Arbitrarily Accurate Aggregation Scheme for Byzantine SGD

Alexandre Maurer
School of Computer Science, UM6P, Ben Guerir, Morocco

Abstract

A very common optimization technique in Machine Learning is Stochastic Gradient Descent (SGD). SGD can easily be distributed: several *workers* try to estimate the gradient of a loss function, and a central *parameter server* gathers these estimates. When all workers behave correctly, the more workers we have, the more accurate the gradient estimate is. We call this the Arbitrary Aggregation Accuracy (AAA) property.

However, in practice, some workers may be Byzantine (i.e., have an arbitrary behavior). Interestingly, when a fixed fraction of workers is assumed to be Byzantine (e.g. 20%), no existing aggregation scheme has the AAA property. In this paper, we propose the first aggregation scheme that has this property despite a fixed fraction of Byzantine workers (less than 50%). We theoretically prove this property, and then illustrate it with simulations.

2012 ACM Subject Classification Computing methodologies → Machine learning; Computing methodologies → Distributed algorithms

Keywords and phrases distributed machine learning, Byzantine failures, stochastic gradient descent

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2021.4

Supplementary Material *Software (Source Code)*: `https://tinyurl.com/sim-aaa-paper`

1 Introduction

Many machine learning models are trained using *stochastic gradient descent* (SGD) [17], an optimization technique that can easily be parallelized on multiple computers. As machine learning models become larger and larger, parallelizing their training becomes more and more important, if we want to train them in a reasonable amount of time.

If all computers are assumed to work correctly, parallelizing the training is relatively simple. The classical architecture is the following. A central *parameter server* is trying to minimize a *loss function*. To do so, it uses the gradient descent algorithm, which requires to compute (an approximation of) the gradient of the loss function, at several points of the loss function. As this is the most time-consuming task, the parameter server distributes this task among multiple *workers*. Each worker computes a vector which is an approximation of the desired gradient. The parameter server then collects and aggregates these vectors, to obtain a (reasonably good) approximation of the gradient. This process is repeated multiple times, until we reach a minimum of the loss function. We explain this in more details in Section 2.3.

However, when the number of workers becomes very large, one should assume that some workers will *not* behave correctly. Some workers may even be malicious agents trying to prevent a successful training of the model. This is especially true when workers are not identical computers stored in a data center, but personal computers or smartphones participating in the training in a collaborative way.

Therefore, a recent line of work is *robust distributed SGD*: the goal is to propose distributed architectures that manage to train the model despite the presence of malicious workers. In order to achieve very strong safety guarantees, we assume that these malicious workers are *Byzantine* [15], that is: they are omniscient, and can have any arbitrary behavior.

Let us give a quick overview of this literature.[1] In [18], a first solution was proposed for problems of dimension one. In [3], a score is associated to each proposed vector, defined as the sum of distances with some of its closest neighbors. The vector with the smallest score is then selected. In [9], several outputs of [3] are selected (without replacement), and then averaged. In [21], a scalar aggregation rule (SAR) is applied to each coordinate. This SAR can be, for instance, the median, or a trimmed mean (a mean after removing the $x\%$ largest and $x\%$ smallest values). In [19] (resp. [20]), the proposed values closest from the median (resp. trimmed mean) are selected, then averaged. In [12], a variant of the trimmed mean is applied to a selection of vectors obtained from [9]. In [7], several batches of vectors are averaged; then, their geometric median is computed. In [6], the vectors are aggregated using coding theory and a redundancy scheme. In [1] and [10], historical information is used to identify dishonest workers. The only solution tolerating asynchrony so far is [10].

Note that most of these works assume a centralized and reliable parameter server. However, as shown in [11], these schemes[2] can be transformed into fully decentralized schemes, where no entity is a single point of failure. In order to focus on the aggregation strategy, we also assume a centralized parameter server in this paper.

In the following, we focus on aggregation schemes where an approximation of the gradient is computed independently at each step, like in classical SGD. Aside from enabling a clearer mathematical analysis of the gradient, it also makes the system resilient to transient failures, that is: in addition of Byzantine workers, the system can recover from any temporary failures of correct workers.

The Arbitrary Aggregation Accuracy (AAA) property

Now, let us come back to the case where all workers are correct, and consider a given step of the gradient descent algorithm. In this setting, the parameter server simply computes the mean of the received vectors. If each workers processes a given share of the dataset, and these shares are independent and identically distributed (which is usually assumed), then the more workers we have, the more *accurate* our approximation of the true gradient will be. Actually, for any arbitrary level of precision, there exists a number of workers that can achieve this level of precision. We call this the *Arbitrary Aggregation Accuracy* (AAA) property.

Formally, if G is the true gradient of the loss function (at a given step), n is the number of workers, and A_n is the vector aggregated by the parameter server (the approximation of the true gradient), then this property can be expressed as follows:[3]

$$\lim_{n \to +\infty} \mathbb{E}\|A_n - G\| = 0.$$

We now assume that there is a fixed fraction of Byzantine workers (for instance, 20% of workers, independently of the total number of workers). We may ask the following question: is it possible to have the AAA property in this setting?

---

[1] This problem shares some similarities with the more general problem of executing arbitrary tasks in a setting with a "master" and several (unreliable) "workers". In [14], a bound is shown on the complexity of performing n tasks correctly with high probability. However, doing so does not give meaningful guarantees when the goal is to perform SGD.

[2] This applies to any scheme of the same (centralized) nature as those presented above.

[3] In this paper, $\|\cdot\|$ refers to the L2 norm, and the expectation is both on the i.i.d. samples of the dataset and on the randomness involved in the algorithm.

Interestingly, no existing Byzantine-resilient version of SGD has this property (more details on this in Section 2.5): a fixed fraction of Byzantine workers results in a fixed error w.r.t. the true gradient, no matter how large the number of workers is.

