Trading Communication for Computation in Byzantine-Resilient Gradient Coding

Christoph Hofmeister, Luis Maßny, Eitan Yaakobi, and Rawad Bitar

***Abstract*—We consider gradient coding in the presence of an adversary controlling so-called malicious workers trying to corrupt the computations. Previous works propose the use of MDS codes to treat the inputs of the malicious workers as errors and correct them using the error-correction properties of the code. This comes at the expense of increasing the replication, i.e., the number of workers each partial gradient is computed by. In this work, we reduce replication by proposing a method that detects the erroneous inputs from the malicious workers, hence transforming them into erasures. For s malicious workers, our solution can reduce the replication to $s+1$ instead of $2s+1$ for *each partial gradient* at the expense of only s additional computations at the main node and additional rounds of light communication between the main node and the workers. We give fundamental limits of the general framework for fractional repetition data allocation. Our scheme is optimal in terms of replication and local computation but incurs a communication cost that is asymptotically, in the size of the dataset, a multiplicative factor away from the derived bound.**

I. Introduction

Consider the setting of a main node possessing large amounts of data on which a machine learning model shall be trained using gradient descent. To speed up the learning process, the main node distributes the computations to several worker nodes [1], [2]. One of the main vulnerabilities of distributed gradient descent is the presence of Byzantine errors corrupting some workers' computation results [3]. Even a single corrupted computation result can drastically deteriorate the performance of the algorithm [4].

The problem of tolerating Byzantine errors in distributed computing has been considered in different settings. For example, for linear computations [5], [6] use Freivalds' algorithm to detect Byzantine errors with high probability and exclude them in further processing. For polynomial computations, [7] uses properties of error-correcting codes to correct errors. Other approaches towards the mitigation of erroneous results include group testing and Reed-Solomon codes [8], and homomorphic hash functions [9]. For a more comprehensive review of the existing literature, we refer the interested reader to [5].

For distributed gradient descent each worker computes a gradient of a so-called loss function for local training data. The main node aggregates these partial gradients into a total gradient. The problem of Byzantine errors in this context has been first tackled using robust aggregation functions, which select only a subset of the workers' results. The selection is based on minimizing the distance to other results [4], [10]–[14], by assigning a sanity score to each result [15], or by general consistency checks [16]. Using such aggregation functions, however, the resulting gradient estimate may be only an inexact approximation of the desired total gradient in the error-free case. This increases the runtime for the gradient descent algorithm, see e.g. [17] and references therein, and might perform poorly in some particular settings, e.g., when the distribution of the training data is not identical among the workers [18], [19]. Moreover, advanced gradient descent techniques, such as the momentum method [20], are in general not compatible with approximate schemes [19].

Due to the latter, the problem of tolerating Byzantine errors in distributed gradient descent with exact recovery has been approached from a coding-theoretic perspective. For linear and polynomial computations, coding over the input data at the main node has been proposed in [7] and [21]. Since the computations in gradient descent are highly non-linear in general, these approaches might not be applicable. Instead, [22] introduces DRACO, a framework that performs coding over the computation results at the workers. The authors build on the idea of gradient coding [19], which was originally designed to mitigate the effect of stragglers, i.e., slow or unresponsive workers. By replicating each gradient computation to $s+1$ different workers, gradient coding can tolerate s stragglers by treating them as erasures. Applying the same ideas, DRACO can tolerate s malicious workers instead, treating the computations of malicious workers as errors. This comes at the cost of increasing the replication of each gradient computation to $2s+1$, hence, causing a large computation overhead. Both DRACO and gradient coding are shown to achieve an optimal replication for the respective problem settings.

In this work, we also consider the problem of exact gradient coding in the presence of an adversary controlling s malicious workers, that introduce Byzantine errors in their computation results. In contrast to [22], we define a more general framework in which the main node can run a small number of computations itself to aid in decoding. We propose a scheme that requires a replication of only $s+1$ at the expense of running s local gradient computations[1]. The idea of running local computations at the main node is also used in [23], [24], where the main node computes an estimate of the true gradient from only few data samples and discards worker results that have a large distance to this estimate. In contrast to our work, those solutions do not guarantee exact recovery of the total gradient at the main node. We design additional light communication between the workers and the main

CH, LM and RB are with the School of Computation, Information and Technology at the Technical University of Munich, Germany. Emails: {christoph.hofmeister, luis.massny, rawad.bitar}@tum.de

EY is with the CS department of Technion—Israel Institute of Technology, Israel. Email: yaakobi@cs.technion.ac.il

This project is funded by the Bavarian Ministry of Economic Affairs, Regional Development and Energy within the scope of the 6G Future Lab Bavaria, and by DFG (German Research Foundation) projects under Grant Agreement No. WA 3907/7-1 and No. BI 2492/1-1.

[1]A replication of $s+1$ means that *each* gradient computation is run $s+1$ times; the cost of s local computations *in total* is very small in comparison.

978-1-6654-7554-9/23/$31.00 ©2023 IEEE

node to help identify which gradients should be computed locally. We generalize the framework to the case where less than s local computations are allowed at the main node and explore the tradeoff between replication and local computation.

