SGD: Decentralized Byzantine Resilience

El Mahdi El Mhamdi Rachid Guerraoui Arsany Guirgui Sébastien Rouault
EPFL
{elmahdi.elmhamdi, rachid.guerraoui, arsany.guirguis, sebastien.rouault}@epfl.ch

May 13, 2019

Abstract

The size of the datasets available today leads to distribute Machine Learning (ML) tasks. An SGD–based optimization is for instance typically carried out by two categories of participants: parameter servers and workers. Some of these nodes can sometimes behave arbitrarily (called *Byzantine* and caused by corrupt/bogus data/machines), impacting the accuracy of the entire learning activity. Several approaches recently studied how to tolerate Byzantine workers, while assuming honest and trusted parameter servers. In order to achieve total ML robustness, we introduce GuanYu, the first algorithm (to the best of our knowledge) to handle Byzantine parameter servers as well as Byzantine workers. We prove that GuanYu ensures convergence against 1/3 Byzantine parameter servers and 1/3 Byzantine workers, which is optimal in asynchronous networks (GuanYu does also tolerate unbounded communication delays, i.e. asynchrony). To prove the Byzantine resilience of GuanYu, we use a contraction argument, leveraging geometric properties of the median in high dimensional spaces to prevent (with probability 1) any drift on the models within each of the non-Byzantine servers. To convey its practicality, we implemented [1] GuanYu using the low-level TensorFlow APIs and deployed it in a distributed setup using the CIFAR-10 dataset. The overhead of tolerating Byzantine participants, compared to a vanilla TensorFlow deployment that is vulnerable to a single Byzantine participant, is around 30% in terms of throughput (model updates per second) - while maintaining the same convergence rate (model updates required to reach some accuracy).

[1] The code to reproduce our experiments and assess GuanYu on other situations is provided at: `https://github.com/anonconfsubmit/submit-1`. Its sources are encrypted with the following key: `ohd1iyxnigrbsojo8a1z1a9vtnt7mft5`

1 Introduction

Following the recent celebrated successes [28], machine learning (ML) is expected to play a central role in safety-critical tasks such as driving and flying people, diagnosing their diseases, and recommending treatments to their doctors [22]. In such tasks, little room should be left for the vulnerabilities of today's machine learning solutions, as routinely reported [9, 24, 33].

One of the most active lines of research in machine learning, coined *adversarial machine learning*, consists in highlighting a series of vulnerabilities, and proposing schemes to solve each of them. These vulnerabilities boil down to roughly three classes. (1) *Evasion Attacks*, which are possible after a model has already been trained, are those that allow a malicious input to be classified as a benign one, and therefore 'evade' a potential security filter. This line of research is the most covered so far, with hundreds of papers per year [9]. It benefits from large media coverage to the extent in which 'adversarial machine learning' is often understood only in terms of 'adversarial examples' (i.e. evasion attacks). However, there is more than evasion to adversarial ML. Privacy concerns are also raised by (2) *Exploration Attacks*, which lead to the discovery of privacy-sensitive information used to train the model [4]. There is a third, yet less covered line, although chronologically among the first set of vulnerabilities to be discovered in ML [16, 31, 35, 25, 8]: (3) *Poisoning Attacks*. Here, misleading data in the training phase influence the quality of the learned model. In this work, we focus on the latter and encompass it inside the bigger quest for *Byzantine Resilient Machine Learning*, which we discuss below, after emphasizing the danger of data poisoning.

Data poisoning kills. AI safety is often depicted as a far-future concern, with most illustrative examples using rogue robots and, in the best case, self-driving cars. In fact, the lack of safety in *already deployed* and primitive forms of AI is a present threat. Data Poisoning spans many cases of vulnerable ML applications having a negative social impact. To illustrate this, consider a video that convinces young parents that vaccines are dangerous for their babies and is labeled as 'health advice'; this is arguably data poisoning, given the medical consensus on vaccines. For instance, the World Health Organization has declared vaccine hesitancy as a global health threat in 2019, reporting a surge of 30% in measles infections and resurgence in countries that were close to eliminating the disease [32]. Recent research suggests a negative role of social medias' recommender systems in the surge of the anti-vaccine resentment [38, 30, 37, 20]. Obviously, the surge of anti-vaccine resentment

is more complex than mislabeled content poisoning a recommender system. Striking cases of Data Poisoning are also present in political debates where a minority of data-points could lead a recommender system to disastrous consequences: In February 2018, after the tragic parkland high school shooting in Florida, YouTube's front page featured a video defaming teenager survivors Emma Gonzalez and David Hogg among others as "crisis actors" as they rose to prominence while campaigning for gun control. YouTube had to publicly apologize but the harm [2] was done and the debate on gun control in the U.S. successfully poisoned by false claims, some survivors of the shooting even received death threats [3].

Byzantine resilient machine learning. At a very high level, machine learning could be seen as an attempt to *aggregate* knowledge from data-points and update decision-making algorithms accordingly. Considering the social media example, users *behave* (like, comment, follow etc) and create data-points that are, in turn, *aggregated* at the learner (to update the model). To secure machine learning, be it distributed or centralized, robust aggregation rules have to be devised, far from the vulnerable *averaging*, which is still the backbone of the most popular forms of ML [3].

Over the past two years, a growing body of work [10, 5, 14, 42, 7, 18, 40, 41] (and about 30 other papers) encompassed resilience to Poisoning inside a wider project, *Byzantine Resilient Machine Learning*. Poisoning attacks represent a special case of the broader Byzantine Failure model of [27]. The latter encompasses not only Data Poisoning but goes beyond to include software bugs, latency and even the worst case: hacked machines behaving arbitrarily. The work on Byzantine-resilient ML relied on the now standard parameter-server abstraction [29], in which a *server*, holding the model, leverages *workers* to compute model updates. Not all the workers are assumed to be honest as a fraction could be Byzantine, but the server is however assumed to be honest. While the aggregation rules produced by this line of research are useful in many applications where the server is trusted, in particular in the single learner case, none of these aggregation schemes secure a fully decentralized learning scheme, where servers could also be subject to a malicious behavior.

[2] We are not inferring that this falls necessarily into pure poisoning. Whatever the vulnerability is, the safety of recommender systems such as YouTube's against poisoning is an urgent question.

[3] `https://goo.gl/DaK5X9`

Contributions. We present GuanYu [4], the first Byzantine tolerant learning algorithm that combines (a) the resilience to Byzantine workers with (b) the (so far unsolved issue of) resilience to Byzantine parameter servers. We prove that GuanYu tolerates up to 1/3 Byzantine servers and 1/3 Byzantine workers, which is optimal in the asynchronous setting. A major technical scheme, underlying the resilience of GuanYu to Byzantine parameter servers, is a contraction argument leveraging geometric properties of the median in high dimensional spaces. Using this argument, we prevent (with probability 1) any drift on the models within each of the non-Byzantine servers. Interestingly, we show that the dimension plays against the adversary.

To convey its practicality, we build GuanYu on top of TensorFlow low-level APIs [3] and experimentally assess our implementation. Through our experiments, we show that GuanYu can indeed tolerate Byzantine behavior with a reasonable overhead compared to the non Byzantine-resilient vanilla TensorFlow deployment. Specifically, we study the overhead of building GuanYu over TensorFlow low-level APIs instead of leveraging its distributed runtime (which we quantify by 65%) and the cost of Byzantine resilience (which we quantify by around 30%).

The paper is organized as follows. Section 2 presents the formal setting of distributed learning and specify the threat model. Section 3 describes GuanYu both on the server side as well as on the worker side. Section 4 discusses the practical aspects of our system implementation and Section 5 describes our evaluation results. We conclude in Section 6. The full convergence proof of GuanYu is given as a supplementary material.

The code to reproduce our experiments and assess GuanYu on other situations is provided at: `https://github.com/anonconfsubmit/submit-1`. Its sources are encrypted with the following key: `ohd1iyxnigrbsojo8a1z1a9vtnt7mft5`.

2 Model

We build on the standard synchronous parameter server model [29], with two main differences. Figure 1 gives a schematic overview.

1. We assume some minority of the nodes (i.e. machines) involved in the distributed SGD process to be *adversarial*, or *Byzantine*, in the parlance of [10, 14, 21]. We give the exact specification of the adversary's capabilities in Section 2.2.

[4] Guan Yu was a Chinese general of the 3^{rd} century AD. According to common belief, he held very high moral standards and is widely respected today.