Our contribution

In this paper, we propose CompAss, the first aggregation scheme that has the AAA property despite a fixed fraction f of Byzantine workers ($f < \frac{1}{2}$). We describe this scheme, then prove that it has the AAA property. We then illustrate this property with simulations: we compare the accuracy of CompMed (a modified version of CompAss[4]) to an existing aggregation scheme.

The rest of the paper is organized as follows. In Section 2, we describe the general setting. In Section 3, we describe the CompAss aggregation scheme. In Section 4, we prove that CompAss has the AAA property. In Section 5, we illustrate the AAA property with simulations. We conclude in Section 6.

Remarks and clarifications

Before going further, let us clarify several points about the contribution of this paper.

- This work is mostly a theoretical work.
- This work, as well as many previous works, proposes a scheme to approximate the gradient of the loss function. The precision of this approximation (of the gradient) should not be confused with the precision of the learned model. For a given gradient descent step, having an accurate gradient is always a desirable property, since the goal of a gradient descent step is to decrease the value of the loss function. Therefore, an accurate gradient approximation will not be the cause of problems like overfitting.
- Similarly, guarantees on the quality of the gradient approximation (as provided in this work and several previous works) should not be confused with guarantees on the accuracy of the trained model. Such guarantees can be found, for instance, in [1] and [5], under specific hypotheses (e.g., convex loss function, bounded gradient...).
- The gradient approximations proposed by correct workers may have a variance high enough to allow Byzantine workers to collude to move the mean without being detected (as described in [2]). However, they can only do so in aggrergation schemes where the AAA property is not satisfied: this property ensures that the estimated gradient is arbitrarily close to the true gradient, independently of the behavior of Byzantine workers.
- In the SGD algorithm, there is always a probability of error w.r.t. the true gradient. It is also the case for distributed SGD (without failures), and for Byzantine-resilient solutions (including ours). Nevertheless, we ensure that this error shrinks to zero when the number of nodes increases.

2 Preliminaries

2.1 Setting

We want to train a machine learning model of d parameters, using the gradient descent algorithm.

[4] The reason for this change is motivated in Section 5.1.

The model can be represented by a function $\mathcal{M}(P, X)$, where $P = (p_1, \ldots, p_d)$ are the *parameters* of the model (for instance, the weights and biases of a neural network), and X is the current input of the model (usually a vector of real values). $\mathcal{M}(P, X)$ returns a single real value y.

To train the model, we have a dataset consisting in two lists $(X_1, \ldots, X_m)$ and $(y_1, \ldots, y_m)$, where X_i is an input of the model (*feature*), and y_i is the corresponding desired output (*label*). In the following, we denote y_i by $y(X_i)$. We define a *loss function* $L(P)$, measuring the "distance" between the current model and the desired outputs. For instance, a classic form of the loss function is:

$$L(P) = \frac{1}{m} \sum_{i=1}^{m} \left(\mathcal{M}(P, X_i) - y(X_i)\right)^2.$$

We make no hypotheses on the shape of this function, except that it has, in each point, a gradient with finite coordinate values.[5]

Training the model consists in finding a set of parameters minimizing the loss function. The standard way to do this is to use the *gradient descent* algorithm, which consists of repeating the two following steps:

1. Compute the gradient $\nabla L(P)$ of the loss function.
2. Update the vector of parameters P as follows: $P \leftarrow P - \alpha \nabla L(P)$ (where α is an arbitrarily small constant).

2.2 Stochastic gradient descent (SGD)

In practice, computing the exact gradient may be very long when the dataset contains a lot of elements (which is usually the case). An alternative is to use *stochastic gradient descent* (SGD), that is: at each step, we randomly select a set S of elements from the dataset, and use it to compute an approximation $\nabla L^*(P, S)$ of $\nabla L(P)$. For instance, if $L(P)$ has the aforementioned classical form, then:

$$L^*(P, S) = \frac{1}{|S|} \sum_{X \in S} \left(\mathcal{M}(P, X) - y(X)\right)^2.$$

Over several steps, the errors due to randomness tend to cancel each other, and we generally achieve the same result with a much shorter computation time.

2.3 Distributed SGD

A convenient property of SGD is that it can easily be parallelized. The classical architecture is the following. We have a *parameter server*, that stores and updates the parameters of the model, and n *workers* $(w_1, \ldots, w_n)$.

Let $\alpha > 0$ be an arbitrarily small constant. At each step:

1. The parameter server sends the current vector of parameters P to each worker.
2. Each worker w_i selects a random subset S of the dataset, computes $V_i = \nabla L^*(P, S)$, and sends it to the parameter server.
3. The parameter server computes the mean A_n of the received vectors V_i ($A_n = \frac{1}{n} \sum_{i=1}^{n} V_i$), and uses it to update the model ($P \leftarrow P - \alpha A_n$).

[5] This function may have multiple local minima. However, for most machine learning models (e.g. neural networks), most local minima are sufficient to reach a satisfying accuracy [8].

Here, we assume that each worker possesses a copy of the whole dataset, and can randomly select elements from the dataset at each step. This is a reasonable hypothesis, given the current memory capacities of computers (or even smartphones), and the cheap cost of memory units (relatively to the cost of computing power). We make this hypothesis in the rest of the paper. Another justification might be that workers have remote access to the dataset through the internet (and may copy specific parts of it).

2.4 Failure model

In the aforementioned distributed setting, all workers are assumed to behave correctly. However, in practice, this may not always be the case.

Let $f < 0.5$ be a fixed parameter of the problem. Let k be the largest integer such that $k \leq fn$. Among the n workers, k are assumed to be *Byzantine*, that is: their behavior is completely arbitrary. Here, as the workers send vectors to the parameter server, this means that up to k workers can send arbitrary vectors to the parameter server. The parameter server does not know which workers are Byzantine.