II. Problem Setting

We first set the notation. Matrices and vectors are denoted by upper-case and lower-case bold letters, respectively. $\mathbf{A}_{i,j}$ refers to the element in row i and column j of the matrix $\mathbf{A}$. Scalars are denoted by lower-case letters, sets by calligraphic letters, and lists by fractal letters, respectively, e.g., a, $\mathcal{A}$ and $\mathfrak{A}$. For an integer $a \geq 1$, we define $[a] \stackrel{\text{def}}{=} \{1, 2, \ldots, a\}$. Let $\mathfrak{A}_i$, $i = 1, \ldots, t$, be a collection of lists, we define $\mathfrak{A}^{(t)}$ to be their concatenation. We use $\mathbf{1}_{m\times n}$ and $\mathbf{0}_{m\times n}$ to denote the all-one and all-zero matrices of dimension $m \times n$.

We consider a *synchronous distributed gradient descent* setting, in which the goal is to fit the parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ of a model to a dataset consisting of p samples $\mathbf{x}_i \in \mathbb{R}^d, i \in [p]$. This is done by finding (local) optima for the problem $\arg\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \sum_{i\in[p]} \ell(\boldsymbol{\theta}, \mathbf{x}_i)$ for a per-sample loss function $\ell(\boldsymbol{\theta}, \mathbf{x}_i)$. The gradient descent algorithm starts with a random initialization for the parameter vector, defined as $\boldsymbol{\theta}^{(0)}$, and then iteratively applies the update rule $\boldsymbol{\theta}^{(\tau+1)} = \boldsymbol{\theta}^{(\tau)} - \frac{\eta^{(\tau)}}{p} \sum_{i\in[p]} \nabla\ell(\boldsymbol{\theta}^{(\tau)}, \mathbf{x}_i)$, where τ is the iteration index and $\eta^{(\tau)} \in \mathbb{R}$ is referred to as the learning rate. For notational convenience, we define the evaluation of the gradient of the loss function at individual samples as $\mathbf{g}_i^{(\tau)} \stackrel{\text{def}}{=} \nabla\ell(\boldsymbol{\theta}^{(\tau)}, \mathbf{x}_i)$ and call them *partial gradients*. Since in practice data is quantized to a finite set of values, we take the partial gradients to be vectors over a finite alphabet $\mathcal{A}$, i.e., $\mathbf{g}_i^{(\tau)} \in \mathcal{A}^d$.

Consider a system comprising a main node and n worker nodes, s of which might be malicious. The malicious workers can send arbitrarily corrupted information to the main node. At the start of the procedure, the main node distributes the samples to the workers with some redundancy. Then, each iteration τ starts with the main node broadcasting the current parameter vector $\boldsymbol{\theta}^{(\tau)}$. The workers then compute the partial gradients $\mathbf{g}_i^{(\tau)}$ corresponding to the samples they store. At the end of the iteration, the main node must obtain the *full gradient* $\mathbf{g}^{(\tau)} \stackrel{\text{def}}{=} \sum_{i\in[p]} \mathbf{g}_i^{(\tau)}$ irrespective of the actions of the s malicious workers.

In this work, we are concerned with the problem of reliably reconstructing $\mathbf{g}^{(\tau)}$ *exactly* at the main node in each iteration τ. In the sequel, we only consider a single iteration of gradient descent and omit the superscript τ.

III. Byzantine-Resilient Gradient Coding

For a better grasp of our ideas, we start with a toy example that captures the concepts introduced and studied.

A. How to Catch Liars Efficiently?

Consider a game among three friends Alice (A), Bob (B) and Dan (D). A and B have p private numbers $g_1, \ldots, g_p$ whose sum should be communicated correctly to D. The problem is that one player is trying to cheat D.

At the first stage, each player sends the sum of all numbers to D. Then, the game is played in rounds. At each round, D can first ask A and B questions about $g_1, \ldots, g_p$. Only one player is guaranteed to reply truthfully. Then, D can query an oracle to uncover the true value of some of the g_i's. The game is repeated until D correctly obtains the desired sum.

Example 1. *Assume w.l.o.g that A is cheating D, $p = 4$, and that $g_i = i$. In a first stage, A and B send the sum of their numbers to D. Assume that A sends the value 2 and B sends the correct value 10. At the first round, D asks A and B to send the sum $g_1 + g_2$. To create confusion, A acts truthfully and also sends the value 3. Now, D knows that $g_3 + g_4$ is either -1 or 8. So, D decides not to query the oracle yet. He instead moves to the second round and asks A and B to send the value of g_3. Again, both players send the value 3. Hence, D now knows that g_4 has to be -4 if A is acting honestly or 5 if B is acting honestly. At this point, D decides to query the oracle for g_4, thus catching the liar A and obtaining the true sum 10.*

This game is the crux of our framework. The private numbers are the partial gradients computed at the workers, querying the oracle represents local computations at the main node, and asking questions is the light communication between the main node and the workers. The figures of merit of this game are: *i)* the minimum number of queries to the oracle that D needs; and *ii)* given that D can only obtain the minimum number of oracle queries and can play several rounds, how many questions does D need to ask the other players to recover the desired sum correctly. For a more elaborate example see [25, Appendix A].