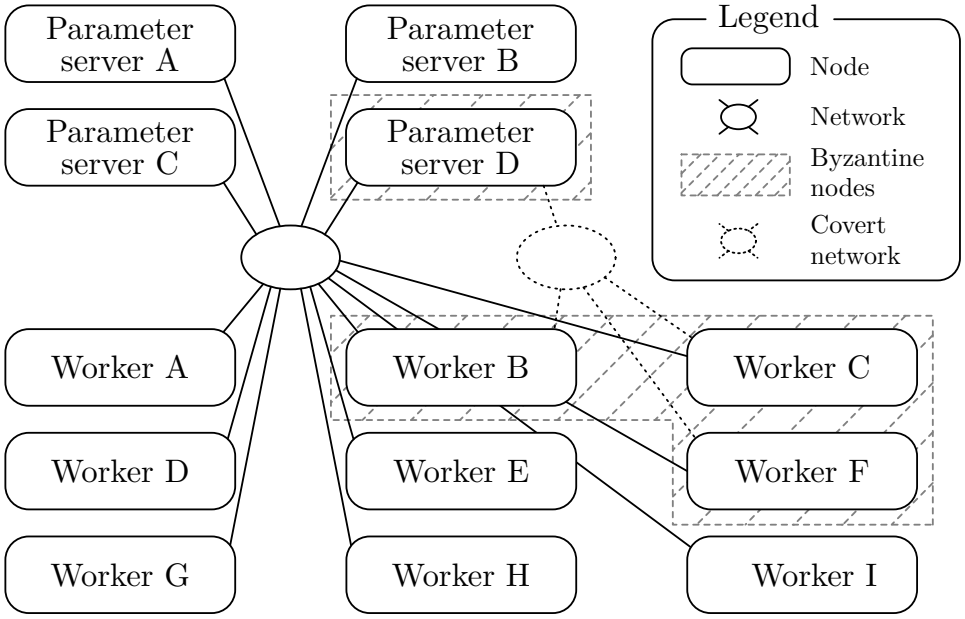

Figure 1: A distributed ML setup with four parameter servers and nine workers, including respectively one and three Byzantine nodes. The network is assumed to be *asynchronous*, i.e. we assume no bound on the time it takes to deliver a message. This is not to be confused with *asynchronous SGD* (see Section 2.1). Byzantine nodes can cooperate by communicating with each other, using a different, arbitrarily fast network. They are viewed as one, cooperating adversarial entity (the *adversary*).

2. The second difference with the standard parameter server model (and previous works) is that we consider several *replicas* of a parameter server, instead of one. Replicating the parameter server prevents it from being a single *point–of–failure*, unlike in classical approaches.

While removing the strong assumption of a single correct parameter server makes our algorithm more robust than its predecessors, it also increases its complexity. We could use standard *state machine replication* [36] (SMR) to create the abstraction of one parameter server (which would allow the transparent use of published algorithms that tolerate Byzantine workers [10, 14, 21]) while benefiting from the resilience of having many underlying replicas. But the SMR approach raises two serious issues:

1. The overhead of classical SMR approaches (e.g. [36]) in the case

of distributed SGD is potentially huge, as nodes exchange gradients and parameter vectors that can be several hundreds MB large. In this context, re–transmissions and synchronizations guaranteeing that all the parameter servers share the same global state can induce severe additional performance loss. In fact, excessive synchronization of the parameter server can be useless for distributed SGD. Indeed, using imprecise or mildly diverging parameters have proven themselves beneficial in terms of convergence speed and quality [44, 6, 26].

2. SMR has been proven to be impossible in *asynchronous networks* [23], and requires a bound on communication delays between non–Byzantine nodes (at least to hold eventually). We argue the assumption of such a time bound is, in the presence of adversarial nodes, at least inefficient, if not dangerous[5]. The rationale is that, either the estimation of such a time bound is large enough so that it always remains true in the actual system (but incurs long *timeouts*), or this time bound is small enough to be violated in practice (which is prone to breaking any proof of convergence). The latter case may not be a rare event, as in practice the adversary has a wide range of possible actions that are not (or imperfectly) modeled, e.g. congesting some parts of the network for some short periods of time.

2.1 Asynchronous Network / Synchronous Training

In this paper, we study *bulk–synchronous training* over an *asynchronous network*. Respectively: **1.** A parameter server does not need to wait for all the workers' gradients to make progress, and vice versa. Only the gradients computed at learning step t (which could be more than one) are used for the parameter updates at the same step t. **2.** No assumption is made on the time it takes for a non–Byzantine node's response to be computed, sent and delivered back to another non–Byzantine node.

A remark is in order with respect to the different meanings of *asynchrony*, respectively in the distributed computing and the distributed ML literature. In the latter, *asynchrony* is classically used in the context of *SGD training*, for algorithms where nodes (most often the *workers*) are not always working on the same *learning step* [13, 17, 3]. In the former, *asynchrony* expresses an orthogonal concept: the lack of any bound on communication delays.

[5] *Optimistic asynchronous* algorithms do exist, like [12], making "liveness" timing assumptions (also arguably challenged in an adversarial setting). Our GuanYu algorithm also builds on "liveness" assumptions, but from the intrinsic "resilience capabilities" of SGD (see Section 3.4) and not from any network timing.

2.2 Adversary Capabilities

The adversary is an entity that controls all Byzantine nodes (Figure 1) and which goal is to prevent the distributed SGD process from converging to a state that could have been achieved if there was no adversary.

As in [10, 14, 21], we assume an omniscient adversary. At any time, it has access to the full memory of every other node and the packets being transferred over the network. We assume that the adversary is however not omnipotent: it can only send messages from the nodes it controls, of course not necessarily following the protocol described in Section 3.3.

3 GuanYu

3.1 Background

GuanYu uses several *Gradient Aggregation Rules* (GARs). A GAR is a function of $\left(\mathbb{R}^d\right)^n \rightarrow \mathbb{R}^d$, where $(n, d) \in (\mathbb{N} - \{0\})^2$. For each step of a synchronous distributed SGD [29], the GAR's role is to *aggregate* the n workers' gradient estimations into 1 gradient, in order to update the parameter vector. In a non–Byzantine context, the arithmetic mean is commonly used [3].

A (α, f)–Byzantine resilient GAR tolerates the presence of some minority, noted $f < n$, of Byzantine workers (i.e. able to send arbitrary gradients). In the context of a single parameter server (assumed to be honest), *Multi–Krum*, introduced in [10], enables to tolerate Byzantine workers. It ensures the almost–sure convergence of the parameter vector to a point where the gradient of the loss function is null.

More specifically, assuming $f \in \mathbb{N} - \{0\}$, *Multi–Krum* requires $n \geq 2f + 3$ to be (α, f)–Byzantine resilient, along with other assumptions (see Section 3.4). *Multi–Krum* works by assigning a score to each input gradient. The score of input gradient x is the sum of the distances between x and its $n - f - 2$ closest neighbor input gradients. *Multi–Krum* then outputs the arithmetic mean of the $n - f - 2$ smallest–scoring gradients.

3.2 Notations

We note M the *coordinate–wise median*, i.e. each coordinate i of the output vector is the *median* applied over all the input vectors' i^{th} coordinates. *Multi–Krum* is noted F.

Let $\left(n, \overline{n}, f, \overline{f}, q, \overline{q}, d\right) \in (\mathbb{N} - \{0\})^7$, each representing:

- $n \geq 3f + 3$ the total number of parameter servers, among which f are Byzantine.
- $\overline{n} \geq 3\overline{f} + 3$ the total number of workers, among which $\overline{f}$ are Byzantine.
- $2f + 3 \leq q \leq n - f$ the *quorum* used for M.
- $2\overline{f} + 3 \leq \overline{q} \leq \overline{n} - \overline{f}$ the *quorum* used for F.
- d the dimension of the parameter space $\mathbb{R}^d$.

The *quorums* indicate the number of messages (either gradients or parameter vectors in our algorithm) to wait before aggregating them (using either F or M). See Figure 2.

Let (without loss of generality):

- $[1\,..\,n - f]$ be indexes of non–Byzantine servers
  $[n - f + 1\,..\,n]$ be indexes of Byzantine servers
- $\left[1\,..\,\overline{n} - \overline{f}\right]$ be indexes of non–Byzantine workers
  $\left[\overline{n} - \overline{f} + 1\,..\,\overline{n}\right]$ be indexes of Byzantine workers

Let $\theta_t^{(i)}$ be a notation for the parameter vector at parameter server $i \in [1\,..\,n - f]$ for step $t \in \mathbb{N}$. Let $G_t^{(i)}$ be a notation for the gradient distribution at worker $i \in \left[1\,..\,\overline{n} - \overline{f}\right]$ for step $t \in \mathbb{N}$. Let L be the loss function we aim to minimize. Let $\nabla L\,(\theta)$ be the real gradient of the loss function at θ, and let $\widehat{\nabla L}\,(\theta)$ be a stochastic estimation of the gradient, following G, of L at θ.