Note that, since the behavior of Byzantine workers is arbitrary, it does not matter whether or not they keep track of past events: we must always assume the worst-case scenario.

2.5 The Arbitrary Aggregation Accuracy (AAA) property

An *aggregation scheme* is a distributed system that, for a given step of the gradient descent algorithm, produces an approximation A of the true gradient $G = \nabla L(P)$ of the loss function. The scheme presented in 2.3 is an example of aggregation scheme. If such a system allows an arbitrary number n of workers, we call the resulting aggregated vector A_n.

We say that an aggregation scheme has the Arbitrary Aggregation Accuracy (AAA) property if the expected value of the distance between A_n and G approaches 0 when n increases:

$$\lim_{n \to +\infty} \mathbb{E}\|A_n - G\| = 0.$$

A classical metric for Byzantine-resilient versions of SGD is the *angular error*, that is: the angle θ_n between $E[A_n]$ and G. An asymptotic comparison of the angular errors of existing aggregation schemes is provided in [4]. When the fraction of Byzantine workers is constant (independently of n), these angular errors are at best $\Theta(1)$ (i.e. constant). This contradicts the AAA property: when this property is satisfied, $lim_{n \to +\infty} \theta_n = 0$.

To give some intuition on why these previous solutions do not satisfy the AAA property, let us consider, for instance, the coordinate-wise median: for each coordinate, we take the median of the values proposed by workers (see Section 5.1 for a more formal description). In this setting, the worst thing Byzantine workers can do is to propose extreme values (all positive or all negative): even if they are a minority, this will push the median towards these extreme values. Here, for a given distribution of values, a fixed fraction of Byzantine workers will have a fixed impact of the median value. This phenomena is illustrated experimentally in Section 5.

3 Our aggregation scheme

In this section, we present the Compass aggregation scheme. In 3.1, we give some preliminary definitions. In 3.2, we describe Compass. In 3.3, we explain its general idea, and comment it step by step.

3.1 Definitions

We consider n workers $(w_1, \ldots, w_n)$. Let N be the largest integer such that $N^2 \leq n$. Let M be a fixed parameter, corresponding to the number of elements of the dataset that a worker uses to compute an approximation of the gradient at each step.

We now define several notions and functions used in our aggregation scheme:

- A *random split* consists in randomly selecting a set S of N^2 workers among $(w_1, \ldots, w_n)$, then randomly splitting the elements of S into N sets $(W_1, \ldots, W_N)$, each one containing N elements.
- A *random pick* is a set of M randomly selected integers between 1 and m (as a reminder, m is the size of the dataset).
- For a set S of vectors of dimension d, we define the function $Maj(S)$ as follows:
  - If there exists a vector V of S such that a strict majority of vectors of S are equal to V, then, $Maj(S) = V$.
  - Otherwise, $Maj(S)$ returns a null vector $(0, 0, \ldots, 0)$ of dimension d.
- For two values p and x (with $x \geq 0$), we define $Cut_0(p, x)$ as follows:
  - If $p > x$, $Cut_0(p, x) = x$
  - If $p < -x$, $Cut_0(p, x) = -x$
  - Otherwise, $Cut_0(p, x) = p$
- For a vector $V = (v_1, v_2, \ldots, v_d)$ and a value x, we define $Cut(V, x)$ as follows:

$$Cut(V, x) = (Cut_0(v_1, x), Cut_0(v_2, x), \ldots, Cut_0(v_d, x)).$$

3.2 Description of the aggregation scheme

We now describe the Compass aggregation scheme (similarly to the distributed SGD scheme described in 2.3).

Let $\alpha > 0$ be an arbitrarily small constant. At each step:

1. The parameter server generates a random split $(W_1, \ldots, W_N)$ and N random picks $(Z_1, \ldots, Z_N)$.
2. $\forall i \in \{1, \ldots, N\}$, the parameter server sends Z_i and the current vector of parameters P to each worker of the set W_i.
3. $\forall i \in \{1, \ldots, N\}$, let Ω_i be the set containing the elements X_j of the dataset such that $j \in Z_i$. Each worker of the set W_i computes $\nabla L^*(P, \Omega_i)$, and sends it to the parameter server.
4. $\forall i \in \{1, \ldots, N\}$, let S_i be the set of vectors sent by the workers of W_i (the parameter server only accepts one vector per worker[6]). The parameter server aggregates the received vectors as follows:

$$A_n = Cut\left(\frac{1}{N}\sum_{i=1}^{N} Maj(S_i), \sqrt{N}\right)$$

... and uses it to update the model $(P \leftarrow P - \alpha A_n)$.

[6] If a worker does not send any vector before the end of the round, we consider that it sent a null vector $(0, 0, \ldots, 0)$. Therefore, $|S_i| = |W_i|$.

3.3 Detailed explanation

General idea

A common strategy to defeat Byzantine processes is replication: many processes perform the same computing task, and a majority vote selects the correct output. Here, however, if all correct workers compute the same vector, adding more workers will not improve the quality of the gradient approximation. To do so, we have to *aggregate* many (independent) approximations. This is done, for instance, in the simple scheme described in Section 2.3 (with a mean). However, this scheme is not robust to even one Byzantine worker (as shown in [3]).

Here, we propose a balanced mix of replication and aggregation: if we have N^2 workers ($N \approx \sqrt{n}$), then, we can choose N sets of N workers each. Each set computes the same vector, and a majority vote determines the output of this set. We then aggregate these outputs. As N grows with n, increasing the number of workers increases *both* the reliability of replication *and* the quality of aggregation.

Step-by-step description

Now, let us comment on our aggregation scheme step by step.

