Democratizing Machine Learning

Resilient Distributed Learning with Heterogeneous Participants

Anonymous Authors

***Abstract*—Classical algorithms in distributed learning usually impose a uniform workload on all participating devices, typically discarding slower workers in each iteration of the learning process. This not only induces a sub-optimal utilization of the available computational resources, but also a poor learning accuracy, as critical data only held by slower devices are discounted from the learning. We propose HGO, a distributed learning algorithm with parameterizable iteration costs that can be adjusted to the computational capabilities of different devices, thereby encouraging the participation of slower devices. Besides being adaptive, HGO is also robust in the sense that it tolerates faulty devices, by combining a randomized sampling of devices with a robust aggregation scheme. We convey the convergence of HGO for both non-convex and strongly convex settings, without assuming any specific partitioning of the data over the devices. By doing so, we characterize, for the first time, the interplay between the statistical error, the convergence rate, the Byzantine resilience, the per-iteration cost and the run-time complexity of distributed learning. Particularly, we capture the trade-off between the impact of Byzantine workers (which is a function of the mini-batch size, as we show) and the computation rate of workers (which is critical for convergence). Empirically, we evaluate HGO in a distributed setting, for a panoply of models and datasets.**

***Index Terms*—Distributed computing, Byzantine failures, Heterogeneity, Machine learning, Optimization**

I. Introduction

Mobile devices are now an essential component of our modern society. Their number of users will reach 5.1 billion by 2025, representing 70% of the world's population.[1] Each day, these devices generate a massive amount of data that must be processed locally (on the device that generated it), both for infrastructure cost efficiency and privacy reasons. Distributed machine learning (ML) proposes to implement this requirement as follows: several devices collaboratively learn a common model by exchanging parameters, possibly through a server machine [1], [2]. However, standard solutions to distributed ML do not provide an obvious way to account for heterogeneity between devices, neither in terms of data generation nor in terms of hardware.

While classical optimization algorithms like *Stochastic Gradient Descent* (SGD) have proved their utility amongst practitioners, they typically assume that participating devices have similar hardware specifications. While this requirement might be satisfied in data centers equipped with powerful co-located computers steadily connected to the power grid and to the network, personal computing devices owned by general public (such as smartphones or laptops) often behave in an unpredictable way. For example, they may often disconnect from the network, become frequently inactive to save battery consumption, or crash accidentally. Classical algorithms are not well suited for such handheld devices, as they notoriously discard weaker devices (with low computational powers) from the learning process. This leads to (1) sub-optimal usage of available computing resources and (2) poorer learning accuracy, as data held by weaker devices is excluded from the learning process. On top of that, some devices might get hacked or manipulated by malicious parties, and inject arbitrary perturbations to the system. Such devices are technically referred to as *Byzantine* [3].

In particular, we recognize three critical issues when deploying classical distributed ML algorithms on handheld devices, namely: *computational heterogeneity*, *data heterogeneity*, and *Byzantine failures*. These issues have received a lot of attention so far, but prior works either consider them individually (e.g. [4]–[17]), or consider only two of these issues at a time (e.g. [18]–[21]).[2] We propose HGO,[3] a distributed optimization algorithm that mitigates the impact of all three issues. HGO achieves this by using a scheme for workload distribution adaptive to devices, depending on their computational capabilities. The adaptation tackles both computational and data heterogeneity, thereby resulting in a more efficient use of available computing resources and improved learning accuracy, respectively. To tolerate Byzantine devices, HGO makes use of a robust *aggregation rule* (e.g., Trimmed Mean [9], MeaMed [8], Krum [4], or Aksel [5]).

Summary of Key Contributions: HGO is inspired by the stochastic *block coordinate descent* method. In particular, the server initiates each iteration by randomly choosing a subset of workers, and sending them its current estimates of the learning parameters. Then, an honest (non-Byzantine) worker sends a fragment (i.e., some coordinates) of its mini-batch stochastic gradient. The server identifies a block of coordinates for which it receives values from a quorum of workers, and applies the robust aggregation rule on this block of workers' gradients to update its current parameter estimates. We show linear and sub-linear convergence of HGO in the strongly convex and non-convex settings, respectively. Moreover, we also report on the learning performance of HGO through simulations (in

[1] According to the Mobile Economy 2019 report by the GSM Association: https://data.gsmaintelligence.com/api-web/v2/research-file-download?id=39256194&file=2712-250219-ME-Global.pdf

[2] Please refer to Section VI for additional related work.

[3] Abbreviation for *Heterogeneous Gradient-based Optimization*. HgO is also the chemical formula for mercuric oxide, a compound used as an antiseptic in medicine, and as a fungicide in agriculture.

the full version of the paper, we also present a deployment on actual Android-based smartphones). Our key theoretical and empirical findings are summarized below.

1) The worst case time complexity of HGO is still better than the one of classical SGD.
2) We demonstrate for the first time a critical trade-off between the convergence quality (i.e., the rate and the Byzantine robustness) of a gradient-based distributed ML algorithm and the computation costs for the devices.
3) Our experiments highlight the fact that the convergence quality of HGO is positively correlated with the computation power of the network, thereby enabling any device to participate in training ML models according to its computation capabilities.

Our theoretical analysis also apply to a much larger family of gradient-based protocols (including CD, SCD, Block CD, GD, SGD) as well as that of robust aggregation rules (including Median, Trimmed Mean, Krum, Aksel, MeaMed). To our knowledge, none of these learning algorithms have been analyzed under the combination of issues we consider.

The rest of the paper is organized as follows. We present the model and preliminary concepts in Section II. Section III describes the algorithm and its time complexity. Section IV presents our theoretical analysis of HGO and a discussion on the trade-offs. Section V summarizes its empirical performance. Section VI discusses additional related works, and Section VII concludes the paper.

For space limitations, **we defer all the proofs, as well as the detailed description of our evaluation, to the full version of the paper**, which is available on GitHub: https://anonymous.4open.science/r/HgO-5136

II. Preliminaries

We present in this section the model setting and introduce the concepts of heterogeneity and robust aggregation. We also state all assumptions used in our theoretical results.

A. Model setting

We consider a set on N data points $X = \{x_1, \ldots, x_N\}$ that is arbitrarily partitioned over n workers (e.g., smartphones) in a parameter-server architecture. The set of data points held by a worker i is denoted by $\mathcal{S}_i$. Thus, $\bigcup_{i=1}^{n} \mathcal{S}_i = X$, and $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for all $i \neq j$. The server is assumed to be trustworthy, but some of the workers may be faulty, and their identity is a priori unknown [3]. Specifically, there are two types of workers:

- **Correct workers:** These are honest processes that correctly follow the instructions prescribed by the server. We represent the correct workers by set $\mathcal{C} \subseteq [n]$ where $[n]$ denotes the set $\{1, \ldots, n\}$.
- **Byzantine workers:** The remaining workers, i.e., $[n] \setminus \mathcal{C}$, are assumed to be Byzantine, i.e., they have an arbitrary behavior. For example, they may crash, or inject adversarial perturbations in the system (e.g., see [4]).