3.3 GuanYu's Algorithm

We now overview our algorithm. Each non–Byzantine parameter server i starts (at step $t = 0$) with the same parameter vector:

$$\forall i \in [1\,..\,n - f]\,,\ \theta_0^{(i)} \triangleq \theta_0$$

One full step of the algorithm is summarized in Figure 2, and described in the following three paragraphs:

1. Step $t \in \mathbb{N}$ begins with each (non–Byzantine) parameter server i broadcasting its current parameter vector $\theta_t^{(i)}$ to every worker. Each (non–Byzantine) worker j *aggregates* with M the q first received $\theta_t^{(i)}$, and computes an estimate $g_t^{(j)}$ of the gradient at the aggregated parameters.

2. Then, each (non–Byzantine) worker j broadcasts its computed gradient estimation $g_t^{(j)}$ to every parameter server. Each (non–Byzantine) parameter server i *aggregates* with F the $\overline{q}$ first received $g_t^{(j)}$, and performs a local parameter update with the aggregated gradient, resulting in $\overline{\theta}_t^{(i)}$.

3. Finally, each (non–Byzantine) parameter server i broadcasts $\overline{\theta}_{t+1}^{(i)}$ to every other parameter servers, aggregating with M the q first received $\overline{\theta}_{t+1}^{(k)}$. This aggregated parameter vector is $\theta_{t+1}^{(i)}$.

Parameter server $i \in [1 \,..\, n-f]$		**Worker $j \in \left[1 \,..\, \overline{n-f}\right]$**
$\theta_t^{(i)}$	$\overset{q}{\rightarrow}$	
	$\overset{\overline{q}}{\leftarrow}$	$g_t^{(j)} = \widehat{\nabla L}\left(M\left(\theta_t^{(a)} \dots \theta_t^{(b)}\right)\right) \sim G_t^{(j)}$
$\theta_t^{(i)} - \eta_t F\left(g_t^{(x)} \dots g_t^{(y)}\right) = \overline{\theta}_{t+1}^{(i)}$	$\circlearrowright q$	
$M\left(\overline{\theta}_{t+1}^{(z)} \dots \overline{\theta}_{t+1}^{(w)}\right) = \theta_{t+1}^{(i)}$	$\overset{q}{\rightarrow}$	
		$\cdots\cdots\cdots$

Figure 2: Operations of GuanYu, as described in Section 3.3. The arrow indicates a *broadcast* (and its direction), along with the number of messages *delivered* by the receiving ends, late messages being discarded. For instance in the second row, in all the workers (the receiving ends) M is always called with the first $\overline{q}$ received *parameter vectors*, and in the third row, F is called in each parameter server with the first q received *gradients*. The rest of the notation is defined in Section 3.2.

3.4 Assumptions

Besides the assumptions[6] from [10], which allow convergence in the case of one trusted parameter server, we assume that:

1. L is Lipschitz continuous, i.e. $\exists l > 0$,
   $\forall (x, y) \in \left(\mathbb{R}^d\right)^2, \ \|\nabla L(x) - \nabla L(y)\| \leq l \|x - y\|$
2. $\exists t_s \in \mathbb{N}, \ \forall t > t_s, \ \exists (u_t, o_t) \in \mathbb{R}^d \times \mathbb{R}^d,$
   $\exists \left(\delta_t^{(1)} \dots \delta_t^{(\alpha)}, r_t^{(1)} \dots r_t^{(\alpha)}\right) \in \mathbb{R}^\alpha \times \left(\mathbb{R}^{d^2}\right)^\alpha,$

[6]The exhaustive list is available in the supplementary material

$\forall i \in [1\,..\,n-f]\,,$
$a_t^{(i)} \sim \mathcal{N}\left(0, \delta_t^{(i)}\right),\ b_t^{(i)} \sim \mathcal{N}\left(o_t, r_t^{(i)}\right),$
with $a_t^{(1)} \dots a_t^{(\alpha)},\ b_t^{(1)} \dots b_t^{(\alpha)}$ mutually independent,
and: $\overline{\theta}_t^{(i)} = a_t^{(i)} \cdot u_t + b_t^{(i)}$

Our Lipschitz continuity assumption acts as the only bridge between the parameter vectors $\theta_t^{(i)}$ and the stochastic gradients $\widehat{\nabla L}$. To our knowledge, the validity of this assumption has not been asserted in practical cases. Nevertheless, we hardly believe that a Byzantine resilient algorithm can be proven without assuming some kind of *similarities* between gradients estimated at close but different parameters.

Building upon [19], the second assumption conveys that, after some step $t_s \in \mathbb{N}$, all the non–Byzantine parameter vectors are roughly aligned. This assumption has also been supported by the experiments we conducted, available in the supplementary materiel (Section 3.4).

3.5 Proof Intuition

Under the assumption established in Section 3.4, and:

$$\forall t \in \mathbb{N},\ \forall \left(\theta_t^{(n-f+1)} \dots \theta_t^{(n)}\right) \in \left(\mathbb{R}^d\right)^f,$$

our algorithm guarantees that:

$$\lim_{t \to +\infty} \mathbb{E} \left\| \nabla L \left(M \left(\theta_t^{(1)} \dots \theta_t^{(n)} \right) \right) \right\| = 0. \tag{1}$$

Equation 2 is an adapted formulation of the commonly admitted termination criteria for non–convex optimization tasks [11].

We provide below the key intuitions and steps of the proof. The full proof is provided in the supplementary material.

1. Regarding the global strategy, we identify two independent propositions that are sufficient for Equation 2. Namely, it is enough to prove that: **1.** Every pair of non–Byzantine parameter vectors $\theta_t^{(i)}$ and $\theta_t^{(j)}$ gets almost–surely arbitrary close to each other after some (possibly large) step $t \in \mathbb{N}$. **2.** The algorithm does not prevent almost–sure convergence of the gradient computed at any single non–Byzantine parameter vector $\theta_t^{(i)}$, e.g. $\theta_t^{(1)}$. The intuition is straightforward here: if the gradient at one (non–Byzantine) parameter vector converges, and the other non–Byzantine parameter vectors get arbitrarily close to it, then by construction of the *median* (because there

is a majority of non–Byzantine parameter servers) and the Lipschitz continuity of L (*small differences* in the parameter vectors always make *small differences* in the gradients), Equation 2 is verified.

2. Regarding proposition **1.**, two properties come into effect: the learning rate η_t converging toward 0, and the (average) *contraction effect* of the coordinate–wise *median*. This *contraction effect* states that, on average, the distance between any two $\theta_t^{(i)}$ and $\theta_t^{(j)}$ is strictly smaller than the maximum (for some a, b) distance between $\overline{\theta}_t^{(a)}$ and $\overline{\theta}_t^{(b)}$. Then, the fact that the learning rate goes toward 0, along with the Lipschitz continuity of L, intuitively translate into the possible existence of two *phases*. Before some step t_{inflex}, the (bounded) stochastic noise of the (non–Byzantine) gradient estimations, on average, pushes the (non–Byzantine) parameter vectors away from each other; the "exploratory phase". After step t_{inflex}, the learning rate becomes small enough so that, on average, the *contraction effect* pulls back together the (non–Byzantine) parameter vectors.

Regarding proposition **2.**, the studied parameter vector $\theta_t^{(1)}$ is modified by the gradient descent itself, and the coordinate–wise *median*. We already know that (stochastic) gradient descent (with *Multi–Krum*) alone would converge [10]. Using proposition **1.**, the intuition is that, because of Lipschitz continuity of L, the non–Byzantine gradient estimations are all following arbitrary close random distributions after some step t, meeting the original conditions of [10].

We also argue that $1/3$ of Byzantine nodes, both for the workers and the parameter servers, is (asymptotically) optimal in asynchronous networks. The rationale stems from the optimal breakdown point [7] of $1/2$ for any synchronous robust aggregation scheme (e.g. F and M), identified in [34]. Network asynchrony makes it impossible to differentiate a slow non–Byzantine node from a non–responding Byzantine node. This observation implies the over–provisioning of (at least) $1/2$ non–Byzantine nodes for $1/2$ Byzantine ones. Hence the final optimal breakdown point of $\frac{1/2}{3/2}\left(\frac{\text{one half Byzantine nodes}}{\text{total of three halves}}\right) = 1/3$.

