbf{v}_{i\mathcal{U}}^t$, $\forall i \in [m]$, according to (5), it automatically updates $\mathbf{w}_{f(\mathcal{U})}^t$ according to

$$\mathbf{w}_{f(\mathcal{U})}^{t+1} = \mathbf{w}_{f(\mathcal{U})}^t - \alpha\mathbf{X}_{f(\mathcal{U})}^T\phi'(\mathbf{X}\mathbf{w}^t). \tag{6}$$

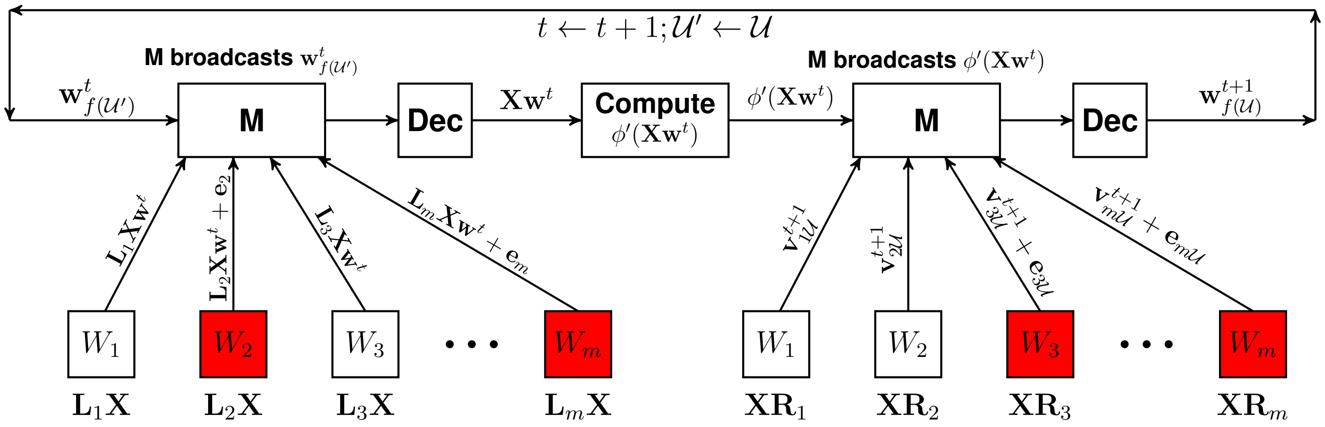
   If we set $\mathbf{v}_{i\overline{\mathcal{U}}}^{t+1} := \mathbf{v}_{i\overline{\mathcal{U}}}^t$ and $\mathbf{w}_{\overline{f(\mathcal{U})}}^{t+1} := \mathbf{w}_{\overline{f(\mathcal{U})}}^t$, then $\mathbf{v}^{t+1} = \mathbf{R}^+\mathbf{w}^{t+1}$, i.e., our invariant holds.

Note that (6) is the same update rule if we run the plain CD algorithm to update $\mathbf{w}_{f(\mathcal{U})}$. In fact, our encoding matrix satisfies a stronger property, that $\mathbf{v}_{i\mathcal{U}}^{t+1} = \mathbf{R}_{i\mathcal{U},f(\mathcal{U})}^+\mathbf{w}_{f(\mathcal{U})}^{t+1}$ holds for every $i \in [m]$, $\mathcal{U} \subseteq [p]$, where $\mathbf{R}_{i\mathcal{U},f(\mathcal{U})}^+$ denotes the $|\mathcal{U}| \times |f(\mathcal{U})|$ matrix obtained from $\mathbf{R}_{i\mathcal{U}}^+$ by restricting its column indices to the elements in $f(\mathcal{U})$.