B. The Framework

Carrying over this idea to the gradient coding framework, we next define gradient coding schemes resilient against an adversary controlling s malicious workers.

Definition 1 (Byzantine-resilient gradient coding scheme)**.** *A Byzantine-resilient gradient coding scheme tolerating s malicious workers, referred to as s-BGC, is a tuple $(\mathbf{A}, \mathfrak{E}, \mathrm{dec}, \Pi)$ where*

- $\mathbf{A} \in \{0,1\}^{p\times n}$ *is a* ***data assignment matrix*** *in which $\mathbf{A}_{i,j}$ is equal to 1 if the i-th data sample is given to the j-th worker and 0 otherwise,*
- $\mathfrak{E} \stackrel{def}{=} (\mathrm{enc}_{j,e} \mid j \in [n], e \in [v])$ *is the list of nv* ***encoding functions*** *used by the workers such that $\mathrm{enc}_{j,1}$ corresponds to a gradient code dictated by $\mathbf{A}$ and $\mathrm{enc}_{j,e}$ depends only on the gradients assigned to W_j,*
- $\Pi = (\Pi_1, \Pi_2)$ *is a* ***multi-round protocol*** *in which Π_1 selects the indices of the encoding functions to be used by the workers and Π_2 selects gradients to be locally computed at the main node,*
- *and* dec *is a* ***decoding function*** *used by the main node after running the protocol Π to always output the correct full gradient if the number of malicious workers is at most s.*

Each worker initially ($t = 0$) sends a vector $\mathbf{z}_{0,j} \stackrel{\text{def}}{=} \mathrm{enc}_{j,1}(\mathbf{g}_1, \ldots, \mathbf{g}_p) \in \mathcal{A}^d$ that is a codeword symbol of a gradient code [19]. The protocol Π then runs for $r \in \mathbb{N}$ rounds. In each round $t \in [r]$, the main node uses Π_1 to select an encoding function $\mathrm{enc}_{j,e_{t,j}}$ for each worker W_j and communicates its index $e_{t,j}$ to the respective worker. Each worker W_j then computes a response $\mathbf{z}_{t,j} =$

$\mathrm{enc}_{j,e_{j,t}}(\mathbf{g}_1,\dots,\mathbf{g}_p) \in \mathcal{A}^{d_{t,j}}$ for some $d_{t,j} \in \mathbb{N}_0$ and sends a vector $\widetilde{\mathbf{z}}_{t,j} \in \mathcal{A}^{d_{t,j}}$ to the main node. For honest workers $\widetilde{\mathbf{z}}_{t,j} = \mathbf{z}_{t,j}$, while for malicious workers, $\widetilde{\mathbf{z}}_{t,j}$ may be chosen arbitrarily.

In every round, the main node uses Π_2 to choose a set of partial gradients to compute locally. We denote the list of indices of the locally computed partial gradients in round t by $\mathfrak{I}_t$ and the corresponding list of partial gradient values by $\mathfrak{G}_t \stackrel{\text{def}}{=} (\mathbf{g}_i \mid i \in \mathfrak{I}_t)$. Analogously, we define $\mathfrak{Z}_t \stackrel{\text{def}}{=} (\mathbf{z}_{t,j} \mid \forall j \in [n])$ and $\widetilde{\mathfrak{Z}}_t \stackrel{\text{def}}{=} (\widetilde{\mathbf{z}}_{t,j} \mid \forall j \in [n])$.

The protocol Π_1 selects the indices of the encoding functions to be used in the current round t based on the received results and locally computed gradients from previous rounds. After receiving the results in the current round, the main node uses Π_2 to select the gradients to compute locally in this round, i.e.,

$$e_{1,t},\dots,e_{n,t} = \Pi_1\Big(t, \widetilde{\mathfrak{Z}}^{(t-1)}, \mathfrak{I}^{(t-1)}, \mathfrak{G}^{(t-1)}\Big), \quad (1)$$

$$\mathfrak{I}_t = \Pi_2\Big(t, \widetilde{\mathfrak{Z}}^{(t)}, \mathfrak{I}^{(t-1)}, \mathfrak{G}^{(t-1)}\Big). \quad (2)$$

After round r, the main node computes an estimate $\widehat{\mathbf{g}}$ of $\mathbf{g}$ using the decoding function $\widehat{\mathbf{g}} = \mathrm{dec}\Big(\widetilde{\mathfrak{Z}}^{(r)}, \mathfrak{I}^{(r)}, \mathfrak{G}^{(r)}\Big)$. The total number of partial gradients computed at the main node is defined as $c \stackrel{\text{def}}{=} \big|\mathfrak{I}^{(r)}\big|$.

A valid s-BGC scheme must output $\widehat{\mathbf{g}} = \mathbf{g}$ if the number of malicious workers is at most s.

C. Figures of Merit

An s-BGC scheme is evaluated by the maximum number of rounds r and the maximum number of local computations c required by Π, and its replication factor and communication overhead, which we define next.