4 Implementation

We describe in this section our design and implementation of the GuanYu system. We use TensorFlow [3], a popular framework for distributed Machine Learning, as an underlying implementation framework. This decision has

[7] Defined as the faction of Byzantine nodes beyond which no resilient aggregation exists.

its pros and cons. On the one hand, TensorFlow is one of the most used distributed ML frameworks with a wide community support. It also provides good performance in terms of scalability and distributed GPUs support. On the other hand, TensorFlow is not resilient to even one Byzantine node, either a worker or a parameter server. Adding Byzantine resilience to TensorFlow requires modifying its core runtime to support extra communication and processing on top of the learning low-level abstractions. Tweaking the distributed core runtime of TensorFlow to support parameter server replication is cumbersome.

We pursued another route: we use TensorFlow as a local library, not as a distributed framework. More specifically, we use the TensorFlow low-level abstractions for calculating the gradients and updating the model, but we handle communication ourselves between all the nodes (workers and parameter servers) in the learning process. However, to be consistent with TensorFlow design, we encode all messages in protocol buffers and we use gRPC for communication. A caveat is worth noting here: handling communication outside the dataflow graph introduces an overhead due to the context switches between TensorFlow and Python runtimes. This includes, for example, converting tensors to Numpy arrays and vice versa. We quantify this overhead in Section 5.

Following our design, each worker builds an independent but similar graph to all other graphs built by other workers. The same is applied by the parameter servers (Figure 2). Each parameter server holds a copy of the model so that a worker can pull the copy of the current model to work at each iteration. Workers store only a copy of the model so that they can do the backpropagation step each iteration. We implement the extra processing functions that we use (e.g. *median*) as a TensorFlow operation so that they can be integrated in the graph built by either workers or parameter servers. Each node also initiates a gRPC server to respond to other nodes' requests. Moreover, each node implements the procedures that can be called by other nodes to pull whatever information they want either the model or the gradients. We assume an asynchronous network, though our learning algorithm is synchronous in the sense of Section 2.1.

5 Experimental Evaluation

We evaluate the performance of GuanYu following a rather standard methodology in the distributed ML literature. In particular, we consider the image classification task due to its wide adoption as a benchmark for distributed

ML literature [3, 15, 43].

5.1 Setting

Platform. We use Grid5000 [2] as an experimental platform. We employ 18 compute/worker nodes in addition to one (in a non-Byzantine resilient deployment) or six (in the case of employing GuanYu for Byzantine resilience) parameter servers, all from the same cluster, each having 2 CPU (Intel Xeon E5-2630) with 8 cores, 128 GiB RAM and 10 Gbps Ethernet.

Dataset. We use the CIFAR-10 dataset [1], a widely used dataset in image classification [39, 43], which consists of 60,000 color images (we use 50,000 images for training and the rest for testing) in 10 classes. We employ a convolutional neural network with a total of 1.75M parameters as shown in Table 1. Unless stated otherwise, we use a mini-batch of size 128 and learning rate 0.001.

To respect the limits imposed by GuanYu on the ratio of Byzantine participants, we use up to five Byzantine workers (out of total of 18 nodes) and one Byzantine server (out of total of six nodes). A severe attack that a Byzantine participant can perform is to send bad data (gradients or model) that pulls the learning process out of the convergence area. We experimented with such attack by modeling a Byzantine behavior in a way that sends a totally corrupted data compared to the correct one it should send. We also experimented a scheme in which a parameter server sends different (bad) models to different workers in the same iteration. We show that the vanilla deployment of TensorFlow cannot tolerate even one Byzantine player performing any of these attacks.

Table 1: CNN Model parameters.

	Input	Conv1	Pool1	Conv2	Pool2	FC1	FC2	FC3
Kernel size	32×32×3	5×5×64	3×3	5×5×64	3×3	384	192	10
Strides		1×1	2×2	1×1	2×2			

5.2 Evaluation Metrics

We evaluate the performance of GuanYu using the following standard metrics.

Throughput.[8] This metric measures the total number of updates that the deployed system can do per second. The factors that affect the throughput are the time to compute a gradient, communication delays (workers receive the model, send the gradient, and servers also exchange the computed model), and the aggregation time on both sides, workers and parameter servers. *Accuracy.* This metric measures the top–1 cross–accuracy: the fraction of correct predictions among all the predictions, using the *testing* dataset. We measure accuracy both with respect to time and model updates.

We discuss the breakdown for the overhead imposed by GuanYu and differentiate between the overhead due to using TensorFlow low-level APIs (while handling the communication outside the graph) and the overhead due to the replicated parameter server and the communicated messages (the Byzantine resilience cost). First, we quantify the overhead of our system in a non-Byzantine environment. Second, we confirm that vanilla TensorFlow cannot tolerate even one Byzantine parameter server while GuanYu does.

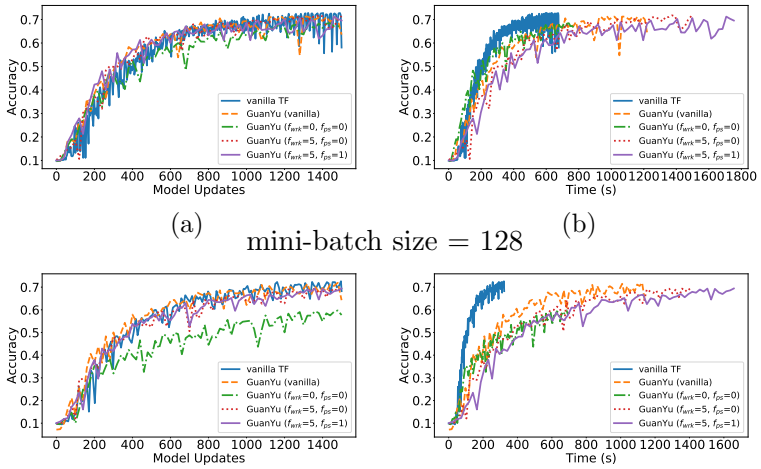

Figure 3: Overhead of GuanYuin a non-Byzantine environment.

5.3 Non–Byzantine Environment

In this section, we study the overhead of GuanYu, in the absence of Byzantine players, be they workers or parameter servers. We consider

[8] Although there is no figures (for brevity) that shows it on the y–axis, throughput can be, roughly, derived from the given figures.

two non–Byzantine resilient baselines: (1) vanilla TensorFlow (vanilla TF), in which we use the TensorFlow distributed core runtime and (2) vanilla GuanYu in which we execute exactly the same graph, as in vanilla TF, while handling the communication ourselves, as described in Section 4. The difference in performance between both baselines quantifies the overhead of abandoning the TF distributed runtime and opting for the design we use for GuanYu. We compare different deployments of GuanYu (which does tolerate Byzantine players) against these baselines while changing the *declared* numbers of Byzantine workers and parameter servers, keeping these numbers in the ranges instructed by our theory.

Figure 3(a) shows that all compared systems achieve almost the same convergence rate. Moreover, it shows that deploying the Byzantine variant of GuanYu does not add any overhead compared to the non–Byzantine one (vanilla GuanYu), in terms of convergence steps. However, the cost of our design and that of being Byzantine resilient appears in Figure 3(b) (and even more obvious while using a mini-batch size of 32 in Figure 3(d)) which depicts the convergence rate against time. For example, vanilla TF reaches 60% accuracy in 268 seconds which is 65% better than the vanilla deployment of GuanYu where the Byzantine deployment reaches the same accuracy level in (up to) 33% more time (which is the cost of Byzantine–resilience) compared to the latter.

It may look counter-intuitive that tolerating more Byzantine players helps achieve a better convergence rate in terms of model updates. This is in fact due to the design we use while implementing GuanYu. Parameter servers wait for a quorum of $2f + 3$ replies from workers before doing the aggregation step as discussed in Section 3.3 (while f is the *declared* number of Byzantine workers). Thus, increasing f forces servers to wait for more replies. Although this decreases the throughput of the learning process, parameter servers gather more gradients from workers, helping them to do better steps and converge faster (in terms of number of epochs). Using a smaller batch size (Figure 3(c)) emphasizes the convergence rate overhead while declaring a small number of Byzantine participants.

Overhead We attribute the performance difference between vanilla TF and vanilla GuanYu to two main reasons.

1. Abandoning the highly optimized distributed runtime of TensorFlow and replacing it with our implementation. This includes device placement, using operations that accept and output the same type of data structures (tensors), and the optimized communication between par-

ticipants. In the current implementation of GuanYu, we use naive approach to implement all of these blocks.