P.2 We can efficiently recover $\mathbf{w}_{f(\mathcal{U})}^{t+1}$ from the erroneous vectors $\{\mathbf{v}_{i\mathcal{U}}^{t+1} + \mathbf{e}_{i\mathcal{U}}\}_{i=1}^m$, where at most t of $\mathbf{e}_{i\mathcal{U}}$'s are non-zero and may have arbitrary entries. Since $\mathbf{v}_{i\mathcal{U}}^{t+1} = \mathbf{R}_{i\mathcal{U},f(\mathcal{U})}^+\mathbf{w}_{f(\mathcal{U})}^{t+1}$, for every $i \in [m], \mathcal{U} \subseteq [p]$, this property requires that not only $\mathbf{R}^+$, but most of its sub-matrices also have error correcting capabilities.

Remark 1. *Note that* **P.1** *implies that for every* $i \in [p]$*, we have* $|f(i)| \leq d/p$*. As we see later, this will be equal*

[4]If such a $\mathbf{w}$ exists, then it is unique. This follows from the fact that $\mathbf{R}^+$ is of full column-rank



$t \leftarrow t+1; \mathcal{U}' \leftarrow \mathcal{U}$
M broadcasts $\mathbf{w}^t_{f(\mathcal{U}')}$
$\mathbf{w}^t_{f(\mathcal{U}')}$
M
Dec
$\mathbf{X}\mathbf{w}^t$
Compute $\phi'(\mathbf{X}\mathbf{w}^t)$
$\phi'(\mathbf{X}\mathbf{w}^t)$
M broadcasts $\phi'(\mathbf{X}\mathbf{w}^t)$
M
Dec
$\mathbf{w}^{t+1}_{f(\mathcal{U})}$
$\mathbf{L}_1\mathbf{X}\mathbf{w}^t$
$\mathbf{L}_2\mathbf{X}\mathbf{w}^t + \mathbf{e}_2$
$\mathbf{L}_3\mathbf{X}\mathbf{w}^t$
$\mathbf{L}_m\mathbf{X}\mathbf{w}^t + \mathbf{e}_m$
$\mathbf{v}^{t+1}_{1\mathcal{U}}$
$\mathbf{v}^{t+1}_{2\mathcal{U}}$
$\mathbf{v}^{t+1}_{3\mathcal{U}} + \mathbf{e}_{3\mathcal{U}}$
$\mathbf{v}^{t+1}_{m\mathcal{U}} + \mathbf{e}_{m\mathcal{U}}$
W_1 W_2 W_3 W_m
$\mathbf{L}_1\mathbf{X}$ $\mathbf{L}_2\mathbf{X}$ $\mathbf{L}_3\mathbf{X}$ $\mathbf{L}_m\mathbf{X}$
W_1 W_2 W_3 W_m
$\mathbf{X}\mathbf{R}_1$ $\mathbf{X}\mathbf{R}_2$ $\mathbf{X}\mathbf{R}_3$ $\mathbf{X}\mathbf{R}_m$