Our goal is to design a distributed learning algorithm that allows all the correct workers to learn a ML model over the union of their data points $X_{\mathcal{C}} = \bigcup_{i \in \mathcal{C}} \mathcal{S}_i$, despite the presence of up to f Byzantine workers. Specifically, we fix a learning model Π parameterized by $w \in \mathbb{R}^d$ for which each data point x incurs a loss of $\ell(w, x)$, where the function $\ell : \mathbb{R}^d \times X \to \mathbb{R}$ is assumed differentiable. The algorithm aims to find a local minimum of the function

$$\mathcal{L}(w) := \frac{1}{|X_{\mathcal{C}}|} \sum_{x \in X_{\mathcal{C}}} \ell(w, x). \tag{1}$$

Besides the presence of faulty workers, another challenge we consider in our model is the heterogeneous computational capabilities of the workers. To formally discuss this issue, we first briefly describe the classical algorithm of distributed stochastic gradient descent (SGD).

a) Distributed SGD Algorithm: This is an iterative algorithm where the server maintains an estimate of a solution to the learning problem, which is updated iteratively with the help of the workers. The initial estimate w_1 may be chosen arbitrarily. In each iteration $k = 1, 2, \ldots$, the server broadcasts its current estimate w_k to all workers, and waits a certain amount of time for the workers to return their stochastic gradients. Each worker i samples a random *mini-batch* $\zeta_{i_k} \subset \mathcal{S}_i$ of size s_i to compute the corresponding *stochastic* gradient:

$$G_i(w_k) = \frac{1}{s_i} \sum_{x \in \zeta_{i_k}} \nabla \ell(w_k, x) \tag{2}$$

Then the workers send back their stochastic gradients to the server, which updates the model parameters as follows:

$$w_{k+1} = w_k - \gamma \left(\frac{1}{n} \sum_{i=1}^{n} G_i(w_k) \right) \tag{3}$$

where $\gamma \in \mathbb{R}_{>0}$ is the learning rate. Under certain conditions, the algorithm eventually outputs a solution to the learning problem. However, the convergence of the above algorithm often relies on the assumption that all workers have comparable computational capabilities.

B. Computational and data Heterogeneity

In addition to faults, when deploying distributed learning on low-powered hand-held devices, such as tablets or smartphones, workers may have very different computational capabilities. Specifically, we consider that each honest (or correct) worker i computes a single coordinate of gradient $\nabla \ell(w, x)$, for a given parameter w and data point $x \in \mathcal{S}_i$, with a rate of $\lambda_i \in \mathbb{R}_{>0}$. Therefore, if the server allocates τ amount of time to each worker in an iteration k, worker i computes $\min\{\frac{\lambda_i \tau}{s_i}, d\}$ coordinates of its stochastic gradient $G_i(w_k)$. That is, slower workers (with lower computational capabilities) may only be able to compute their stochastic gradients partially, while the faster workers (with higher computational capabilities) may compute all the coordinates of their stochastic gradients. In what follows, we denote $\mathcal{Q}_j$ the set of workers that computed coordinate j of their stochastic gradients.

Besides the differences in workers' computational capabilities, distributed learning also faces the challenge of *data heterogeneity*, i.e., the data partitioning amongst the workers need not be uniform. Thus, even the correct workers are unable to compute unbiased estimates of the true gradient $\nabla \mathcal{L}(w)$, as pictorially shown in Figure 1 below.

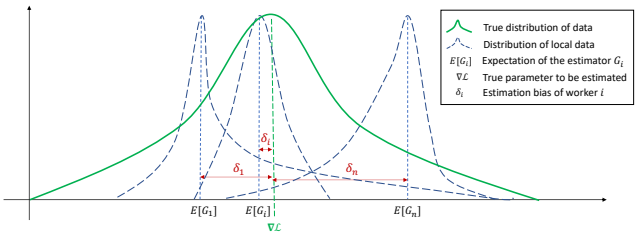


Fig. 1: The dashed blue curves model the distributions of local data as seen by each worker. The green curve models the true distribution of data. In our setting, each worker tries to estimate a block of partial derivatives of the true cost function using only its local data. Due to data heterogeneity, the estimates of the workers are variously biased.

C. Robust Aggregation

In presence of Byzantine workers, the server cannot rely on the simple averaging of all the workers' gradients as in the classical distributed SGD algorithm. Rather it uses a coordinate-wise robust *aggregation rule* AR, which we formally define as follows in our context. For a vector $v \in \mathbb{R}^d$, its j-th element is denoted by $\{v\}_j$. Recall that $\mathcal{C}$ denotes the set of correct workers , and the function $\mathcal{L}(w)$ is defined in (1) above. For a correct worker $i \in \mathcal{C}$, recall that its gradient $G_i(w)$ at any parameter $w \in \mathbb{R}^d$ is defined by (2). The gradients of a Byzantine worker i may be arbitrary.

Definition 1. *For a given $q \in [d]$, $w \in \mathbb{R}^d$, an aggregation rule* $\mathrm{AR} : \mathbb{R}^q \to \mathbb{R}$ *is* $\Delta(q, f)$-robust *if for every subset $\mathcal{Q} \subseteq [n]$ with $|Q| \geq q$,*

$$\mathbb{E}\left[(\mathrm{AR} - \{\nabla \mathcal{L}(w)\}_j)^2\right] \leq \Delta(q, f) \left(\frac{1}{|\mathcal{C}_\mathcal{Q}|} \sum_{i \in \mathcal{C}_\mathcal{Q}} \sigma_i^2(w)\right), \ \forall j \in [d]$$

where $\mathrm{AR} = \mathrm{AR}\left(\{G_i(w)\}_j;\ i \in \mathcal{Q}\right)$, $\mathcal{C}_\mathcal{Q} = \mathcal{C} \cap \mathcal{Q}$, *and* $\sigma_i^2(w) = \mathbb{E}\left[\left(\{G_i(w)\}_j - \{\nabla \mathcal{L}(w)\}_j\right)^2\right]$ *for all $i \in \mathcal{C}$. The robustness coefficient $\Delta(q, f)$ is a bounded positive real value that depends upon q and f.*

Remark 1. *It is only reasonable that the error of a robust aggregation rule should reduce when the number of correct workers increases. That is, the robustness coefficient $\Delta(q, f)$ should be a decreasing function of q. While earlier aggregation rules (e.g., [4], [6]) behaved against this reason, new improved aggregation rules (e.g., [5], [8], [9], [15]) satisfy this property. We only consider aggregation rules that satisfy this property.*

D. Assumptions

To formally analyze the convergence of our algorithm, we make the following standard assumption on the smoothness of the loss function [22]. For a vector v, we let $\{v\}_i$ denote its i-th element.

Assumption 1 (Smoothness)**.** *For all $j \in [d]$, there exists $L_j < \infty$ such that for all $w, w' \in \mathbb{R}^d$,*
$|\{\nabla \mathcal{L}(w')\}_j - \{\nabla \mathcal{L}(w)\}_j| \leq L_j \|w' - w\|$.

Although most of our results do not rely on the convexity assumption, we do present a stronger convergence guarantee when the loss function is strongly convex, as stated below.