2. The context switch between TensorFlow and numpy/python runtimes. In our implementation, each participant, being a worker or a parameter server, builds its own graph independently to do some processing. Transferring the result to other contributing processes requires converting the resulting tensor to numpy array, serializing it in a protocol buffer then sending it over a gRPC channel. On the other side, a receiver endpoint should do exactly the same steps in a reverse order while feeding back the resulting tensor to its graph. Our measurements show that converting tensors to numpy arrays (and vice versa) and feeding tensors to a graph incur a big overhead.

The overhead of tolerating Byzantine failures is more obvious. First, the parameter server should be replicated; this increases the communication overhead and the idle time of workers while waiting to gather a quorum of replies from these parameter servers. Second, parameter server(s) processing time increases because of using a robust aggregation rule (e.g., coordinate-wise *median*) on the received gradients rather than a simple, fast one like averaging. Third, parameter servers should communicate towards the end of each iteration before updating the model, gathering a quorum and filtering our Byzantine responses (if any) through applying (again) a robust aggregation rule (e.g., *median*). We believe that this overhead is inevitable in any totally Byzantine ML system.

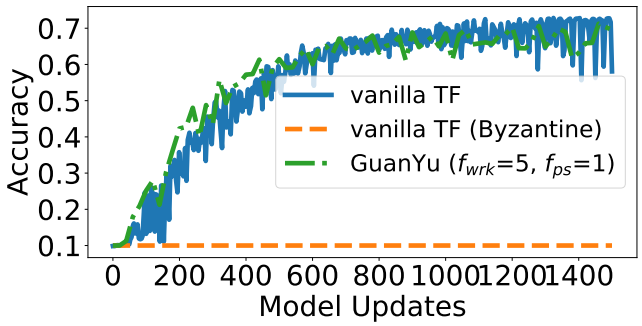

Figure 4: Impact of Byzantine players on convergence.

5.4 Byzantine Environment

We discuss here the result of deploying GuanYu in the presence of Byzantine behavior. Figure 4 shows that GuanYu can survive the presence of

Byzantine participants while vanilla TensorFlow cannot tolerate even one. We put a Byzantine attack in one of the following categories: (1) sending corrupted gradients to parameter servers, (2) sending corrupted parameter vectors/model to workers, (3) sending different replies to different participants, or (4) not responding at all to requests. We believe that, as a Byzantine player, choosing the last option is not wise because this will not harm convergence anyway (even a vanilla TF deployment can converge in the existence of a *silent* Byzantine player). We tested different possible Byzantine behaviors (on both ends: workers and parameter servers) and we got approximately similar results; we present an instance in Figure 4. Thanks to using robust aggregation rules at both ends (parameter servers and workers) and to our robust algorithm, GuanYu can converge (as depicted in the figure) in all cases.

6 Conclusion

In this paper, we present GuanYu, the first algorithm to provide theoretical guarantees against Byzantine parameter servers, in addition to tolerating Byzantine workers. We show that GuanYu guarantees convergence in the presence of a minority of Byzantine participants as well as asynchrony. Through building GuanYu over TensorFlow and deploying it in a distributed setup, we show that GuanYu can tolerate Byzantine behavior with a reasonable overhead compared to a non-Byzantine vanilla TensorFlow deployment.

References

[1] Cifar dataset. `https://www.cs.toronto.edu/~kriz/cifar.html`.

[2] Grid5000. `https://www.grid5000.fr/`.

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[4] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *SIGSAC*, pages 308–318, 2016.

[5] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In *Neural Information Processing Systems, to appear*, 2018.

[6] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Randomized quantization for communication-optimal stochastic gradient descent. *arXiv preprint arXiv:1610.02132*, 2016.

[7] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD with majority vote is communication efficient and fault tolerant. In *International Conference on Learning Representations*, 2019.

[8] Battista Biggio and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, 2012.

[9] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *arXiv preprint arXiv:1712.03141*, 2017.

[10] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Neural Information Processing Systems*, pages 118–128, 2017.

[11] Léon Bottou. Online learning and stochastic approximations. *Online learning in neural networks*, 17(9):142, 1998.

[12] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[13] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[14] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients. In *International Conference on Machine Learning*, pages 902–911, 2018.

[15] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *OSDI*, volume 14, pages 571–582, 2014.

[16] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[17] 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.

[18] Ilias Diakonikolas, Gautam Kamath, Daniel M Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Robustly learning a gaussian: Getting optimal error, efficiently. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2683–2702. Society for Industrial and Applied Mathematics, 2018.

[19] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred A Hamprecht. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

[20] Mark Dredze, David A Broniatowski, Michael C Smith, and Karen M Hilyard. Understanding vaccine refusal: why we need social media now. *American journal of preventive medicine*, 50(4):550–552, 2016.

[21] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in Byzantium. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3521–3530, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[22] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[23] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *JACM*, 32(2):374–382, 1985.

[24] Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. Adversarial spheres. *arXiv preprint arXiv:1801.02774*, 2018.

[25] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[26] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[27] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *TOPLAS*, 4(3):382–401, 1982.

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[29] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, volume 1, page 3, 2014.

[30] Tanushree Mitra, Scott Counts, and James W Pennebaker. Understanding anti-vaccination attitudes in social media. In *ICWSM*, pages 269–278, 2016.

[31] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D Joseph, Benjamin IP Rubinstein, Udam Saini, Charles A Sutton, J Doug Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. *LEET*, 8:1–9, 2008.

[32] World Health Organization and Rada Akbar. Ten threats to global health in 2019. `https://www.who.int/emergencies/ten-threats-to-global-health-in-2019`, 2019. [Online; accessed 21-January-2019].

[33] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, pages 506–519, 2017.

[34] Peter J Rousseeuw. Multivariate estimation with high breakdown point. *Mathematical statistics and applications*, 8:283–297, 1985.

[35] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *SIGCOMM*, pages 1–14, 2009.

[36] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *CSUR*, 22(4):299–319, 1990.

[37] Naomi Smith and Tim Graham. Mapping the anti-vaccination movement on facebook. *Information, Communication & Society*, pages 1–18, 2017.

[38] Melodie Yun-Ju Song and Anatoliy Gruzd. Examining sentiments and popularity of pro-and anti-vaccination videos on youtube. In *Proceedings of the 8th International Conference on Social Media & Society*, page 17. ACM, 2017.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[40] Lili Su and Shahin Shahrampour. Finite-time guarantees for byzantine-resilient distributed state estimation with noisy measurements. *arXiv preprint arXiv:1810.10086*, 2018.

[41] Pooja Vyavahare, Lili Su, and Nitin H Vaidya. Distributed learning with adversarial agents under relaxed network condition. *arXiv preprint arXiv:1901.01943*, 2019.

[42] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Zeno: Byzantine-suspicious stochastic gradient descent. *arXiv preprint arXiv:1805.10032*, 2018.

[43] Hao Zhang, Zeyu Zheng, Shizhen Xu, Wei Dai, Qirong Ho, Xiaodan Liang, Zhiting Hu, Jinliang Wei, Pengtao Xie, and Eric P. Xing. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In *USENIX ATC*, pages 181–193, 2017.

[44] Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. In *NIPS*, pages 685–693, 2015.

SGD: Decentralized Byzantine Resilience

Convergence proof of GuanYu

---

7 Preliminary material

GuanYu uses several components we introduce below.

7.1 *Multi–Krum*

Multi–Krum, noted F in the protocol and its convergence proof, is a *Gradient Aggregation Rule* (GAR) that has been introduced, along with the concept of (α, f)–Byzantine resilience, in [10].

Consider $(n, d) \in (\mathbb{N} - \{0\})^2$. A GAR is a function of $\left(\mathbb{R}^d\right)^n \to \mathbb{R}^d$, e.g. the arithmetic mean. Supposing some majority of its inputs follow i.i.d. the same distribution $\mathcal{X}$, a (α, f)–Byzantine resilient GAR outputs a gradient y such that:

- $\mathbb{E}\, y$ is in the same half–space as $\mathbb{E}\, \mathcal{X}$
- the first 4 statistical moments of y are bounded above by a linear combination of the first 4 statistical moments of $x \sim \mathcal{X}$

These conditions enable such a GAR to be *plugged–in* the "global confinement" proof of [11].

Let $f \in \mathbb{N} - \{0\}$, and $n \geq 2f + 3$. *Multi–Krum* works by assigning a score to each input gradient. This score of x is the sum of the distances between x and its $n - f - 2$ closest input gradients. Finally, *Multi–Krum* outputs the arithmetic mean of the $n - f - 2$ smallest–scoring gradients.