Fig. 1 This figure shows our 2-round approach to the Byzantine-resilient distributed coordinate descent (CD) for solving (1) using data encoding and real error-correction. We encode $\mathbf{X}$ with the encoding matrix $[\mathbf{R}_1 \ \ldots \ \mathbf{R}_m] \in \mathbb{R}^{d\times pm}$ and store $\widetilde{\mathbf{X}}_i^R := \mathbf{X}\mathbf{R}_i$ at the i'th worker and solve (3) over an enlarged parameter vector $\mathbf{v} = \mathbf{R}^+\mathbf{w}$. At the t'th step, for some $\mathcal{U} \subseteq [p]$, the update at the i'th worker is $\mathbf{v}^{t+1}_{i\mathcal{U}} = \mathbf{v}^t_{i\mathcal{U}} - \alpha\mathbf{R}^T_{i\mathcal{U}}\mathbf{X}^T\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$, which requires $\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$, where $\widetilde{\mathbf{X}}^R\mathbf{v}^t = \mathbf{X}\mathbf{w}$. The first part of the figure is for providing $\phi'(\mathbf{X}\mathbf{w}^t)$ to every worker in each iteration so that they can update $\mathbf{v}^t_{i\mathcal{U}}$'s. For this, we encode $\mathbf{X}$ using the encoding matrix $[\mathbf{L}_1^T \ \ldots \ \mathbf{L}_m^T]^T \in \mathbb{R}^{p'm\times n}$ and store $\widetilde{\mathbf{X}}_i^L := \mathbf{L}_i\mathbf{X}$ at worker i. The encoding has the property that we can recover $\mathbf{X}\mathbf{w}^t$ from the erroneous vectors $\{\widetilde{\mathbf{X}}_i^L\mathbf{w}^t + \mathbf{e}_i\}_{i=1}^m$, where at most t of the $\mathbf{e}_i$'s are non-zero and can be arbitrary. We can make it computationally more efficient by observing that in each iteration, only a subset of coordinates of $\mathbf{w}$ are being updated: suppose we updated $\mathbf{v}^t_{i\mathcal{U}'}$'s in the $(t-1)$'st iteration, which automatically updated $\mathbf{w}^t_{f(\mathcal{U}')}$. Since $\mathbf{w}^t_{[d]\setminus f(\mathcal{U}')}$ remain unchanged, we need to send only $\mathbf{w}^t_{f(\mathcal{U}')}$ to the workers – worker i can store the result from the previous iteration with itself, combines it with the current local computation, and sends $\widetilde{\mathbf{X}}_i^L\mathbf{w}^t$ to the master. The set of corrupt workers may be different in different rounds – the corrupt ones are shown in red color and they can send arbitrary outcomes to master. Once master has recovered $\mathbf{X}\mathbf{w}^t$, it computes $\phi'(\mathbf{X}\mathbf{w}^t)$ and broadcasts it; upon receiving it worker i updates $\mathbf{v}^{t+1}_{i\mathcal{U}}$ and sends it back. By **P.1**, this reflects an update on $\mathbf{w}^{t+1}_{f(\mathcal{U})}$ according to (6); and by **P.2**, the master can recover $\mathbf{w}^{t+1}_{f(\mathcal{U})}$.

to $m/(1+\epsilon)$ for some $\epsilon > 0$ which is determined by the corruption threshold. This means that in each iteration of the CD algorithm running on the modified encoded problem, we will be effectively updating the coordinates of the parameter vector $\mathbf{w}$ in chunks of size $m/(1+\epsilon)$ or its integer multiples. In particular, if each worker i updates k coordinates of $\mathbf{v}_i$, then $km/(1+\epsilon)$ coordinates of $\mathbf{w}$ will get updated. For comparison, Algorithm 1 updates km coordinates of the parameter vector $\mathbf{w}$ in each iteration, if each worker updates k coordinates in that iteration.

In the next section, we design an encoding matrix $\mathbf{R}^+$ and a decoding method that satisfy **P.1** and **P.2**. The main result of this paper is stated below, which is proved in the full version [17]. Our algorithm is described in Figure 1.

Theorem 1 (Main Result)**.** *Our Byzantine-resilient distributed CD algorithm has the following guarantees.*

- *It can tolerate $t \leq \left\lfloor \frac{\epsilon}{1+\epsilon} \cdot \frac{m}{2} \right\rfloor$ corrupt worker nodes.*
- *Total storage requirement is roughly $2(1+\epsilon)|\mathbf{X}|$.*
- *Total encoding time is $O\left(nd\left(\frac{\epsilon}{1+\epsilon}m+1\right)\right)$.*
- *If each worker i updates τ coordinates of $\mathbf{v}_i$, then the computational complexity in each iteration*
  - *at each worker node is $O((1+\epsilon)n\tau)$.*
  - *at the master node is $O((1+\epsilon)nm+\tau m^2)$.*

Here $\epsilon > 0$ is a free parameter.