Definition 2 (Replication factor and communication overhead)**.** *The **replication factor** of an s-BGC scheme is the average number of workers to which each sample is assigned, i.e.,*

$$\rho \stackrel{\text{def}}{=} \frac{\sum_{i\in[p],j\in[n]} \mathbf{A}_{i,j}}{p}.$$

*The **communication overhead** is the maximum number of symbols from $\mathcal{A}$ transmitted from the workers to the main node during Π, i.e.,*

$$\kappa \stackrel{\text{def}}{=} \sum_{t\in[r],j\in[n]} d_{t,j}.$$

We say that a tuple $(\mathbf{A}, \mathfrak{E}, \mathrm{dec}, \Pi)$ is a (r, c, ρ, κ)-s-BGC scheme if in the presence of at most s malicious workers, the scheme always outputs $\widehat{\mathbf{g}} = \mathbf{g}$ by requiring at most r communication rounds, at most c local computations, and has replication factor ρ and communication overhead less than or equal to κ.

We study settings in which the number of workers n divides $s+1$, i.e., $n = m(s+1)$ for some integer m and only consider balanced data assignments, i.e., every worker computes the same number of gradients. We focus on the particular case of a fractional repetition data assignment [19]. That is, the main node partitions the workers into m groups of size $\frac{n}{m}$ each and assigns the same data samples to all workers within a group. The data assignment matrix is constructed as

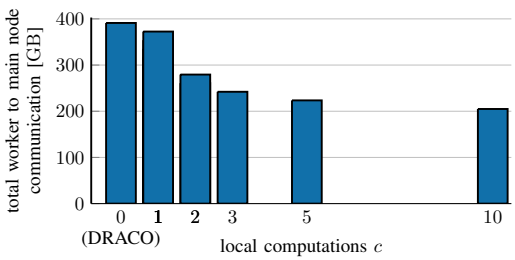

Fig. 1. Tradeoff between total worker to main-node communication and the number of local computations for our scheme. The parameters are $s = 10$, $1 \le u \le 11$, $m = 1$, $p = 1 \times 10^4$, $d = 1 \times 10^6$, $|\mathcal{A}| = 2^{16}$. When considering total communication, the cost of the protocol is outweighed by the cost of transmitting the gradient values. The gap to our bound, given in Theorem 2, is less than 5 kB. For $c = 0$ local computations, our scheme is equivalent to DRACO [22]. By requiring fewer workers, our scheme with $u = 1$ reduces communication by $48\,\%$ at the expense of at most $c = 10$ local gradient computations at the main node. For comparison, each worker node performs $p = 1 \times 10^4$ gradient computations.

$$\mathbf{A} = \begin{bmatrix} \mathbf{1}_{\frac{p}{m}\times\frac{n}{m}} & \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} & \dots & \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} \\ \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} & \mathbf{1}_{\frac{p}{m}\times\frac{n}{m}} & \dots & \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} & \mathbf{0}_{\frac{p}{m}\times\frac{n}{m}} & \dots & \mathbf{1}_{\frac{p}{m}\times\frac{n}{m}} \end{bmatrix}. \quad (3)$$

Since we focus on this particular data assignment, the initial worker responses are given by the sum of all computed gradients $\mathbf{z}_{0,j} = \sum_{\substack{i\in[p] \\ \mathbf{A}_{i,j}=1}} \mathbf{g}_i$, which form a valid gradient code.

IV. Bounds and Code Constructions of s-BGC Schemes

For the non-trivial case of $c < p$, i.e., the main node does not compute all the partial gradients locally, the replication factor of any s-BGC scheme is bounded from below by $\rho \ge s+1$. In addition, we note that if $c = 0$, then the replication factor of any s-BGC scheme is bounded from below as $\rho \ge 2s+1$ and can be achieved through DRACO [22]. Conversely, if $\rho \ge 2s+1$, then $c = \kappa = 0$ was shown to be achievable. Thus, we focus on the case $s+1 \le \rho \le 2s+1$, and we investigate the fundamental tradeoff between c, ρ and κ for any r.

In particular, we show that for $c = s$ the replication factor $\rho = s+1$ is achievable. For $0 < c < s$, we show that for any $1 \le u \le s+1$, if $\rho \le s+u$, then $c \ge \lfloor \frac{s}{u} \rfloor$. For $\rho = s+u$ and $c = \lfloor \frac{s}{u} \rfloor$, we give a lower bound on κ. We then construct an s-BGC scheme that requires $c \le \lfloor \frac{s}{u} \rfloor$ local computations, $\rho = s+u$, $r \le (s+1-u)\lceil \log_2(\frac{p}{m}) \rceil$ rounds and $\kappa \le (s+1-u)\Big(2\lceil \log_2(\frac{p}{m}) \rceil + \frac{s+3u}{2\log_2|\mathcal{A}|}\Big)$. As depicted in Fig. 1, for realistic parameter ranges, the local computations drastically reduce the required communication. The communication overhead of the protocol κ is outweighed by the initial transmission of $\mathbf{z}_{0,j}$. For a comparison between our achievability result and converse for κ, cf. [25, Appendix F].

Our lower bounds consider an adversary that chooses the following behavior for the malicious workers.