7.2 Coordinate–wise *median*

We formally define the *median*, noted $\mathcal{M}$, as follows:

$$\forall n \in \mathbb{N} - \{0\}\,,\ \forall\, (x_1 \dots x_n) \in \mathbb{R}^n,\ \mathcal{M}\,(x_1 \dots x_n) \triangleq x_s \in \mathbb{R}$$

such that:

$$\exists\,(L,H)\subset\left([1\,..\,n]-\{s\}\right)^2,\ L\cap H=\emptyset,\ |L|=|H|=\left\lfloor\frac{n}{2}\right\rfloor,$$
$$\forall l\in L,\ x_l\leq x_s,\ \forall h\in H,\ x_h\geq x_s$$

and:

$$\begin{cases}\exists\,(a,b)\in[1\,..\,n]^2,\ x_s=\frac{x_a+x_b}{2} & \text{if } n \text{ is even}\\ x_s\in\{x_1\dots x_n\} & \text{if } n \text{ is odd}\end{cases}$$

We can note that: $\forall n\in\mathbb{N}-\{0\},\ \forall\,(x_1\dots x_n)\in\mathbb{R}^n,\ \ |\mathcal{M}\,(x_1\dots x_n)|\geq 1.$

We formally define the coordinate–wise *median*, noted M, as follows:

$$\forall\,(d,n)\in(\mathbb{N}-\{0\})^2,\ \forall\,(x_1\dots x_n)\in\left(\mathbb{R}^d\right)^n,\ M\,(x_1\dots x_n)\triangleq x_s\in\mathbb{R}^d$$

such that:

$$\forall i\in[1\,..\,d],\ x_s[i]=\mathcal{M}\,(x_1[i]\dots x_n[i])$$

8 GuanYu

8.1 Constants and notations

Let $\left(n,\overline{n},f,\overline{f},q,\overline{q},d\right)\in(\mathbb{N}-\{0\})^7$, each representing:

- $n\geq 3f+3$ the total number of parameter servers, among which f are Byzantine
- $\overline{n}$ the total number of workers, among which $\overline{f}$ are Byzantine
- $2f+3\leq q\leq n-f$ the *quorum* (see Section 8.3) used for M
  $2\overline{f}+3\leq\overline{q}\leq\overline{n}-\overline{f}$ the *quorum* used for F
- d the dimension of the parameter space $\mathbb{R}^d$

Let (without loss of generality):

- $[1\,..\,n-f]$ be indexes of non–Byzantine parameter servers
  $[n-f+1\,..\,n]$ be indexes of Byzantine parameter servers
- $\left[1\,..\,\overline{n}-\overline{f}\right]$ be indexes of non–Byzantine workers
  $\left[\overline{n}-\overline{f}+1\,..\,\overline{n}\right]$ be indexes of Byzantine workers

Let $\theta_t^{(i)}$ be a notation for the parameter vector at parameter server $i\in[1\,..\,n-f]$ for step $t\in\mathbb{N}$.

Let $G_t^{(i)}$ be a notation for the gradient distribution at worker $i\in\left[1\,..\,\overline{n}-\overline{f}\right]$ for step $t\in\mathbb{N}$.

Let L be the loss function we aim to minimize, let $\nabla L\,(\theta)$ be the real gradient of the loss function at θ, and let $\widehat{\nabla L}\,(\theta)$ be a stochastic estimation of the gradient, following G, of L at θ.

Let $\xi_t \times (k)^{(a)}$ denote the subset of size k of some set $\left\{\xi_t^{(1)} \dots \xi_t^{(n)}\right\}$ delivered by node a at step t. To highlight the fact that such a subset can contain up to f arbitrary (Byzantine) vectors, we will also denote it by $\left(\xi_t \times (k-f)^{(a)}, \xi_t \times (f)^{(a)}\right)$. Also, the exact value of n depends on the context: in the proof, it will always be $\overline{n}$ when $\xi_t^{(i)}$ denotes a gradient, and n otherwise.

8.2 Initialization

Each non–Byzantine parameter server i starts (at step $t = 0$) with the same parameter vector:

$$\forall i \in [1 \,..\, n-f]\,,\ \theta_0^{(i)} \triangleq \theta_0$$

8.3 Proceeding of step $t \in \mathbb{N}$

Parameter server		**Worker**
$\theta_t^{(i)}$	$\overset{q}{\rightarrow}$	
		$g_t^{(j)} = \widehat{\nabla L}\left(M\left(\theta_t^{(a)} \dots \theta_t^{(b)}\right)\right) \sim G_t^{(j)}$
	$\overset{\overline{q}}{\leftarrow}$	
$\theta_t^{(i)} - \eta_t\, F\left(g_t^{(x)} \dots g_t^{(y)}\right) = \overline{\theta}_{t+1}^{(i)}$		
	$\curvearrowright q$	
$M\left(\overline{\theta}_{t+1}^{(z)} \dots \overline{\theta}_{t+1}^{(w)}\right) = \theta_{t+1}^{(i)}$		
	$\overset{q}{\rightarrow}$	
		$\cdots\cdots\cdots$

- $\overset{x}{\rightarrow}$: each *non–Byzantine* PS broadcasts their message to all the workers
  each *non–Byzantine* worker only waits for the first x messages it receives
- $\overset{x}{\leftarrow}$: each *non–Byzantine* worker broadcasts their message to all the PS
  each *non–Byzantine* PS only waits for the first x messages it receives
- $\curvearrowright^x$: each *non–Byzantine* PS broadcasts their message to all the PS
  each *non–Byzantine* PS only waits for the first x messages it received

9 Convergence

9.1 Formulation

Assuming that[9]:

1. L is Lipschitz continuous, i.e. $\exists l > 0,\ \forall (x, y) \in \left(\mathbb{R}^d\right)^2,\ \|\nabla L(x) - \nabla L(y)\| \leq l \|x - y\|$
2. $\exists t_s \in \mathbb{N},\ \forall t > t_s,\ \exists \left(\delta_t^{(1)} \dots \delta_t^{(\alpha)}, r_t^{(1)} \dots r_t^{(\alpha)}\right) \in \mathbb{R}^\alpha \times \left(\mathbb{R}^{d^2}\right)^\alpha$ (see Section 9.2.3) , $\exists (u_t, o_t) \in \mathbb{R}^d \times \mathbb{R}^d,\ \forall i \in [1 \,..\, n - f],\ a_t^{(i)} \sim \mathcal{N}\left(0, \delta_t^{(i)}\right),\ b_t^{(i)} \sim \mathcal{N}\left(o_t, r_t^{(i)}\right)$, with $a_t^{(1)} \dots a_t^{(\alpha)},\ b_t^{(1)} \dots b_t^{(\alpha)}$ mutually independent and $\overline{\theta}_t^{(i)} = a_t^{(i)} \cdot u_t + b_t^{(i)}$
3. $\forall t \in \mathbb{N},\ g_t^{(1)} \dots g_t^{\left(\overline{n} - \overline{f}\right)}$ are mutually independent
4. $\exists \sigma' \in \mathbb{R}_+,\ \forall (i, t) \in \left[1 \,..\, \overline{n} - \overline{f}\right] \times \mathbb{N},\ \mathbb{E}\left\|g_t^{(i)} - \mathbb{E}\, g_t^{(i)}\right\| \leq \sigma'$
5. L is positive, and 3–times differentiable with continuous derivatives
6. the sequence of learning rates η_t satisfies $\sum_t \eta_t = +\infty$ and $\sum_t {\eta_t}^2 < +\infty$
7. $\forall (i, t) \in \left[1 \,..\, \overline{n} - \overline{f}\right] \times \mathbb{N},\ \theta_t^{(i)} \triangleq M\left(\theta_t^{(a)} \dots \theta_t^{(b)}\right),\ \mathbb{E}\, g_t^{(i)} = \nabla L\left(\theta_t^{(i)}\right)$, where $\theta_t^{(a)} \dots \theta_t^{(b)}$ represents the q first received parameter vectors from the parameter servers by worker i.
8. $\forall r \in [2 \,..\, 4],\ \exists (A_r, B_r) \in \mathbb{R}^2,\ \forall (i, t, \theta) \in \left[1 \,..\, \overline{n} - \overline{f}\right] \times \mathbb{N} \times \mathbb{R}^d,\ \mathbb{E}\left\|g_t^{(i)}\right\| \leq A_r + B_r \|\theta\|^r$
9. $\exists \gamma \in [0, \pi/2[,\ \forall (i, t, \theta) \in \left[1 \,..\, \overline{n} - \overline{f}\right] \times \mathbb{N} \times \mathbb{R}^d,\ \kappa \sqrt{\mathbb{E}\left(\left\|g_t^{(i)} - \mathbb{E}\, g_t^{(i)}\right\|^2\right)} \leq \|\nabla L(\theta)\| \sin(\gamma)$
10. $\exists D \in \mathbb{R},\ \forall \theta \in \mathbb{R}^d,\ \|\theta\|^2 \geq D,\ \exists (\varepsilon, \beta) \in \mathbb{R}_+ \times [0, \pi/2 - \gamma[,\ \|\nabla L(\theta)\| \geq \varepsilon,\ \langle \theta, \nabla L(\theta) \rangle \geq \cos(\beta)\ \|\theta\|\ \|\nabla L(\theta)\|$

where $\exists k \in \,]1, +\infty[$:

$$\kappa \triangleq k \sqrt{2\left(\overline{n} - \overline{f} + \frac{\overline{f}\left(\overline{n} - \overline{f} - 2\right) + \overline{f}^2\left(\overline{n} - \overline{f} - 1\right)}{\overline{n} - 2\,\overline{f} - 2}\right)}$$

[9] Assumptions 3 to 10 are directly imported or adapted from [10].