Remark 2. *In the plain distributed CD described in Algorithm 1 that does not provide any adversarial protection, updating τ coordinates of the parameter vector requires $O(n\tau)$ time at each worker node (which is a $(1+\epsilon)$ factor less than ours) and $O(nm)$ time at the master node, which, again, is a $(1+\epsilon)$ factor less than ours if each worker updates $\tau \leq (1+\epsilon)\frac{n}{m}$ coordinates in each iteration. In particular, for $\epsilon = 2$, we can tolerate up to $m/3$ corrupt worker nodes, without sacrificing upon the computational complexity and space requirement beyond a constant factor.*

IV. Encoding and Decoding

In this section we design our generic encoding matrix and give an overview of our decoding algorithm for that. Let $\mathbf{F} \in \mathbb{R}^{k\times m}$ $(k < m)$ be an error correction matrix over $\mathbb{R}$, i.e., if $\mathbf{e} \in \mathbb{R}^m$ is sparse, then we can efficiently recover $\mathbf{e}$ from $\mathbf{F}\mathbf{e}$. There are many choices for the matrix $\mathbf{F}$ that can handle different levels of sparsity with varying decoding complexity. We can choose any of these matrices depending on our need, and this will not affect the design of our encoding matrix later. In particular, we can use a $k \times m$ Vandermonde matrix along with the Reed-Solomon decoding, which can correct up to $k/2$ errors (i.e., we can recover $\mathbf{e}$ from $\mathbf{F}\mathbf{e}$, provided $\mathbf{e}$ has at most $k/2$ non-zero entries) and has $O(m^2)$ decoding complexity [18]. We take $k = 2t$. Let $\mathcal{N}(\mathbf{F}) \subset \mathbb{R}^m$ denote the null-space of $\mathbf{F}$. Since $\mathsf{rank}(\mathbf{F}) = k$, dimension of $\mathcal{N}(\mathbf{F})$ is $q = (m-k)$. Let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_q\}$ be an orthonormal basis of $\mathcal{N}(\mathbf{F})$, and let $\mathbf{b}_i = [b_{i1}\ b_{i2} \ldots b_{im}]^T$, for $i \in [q]$. Define the following matrix for each $i \in [m]$:

$$\mathbf{S}_i = \begin{bmatrix} b_{1i} \ldots b_{qi} & & & \\ & \ddots & & \\ & & b_{1i} \ldots b_{qi} & \\ & & & b_{1i} \ldots b_{li} \end{bmatrix}_{p\times d} \tag{7}$$

where $q = (m-2t)$, $l = d-(p-1)q$, and $p = \lceil \frac{d}{q} \rceil$. Note that $1 \leq l < q$, and if q divides d, then $l = q$. All the unspecified entries of $\mathbf{S}_i$ are zero. Let $\mathbf{S} = [\mathbf{S}_1^T \ \mathbf{S}_2^T \ \ldots \ \mathbf{S}_m^T]^T \in \mathbb{R}^{pm \times d}$. It is easy to verify that $\mathbf{S}$ has orthonormal columns, which implies that $\mathbf{S}^+ = \mathbf{S}^T$. We take $\mathbf{R} = \mathbf{S}^T$, which implies that $\mathbf{R}^+ = \mathbf{S}$. In order to show that $\mathbf{S}$ satisfies **P.1**-**P.2**, define the function f as follows:

$$f(i) := \begin{cases} [(i-1)*q+1 : i*q] & \text{if } 1 \leq i < p, \\ [(p-1)*q+1 : d] & \text{if } i = p. \end{cases}$$

It is evident from the definition that f satisfies the first two conditions of **P.1**. For the 3rd condition, we crucially use the fact that $\mathbf{S}$ has orthonormal columns and the observation that, for any $\mathcal{U} \subseteq [p]$, all the columns of $\mathbf{S}_{i\mathcal{U}}$ whose indices belong to $[d] \setminus f(\mathcal{U})$ are identically zero, which implies that $\mathbf{S}_{i\mathcal{U}}\mathbf{w} = \mathbf{S}_{i\mathcal{U},f(\mathcal{U})}\mathbf{w}_{f(\mathcal{U})}$ holds for every $\mathbf{w} \in \mathbb{R}^d$.