A. Symmetrization Attack for Bounds

We base parts of our proofs on the concept of a symmetrization attack, as explained in the following. The core idea is for the adversary to choose errors such that the main node cannot distinguish between different cases. The adversary chooses a (potentially corrupted) value for each malicious worker and each partial gradient. We denote

worker W_j's *claimed* partial gradient results for $\mathbf{g}_i$ as $\widetilde{\mathbf{g}}_i^{(j)} \in \mathcal{A}^d$ for all $j \in [n]$ and $i \in [p]$. For honest workers we say $\widetilde{\mathbf{g}}_i^{(j)} = \mathbf{g}_i$. Each worker W_j computes their responses consistently based on those values, i.e.,

$$\widetilde{\mathbf{z}}_{t,j} = \mathrm{enc}_{j,e_{j,t}}\Big(\widetilde{\mathbf{g}}_1^{(j)}, \dots, \widetilde{\mathbf{g}}_p^{(j)}\Big), \quad \forall t \in \{0, \dots, r\}, j \in [n], e_j \in [v].$$

For clarity of exposition, we first lay out how the adversary chooses the claimed gradient values for $m = 1$ groups of size $s+1$, i.e., for $u = 1$, before we generalize to arbitrary $m, u \in \mathbb{N}$.

For $u = 1$ and $m = 1$, the adversary draws a set $\widetilde{\mathcal{I}} \subseteq [p]$ of size $|\widetilde{\mathcal{I}}| = \lfloor \frac{s}{u} \rfloor = s$ uniformly at random and assigns each malicious worker W_j a unique gradient index from $\widetilde{\mathcal{I}}$. The adversary will introduce errors only for gradients $\mathbf{g}_{\tilde{i}}$, $\tilde{i} \in \widetilde{\mathcal{I}}$ and only for the one malicious worker that got assigned gradient index $\tilde{i}$. With probability $\frac{1}{2}$, it picks a single gradient index from $\widetilde{\mathcal{I}}$ uniformly at random and sets all malicious workers' values to the same erroneous random partial gradient value. The resulting claimed gradients take on the form as depicted in Table I. For partial gradients with index in $\widetilde{\mathcal{I}}$, there are two competing values $\mathbf{g}'_i$ and $\mathbf{g}''_i$ whereas for all other gradient indices the claimed values by all workers agree.

	$\widetilde{\mathbf{g}}_1^{(j)}$	$\widetilde{\mathbf{g}}_2^{(j)}$	$\dots$	$\widetilde{\mathbf{g}}_s^{(j)}$	$\widetilde{\mathbf{g}}_{s+1}^{(j)}$	$\dots$	$\widetilde{\mathbf{g}}_p^{(j)}$
W_1	$\mathbf{g}''_1$	$\mathbf{g}'_2$	$\dots$	$\mathbf{g}'_s$	$\mathbf{g}'_{s+1}$	$\dots$	$\mathbf{g}'_p$
W_2	$\mathbf{g}'_1$	$\mathbf{g}''_2$	$\dots$	$\mathbf{g}'_s$	$\mathbf{g}'_{s+1}$	$\dots$	$\mathbf{g}'_p$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\dots$	$\vdots$
W_s	$\mathbf{g}'_1$	$\mathbf{g}'_2$	$\dots$	$\mathbf{g}''_s$	$\mathbf{g}'_{s+1}$	$\dots$	$\mathbf{g}'_p$
W_{s+1}	$\mathbf{g}'_1$	$\mathbf{g}'_2$	$\dots$	$\mathbf{g}'_s$	$\mathbf{g}'_{s+1}$	$\dots$	$\mathbf{g}'_p$

TABLE I
CLAIMED PARTIAL GRADIENTS FOR SYMMETRIZATION ATTACK.

For $u > 1$, the adversary randomly partitions the malicious workers into $\lfloor s/u \rfloor - 1$ sets of size u and (depending on divisibility) one group of size $s \mod u$. Each of the sets of size u behaves like one malicious worker in the case $u = 1$. The workers in the remaining set of less than u workers (if non-empty) pick their claimed gradients randomly either like a random other set of workers or like the honest workers. The resulting claimed gradients take on the form as depicted in [25, Appendix B].

For $m > 1$, the adversary chooses the claimed gradients in the first group according to the attack strategy for $m' = 1$ and $p' = p/m$. In all other groups, the claimed gradients equal their true values.

B. Fundamental Limits

Theorem 1 (Lower bound on c and ρ)**.** *Suppose that $n = m(s+u)$ for integers $m, u \geq 1$. For any tuple ($\mathbf{A}$, $\mathfrak{E}$, dec, Π), with $\mathbf{A}$ as in* (3)*, to be a (r,c,ρ,κ)-s-BGC scheme, it holds that if $\rho \leq s+u$, then $c \geq \lfloor \frac{s}{u} \rfloor$ (conversely, if $c < \lfloor \frac{s}{u} \rfloor$ then $\rho > s+u$) for any number of rounds r and any communication overhead κ.*

Proof: We demonstrate that any tuple $(\mathbf{A}, \mathfrak{E}, \mathrm{dec}, \Pi)$ with fractional repetition data allocation which uses $c \leq \lfloor \frac{s}{u} \rfloor - 1$ local computations cannot be an s-BGC scheme. Specifically, we show that the malicious workers can always perform a symmetrization attack preventing the main node from deterministically computing the full gradient. Let the malicious workers behave as explained in Section IV-A.