We prove that $\forall t \in \mathbb{N},\ \forall \left(\theta_t^{(n-f+1)} \dots \theta_t^{(n)}\right) \in \left(\mathbb{R}^d\right)^f,\ \theta_t \triangleq M\left(\theta_t^{(1)} \dots \theta_t^{(n)}\right)$:

$$\lim_{t \to +\infty} \mathbb{E} \left\| \nabla L\left(\theta_t\right) \right\| = 0 \tag{2}$$

9.2 Lemmas

9.2.1 Convergence of $\sum_{i=0}^{n} k^{n-i}\,\eta_i$

Let $k \in [0, 1[$ and $\eta_i > 0$ be the general term of a sequence such that $\lim_{i \to +\infty} \eta_i = 0$.

Then:

$$\lim_{n \to +\infty} \left(\sum_{i=0}^{n} k^{n-i}\,\eta_i \right) = 0 \tag{3}$$

Proof. First, we observe that:

$$\begin{aligned} \lim_{t \to +\infty} \eta_t = 0 &\implies \forall \varepsilon > 0,\ \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \eta_t < \varepsilon \\ &\implies \exists \tau \in \mathbb{N},\ \forall t \geq \tau,\ \eta_t < 1 \end{aligned} \tag{4}$$

Then, reusing τ from (4), it holds $\forall n \geq 2\,\tau$:

$$\begin{aligned} \sum_{i=0}^{n} k^{n-i}\,\eta_i &= \sum_{i=0}^{\tau-1} k^{n-i}\,\eta_i + \sum_{i=\tau}^{\lfloor n/2 \rfloor} k^{n-i}\,\eta_i + \sum_{i=\lfloor n/2 \rfloor+1}^{n} k^{n-i}\,\eta_i \\ &< k^{n-\tau} \left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i \right) + \sum_{i=\tau}^{\lfloor n/2 \rfloor} k^{n-i} + \sum_{i=\lfloor n/2 \rfloor+1}^{n} k^{n-i}\,\eta_i \\ &< k^{n-\tau} \left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i \right) + k^{\lceil n/2 \rceil} \sum_{i=\tau}^{\lfloor n/2 \rfloor} k^{\lfloor n/2 \rfloor - i} + \max_{i \in \left[\frac{n}{2}+1\,..\,n\right]} (\eta_i) \sum_{i=\lfloor n/2 \rfloor+1}^{n} k^{n-i} \\ &< k^{n-\tau} \left(\sum_{i=0}^{\tau-1} k^{\tau-i}\,\eta_i \right) + \frac{1}{1-k} \left(k^{\lceil n/2 \rceil} + \max_{i \in \left[\frac{n}{2}+1\,..\,n\right]} (\eta_i) \right) \end{aligned}$$

Finally, since $k \in]0, 1[$ and $\lim_{n \to +\infty} \left(\max_{i \in \left[\frac{n}{2}+1\,..\,n\right]} (\eta_i) \right) = \lim_{n \to +\infty} (\eta_n) = 0$, we conclude:

$$\lim_{n \to +\infty} \left(\sum_{i=0}^{n} k^{n-i}\,\eta_i \right) = 0$$

□

9.2.2 *Multi–Krum* bounded deviation from majority

Let $(d, f) \in (\mathbb{N} - \{0\})^2$, let $q \in \mathbb{N}$ such that $q \geq 2\,f + 3$, and let note $\alpha \triangleq 2\,q$.

Let note $H_1 \triangleq [1\,..\,q - f]$ and $H_2 \triangleq [q + 1\,..\,2\,q - f]$.

We will show that:

$$\exists c' \in \mathbb{R}_+,\ \forall (x_1 \dots x_q) \in \left(\mathbb{R}^d\right)^q,\ \ \|F\,(x_1 \dots x_q)\| \leq c' \max_{(i,j) \in {H_1}^2} \|x_i - x_j\|$$

One immediate corollary follows:

$$\exists c \in \mathbb{R}_+,\ \forall (x_1 \dots x_\alpha) \in \left(\mathbb{R}^d\right)^\alpha,\ \ \|F\,(x_1 \dots x_q) - F\,(x_{q+1} \dots x_\alpha)\| \leq c \max_{(i,j) \in {(H_1 \cup H_2)}^2} \|x_i - x_j\|$$

Proof. We will proceed by construction of F (Section 7.1).

Let note $B_1 \triangleq [q - f + 1\,..\,q]$, and observe that:

$$|B_1| < q - f - 2 \leq |H_1|$$

Noting by $s\,(x_i)$ the score of input vector x_i, we then observe that:

$$\forall i \in H_1,\ s\,(x_i) \leq (q - f) \max_{(i,j) \in {H_1}^2} \|x_i - x_j\|$$

So, to be selected, an input vector indexed by $b \in B_1$ (the "Byzantine group") must satisfies:

$$s\,(x_b) \leq (q - f) \max_{(i,j) \in {H_1}^2} \|x_i - x_j\|$$

Also, because $|B_1| < q - f - 2$, we observe that:

$$\forall b \in B_1,\ s\,(x_b) \geq \min_{i \in H_1} \|x_b - x_i\|$$

Hence, since the output vector is the arithmetic mean of some input vectors in $\{x_1 \dots x_q\}$, any input vector indexed by $b \in B_1$ selected in the arithmetic mean is bounded by:

$$\min_{i \in H_1} \|x_b - x_i\| \leq (q - f) \max_{(i,j) \in {H_1}^2} \|x_i - x_j\|$$

And we conclude:

$$\exists c' \in \mathbb{R}_+,\ \forall (x_1 \dots x_q) \in \left(\mathbb{R}^d\right)^q,\ \ \|F\,(x_1 \dots x_q)\| \leq c' \max_{(i,j) \in {H_1}^2} \|x_i - x_j\|$$

The corollary follows immediately since:

$$\|F(x_1 \dots x_q) - F(x_{q+1} \dots x_\alpha)\| \leq \|F(x_1 \dots x_q)\| + \|F(x_{q+1} \dots x_\alpha)\|$$

And $\forall k \in \{1, 2\}$:

$$\max_{(i,j) \in H_k{}^2} \|x_i - x_j\| \leq \max_{(i,j) \in (H_1 \cup H_2)^2} \|x_i - x_j\|$$

□

9.2.3 Coordinate–wise *median* contraction effect

Let $(d, f) \in (\mathbb{N} - \{0\})^2$, let $(n, q) \in \mathbb{N}^2$ such that $3f + 3 \leq n$ and $2f + 3 \leq q \leq n - f$.

Let note $\alpha \triangleq n - f$ and $\beta \triangleq q - f$.