In Step 1, the parameter server generates the N aforementioned sets $(W_1, \dots, W_N)$. To prevent any strategic placement of Byzantine workers, these sets are chosen randomly at each step. The parameter server also generates N random picks $(Z_1, \dots, Z_N)$. Each Z_i is a set of identifiers of elements of the dataset. These elements will be processed by the corresponding group of workers W_i.

In Step 2, the parameter server sends P (the current vector of parameters) and Z_i (the aforementioned set of identifiers) to each worker of the set W_i, for each $i \in \{1, \dots, N\}$.

In Step 3, each worker of the set W_i (for each $i \in \{1, \dots, N\}$) computes the approximation $\nabla L^*(P, \Omega_i)$ of the gradient $\nabla L(P)$, where Ω_i is the set of elements of the dataset corresponding to Z_i. Then, it sends it back to the parameter server.

In Step 4, the parameter server aggregates the received vectors. First, it uses a majority vote ($Maj(S_i)$) to determine the output of the set W_i (for each $i \in \{1, \dots, N\}$). Then, it computes the mean of these outputs. Finally, it applies the Cut function to ensure that the coordinates of the aggregated vector remain smaller than $\sqrt{N}$. Doing so is important, because there is always a probability $\mu > 0$ that a set W_i contains a majority of Byzantine workers. If so, the output of this set (after the majority vote) will be determined by Byzantine workers, that could (for instance) propose a vector with coordinate values inversely proportional to μ, to ensure that the expected mean of outputs remains far away from the true gradient. Applying the Cut function enables to prove the result of Lemma 3 (see Section 4), and then the AAA property.

4 Analysis

In this section, we prove that Compass has the AAA property (see Theorem 4).

In the following proofs, we introduce several variables in order to constrain some values to be integers. For instance: "Let k be the largest integer such that $k < n^2$". Some readers may consider that it would be simpler to just write "$\sqrt{n}$" here; however, some other readers may object that doing so would make the proofs less rigorous, and make their validity unclear. For this reason, we choose to use the first notation.

▶ **Lemma 1.** *Let $p < \frac{1}{2}$ be a probability. Consider N sets of N workers, where each worker has an independent probability p to be Byzantine. Let E_N be the following event: "All N sets contain a strict minority of Byzantine workers". Then, there exists N_0 such that, $\forall N \geq N_0$, $P(E_N) \geq 1 - \frac{2}{N}$.*

Proof. As $\lim_{x \to +\infty} \frac{\ln x}{x} = 0$, let N_0 be the smallest integer such that, $\forall N \geq N_0$:

$$\frac{\ln N_0}{N_0} < \left(\frac{1-2p}{4}\right)^2.$$

Consider a set of workers. Let k be the number of Byzantine workers in this set. According to Hoeffding's inequality:

$$P(|k - Np| \geq \sqrt{N \ln N}) \leq \frac{2}{N^2}.$$

Therefore:

$$P\left(\left|\frac{k}{N} - p\right| < \sqrt{\frac{\ln N}{N}}\right) \geq 1 - \frac{2}{N^2}$$

... and, for $N \geq N_0$:

$$P\left(\left|\frac{k}{N} - p\right| < \frac{1-2p}{4}\right) \geq 1 - \frac{2}{N^2}.$$

Now, we can remark that:

$$P\left(\left|\frac{k}{N} - p\right| < \frac{1-2p}{4}\right) \leq P\left(\frac{k}{N} - p < \frac{1-2p}{4}\right) = P\left(\frac{k}{N} < \frac{1+2p}{4}\right).$$

As $p < \frac{1}{2}$, we have $1 + 2p < 2$, and:

$$P\left(\frac{k}{N} < \frac{1+2p}{4}\right) \leq P\left(\frac{k}{N} < \frac{2}{4}\right) = P\left(k < \frac{N}{2}\right).$$

Thus, $\forall N \geq N_0$:

$$P\left(k < \frac{N}{2}\right) \geq 1 - \frac{2}{N^2}$$

and

$$P(E_N) \geq \left(1 - \frac{2}{N^2}\right)^N \geq 1 - \frac{2N}{N^2} = 1 - \frac{2}{N}.$$ ◀

▶ **Lemma 2.** *Let $N \geq 1$ be an integer, and let $f < \frac{1}{2}$ be a positive value. Consider N^2 workers, among which k are Byzantine, with $k \leq fN^2$. Assume that these N^2 workers are randomly assigned to N sets. Let E'_N be the following event: "All N sets contain a strict minority of Byzantine workers". Then, there exists N_1 such that, $\forall N \geq N_1$, $P(E'_N) \geq 1 - \frac{3}{N}$.*

Proof. The proof of this lemma can be found in the appendix. ◀

▶ **Lemma 3.** *Let G be a vector of dimension d, and let $p < \frac{3}{N}$ be a probability. Let $(V_1, V_2, V_3, \dots)$ be a sequence of vectors of dimension d, such that $\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} V_i = G$.*[7] *Let $(B_1, B_2, B_3, \dots)$ be an arbitrary sequence of vectors of dimension d. Let R_N be a random vector defined as follows: with probability p, $R_N = Cut\left(B_N, \sqrt{N}\right)$; otherwise, $R_N = Cut\left(\frac{1}{N}\sum_{i=1}^{N} V_i, \sqrt{N}\right)$. Then, $\lim_{N\to+\infty} \mathbb{E}\|R_N - G\| = 0$.*

Proof. Before going further, let us clarify one possible misunderstanding. Some readers may confuse N with the number of parameters of the model (which is not the case). According to the lemma's statement, N is related to the numbers of vectors used to approximate the true gradient. The number of parameters of the model is not used in the proofs of this paper.

$$\mathbb{E}\|R_N - G\| = \underbrace{p\left\|Cut\left(B_N, \sqrt{N}\right) - G\right\|}_{X_N} + \underbrace{(1-p)\left\|Cut\left(\frac{1}{N}\sum_{i=1}^{N} V_i, \sqrt{N}\right) - G\right\|}_{Y_N}$$

By definition of the Cut function, $\forall N \geq 1$, $\left\|Cut\left(B_N, \sqrt{N}\right)\right\| \leq \sqrt{d}\sqrt{N}$.