Assumption 2 (Strong convexity)**.** *There exists $0 < \mu < \infty$ such that for all $w, w' \in \mathbb{R}^d$,*
$\mathcal{L}(w') \geq \mathcal{L}(w) + \langle \nabla \mathcal{L}(w), w' - w \rangle + \frac{\mu}{2} \|w' - w\|^2$ *where $\langle \cdot, \cdot \rangle$ denotes the inner product.*

To formally model the inherent inaccuracies of estimation,, we make the following two assumptions, that are satisfied in many learning problems [22], [23]. Note that we do not rely on the stronger assumption of uniformly bounded variance, which is indeed quite difficult to satisfy, and perhaps even impossible in the presence of strong convexity [23]. Instead, we assume a non-uniform bound, as stated below in Assumption 4.

Assumption 3 (Bounded bias)**.** *For all $i \in [n]$ and $j \in [d]$, there exist $\delta_{i,j} < \infty$ such that for all $w \in \mathbb{R}^d$,*
$|\mathbb{E}_{x \sim \mathcal{S}_i}\left[\{\nabla \ell(w, x)\}_j\right] - \{\nabla \mathcal{L}(w)\}_j| \leq \delta_{i,j}$ *where $x \sim \mathcal{S}_i$ denotes uniform distribution of x in $\mathcal{S}_i$.*

Assumption 4 (Non-uniform variance)**.** *For all $w \in \mathbb{R}^d$, there exists $M_w < \infty$ and $Q_w < \infty$ such that for all $i \in [n]$ and $j \in [d]$,* $\mathbb{E}_{x \sim \mathcal{S}_i}\left[\left(\{\nabla \ell(w, x)\}_j - \mathbb{E}_{x \sim \mathcal{S}_i}\left[\{\nabla \ell(w, x)\}_j\right]\right)^2\right] \leq M_w + Q_w \|\{\nabla \mathcal{L}(w)\}_j\|^2$.

III. Our Algorithm, and its Convergence Guarantee

We now present our proposed algorithm HGO.

The HGO algorithm, summarized in Algorithm 1, follows T steps of a *generalized* distributed SGD algorithm. At each iteration k, the server maintains an estimate w_k of an optimal parameter vector. To update its estimate, the server chooses a random order of coordinate indices C_k (i.e., C_k is a permutation of the set $\{1, \ldots, d\}$), selects a random set of q workers, and broadcasts (w_k, C_k) to these workers. Then, the server waits for τ time units for workers' responses. Upon receiving the broadcast message from the server, an honest worker i randomly samples a mini-batch of size s_i from its local data, and sends back to the server the first $b_i = \min\{\frac{\tau \lambda_i}{s_i}, d\}$ coordinates in C_k of its stochastic gradient $G_i(w_k)$ defined in (2) above. Note that a faster worker (with higher computational capabilities) may compute more coordinates of its stochastic gradient compared to a slower worker. However, a Byzantine worker may arbitrary values for its partial derivatives. After τ time units, the server identifies the set of indices $\mathcal{B}_k \subseteq C_k$ that have been computed by at least v workers out of the q selected

Algorithm 1 HGO: **H**eterogeneous **g**radient-based **O**ptimization

Parameter Server

1: **Input:** q, AR, w_1, v, T
2: **Execute the following instructions in each step** $k = 1, \ldots, T$
3: $C_k \longleftarrow$ random permutation of set $\{1, \ldots, d\}$
4: Select q random workers $\{i_1, \ldots, i_q\} \subseteq \{1, \ldots, n\}$
5: Broadcast (w_k, C_k) to the selected q workers
6: Wait τ time units to receive workers' gradients
7: $V_{i_1}, \ldots, V_{i_q} \longleftarrow$ partial derivatives sent by the q workers, i.e., elements of their gradients at w_k
8: $R \longleftarrow$ sort $\left(|V_{i_1}|, \ldots, |V_{i_q}|, \text{ decreasing order}\right)$
9: $b_v \longleftarrow v^{\text{th}}$ item in the set R
10: $\mathcal{B}_k \longleftarrow$ first b_v elements in C_k
11: $\forall j \in \mathcal{B}_k$, identify the set $\mathcal{Q}_j \subseteq \{i_1, \ldots, i_q\}$ of workers that sent the j-th coordinate of their gradients.
12: Compute the aggregate of workers' partial derivatives $AG(w_k) \in \mathbb{R}^d$ such that for all $j \in [d]$,

$$\{AG(w_k)\}_j = \begin{cases} \text{AR}\left(\{V_i\}_j,\ i \in \mathcal{Q}_j\right) & , \quad j \in \mathcal{B}_k \\ 0 & , \quad \text{otherwise} \end{cases}$$

where $\{V_i\}_j$ is sent by worker i for the j-th element of its gradient.
13: Update the parameter vector to $w_{k+1} = w_k - \gamma AG(w_k)$

Honest Worker i

1: **Input:** local dataset S_i, mini-batch size s_i, computation rate λ_i
2: **If** the broadcast message (w_k, C_k) is received **then** execute the following steps:
3: Sample a random mini-batch ζ_{i_k} of size s_i from dataset S_i
4: $\mathcal{B}_k^i \longleftarrow$ first b_i elements in C_k where $b_i = \min\{\frac{\tau\lambda_i}{s_i}, d\}$
5: Construct the set $V_i = \left(\{G_i(w_k\}_j, j \in \mathcal{B}_k^i\right)$ where $G_i(w_k)$ is as defined in (2)
6: Send back to the Server V_i

workers, described in steps 8, 9 and 10 of Algorithm 1. The server then applies a robust *aggregation rule* AR (defined above in Definition 1) on the consistent coordinates of the gradients sent by the workers, and uses the outputs to update its current parameter vector w_k.

Remark 2. *We do not force the workers to complete the whole training cycle, for this can last longer than the average battery can allow. Therefore, a worker can join the training, disconnect at timestamp k_1, then reconnect at timestamp k_2 (after a battery, network or hardware issue). The server keeps track of all the active devices in each iteration. The algorithm works as long as a quorum of workers are active at each iteration. The number of Byzantine workers that can be tolerated inside this quorum is dictated by the aggregation rule.*

Time complexity of HGO. We give now the total time complexity of the algorithm:

Proposition 1 (HGO time complexity)**.** *Let $\bar{\lambda}$ be the average computation rate and $\bar{s}$ the average mini-batch size of the workers. Let C_{pd} denote the cost of computing one partial derivative of a stochastic gradient. If* Trimmed Mean *is used as an aggregation rule, then the total time complexity of* HGO *running for T iterations is $\mathcal{O}\left(T\frac{\bar{\lambda}\tau}{\bar{s}}(C_{pd} + q\log(q))\right)$ on average, and $\Omega\left(Td(q^2 + C_{pd})\right)$ at worse.*