Now we show that $\mathbf{S}$ satisfies **P.2**. Upon receiving $\{\mathbf{v}_{i\mathcal{U}}^{t+1} + \mathbf{e}_{i\mathcal{U}}\}_{i=1}^m$, where $\mathbf{v}_{i\mathcal{U}}^{t+1} = \mathbf{S}_{i\mathcal{U}}\mathbf{w}^{t+1}$, and at most t of $\mathbf{e}_{i\mathcal{U}}$'s are non-zero and may have arbitrary entries, master rearranges this and equivalently writes $|\mathcal{U}|$ systems of linear equations

$$\tilde{h}_i = \tilde{\mathbf{S}}_{i,f(\mathcal{U})}\mathbf{w}_{f(\mathcal{U})}^{t+1} + \tilde{\mathbf{e}}_i, \quad i \in \mathcal{U}, \tag{8}$$

where, for every $i \in \mathcal{U}$, $\tilde{\mathbf{e}}_i = [e_{1i}, e_{2i}, \ldots, e_{mi}]^T$ with $|\mathsf{supp}(\tilde{\mathbf{e}}_i)| \leq t$, and $\tilde{\mathbf{S}}_{i,f(\mathcal{U})}$ is an $m \times |f(\mathcal{U})|$ matrix whose j'th row is equal to the i'th row of $\mathbf{S}_{j\mathcal{U}}$, for every $j \in [m]$. It is easy to verify that $\mathbf{F}\tilde{\mathbf{S}}_{i,f(\mathcal{U})} = \mathbf{0}$ for every $i \in [m], \mathcal{U} \subseteq [p]$. Now, by multiplying (8) by $\mathbf{F}$ and letting $\mathbf{f}_i = \mathbf{F}\tilde{h}_i$, we get $\mathbf{f}_i = \mathbf{F}\tilde{\mathbf{e}}_i$. In order to find the corrupt worker nodes, master takes a linear combination of $\mathbf{f}_i$'s with coefficients α_i's from an *absolutely continuous* function (e.g., the Gaussian distribution) to obtain $\tilde{\mathbf{f}} = \mathbf{F}\tilde{\mathbf{e}}$, where $\tilde{\mathbf{e}} = \sum_{i=1}^{|\mathcal{U}|} \alpha_i \tilde{\mathbf{e}}_i$ and $\mathsf{supp}(\tilde{\mathbf{e}})$ is equal to the set of corrupt worker nodes with probability 1. Now, using the Reed-Solomon decoding, master can recover $\tilde{\mathbf{e}}$, and discards all the information received from the corrupt nodes. Since we have enough redundancy in the data and our encoding matrix is structured, master can efficiently recover $\mathbf{w}_{f(\mathcal{U})}^{t+1}$ from the remaining information.