As $p < \frac{3}{N}$:

$$p\left\|Cut\left(B_N, \sqrt{N}\right)\right\| < \frac{3}{N}\sqrt{d}\sqrt{N} = \frac{3\sqrt{d}}{\sqrt{N}}.$$

Therefore:

$$X_N \leq p\left\|Cut\left(B_N, \sqrt{N}\right)\right\| + p\|G\| < \frac{3\sqrt{d}}{\sqrt{N}} + \frac{3}{N}\|G\|.$$

As a result, $\lim_{N\to+\infty} X_N = 0$. Now, let us determine $\lim_{N\to+\infty} Y_N$.

Let $\delta > 0$. Let $j \in \{1, \dots, d\}$, and let $v(i,j)$ (resp. $g(j)$) be the j^{th} coordinate of V_i (resp. G). As $\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} V_i = G$, in particular:

$$\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} v(i,j) = g(j).$$

Thus, there exists n_j such that, $\forall N \geq n_j$:

$$\left|g(j) - \frac{1}{N}\sum_{i=1}^{N} v(i,j)\right| \leq \delta.$$

Thus, $\forall N \geq n_j$:

$$\left|\frac{1}{N}\sum_{i=1}^{N} v(i,j)\right| \leq \delta + |g(j)|.$$

Let N_0 be the smallest integer such that $N_0 \geq \max(n_1, \dots, n_d)$ and $\sqrt{N_0} \geq \delta + \max_{j\in\{1,\dots,d\}} |g(j)|$.

Then, $\forall N \geq N_0$ and $\forall j \in \{1, \dots, d\}$:

$$\left|\frac{1}{N}\sum_{i=1}^{N} v(i,j)\right| \leq \sqrt{N}$$

[7] This sequence represents the vectors proposed by Byzantine workers. The reason why we write them as a sequence is that we further write "$lim_{N\to+\infty} f_N$", where f_N is a function of B_N.

and

$$Cut_0\left(\frac{1}{N}\sum_{i=1}^{N} v(i,j), \sqrt{N}\right) = \frac{1}{N}\sum_{i=1}^{N} v(i,j).$$

Thus, forall $N \geq N_0$:

$$Cut\left(\frac{1}{N}\sum_{i=1}^{N} V_i, \sqrt{N}\right) = \frac{1}{N}\sum_{i=1}^{N} V_i.$$

As a result:

$$\lim_{N\to+\infty} Cut\left(\frac{1}{N}\sum_{i=1}^{N} V_i, \sqrt{N}\right) = \lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} V_i = G$$

and

$$\lim_{N\to+\infty} \left\| Cut\left(\frac{1}{N}\sum_{i=1}^{N} V_i, \sqrt{N}\right) - G \right\| = 0.$$

As $p < \frac{3}{N}$, $\lim_{N\to+\infty}(1-p) = 1$, and $\lim_{N\to+\infty} Y_N = 0$. Therefore:

$$\lim_{N\to+\infty} \mathbb{E}\|R_N - G\| = \lim_{N\to+\infty} X_N + \lim_{N\to+\infty} Y_N = 0. \quad \blacktriangleleft$$

▶ **Theorem 4.** *Compass has the AAA property.*

Proof. As a reminder, N is the largest integer such that $N^2 \leq n$.

Let $(W_1, \ldots, W_N)$ be the random split generated in Step 1 of Compass. According to Lemma 2, with a probability at least $1 - \frac{3}{N}$, each set W_i contains a strict minority of Byzantine workers. In other words, the probability p that these sets do *not* all contain a strict minority of Byzantine workers is such that $p < \frac{3}{N}$.

Besides, when all sets W_i contain a strict minority of Byzantine workers, $Maj(S_i)$ corresponds to the vector sent by the correct workers of W_i (as a reminder, S_i is the set of vectors sent by the workers of W_i). As these vectors $Maj(S_i)$ are all based on random samples of the dataset, $\mathbb{E}[Maj(S_i)] = G$. In other words:

$$\lim_{N\to+\infty} \frac{1}{N}\sum_{i=1}^{N} Maj(S_i) = G.$$

Therefore, the output A_n of Compass can be represented by the random vector R_N of Lemma 3 (where the arbitrary vectors $(B_1, B_2, B_3, \ldots)$ correspond to the cases where not *all* sets W_i contain a strict minority of Byzantine workers).

When $n \to +\infty$, $N \to +\infty$. Therefore, according to Lemma 3, $\lim_{n\to+\infty} \mathbb{E}\|A_n - G\| = 0$, and Compass has the AAA property. ◀

5 Simulations

In this section, we illustrate the AAA property with simulations. We compare CompMed (a modified version of Compass) with an existing aggregation scheme. In 5.1, we describe these two aggregation schemes. In 5.2, we describe the simulation setting. In 5.3, we show how to make simulations both simpler and more general. The simulation results are presented in 5.4.

5.1 Aggregation schemes

Let us describe the aggregation schemes CWMed and CompMed.

- CWMed (*Coordinate-Wise Median*) is an aggregation scheme introduced in [21]. It consists in taking the median value for each coordinate, in order to exclude extreme values proposed by Byzantine workers. Its angular error is constant. As a reminder (see 2.5), among existing aggregation schemes, the angular error is at best constant.
- CompMed is a modified version of Compass. The principle is the same, except that the final aggregation formula is now similar to CWMed. The reason for this change is that Compass is designed to prove a very general result (for any distribution of values), but may be slow to converge in practice. For these simulations, we assume that the coordinates values proposed by correct workers follow a normal distribution (see 5.2). In this setting, CompMed converges much more quickly.[8]

Description of CWMed

We first define the function Med.