Proof. Let us consider the computations at the server level. At the worst case, the q random workers will all be powerful workers and thus able to compute the full gradients. Therefore, upon receiving the gradients, computing the cardinal of the received sets V_i's is $\Omega(qd)$. On average, this same step is done in $\mathcal{O}(qb)$, where $b = \frac{\bar{\lambda}\tau}{\bar{s}}$ is the average block size, given an average computation rate $\bar{\lambda}$ and an average mini-batch size $\bar{s}$. In both cases, the sorting is done on q values. Using *QuickSort*, the worst case complexity is $\Omega(q^2)$ and the average complexity is $\Omega(q\log(q))$. Finally, considering that applying *Trimmed Mean* on q values has an average complexity of $\Omega(q\log(g))$ and a worst complexity of $\Omega(q^2)$, the aggregation step of the algorithm will be in $\mathcal{O}(q\log(q)\frac{\bar{\lambda}\tau}{\bar{s}})$ on average and at worse, $\Omega(q^2 d)$. On the other hand, each worker i computes a number of partial derivatives of its stochastic gradient, depending on its computation rate λ_i, the amount of time available τ and the mini-batch size s_i. Let C_{pd} be the cost of computing one partial derivative of the stochastic gradient. Then, the average time complexity at the worker level is $\mathcal{O}(\frac{\bar{\lambda}\tau}{\bar{s}}C_{pd})$, and at worse $\Omega(dC_{pd}$. Thus, full time complexity of the algorithm at each iteration is $\mathcal{O}\left(\frac{\bar{\lambda}\tau}{\bar{s}}(C_{pd} + q\log(q))\right)$ and at worse $\Omega\left(d(q^2 + C_{pd})\right)$, which concludes the proof. □

IV. Theoretical Guarantees

Inspired by prior works in the homogeneous setting [4], [5], [15], [22], we show that HGO eventually (when $T \to \infty$) outputs an estimate w_k such that

$$\|\nabla \mathcal{L}(w_k)\|^2 \leq \frac{d\Delta(v,f)A_{w_k}}{b\left(1 - \Delta(v,f)B_{w_k}\right)}$$

where

$$A_w := \frac{1}{|C|}\sum_{j=1}^{d}\sum_{i\in\mathcal{C}}\left(\frac{M_w}{s_i'} + \delta_{i,j}^2\right),\ B_w := \frac{1}{|C|}\sqrt{\sum_{j=1}^{d}\left(\sum_{i\in\mathcal{C}}\frac{Q_w}{s_i'}\right)^2} \tag{4}$$

$$b := \lim_{T\to\infty}\min_{k=1,\ldots,T}|\mathcal{B}_k|\,,\ \text{and}\ s_i' := \frac{s_i(|S_i| - 1)}{|\mathcal{S}_i| - s_i}$$

Informally, we show that our algorithm eventually reaches a ball centered at a local optimum, whose radius is a function of the statistical error. Figure 2 below illustrates the trajectory of the algorithm in the context where there exists a unique optimal solution w^*.

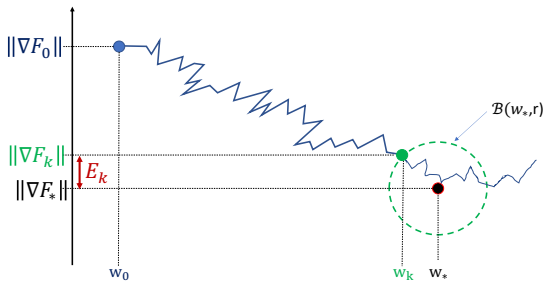


Fig. 2: As long as the the parameter vector w_k is inside $\mathcal{W}$, which means that the gradient norm is still big enough, the algorithm will follow a descent direction, towards a ball $\mathcal{B}(w_*, r)$ centred at a local optimum, with a radius r, which is a function of the statistical error. When the condition is no longer satisfied, the noise (induced by correct and Byzantine workers) will prevent progress towards the optimum.

We present below the formal convergence result of our algorithm. First, we divide the parameter space into two regions $\mathcal{W}$ and $\mathbb{R}^d \setminus \mathcal{W}$ such that

$$\mathcal{W} := \left(w \in \mathbb{R}^d;\ \min_{\mathcal{B}\subseteq[d]} \left\{ (1-\Delta B_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - A_w \right\} > 0 \right) \quad (5)$$

where $\Delta = \Delta(v, f)$.

Remark 3. *The condition* $(1-\Delta B_w) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 - \Delta A_w > 0$ *is a generalization of the* variance-norm ratio *used in the very first work addressing SGD in a Byzantine environment for ensuring* (α, f)*-Byzantine resilience of their proposed aggregation rule (*KRUM*), and thereby convergence of their algorithm in the presence of Byzantine devices [4]. They only considered the special case of SGD (i.e.,* $b = d$*) with unbiased estimation (i.e.,* $\forall (i,j) \in [N] \times [d], \delta_{i,j} = 0$*), uniform upper bound on the variance (i.e.,* $Q_w = 0$*), and a mini-batch of size unity (i.e.,* $\forall i \in [n], s_i = 1$*). In this special case, our generalized condition reduces exactly the same condition as in Proposition 1 of their work. The authors also claim that their condition can be satisfied when using larger mini-batch sizes. Our formulation proves this claim: increasing the mini-batch size* (s_i) *increases the value of the right-hand side of the condition inequality and decreases the left-hand side one, making it easier to achieve a satisfying outcome in many problems.*

A. Convergence analysis

We note that it is impossible to guarantee progress towards a local optimum in any step k if the direction of the aggregated gradient $AG(w_k)$ is opposite to the actual descent direction of the loss function necessary. Specifically, the scalar product between the aggregated gradient $AG(w_k)$ and the gradient $\nabla \mathcal{L}(w_k)$ must be positive. We show below in Lemma 1 that this condition holds true in our setting when $w_k \in \mathcal{W}$ and the aggregation rule AR is Δ-robust.

Lemma 1. *Suppose that Assumptions 3 and 4 hold true. Consider the* k*-th step of Algorithm 1. If* AR *is* $\Delta(v, f)$*-robust at* w_k *and* $w_k \in \mathcal{W}$ *then*

$$\mathbb{E}[\langle AG(w_k), [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \rangle] > \frac{(1-\Delta B_{w_k}) \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2 - \Delta A_{w_k}}{2}.$$

Proof. (Sketch) We ignore the subscript k and write $\Delta(v, f)$ simply as Δ. First, using Definition 1 and Assumptions 3 and 4, we derive an upper bound on the error of the aggregated vector $\mathbb{E} \left\| AG(w) - [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \leq r^2$ where $r^2 = \Delta \left(A_w + B_w \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \right)$. This implies that $\left\| \mathbb{E}[AG(w)] - [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2 \leq r^2$ (by Jensen's inequality). Thus, $\mathbb{E}[AG(w)]$ belongs to a ball centered at $[\nabla \mathcal{L}(w)]_{\mathcal{B}}$ with radius r. Then, we can show, using trigonometric reasoning that $\mathbb{E}[\langle AG(w), [\nabla \mathcal{L}(w_k]_{\mathcal{B}_k} \rangle] > (1 - \sin(\theta)) \left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|^2$ where $\sin(\theta) = \frac{r}{\left\| [\nabla \mathcal{L}(w)]_{\mathcal{B}} \right\|}$. The rest follows from algebraic calculations. □

Using the above result we obtain the convergence properties of HGO for strongly convex and non-convex functions. In the following theorem we show linear convergence of HGO under strong convexity. Under Assumption 1, for a block of coordinates $\mathcal{B}$ we define $L_{\mathcal{B}} = \sum_{j \in \mathcal{B}} L_j$. We let $\mathcal{L}^*$ denote the minimum of $\mathcal{L}(w)$ over the entire space $\mathbb{R}^d$.