We abstract $\mathfrak{E}$ and Π_1 by assuming that the values of all partial gradients $\widetilde{\mathbf{g}}_i^{(j)}$ computed by the workers are available at the main node. Note that regardless of $\mathfrak{E}$ and Π_1 the main node cannot gain any additional information from the workers' responses. We now show that for $c \leq \lfloor \frac{s}{u} \rfloor - 1$ there exists no choice of a decoding function dec and Π_2 for which the main node deterministically outputs the true full gradient. To that end, for any possible Π_2, we give two cases for which the inputs to the decoding function are identical but the true full gradients differ.

The claimed partial gradients take on values as in Table I for $u = 1$ and [25, Appendix B] for $u > 1$. There are $\lfloor \frac{s}{u} \rfloor$ partial gradients, say $\mathbf{g}_1, \dots, \mathbf{g}_{\lfloor \frac{s}{u} \rfloor}$, such that for each partial gradient $\mathbf{g}_i$ worker W_i sends a value $\widetilde{\mathbf{g}}_i^{(i)} = \mathbf{g}''_i$ that is different from the value $\mathbf{g}'_i$ sent by all other workers. For any list $\mathfrak{I}^{(r)}$ of locally computed gradients of size $|\mathfrak{I}^{(r)}| \leq s-1$ (produced by any Π_2) there exists an index $\tilde{i} \in [s]$ such that $\tilde{i} \notin \mathfrak{I}^{(r)}$. Consider these two cases:

Case 1: $\mathbf{g}_i = \mathbf{g}'_i \ \forall i \in [\frac{p}{m}]$ and
Case 2: $\mathbf{g}_i = \mathbf{g}'_i \ \forall i \in [\frac{p}{m}] \setminus \{\tilde{i}\}$ and $\mathbf{g}_{\tilde{i}} = \mathbf{g}''_{\tilde{i}}$.

It is easy to see that both cases occur with non-zero probability according to the attack strategy in Section IV-A. In both cases, the inputs to dec only depend on the fixed $\mathfrak{I}^{(r)}$, the $\widetilde{\mathbf{g}}_i^{(j)}$, $j \in [s+1]$, $i \in [\frac{p}{m}]$ and the $\mathbf{g}_i$, $i \in \mathfrak{I}^{(r)}$, all of which take on identical values in both cases. The value of the full gradient $\mathbf{g}$, however, is $\sum_{i \in [p]} \mathbf{g}'_i$ in Case 1 and $\mathbf{g}''_{\tilde{i}} + \sum_{i \in [p] \setminus \{\tilde{i}\}} \mathbf{g}'_i$ in Case 2. Hence, for $c < \lfloor \frac{s}{u} \rfloor$ no decoding function can deterministically produce the correct full gradient. ∎

Theorem 2 (Lower bound on κ for fixed c and ρ)**.** *Suppose that $n = m(s+u)$ for integers $m, u \geq 1$. For any tuple ($\mathbf{A}$, $\mathfrak{E}$, dec, Π), with $\mathbf{A}$ as in* (3)*, to be a (r,c,ρ,κ)-s-BGC scheme with $\rho = s+u$ and $c = \lfloor \frac{s}{u} \rfloor$, then it must hold that*

$$\kappa \geq \log_{|\mathcal{A}|} \binom{p/m}{\lfloor s/u \rfloor}.$$

Proof: The proof is given in [25, Appendix D]. ∎

C. Construction of an s-BGC Scheme

Theorem 3. *The scheme constructed below is an s-BGC scheme with a parameter u such that $1 \leq u \leq s+1$ and requires $r \leq (s+1-u)\lceil \log_2(\frac{p}{m}) \rceil$ rounds, $c \leq \lfloor \frac{s}{u} \rfloor$ local gradient computations, $\rho = s+u$, and $\kappa \leq (s+1-u)\Big(2\lceil \log_2(\frac{p}{m}) \rceil + \frac{s+3u}{2\log_2 |\mathcal{A}|}\Big)$.*

Proof: The proof is given in [25, Appendix E]. ∎

We construct an (r, c, ρ, κ)-s-BGC scheme that has a replication factor $\rho = s+u$, $u \geq 1$, and requires the optimal local computation load $c \leq \lfloor \frac{s}{u} \rfloor$ at the main node. The protocol Π runs for $r \leq (s+1-u)\lceil \log_2(\frac{p}{m}) \rceil$ rounds and achieves a communication overhead $\kappa \leq (s+1-u)\Big(2\lceil \log_2(\frac{p}{m}) \rceil + \frac{s+3u}{2\log_2 |\mathcal{A}|}\Big)$. Our scheme uses a fractional repetition data assignment with m groups each of size $s+u$. Informally, in each group, the main node runs an elimination tournament consisting of matches (similar to the one explained in Example 1) between pairs of workers that return contradicting responses. For clarity of exposition, we explain the idea of our scheme for the special case of $u = 1$. The general case for $u \geq 2$ follows similar steps [25, Appendix C].