Let L be a list of n values. Let $(x_1, \dots, x_n)$ be a list containing the same values as L, but sorted in increasing order. We define the function $med(L)$ as follows:

- If n is even, $med(L) = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2}$.
- If n is odd, $med(L) = x_{\frac{n+1}{2}}$.

Let $(V_1, \dots, V_n)$ be n vectors. Let $v(i,j)$ be the j^{th} coordinate of V_i. Let $C_j = (v(1,j), v(2,j), \dots, v(n,j))$.

We define $Med(V_1, V_2, \dots, V_n)$ as follows:

$$Med(V_1, V_2, \dots, V_n) = (med(C_1), med(C_2), \dots, med(C_d)).$$

We now describe the CWMed aggregation scheme.

Let $\alpha > 0$ be an arbitrarily small constant. At each step:

1. The parameter server generates n random picks $(Z_1, \dots, Z_N)$.
2. $\forall i \in \{1, \dots, n\}$, the parameter server sends Z_i and the current vector of parameters P to worker w_i.
3. $\forall i \in \{1, \dots, n\}$, let Ω_i be the set containing the elements X_j of the dataset such that $j \in Z_i$. Each worker w_i computes $\nabla L^*(P, \Omega_i)$, and sends it to the parameter server.
4. $\forall i \in \{1, \dots, n\}$, let V_i be the vector sent by worker w_i.[9] The parameter server aggregates the received vectors as follows:

[8] This is due to the fact that Compass computes a mean of several vectors, some of which being potentially Byzantine. Therefore, the size N of the groups of workers must be large enough to ensure that all these vectors are correct with a very high probability (the *Cut* function takes care of the extremely unlikely bad cases).
Here, we assume that the coordinate values proposed by correct workers follow a normal distribution, which means that their expected median value is equal to their expected mean value. Therefore, we can use CWMed, which also excludes extreme values. However, in the general case, the expected median value of a distribution is not always equal to its expected mean value. This is why we used Compass to prove the main theoretical result.
Note that this problem (of the expected median value now always being equal to the expected mean value) is a theoretical limitation of both CWMed and CompMed. Therefore, the comparison we make here is fair with regards to this particular aspect.

[9] If a worker does not send any vector before the end of the round, we consider that it sent a null vector $(0, 0, \dots, 0)$.

$$A_n = Med(V_1, V_2, \ldots, V_n)$$

... and uses it to update the model ($P \leftarrow P - \alpha A_n$).

Description of CompMed

The CompMed aggregation scheme is defined similarly to the Compass aggregation scheme, except that the aggregated vector is now defined as follows:

$$A_n = Med(Maj(S_1), Maj(S_2), \ldots, Maj(S_n)).$$

5.2 Simulation setting

Let $\sigma > 0$ be a positive constants. Let $G = (g_1, \ldots, g_d) = \nabla L(P)$ be the gradient of the loss function.

Let Ω^* be a set of M random elements of the dataset. We assume that $L^*(P, \Omega^*) = (g_1^*, g_2^*, \ldots, g_d^*)$ (i.e., the approximation of the gradient that each correct worker computes) follows a normal distribution centered on the true gradient, that is: $\forall j \in \{1, \ldots, d\}$, g_j follows the normal distribution $\mathcal{N}(g_j, \sigma^2)$. This assumption is backed by recent results in machine learning [13]: many normally distributed datasets result in normally distributed gradients.

Aggregation error

To measure the quality of an aggregation scheme A_n (for a given number of workers n), we define the *aggregation error* λ_n as follows:

$$\lambda_n = \frac{\mathbb{E}[\|A_n - G\|^2]}{d}.$$

This quantity measures the average distance between A_n and G, with regards to the randomness of our model. Dividing by the dimension d (which is a constant of the problem) enables to significantly simplify the simulations, as shown in Section 5.3.

Attack model

Let $f < \frac{1}{2}$ be the fraction of Byzantine workers. We assume that all Byzantine workers send the vector $V_B = (\omega, \omega, \ldots, \omega)$ to the parameter server, where ω is an arbitrarily large positive constant.

For CWMed, this attack has a maximal impact: it "pushes" the median values of coordinates as far a possible from the value they would have had otherwise. The same is true for CompMed: if some groups of workers contain a majority of Byzantine workers, their output will be V_B.

5.3 Making the simulations simpler and more general

Let us show that, for CWMed and CompMed, the aggregation error λ_n can actually be computed without choosing specific values for d and $(g_1, \ldots, g_d)$. Besides simplifying the simulations, this makes the simulation results more general (i.e., not dependent on d and $(g_1, \ldots, g_d)$). Therefore, the only parameters of the simulations (defined above) are: σ, f and ω.

In the following, we explain how to compute two metrics β_n and γ_n, that do not depend on d or $(g_1, \dots, g_d)$. Then, in Theorem 5 and 6, we show that $\lambda_n = \beta_n$ (for CWMed) and $\lambda_n = \gamma_n$ (for CompMed).

Definition of β_n

Let k be the largest integer such that $k \leq fn$. Let $L = (y_1, \dots, y_n)$ be a list of n values, such that:

- $\forall i \in \{1, \dots, n-k\}$, y_i is a random value following the normal distribution $\mathcal{N}(0, \sigma^2)$;
- $\forall i \in \{n-k+1, \dots, n\}$, $y_i = \omega$.

We define β_n as follows: $\beta_n = \mathbb{E}[med(L)^2]$.

Definition of γ_n

For a given step of CompMed, among the N sets of workers $(W_1, \dots, W_N)$, let K be the number of sets that do *not* contain a strict majority of correct workers.