Theorem 1 (Strongly convex)**.** *Suppose that Assumptions 1, 2, 3 and 4 hold true. Consider Algorithm 1 with a constant learning rate* $\gamma < \min_{k \in [T]} \left\{ \frac{1}{L_{\mathcal{B}_k}} \right\}$*. If* $\Delta B_{w_k} < 1$ *then*

$$\mathbb{E}\left[\mathcal{L}(w_T) - \mathcal{L}^*\right] \leq (1-\kappa)^{T-1} \left(\mathcal{L}(w_1) - \mathcal{L}^* - H\right) + H$$

where $\kappa = \frac{b}{d} \mu \gamma (1 - \Delta B)$, $b = \min_{k \in [T]} |\mathcal{B}_k|$, $A = \max_{k \in [T]} A_{w_k}$, $B = \max_{k \in [T]} B_{w_k}$, *and* $H = \frac{d \Delta A}{2b\mu(1-\Delta B)}$.

Proof. (Sketch) Under Assumption 1, for all $k \in [T-1]$,

$$\mathcal{L}(w_{k+1}) \leq \mathcal{L}(w_k) - \gamma_k \langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle + \gamma_k^2 \frac{L_{\mathcal{B}_k}}{2} \|AG(w_k)\|^2$$

Substituting $\|AG(w_k)\|^2 = \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} - AG(w_k) \right\|^2 + 2\langle [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}, AG(w_k) \rangle - \left\| [\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k} \right\|^2$ above and then involving Lemma 1 implies that

$$\mathbb{E}\left[\mathcal{L}(w_{k+1})\right] \leq \mathbb{E}\left[\mathcal{L}(w_k)\right] - \frac{b_k \gamma_k (1 - \Delta B_{w_k})}{2d} \mathbb{E}\left[\|\nabla \mathcal{L}(w_k)\|^2 \right] + \frac{\gamma_k \Delta A_{w_k}}{2}.$$

Subtracting $\mathcal{L}^*$ on both sides, and using Polyak-Lojasiewicz inequality (cf. Assumption 2) we obtain for all $k \in [T-1]$,

$$\mathbb{E}\left[\mathcal{L}(w_{k+1}) - \mathcal{L}^*\right] - H \leq (1-\kappa) \left(\mathbb{E}\left[\mathcal{L}(w_k) - \mathcal{L}^*\right] - H\right)$$

Using telescopic expansion above concludes the proof. □

As known from the literature, it is difficult to obtain convergence result for gradient-based algorithms (especially for CD [24], [25]) in non-convex settings to a global minima. Thus, we only focus on convergence to local minima, and obtain the following bound on the expected squared norms of the true gradients.

Theorem 2 (Non-convex)**.** *Suppose that Assumptions 1, 3 and 4 hold. If* $\Delta B_k < 1$ *and* $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$*, then we have:*

$$\min_{i\in[T]} \mathbb{E}\left[\|\nabla \mathcal{L}(w_i)\|^2\right] \leq \frac{2d(\mathcal{L}(w_1)-\mathcal{L}^*)}{Tb\gamma(1-\Delta B)} + \frac{d\Delta A}{b(1-\Delta B)}$$

where $A = \max_{k\in[T]} A_{w_k}$*,* $B = \max_{k\in[T]} B_{w_k}$*,* $\gamma = \min_{k\in[T]} \gamma_k$ *and* $b = \min_{k\in[T]} b_k$*.*

Proof. (Sketch) Using the same arguments from the previous proof, we can obtain the following inequality:

$$\mathbb{E}\,\mathcal{L}(w_{k+1}) \leq -X\,\|\nabla \mathcal{L}(w_k)\|^2 + Y$$

Introducing the total expectation, rearranging the terms and averaging the gradients through $(1,\ldots,T)$ gives:

$$\mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\|\nabla \mathcal{L}(w_i)\|^2\right] \leq \frac{\kappa}{T}(\mathcal{L}(w_1)-\mathcal{L}(w_*)) + H \quad (6)$$

Since $\min_{i\in[T]} \mathbb{E}\left[\|\nabla \mathcal{L}(w_i)\|^2\right] \leq \mathbb{E}\left[\frac{1}{T}\sum_{i=1}^{T}\|\nabla \mathcal{L}(w_i)\|^2\right]$, we obtain the desired result. □

About the learning rate. In the two theorems, we can see that increasing the learning rate γ improves the convergence rates. On the other hand, the learning rate γ_k at each iteration must satisfy the condition $\gamma_k < \frac{1}{L_{\mathcal{B}_k}}$, where $L_{\mathcal{B}_k}$ is the Lipschitz constant of the truncated gradient $[\nabla \mathcal{L}(w_k)]_{\mathcal{B}_k}$. Since the block $\mathcal{B}_k$ constructed by the server varies through iterations, this upper bound on the learning rate also fluctuates. In practice, choosing a small constant learning rate $\bar{\gamma}$ such that $\bar{\gamma} < \min_{k\in[T]} \frac{1}{L_{\mathcal{B}_k}}$ is a safe choice. However, to improve the convergence rate, one could try to reach this upper bound at each iteration by using a dynamic learning rate γ_k, inversely proportional to $\sqrt{|\mathcal{B}_k|}$.

Remark 4. *Our work can be seen as a generalization of previous works on Byzantine-resilient optimization, encapsulating a large family of descent algorithms (CD, SCD, Block CD, GD, SGD). As a matter of fact, many works can be summarized within our framework. It suffices to replace the variables in our result (depending on the set of assumptions, the algorithm and the defense mechanism) to get the convergence properties. For instance, in [24], the author studies Random Coordinate Descent (RCD), where a single random index (*$b = 1$*) is selected at each iteration, and the adequate coordinate is updated following a descent direction. In this simple algorithm, the descent direction is computed using a learning rate and a partial derivative, which is itself computed using the complete dataset. Since an exact non-distributed computation is executed (no estimation), there will be no variance (*$M_k = Q_k = 0$*), no aggregation (*$\Delta(q,f) = 0$*), no mini-batch (*$\forall i \in [n], s_i = 1$*) and no bias (*$\forall (i,j) \in [N]\times[d], \delta_{i,j} = 0$*), which means that* $A_w = B_w = 0$ *and* $H_k = 0$*. The author also uses a loose bound on the learning rate* $\gamma < \frac{1}{dL_{max}}$*. Replacing these values in Theorem 1 recreates the exact convergence theorem of RCD, presented in the second part of Theorem 1 in [24].*

B. Discussing the trade-offs

Our theoretical results formalize some trade-off relationships between the convergence properties, the behavior and the computation power of the workers, as well as the nature of the dataset's distribution. We discuss some of them in the following.