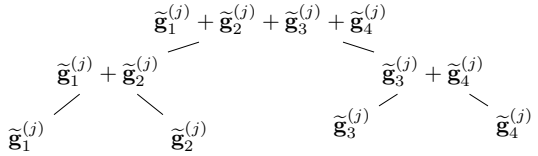

Fig. 2. Example of a match tree for W_j and parameters $m = 1, p = 4$.

The tournament consists of a series of matches between two workers. During a match, each worker constructs a binary tree based on their computed partial gradients, which we refer to as the match tree. The root of the tree is labeled by the sum of all partial gradients computed at that worker ($\widetilde{\mathbf{z}}_{0,j}$). The child nodes are constructed based on the partial gradients that are contained in the parent node. Each node has two children: the first one is labeled by the sum of the first half of the parent node's partial gradients; the second one is labeled by the sum of the second half of the parent node's partial gradients. Thus, the labeling is done such that the sum of the labels of any two siblings gives the label of their parent node.

Proceeding in this way recursively, each worker ends up with the leaves of the tree being labeled by individual partial gradients. For example, when $m = 1$ and $p = 4$, the tree is depicted in Fig. 2. During a match, the main node requests the labels for particular nodes in this tree from the two competing workers, and compares them. If the root labels of two match trees differ, then by definition there must be a child node for which the corresponding label differs in those trees. By induction, it is clear that there has to be a path from the root to a leaf, such that the corresponding labels of all involved nodes differ between the two match trees. Applying this observation to the example in Fig. 2, if $\widetilde{\mathbf{g}}_1^{(j)} + \widetilde{\mathbf{g}}_2^{(j)} + \widetilde{\mathbf{g}}_3^{(j)} + \widetilde{\mathbf{g}}_4^{(j)}$ is different for two workers, then $\widetilde{\mathbf{g}}_1^{(j)} + \widetilde{\mathbf{g}}_2^{(j)}$ or $\widetilde{\mathbf{g}}_3^{(j)} + \widetilde{\mathbf{g}}_4^{(j)}$ must also differ (or both). In the latter case, we end up with $\widetilde{\mathbf{g}}_3^{(j)}$ or $\widetilde{\mathbf{g}}_4^{(j)}$ being different between the workers.

The match starts at the root. In this case, each worker W_j would have already sent the node label in $\mathbf{z}_{0,j}$ to the main node, i.e.,

$$\mathbf{z}_{0,j} = \mathrm{enc}_{j,1}(\mathbf{g}_1, \ldots, \mathbf{g}_p) = \sum_{\substack{i \in [p] \\ \mathbf{A}_{i,j}=1}} \mathbf{g}_i.$$

Note that, without errors, all workers' messages agree within a group. In case of discrepancies between the responses $\widetilde{\mathbf{z}}_{0,j}$ of the workers within a group, the main node selects a pair of disagreeing workers W_{j_1}, W_{j_2} and further descends in the tree as follows.

For every node in the tree, the main node requests and compares the left child's label from both workers, encoded in $\mathbf{z}_{t,j_1}$ and $\mathbf{z}_{t,j_2}$. The main node then moves on to a child whose label the workers disagree on. Based on the current node's label and the left child's label, the main node can infer the right child's label. If the competing workers agree on the left child's label, they must disagree on the right child's label. This procedure is repeated until a leaf is reached. Note that, even if the workers send inconsistent labels at each round, this procedure is guaranteed to reach a leaf node for which the (sent or inferred) values of the individual gradient is different for the two workers.

It is possible to reduce the communication load by picking a coordinate ζ in which the workers' initial responses disagree, i.e., $[\mathbf{z}_{0,j_1}]_\zeta \neq [\mathbf{z}_{0,j_2}]_\zeta$. That is, the workers only encode the ζ-th coordinate of the node labels. By this method, it is still guaranteed that the main node can identify a partial gradient for which the competing workers disagree.

Having identified disagreeing values $\widetilde{\mathbf{g}}_i^{(j_1)}$ and $\widetilde{\mathbf{g}}_i^{(j_2)}$ of a partial gradient $\mathbf{g}_i$, the main node computes the correct value of this partial gradient locally. It then marks the worker(s) whose values differ from the value computed locally as malicious. The algorithm ends up with disagreeing leaf labels by design. Each match is guaranteed to eliminate at least one malicious worker and no honest worker. After performing at most s matches, the main node is guaranteed to identify all malicious workers. The main node takes the encoded gradient of one worker identified as being honest from each group and sums up the group-wise results to obtain $\widehat{\mathbf{g}} = \mathbf{g}$.

Algorithmic descriptions of the elimination tournament and a match between workers are in [25, Appendix H].