Let $L' = (y_1, \dots, y_N)$ be a list of N values, such that:

- $\forall i \in \{1, \dots, N-K\}$, y_i is a random value following the normal distribution $\mathcal{N}(0, \sigma^2)$;
- $\forall i \in \{N-K+1, \dots, N\}$, $y_i = \omega$.

We define γ_n as follows: $\gamma_n = \mathbb{E}[med(L')^2]$.[10]

▶ **Theorem 5.** *For CWMed, $\lambda_n = \beta_n$.*

▶ **Theorem 6.** *For CompMed, $\lambda_n = \gamma_n$.*

The proofs of Theorem 5 and Theorem 6 can be found in the appendix.

5.4 Simulation results

The parameters of the simulations are $\sigma = 1$ and $\omega = 10^5$. The code used for simulations can be found in [16].

We simulated the evolution of the aggregation error λ_n as a function of the number of workers, for both CWMed and CompMed. The results are presented in Figure 1.

For $f = 0$, the aggregation error converges to 0 for both aggregation schemes. We now consider the case $f = 0.2$ (i.e., 20% of Byzantine workers). For CWMed, the aggregation error converges to a value close to 0.12 (the irregularities of the plot are due to the fact than one new Byzantine worker is added for every 5 new workers). For CompMed, the aggregation error quickly becomes indistinguishable from the case $f = 0$ (i.e., it converges to 0).

This illustrates the AAA property of our aggregation scheme: the aggregation error converges to 0 when the number of workers increases, despite a constant fraction of Byzantine workers (which is not the case for existing aggregation schemes, e.g. CWMed).

[10] Note that here, the randomness comes from the values y_i, but also from K.

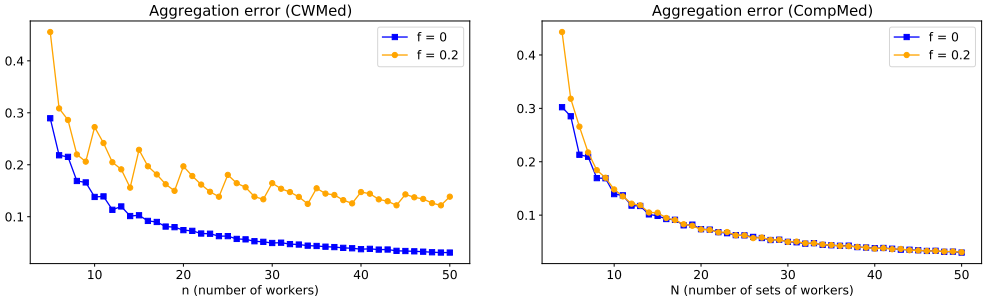

Figure 1 Evolution of the aggregation error for CWMed (left side) and CompMed (right side), as a function of n (number of workers) and N (number of sets of workers) respectively, for $f = 0$ and $f = 0.2$. As a reminder, N is the largest integer such that $N \leq n^2$, where n is the number of workers.

6 Conclusion

In this paper, we presented the first aggregation scheme with the AAA property, and proved its correctness. We illustrated this property with simulations, and compared it to an existing scheme.

The goal of this work was to show that it was possible to have an aggregation error converging to 0 (when n increases) in the presence of Byzantine workers. For future works, an interesting question would be: how *fast* can it converge to zero? The challenge would be to design an aggregation scheme ensuring a faster convergence, both in theory and in simulations.

References

1 Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 4613–4623, 2018.

2 Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS 2019*, pages 8632–8642, 2019. URL: `https://proceedings.neurips.cc/paper/2019/hash/ec1c59141046cd1866bbbcdfb6ae31d4-Abstract.html`.

3 Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems 30*, pages 119–129. Curran Associates, Inc., 2017.

4 Amine Boussetta, El-Mahdi El-Mhamdi, Rachid Guerraoui, Alexandre Maurer, and Sébastien Rouault. AKSEL: Fast Byzantine SGD. In *24th International Conference on Principles of Distributed Systems (OPODIS 2020)*, 2021.

5 Saikiran Bulusu, Prashant Khanduri, Pranay Sharma, and Pramod K. Varshney. On distributed stochastic gradient descent for nonconvex functions in the presence of byzantines. In *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*, pages 3137–3141. IEEE, 2020. `doi:10.1109/ICASSP40776.2020.9052956`.

6 Lingjiao Chen, H. Wang, Zachary B. Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients. In *ICML*, 2018.

7 Yudong Chen, Lili Su, and Jiaming Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(2):44, 2017.

8 Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015. URL: `http://proceedings.mlr.press/v38/choromanska15.html`.

9 Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Sébastien Rouault. Aggregathor: Byzantine machine learning via robust gradient aggregation. In *SysML*, 2019.

10 Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Rhicheek Patra, Mahsa Taziki, et al. Asynchronous byzantine machine learning (the case of sgd). In *ICML*, pages 1153–1162, 2018.

11 El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, and Lê Nguyên Hoang. Geniunely distributed byzantine machine learning. In *PODC*, 2020.

12 El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in Byzantium. In *Proceedings of the 35th International Conference on Machine Learning*, pages 3521–3530. PMLR, 2018.

13 Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8580–8589, 2018. URL: `https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html`.

14 Kishori M. Konwar, Sanguthevar Rajasekaran, and Alexander A. Shvartsman. Robust network supercomputing with malicious processes. In Shlomi Dolev, editor, *Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings*, volume 4167 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2006. `doi:10.1007/11864219_33`.

15 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

16 Alexandre Maurer. Source code for simulations related to this paper. URL: `https://tinyurl.com/sim-aaa-paper`.

17 David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

18 Lili Su and Nitin H. Vaidya. Fault-tolerant multi-agent optimization: Optimal iterative distributed algorithms. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 425–434. ACM, 2016. `doi:10.1145/2933057.2933105`.

19 Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd, 2018.

20 Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent, 2018.

21 Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 2018.