1) Convergence properties vs time complexity: The robustness coefficient $\Delta(q,f)$ introduced in Definition 1 and used in our results is a key factor of the aggregation error. The value of the robustness coefficient indeed depends on the quality of the aggregation rule we use, but also on the number of Byzantine workers within the selection of random workers. In fact, for a fixed estimation of the number f of Byzantine workers, increasing the number q of random workers selected at the beginning of each iteration increases the number of correct workers in this random set. This also means that the Byzantine impact is reduced by making $\Delta(q,f)$ smaller. This has a direct impact on the convergence rate as well as on the statistical error. As a matter of fact, one can see in both theorems that the first term of the right-hand side inequality decreases when $\Delta(v,f)$ decreases. This simply means that it takes less time for this term to decay, therefore implying a faster convergence rate. The statistical error is also positively correlated to this coefficient. However, as presented in Proposition 1, the time complexity of the algorithm also increases with q. In short, better convergence properties imply a bigger time complexity.

2) Heterogeneity vs convergence properties: From the theorems, we can see that increasing the mini-batch size s_i of the workers improves the statistical error as well as the convergence rate. On the other hand, these two convergence properties are also linked to the size b_k of the block $\mathcal{B}_k$ constructed by the server at iteration k. In fact, increasing $b = \min_k b_k$ also increases the convergence rate and reduces the statistical error. There are three ways to increase b. We can either decrease the minimum number of workers v executing the aggregation, but this quantity is lower-bounded by $\frac{f}{\alpha}$, where α is the ratio of Byzantine workers. We can also decrease the mini-batch sizes, so that every worker dedicates its resources to computing more blocks ($\lambda_i\tau = b_i s_i$); but we would rather increase the mini-batch sizes, because of their positive effect on the convergence properties. Finally, the server can only contact the powerful workers (the ones with large computation rates λ_i) in order to increase the block size and the mini-batch size, improving the convergence rate and the statistical error. However, doing this may exclude workers with valuable data from the learning procedure, which certainly leads to a worse statistical error. To sum up, improving the workers' hardware specifications also improves the convergence properties.

3) Byzantine tolerance vs hardware heterogeneity: The condition $\Delta B_k < 1$ was stated in the two theorems in order to guarantee convergence. This inequality formalizes the trade-off between the Byzantine tolerance (through the robustness coefficient) and the computation power profile of the workers (captured in the variable B_k). To tolerate a bigger Byzantine impact, B_k must decrease in order to follow a descent direction

at step k. Decreasing B_k implies either to decrease the size b_k of the block $\mathcal{B}_k$, or to increase the size of the mini-batches selected by the workers. This can be achieved by instructing the workers to favor the mini-batch size over the number of coordinates, when allocating the available computation resources. When the computation rates are high, it will be possible to increase both b_i and s_i, in order to both improve the convergence properties and satisfy the condition. In other words, a network full of weak devices cannot handle a strong Byzantine impact.

V. Evaluation

Stochastic gradient descent (SGD) is the optimization backbone when it comes to training ML models. Many variants have been explored to improve its convergence rate by modifying the update rule using momentum techniques or adaptive steps. However, these optimization algorithms are still used within the standard distributed learning architecture, which imposes the same workload on the workers, often leading to sub-optimal resource allocation and poor learning performance due to the presence of weak workers. HGO can be considered as a new distributed learning scheme, allowing computationally heterogeneous workers to participate in a training process according to their capabilities. In this section, we evaluate the performance of HGO against the standard learning scheme. Due to page limitation, we only present a few experiments that support our theoretical analysis. A description of the experimentation process and a complete set of experiments involving more ML models and datasets, as well as executions under Byzantine attacks are available in the extended version of this paper. An android implementation of our algorithm is also available to execute it on real smartphones. The source code of the computer and android implementation are available in the following GitHub repositories, respectively: https://anonymous.4open.science/r/HgO-5136 and https://anonymous.4open.science/r/HgOApp-CC8E

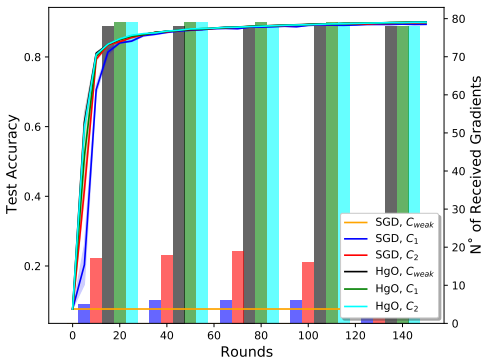
A. Environment setting

HGO can be used with any gradient-based solver by modifying the update rule in step 13 of Algorithm 1. Theoretical results were proven for SGD but can be extended to more advanced solvers with ease. We choose to work with SGD to eliminate the positive impact of advanced solvers and only appreciate the performance of HGO over the standard learning scheme. For clarity, we denote SGD under the standard learning scheme simply by SGD, and SGD under our new scheme by HGO.

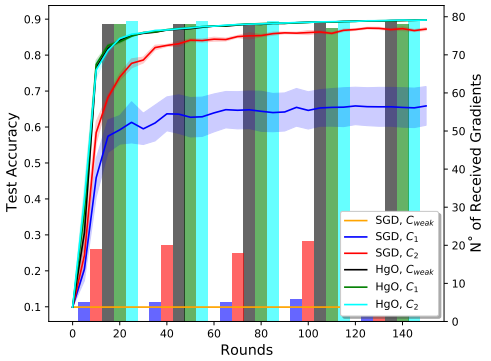
We train a binary and a multinomial logistic regression, as well as a feed forward neural network using the *MNIST* dataset. We consider workers with different computation rates and use the categories "weak", "average" and "powerful" to distinguish the hardware specifications of workers. Each category represents a set of workers whose computation rates are normally distributed over a predetermined mean value. In all the experiments, we assume that the total number of workers n contains 30%, 40% and 30% of weak, average and powerful devices, respectively. In one particular set of experiments, we construct scenarios where these percentages vary, starting from C_{weak} (where all the workers are weak devices) to $C_{powerful}$ (where all the workers are powerful). We consider both homogeneous and heterogeneous partitioning of the dataset over the workers.

B. Experiments

The main advantage of HGO is the fact that the server receives information from all the workers, at each iteration. Even if the worker is weak, it still manage to compute at least a few partial derivatives using a few data points, which is enough to give valuable information about its local dataset. Figure 3 illustrates this concept.