D. Discussion

According to Theorem 1, the interactive protocol of our scheme achieves the lowest possible number of locally computed partial gradients for the fractional repetition data assignment and $\rho = s + u$. Note that for $u \geq s + 1$, i.e., $\rho \geq 2s + 1$, no local computation is necessary. In fact, since there is only one set of consistent workers $\mathcal{W}$ that has $|\mathcal{W}| \geq u$ per fractional repetition group, the scheme immediately identifies $\mathcal{W}$ as the honest set and terminates without any additional computation or communication. Thus, as shown in [22], this is optimal. Although we consider $1 \leq u \leq s+1$ in Theorem 3 for technical reasons, the scheme works for any $u \geq 1$. We remark that since $p \gg s$ in state-of-the-art machine learning deployments, the local computations cause only a relatively small load at the main node. Note that the achievable bound on κ in Theorem 3 is off from the converse bound in Theorem 2 by a constant factor asymptotically, see [25, Appendix F]. The reason is that we use a rather simple and conservative lower bound on the amount of information that is required to be transmitted by the workers in Theorem 2. Among others, we assume that all workers have knowledge about the malicious worker's identities and also the malicious workers contribute useful information. A tighter converse bound is left as future work. Finally, we remark that although the communication complexity of our scheme is quadratic in s, this value is not very large in practice.

V. Conclusion

We considered the problem of distributed learning in the presence of Byzantine computation errors. We introduced a framework that extends the known gradient coding framework by adding an interactive light communication between the main node and the workers and verifying local computations at the main node. In the scope of this framework, we proposed a new scheme that can tolerate s malicious workers with a computational redundancy of $s + u$ for any $u \geq 1$. We showed that with a fractional repetition data assignment, the scheme achieves the optimal number of local computations at the main node. Future work includes the improvements of the converse and achievability bounds, the generalization of the fundamental limits to a broader class of data assignments, and the investigation into optimal Byzantine-resilient gradient coding schemes.

REFERENCES

[1] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[3] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, jul 1982. [Online]. Available: https://dl.acm.org/doi/10.1145/357172.357176

[4] G. Damaskinos, E.-M. El-Mhamdi, R. Guerraoui, A. Guirguis, and S. Rouault, "AggregaThor: Byzantine machine learning via robust gradient aggregation," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 81–106, 2019.

[5] C. Hofmeister, R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, "Secure private and adaptive matrix multiplication beyond the singleton bound," *IEEE Journal on Selected Areas in Information Theory*, vol. 3, no. 2, pp. 275–285, 2022.

[6] T. Tang, R. E. Ali, H. Hashemi, T. Gangwani, S. Avestimehr, and M. Annavaram, "Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 628–638.

[7] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 89. PMLR, 16–18 Apr 2019, pp. 1215–1225.

[8] A. Solanki, M. Cardone, and S. Mohajer, "Non-Colluding Attacks Identification in Distributed Computing," in *2019 IEEE Information Theory Workshop (ITW)*, Visby, Sweden, Aug. 2019, pp. 1–5.

[9] Y. Keshtkarjahromi, R. Bitar, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Secure coded cooperative computation at the heterogeneous edge against Byzantine attacks," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2019, pp. 1–6.

[10] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in Byzantium," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 10–15 Jul 2018, pp. 3521–3530.

[11] Y. Chen, L. Su, and J. Xu, "Distributed statistical machine learning in adversarial settings: Byzantine gradient descent," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 2, pp. 1–25, Dec. 2017.

[12] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[13] S. Rajput, H. Wang, Z. Charles, and D. Papailiopoulos, "Detox: A redundancy-based framework for faster and more robust gradient aggregation," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[14] E.-M. El-Mhamdi, R. Guerraoui, and S. Rouault, "Fast and robust distributed learning in high dimension," in *International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020, pp. 71–80.

[15] C. Xie, S. Koyejo, and I. Gupta, "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 09–15 Jun 2019, pp. 6893–6901.

[16] K. Konstantinidis and A. Ramamoorthy, "Aspis: Robust detection for distributed learning," in *2022 IEEE International Symposium on Information Theory (ISIT)*, 2022, pp. 2058–2063.

[17] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic Gradient Coding for Straggler Mitigation in Distributed Learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, May 2020.

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

[19] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 06–11 Aug 2017, pp. 3368–3376.

[20] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*. PMLR, May 2013, pp. 1139–1147.

[21] D. Data, L. Song, and S. N. Diggavi, "Data encoding for Byzantine-resilient distributed optimization," *IEEE Transactions on Information Theory*, vol. 67, no. 2, pp. 1117–1140, Feb. 2021.

[22] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "DRACO: Byzantine-resilient distributed training via redundant gradients," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 10–15 Jul 2018, pp. 903–912.

[23] X. Cao and L. Lai, "Distributed gradient descent algorithm robust to an arbitrary number of Byzantine attackers," *IEEE Transactions on Signal Processing*, vol. 67, no. 22, pp. 5850–5864, Nov. 2019.

[24] S. Prakash, H. Hashemi, Y. Wang, M. Annavaram, and S. Avestimehr, "Secure and fault tolerant decentralized learning," 2020. [Online]. Available: https://arxiv.org/abs/2010.07541

[25] C. Hofmeister, L. Maßny, E. Yaakobi, and R. Bitar, "Trading communication for computation in Byzantine-resilient gradient coding," 2023. [Online]. Available: https://arxiv.org/abs/2303.13231