A Appendix

Proof of Lemma 2. Let $p = \frac{2f+1}{4}$. Let us describe 4 ways to select some Byzantine workers among N^2 workers, that we call "games".

- **Game A**: k workers are selected randomly, and then turned Byzantine.
- **Game B**: Each worker is turned Byzantine with probability p.
- **Game C**: Game B is executed. Then, if the number of Byzantine workers is k or less: all workers are turned Byzantine.

- **Game D**: Game C is executed. Then, we randomly pick one Byzantine worker, make it correct again, and repeat the process until we have exactly k Byzantine workers.

Let Φ_X be the event: "After Game X, all N sets contain a strict minority of Byzantine workers." As Game D consists in executing Game C, then only removing Byzantine workers, we have: $P(\Phi_D) \geq P(\Phi_C)$.

Then, we can notice that Game D is equivalent to Game A (since each worker is equally likely to end up Byzantine). Therefore, $P(\Phi_A) = P(\Phi_D) \geq P(\Phi_C)$. Now, let us give a lower bound of $P(\Phi_C)$.

Let Ψ_B be the following event: "After Game B, there are strictly more than k Byzantine workers". Then, we can notice that, for Φ_C to be true, it is necessary that both Φ_B and Ψ_B are true. Indeed, if Ψ_B is false, Φ_C cannot be true, because all workers would then be turned Byzantine in Game C (just after executing Game B). And, if Ψ_B is true but Φ_B is false, Φ_C cannot be true, because we would not have a strict minority of Byzantine workers in all N sets. Therefore, $P(\Phi_C) \geq P(\Phi_B \wedge \Psi_B)$.

Now, notice that $P(\Psi_B) = P(\Phi_B \wedge \Psi_B) + P(\neg\Phi_B \wedge \Psi_B)$. Since $P(\neg\Phi_B \wedge \Psi_B) \leq P(\neg\Phi_B) = 1 - P(\Phi_B)$, we have: $P(\Psi_B) \leq P(\Phi_B \wedge \Psi_B) + 1 - P(\Phi_B)$, and $P(\Phi_B \wedge \Psi_B) \geq P(\Phi_B) + P(\Psi_B) - 1$.

Before going further, let us give a lower bound of $P(\Psi_B)$. Let N_0' be the smallest integer such that, $\forall N \geq N_0'$:

$$\frac{\ln N^2}{N^2} < (p-f)^2$$

Let k' be the number of Byzantine workers after Game B. According to Hoeffding's inequality, applied to the N^2 workers:

$$P\left(\left|\frac{k'}{N^2} - p\right| < \sqrt{\frac{\ln N^2}{N^2}}\right) \geq 1 - \frac{2}{N^4}$$

Thus, $\forall N \geq N_0'$:

$$P\left(\left|\frac{k'}{N^2} - p\right| < p - f\right) \geq 1 - \frac{2}{N^4}$$

Now, we can remark that:

$$P\left(\left|\frac{k'}{N^2} - p\right| < p - f\right) \leq P\left(-\frac{k'}{N^2} + p < p - f\right) = P(k' > fN^2)$$

Thus, $\forall N \geq N_0'$:

$$P(\Psi_B) = P(k' > k) \geq P(k' > fN^2) \geq 1 - \frac{2}{N^4}$$

Thus, according to Lemma 1, $\forall N \geq \max(N_0, N_0')$:

$$P(\Phi_B \wedge \Psi_B) \geq \left(1 - \frac{2}{N}\right) + \left(1 - \frac{2}{N^4}\right) - 1 = 1 - \frac{2}{N} - \frac{2}{N^4}$$

Therefore, we have:

$$P(E_N') = P(\Phi_A) = P(\Phi_D) \geq P(\Phi_C) \geq P(\Phi_B \wedge \Psi_B) \geq 1 - \frac{2}{N} - \frac{2}{N^4}$$

Let N_1 be such that $N_1 \geq \max(N_0, N_0')$ and, $\forall N \geq N_1$, $\frac{2}{N^4} \leq \frac{1}{N}$. Then, we have:

$$P(E_N') \geq 1 - \frac{3}{N}$$

◀

Proof of Theorem 5. Let $j \in \{1, \dots, d\}$, and let L^* be a list defined similarly to L, except that we replace $\mathcal{N}(0, \sigma^2)$ by $\mathcal{N}(g_j, \sigma^2)$. Note that this is equivalent to adding g_j to each value of L.

Let us call a_j the j^{th} coordinate of the aggregated vector A_n. Then:

$$\mathbb{E}[(a_j - g_j)^2] = \mathbb{E}[(med(L^*) - g_j)^2] = \mathbb{E}[med(L)^2] = \beta_n$$

Therefore, $\forall j \in \{1, \dots, d\}$, $\mathbb{E}[(a_j - g_j)^2] = \beta_n$, and:

$$\lambda_n = \frac{\mathbb{E}[\|A_n - G\|^2]}{d} = \frac{\sum_{j=1}^{d} \mathbb{E}[(a_j - g_j)^2]}{d} = \frac{d\beta_n}{d} = \beta_n$$ ◀

Proof of Theorem 6. Let $(W_1, \dots, W_N)$ be the N sets of workers chosen at each step of CompMed. Let S_i be the set of vectors sent by the workers of W_i. Let $Maj(S_i) = (h_1, h_2, \dots, h_d)$.

If W_i contains a strict majority of correct workers, then, $\forall j \in \{1, \dots, d\}$, h_j follows the normal distribution $\mathcal{N}(g_j, \sigma^2)$. Otherwise, $\forall j \in \{1, \dots, d\}$, $h_j = \omega$.

Let K be the number of sets of workers W_i that do *not* contain a strict majority of correct workers. Then, the rest of the proof is identical to the proof of Theorem 5, if we replace L by L' (that is, replacing n by N and k by K). Thus, the result. ◀