(a) Homogeneous dataset partitioning

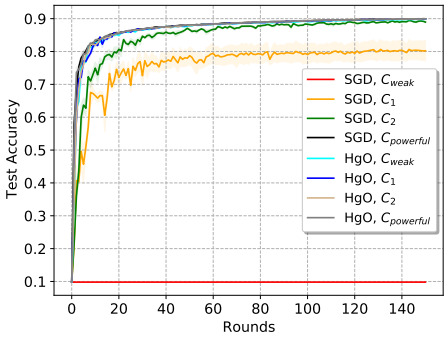


(b) Heterogeneous dataset partitioning

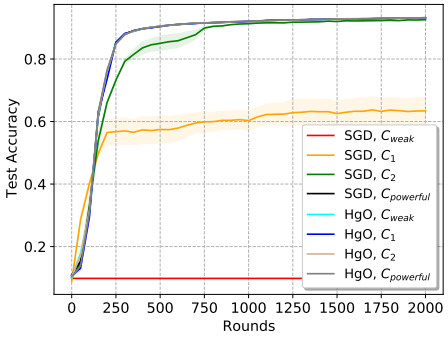
Fig. 3: Training a multinomial logistic regression on MNIST using $n = 100$ workers in an honest setting. The server averages the estimates from $q = 80$ random workers at each iteration. Using HGO, the server receives gradients (partial or full) from all the workers regardless of their computational profile. In contrast, the server only receives full gradients from strong workers when using SGD. HGO outperforms SGD on both final accuracy and convergence rate, especially for heterogeneous partitioning on the dataset.

Interestingly, using HGO allows reaching top performance, even with a network full of weak devices, which is impossible with standard learning schemes. In Figure 4, we can see that HGO shows nearly the same convergence properties, independently of the hardware profile of the workers, as opposed to

SGD where the convergence properties are strongly correlated with the capabilities of the workers. Note that the performance gap between HGO and SGD accentuates when moving from a simple ML model (multinomial logistic regression) to a more complex one (feed-forward neural network). We expect this gap to be even stronger from more complex models and datasets.



(a) Multinomial logistic regression



(b) Feed-forward neural network

Fig. 4: Training a multinomial logistic regression and a feed-forward neural network on MNIST, heterogeneously partitioned over $n = 100$ workers in an honest setting. The server averages the estimates from $q = 80$ random workers at each iteration. In both ML models, HGO shows a faster convergence rate and a better final accuracy than SGD, even in the weakest scenario C_{weak} where all workers have low hardware profiles.

While using HGO with a network full of weak workers shows good performance, we demonstrate that replacing only 10% of these weak devices by powerful workers substantially boost the performance and allows to reach similar convergence properties to a network full of powerful devices. Figure 5 illustrate this concept.

In the previous set of experiments, we plot the accuracy against the rounds which gives an idea on the statistical error as well as the convergence rate of each algorithm. In the next experiment (Figure 6) , we plot the accuracy against the actual wall-clock time, which gives a clear idea on the runtime of HGO, compared to SGD. When there is no time limit, SGD is slow because it is limited by the weakest device

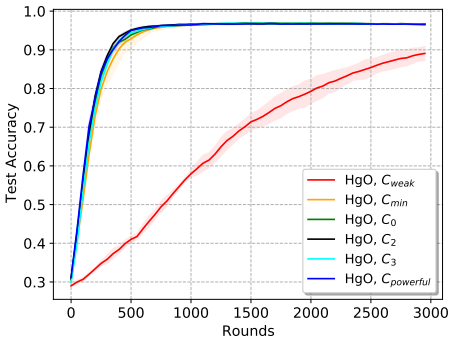


Fig. 5: Training a binary logistic regression on MNIST, with a total of $n = 50$ workers in an honest setting. The server selects $q = 40$ random workers at each iteration, and averages their estimates. We plot the test accuracy of HGO under different hardware profiles. C_{weak} is a scenario with only weak devices. C_{min} replace 10% of the devices in C_{weak} by powerful workers. C_1, C_2 and C_3 are different combinations, increasing the proportion of average and powerful workers, respectively. $C_{powerful}$ is a network with only powerful devices. Interestingly, introducing a network hardware profile equivalent to C_{min} is sufficient to attain convergence properties similar to $C_{powerful}$.

of the network. When the server imposes a time limit, SGD is indeed faster because it discards the weak devices (not able to compute full gradients) and only communicate with the fastest ones. However, doing so has a negative impact on the final accuracy, which is expected since the learning is completed without visiting the data held by the weak devices. On the other hand, HGO enjoys a fast runtime because of the time limit imposed by the server at each iteration, but also reaches the top accuracy because weak workers are still participating (with partial derivatives) in the training.

VI. Related Work

In this section, we discuss additional related works (that were not mentioned in the introduction).

Prior works in distributed ML consider the three critical issues of heterogeneous computational capabilities, heterogeneous data and Byzantine fault-tolerance separately. Many works have studied the Byzantine resilience of gradient-based descent optimization algorithms [4], [5], [7], [11], [12], [15], [21], [26] by leveraging robust statistics, filtering schemes or coding theory to defend against a fraction of Byzantine workers. However, they all assume unbiased estimation and i.i.d. local datasets. Particularly, [15] analyzed the Byzantine resilience of gradient descent coupled with trimmed mean and median, but the analysis only applies to a small number of Byzantine workers (very far from optimal) and, most importantly, it relies on strong assumptions on the distribution of the gradients (bounded skewness and sub-exponential

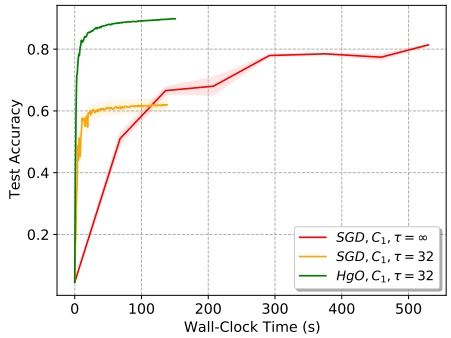


Fig. 6: Training a multinomial logistic regression on MNIST, heterogeneously partitioned over $n = 100$ workers in an honest setting. The server selects $q = 800$ random workers at each iteration, and averages their estimates. When $\tau = \infty$, the server must wait for the slowest worker to send its estimate. When $\tau = 32$, the server only waits for 32 units of time before the aggregation step. HGO outperforms SGD in both final accuracy and runtime.

distribution). A work closely related to ours is [26]: in the proposed synchronous coordinate descent algorithm, an encoding scheme is used to defend against Byzantine workers, which is not suitable for federated learning architectures, due to privacy concerns. To minimize overhead costs, the condition on the maximal number of Byzantine workers falls to $n > 3f$, which is not optimal. Besides, the encoding time is high, and depends on the dimension of the problem ($\mathcal{O}\left(d\left(\frac{\epsilon}{1+\epsilon}n+1\right)\right)$) as well as the quadratic cost of iteration at the PS ($\mathcal{O}(n^2)$).

On the other hand, some papers [13], [14], [16] studied SGD with local iterations on heterogeneous data, without assuming Byzantine workers. [18], [19] analyzed distributed SGD assuming data heterogeneity and Byzantine attacks. However, they use encoding schemes, which are not practical in our setup. Finally, [20] used clustering combined with robust statistics to transform the heterogeneous dataset into homogeneous clusters before the distributed optimization step. Besides the fact that this algorithm uses an extra step (clustering) to handle heterogeneity, which impacts its time complexity negatively, reassigning data to clusters rises some privacy concerns. A common modeling of heterogeneity in all these works is to assume a dissimilarity at the optimum between the local loss functions, gradients or parameter vectors. Most importantly, none of these works consider the heterogeneity of workers in terms of computation capabilities.