A. Providing $\widetilde{\mathbf{X}}^R\mathbf{v}^t$ to Each Worker

In order to update $\mathbf{v}_{i\mathcal{U}}^t$ according to (5), worker i needs $\phi'(\widetilde{\mathbf{X}}^R\mathbf{v}^t)$. Let $\mathbf{v} = \mathbf{v}^t$. Since $\mathbf{v} = \mathbf{R}^+\mathbf{w}$ for some $\mathbf{w}$, we have $\widetilde{\mathbf{X}}^R\mathbf{v} = \mathbf{X}\mathbf{R}\mathbf{v} = \mathbf{X}\mathbf{w}$. So, it suffices to compute $\mathbf{X}\mathbf{w}$ at the master node – once master has $\mathbf{X}\mathbf{w}$, it can locally compute $\phi'(\mathbf{X}\mathbf{w})$ and send it to all the workers. Computing $\mathbf{X}\mathbf{w}$ is the distributed matrix-vector (MV) multiplication problem, where the matrix $\mathbf{X}$ is fixed and we want to compute $\mathbf{X}\mathbf{w}$ for any vector $\mathbf{w}$ in the presence of an adversary. For this we encode $\mathbf{X}$ using an encoding matrix $\mathbf{L} \in \mathbb{R}^{(p'm) \times n}$. Let $\mathbf{L} = [\mathbf{L}_1^T \ \mathbf{L}_2^T \ \ldots \ \mathbf{L}_m^T]^T$, where each $\mathbf{L}_i$ is a $p' \times n$ matrix with $p' = \lceil \frac{n}{m-2t} \rceil$. Each $\mathbf{L}_i$ is similar to $\mathbf{S}_i$ in (7); it has p' rows and n columns, and has the structure as that of $\mathbf{S}_i$. Worker i stores $\widetilde{\mathbf{X}}_i^L = \mathbf{L}_i\mathbf{X}$. To compute $\mathbf{X}\mathbf{w}$, master sends $\mathbf{w}$ to all the workers; worker i responds with $\mathbf{L}_i\mathbf{X}\mathbf{w} + \mathbf{e}_i$, where $\mathbf{e}_i = \mathbf{0}$ if the i'th worker is honest, otherwise can be arbitrary; upon receiving $\{\mathbf{L}_i\mathbf{X}\mathbf{w} + \mathbf{e}_i\}_{i=1}^m$, where at most t of the $\mathbf{e}_i$'s can be non-zero, master applies the decoding procedure and recovers $\mathbf{X}\mathbf{w}$ back. We can improve the computational complexity of this method significantly by observing that, in each iteration of our distributed CD algorithm, only $\mathbf{w}_{f(\mathcal{U})}$ gets updated and $\mathbf{w}_{\overline{f(\mathcal{U})}}$ remains unchanged. This implies that for computing $\mathbf{X}\mathbf{w}$, master only needs to send $\mathbf{w}_{f(\mathcal{U})}$ to the workers, as workers can store the result of local MV product from the previous iteration with themselves. This significantly reduces the local computation at the worker nodes, as now they only need to perform a local MV product of a matrix of size $p' \times |f(\mathcal{U})|$ and a vector of length $|f(\mathcal{U})|$.

Acknowledgements

The work of the authors was partially supported by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196, by the UC-NL grant LFR-18-548554, and by the NSF award 1740047. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[2] J. Konecný, "Stochastic, distributed and federated optimization for machine learning," Ph.D. dissertation, University of Edinburgh, 2017.

[3] L. Chen, H. Wang, Z. B. Charles, and D. S. Papailiopoulos, "DRACO: byzantine-resilient distributed training via redundant gradients," in *ICML*, 2018, pp. 902–911.

[4] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NIPS*, 2017, pp. 118–128.

[5] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[6] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *POMACS*, vol. 1, no. 2, pp. 44:1–44:25, 2017.

[7] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l1-regularized loss minimization," in *ICML*, 2011, pp. 321–328.

[8] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, 2015.

[9] P. Richtárik and M. Takáč, "Parallel coordinate descent methods for big data optimization," *Mathematical Programming*, vol. 156, no. 1, pp. 433–484, Mar 2016.

[10] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Information Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

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

[12] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *ICML*, 2017, pp. 3368–3376.

[13] N. Raviv, R. Tandon, A. Dimakis, and I. Tamo, "Gradient coding from cyclic MDS codes and expander graphs," in *International Conference on Machine Learning, ICML*, 2018, pp. 4302–4310.

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

[15] C. Karakus, Y. Sun, S. N. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *NIPS*, 2017, pp. 5440–5448.

[16] D. Data, L. Song, and S. N. Diggavi, "Data encoding for byzantine-resilient distributed gradient descent," in *Allerton Conference on Communication, Control, and Computing*, 2018.

[17] ——, "Data encoding for byzantine-resilient distributed optimization," [Available Online].

[18] M. Akçakaya and V. Tarokh, "A frame construction and a universal distortion bound for sparse representations," *IEEE Trans. Signal Processing*, vol. 56, no. 6, pp. 2443–2450, 2008.