While compression schemes [27]–[30] are efficient in reducing the communication complexity, they may not provide a solution to computational heterogeneity. Moreover, prior works only considered computational heterogeneity through the window of asynchrony [10], [17], [21] rather than adapting the algorithm itself to the computation capabilities of workers. Particularly, the authors of [17] propose to handle the effect of stragglers in a heterogeneous network of workers, but using a smart learning rate schedule. However, the workers are still using the same optimization algorithm (SGD). Besides, Byzantine resilience and data heterogeneity are not analyzed.

In our paper, we study for the first time the simultaneous execution of a large family of gradient-based algorithms (CD, BCD, SCD, GD, SGD) under the supervision of a parameter server, using a generic defense mechanism. Our algorithm provably converges under a large set of constraints: Byzantine faults, stale and non-fully active workers, data heterogeneity and hardware heterogeneity. We also use a different approach to model data heterogeneity. As a matter of fact, we model all kinds of inaccuracies at the worker level (of which heterogeneity is an example) using a coordinate-wise bias on partial derivatives at each iteration, which is different from assuming dissimilarity at the optimum. Moreover, to our knowledge, we are the first to study the trade-offs between the convergence rate, the statistical error, the Byzantine resilience, the per-iteration cost and the time complexity of the algorithm.

VII. Conclusion

We present HGO, a new generic algorithm that can be equipped with any defense mechanism, and whose iteration cost can be tuned to match the capabilities of workers in a parameter server architecture (e.g., for machine learning on smartphones). We prove convergence for strongly convex as well as non-convex cost functions under a non trivial combination of hypotheses, namely: Byzantine failures, computational heterogeneity and data heterogeneity.We also demonstrate trade-offs between important quantities like the statistical error rate, the impact of Byzantine workers, the level of inaccuracy of honest workers, the time complexity of the algorithm and its convergence rate. We support our theoretical analysis empirically, by demonstrating the interplay between convergence properties and the execution environment of the algorithm. Interestingly, we show that introducing as low as 10% of powerful workers in a network full of weak devices, substantially improves the convergence properties of the algorithm. HGO also outperforms SGD in the case of non-identically distributed datasets: it aggregates the estimates of all workers, regardless of their computation capabilities. This contrasts with SGD, which imposes the same workload on all workers, and therefore discards some workers (which data may be very valuable to the training). On a more ethical side, our work encourages data privacy by making smartphones (which are the most important personal data carrier nowadays) contribute directly to AI projects, without exposing local data to third parties.

HGO only exchange a block of coordinates instead of the full gradient at each iteration, which makes it better than gradient descent and its variants, in terms of communication complexity. However, it is still considered as a network bottleneck in the case of smartphones connected to the internet. A better option would be to locally train for multiple steps, instead of one iteration at a time, before sending the parameter vector. Also, while the assumptions on the variance and the

bias of the estimation concerns each data point and each coordinate (and are therefore significantly stronger than the classical versions), it allowed us to integrate the mini-batch and the block analysis in our theoretical results. A set of more relaxed assumptions could perhaps lead to better results.

Can this work be extended to non-smooth functions, or to second-order optimization? What is the impact of the learning rate on the statistical error? Which computation power profile is needed to reach a target accuracy within a given time budget? Finally, what is the impact of network bandwidth on the convergence properties? We leave these questions open for future works.

REFERENCES

[1] M. Abadi *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.

[2] J. Konecný, H. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *ArXiv*, vol. abs/1610.02527, 2016.

[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://doi.org/10.1145/357172.357176

[4] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," ser. NeurIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 118–128.

[5] A. Boussetta, E.-M. El-Mhamdi, R. Guerraoui, A. Maurer, and S. Rouault, "AKSEL: Fast Byzantine SGD," in *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Q. Bramas, R. Oshman, and P. Romano, Eds., vol. 184. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, pp. 8:1–8:16. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/13493

[6] 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, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3521–3530. [Online]. Available: http://proceedings.mlr.press/v80/mhamdi18a.html

[7] D. Alistarh, Z. Allen-Zhu, and J. Li, "Byzantine stochastic gradient descent," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018, pp. 4613–4623. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/a07c2f3b3b907aaf8436a26c6d77f0a2-Paper.pdf

[8] C. Xie, O. Koyejo, and I. Gupta, "Generalized byzantine-tolerant sgd," 2018.

[9] ——, "Phocas: dimensional byzantine-resilient stochastic gradient descent," 2018.

[10] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for non convex optimization." *In NIPS, pages 2737–2745*, 2015.

[11] L. Chen, H. Wang, Z. Charles, and D. Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," 2018.

[12] Z. Yang and W. U. Bajwa, "Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 5, no. 4, pp. 611–627, 2019.

[13] A. Khaled, K. Mishchenko, and P. Richtarik, "Tighter theory for local sgd on identical and heterogeneous data," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 4519–4529. [Online]. Available: https://proceedings.mlr.press/v108/bayoumi20a.html

[14] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, "A unified theory of decentralized sgd with changing topology and local updates," in *ICML*, 2020, pp. 5381–5393. [Online]. Available: http://proceedings.mlr.press/v119/koloskova20a.html

[15] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," 2018.

[16] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJxNAnVtDS

[17] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 463–478. [Online]. Available: https://doi.org/10.1145/3035918.3035933

[18] D. Data, L. Song, and S. Diggavi, "Data encoding methods for byzantine-resilient distributed optimization," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2719–2723.

[19] S. Rajput, H. Wang, Z. B. Charles, and D. Papailiopoulos, "Detox: A redundancy-based framework for faster and more robust gradient aggregation," in *NeurIPS*, 2019.

[20] A. Ghosh, J. Hong, D. Yin, and K. Ramchandran, "Robust federated learning in a heterogeneous environment," 2019.

[21] G. Damaskinos, E. M. El Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous Byzantine machine learning (the case of SGD)," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1145–1154. [Online]. Available: http://proceedings.mlr.press/v80/damaskinos18a.html

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

[23] L. Nguyen, P. H. NGUYEN, M. van Dijk, P. Richtarik, K. Scheinberg, and M. Takac, "SGD and hogwild! Convergence without the bounded gradients assumption," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3750–3758. [Online]. Available: http://proceedings.mlr.press/v80/nguyen18c.html

[24] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, pp. 3–34, 2015.

[25] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 341–362, 2012.

[26] D. Data and S. Diggavi, "Byzantine-tolerant distributed coordinate descent," in *2019 IEEE International Symposium on Information Theory (ISIT)*, 2019, pp. 2724–2728.

[27] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," 2018.

[28] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Powersgd: Practical low-rank gradient compression for distributed optimization," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/d9fbed9da256e344c1fa46bb46c34c5f-Paper.pdf

[29] H. Sun, Y. Shao, J. Jiang, B. Cui, K. Lei, Y. Xu, and J. Wang, "Sparse gradient compression for distributed sgd," in *Database Systems for Advanced Applications*, G. Li, J. Yang, J. Gama, J. Natwichai, and Y. Tong, Eds. Cham: Springer International Publishing, 2019, pp. 139–155.

[30] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W