Then:

$$\exists (\delta_1 \dots \delta_\alpha, r_1 \dots r_\alpha) \in \mathbb{R}^\alpha \times \left(\mathbb{R}^{d^2}\right)^\alpha$$

such that $\exists m \in [0, 1[$:

$$\forall (u, o) \in \mathbb{R}^d \times \mathbb{R}^d,\ \forall (y_1 \dots y_f, z_1 \dots z_f) \in \left(\mathbb{R}^d\right)^{2f}$$

noting:

$$\begin{aligned}
&a_1 \sim \mathcal{N}(0, \delta_1) \ \dots\ a_\alpha \sim \mathcal{N}(0, \delta_\alpha), \\
&b_1 \sim \mathcal{N}(o, r_1) \ \dots\ b_\alpha \sim \mathcal{N}(o, r_\alpha), \\
&\text{with } a_1 \dots a_\alpha,\ b_1 \dots b_\alpha \text{ mutually independent variables} \\
&x_1 \triangleq a_1 \cdot u + b_1 \ \dots\ x_\alpha \triangleq a_\alpha \cdot u + b_\alpha \\
&y \triangleq M(x_1 \dots x_\beta, y_1 \dots y_f),\ z \triangleq M(x_{\alpha-\beta+1} \dots x_\alpha, z_1 \dots z_f)
\end{aligned}$$

we have:

$$\mathbb{E}\|y - z\| \leq m\ \mathbb{E}\left(\max_{(i,j) \in [1\,..\,\alpha]^2} \|x_i - x_j\|\right) \tag{5}$$

Proof and intuition. It is easy to prove this lemma when $r_1 = \dots = r_\alpha = 0$.

Indeed in this case, by construction of the coordinate–wise *median*, y belongs to the smallest rectangular parallelotope that contains all $x_1 \dots x_\beta$. By construction, we also observe that the diagonal length of this parallelotope is always $\max_{(i,j) \in [1\,..\,\beta]^2} (\|x_i - x_j\|)$.

One property of a parallelotope P of diagonal length d is that:

$$\forall\,(v,w)\in P^2,\ \ \|v-w\|\leq d$$

Similarly, z belongs to the smallest rectangular parallelotope that contains all $x_{\alpha-\beta+1}\dots x_\alpha$, and since $a_1\dots a_\alpha$ are mutually independent, we conclude on the existence of $m\in[0,1[$.

The intuition is that this *contraction effect* still remain true when $x_1\dots x_\alpha$ are not aligned anymore. The term b_i expresses this misalignment. The open problem remains to find a *closed–form expression* that bounds the matrices $r_1\dots r_\alpha$ and guarantee the existence of this effect.

9.3 Proof of GuanYu

We derive a sufficient condition for (2). Namely, we observe that:

$$\left.\begin{array}{r}\lim\limits_{t\to+\infty}\mathbb{E}\left(\max\limits_{(a,b)\in[1\,..\,n-f]^2}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right)=0\\ \lim\limits_{t\to+\infty}\mathbb{E}\left\|\nabla L\left(\theta_t^{(1)}\right)\right\|=0\end{array}\right\}\implies(2)$$

The first proposition will be proven in Section 9.3.1, and the second proposition in Section 9.3.2. The above implication is proven below.

Indeed, by one of the triangular inequalities and Lipschitz continuity, we have:

$$\begin{aligned}\forall t\in\mathbb{N},\ \theta_t\triangleq M\left(\theta_t^{(1)}\dots\theta_t^{(n)}\right),\ \left|\|\nabla L\left(\theta_t\right)\|-\left\|\nabla L\left(\theta_t^{(1)}\right)\right\|\right|&\leq\left\|\nabla L\left(\theta_t\right)-\nabla L\left(\theta_t^{(1)}\right)\right\|\\&\leq l\left\|\theta_t-\theta_t^{(1)}\right\|\end{aligned}$$

Hence, using the above t and θ_t, we get:

$$\mathbb{E}\left\|\nabla L\left(\theta_t^{(1)}\right)\right\|-l\,\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\|\leq\mathbb{E}\left\|\nabla L\left(\theta_t\right)\right\|\leq\mathbb{E}\left\|\nabla L\left(\theta_t^{(1)}\right)\right\|+l\,\mathbb{E}\left\|\theta_t-\theta_t^{(1)}\right\|$$

We now recall that, by construction of the coordinate–wise *median*:

$$\theta_t\triangleq M\left(\theta_t^{(1)}\dots\theta_t^{(n)}\right),\ \forall i\in[1\,..\,d],\ \left|\theta_t[i]-\theta_t^{(1)}[i]\right|\leq\max_{(a,b)\in[1\,..\,n-f]^2}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|$$

Hence, observing that:

$$\sum_{i=1}^{d}\left(\max_{(a,b)\in[1\,..\,n-f]^2}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|\right)\leq\frac{(n-f)\,(n-f-1)}{2}\max_{(a,b)\in[1\,..\,n-f]^2}\left(\sum_{i=1}^{d}\left|\theta_t^{(a)}[i]-\theta_t^{(b)}[i]\right|\right)$$

The equivalence of norms yields here:

$$\mathbb{E}\left\|\theta_t - \theta_t^{(1)}\right\| \leq \frac{\sqrt{d}\,(n-f)\,(n-f-1)}{2}\,\mathbb{E}\left(\max_{(a,b)\in[1\,..\,n-f]^2}\left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|\right)$$

Finally, using both the first and second propositions, we conclude:

$$\lim_{t\to+\infty}\mathbb{E}\left\|\nabla L\left(\theta_t\right)\right\| = 0$$

9.3.1 Almost–sure contraction of the non–Byzantine parameter vectors

We reuse the constant $m \in [0, 1[$ from the *contraction effect* of M (Section 9.2.3). Assumption 2 states that this contraction effect appears only after step t_s.

We reuse the constant $c \in \mathbb{R}_+$ from the *bounded deviation* lemma of F (Section 9.2.2).

Using the assumption that non–Byzantine gradient estimations' deviation is bounded (Assumption 4), since the number of non–Byzantine workers is also bounded, it holds that:

$$\exists\sigma \in \mathbb{R}_+,\ \forall t \in \mathbb{N},\ \ \mathbb{E}\left(\max_{i\in\left[1\,..\,\overline{n}-\overline{f}\right]}\left\|g_t^{(i)} - \mathbb{E}\,g_t^{(i)}\right\|\right) \leq \sigma$$

For step $t > t_s$:

$$\begin{aligned}
&\mathbb{E}\left(\max_{(a,b)\in[1\,..\,n-f]^2}\left\|\theta_t^{(a)} - \theta_t^{(b)}\right\|\right)\\
&\qquad\leq m\ \mathbb{E}\left(\max_{a,b}\left\|\theta_{t-1}^{(a)} - \theta_{t-1}^{(b)}\right\|\right) + c\,m\,\eta_{t-1}\,\mathbb{E}\left(\max_{a,b}\left\|G_{t-1}^{(a)} - G_{t-1}^{(b)}\right\|\right)\\
&\qquad\leq m\ \mathbb{E}\left(\max_{a,b}\left\|\theta_{t-1}^{(a)} - \theta_{t-1}^{(b)}\right\|\right) +\\
&\qquad\quad c\,m\,\eta_{t-1}\left(2\,\sigma + \mathbb{E}\left(\max_{(x,y)\in\left[1\,..\,\overline{n}-\overline{f}\right]^2}\left\|\nabla L\left(M\left(\theta_{t-1}\times(q-f)^{(x)}, \theta_{t-1}\times(f)^{(x)}\right)\right) -\right.\right.\right.\\
&\qquad\qquad\left.\left.\left.\nabla L\left(M\left(\theta_{t-1}\times(q-f)^{(y)}, \theta_{t-1}\times(f)^{(y)}\right)\right)\right\|\right)\right)\\
&\qquad\leq m\ \mathbb{E}\left(\max_{a,b}\left\|\theta_{t-1}^{(a)} - \theta_{t-1}^{(b)}\right\|\right) + c\,m\,\eta_{t-1}\left(2\,\sigma + l\,m\ \mathbb{E}\left(\max_{a,b}\left\|\theta_{t-1}^{(a)} - \theta_{t-1}^{(b)}\right\|\right)\right)
\end{aligned}$$

Noting $\mathbb{E}\left(\max_{a,b}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right) \triangleq 2\,\sigma\,c\,m\cdot u_t$, it holds for step $t > t_s$:

$$0 < u_t \leq m\,u_{t-1} + \left(l\,c\,m^2\right)\eta_{t-1}\,u_{t-1} + \eta_{t-1}$$

We now prove that $\lim_{t\to+\infty} u_t = 0$.

First, since $\sum_{t\in\mathbb{N}} {\eta_t}^2$ converges, we make the trivial observation that $\lim_{t\to+\infty}\eta_t = 0$, and:

$$\begin{aligned}\lim_{t\to+\infty}\eta_t = 0 &\implies \forall\varepsilon>0,\ \exists\tau\in\mathbb{N},\ \forall t\geq\tau,\ \eta_t<\varepsilon\\ &\implies \exists k\in\,]m;1[\,,\ \exists\tau\in\mathbb{N},\ \forall t\geq\tau,\ \eta_t<\frac{k-m}{l\,c\,m^2}\end{aligned}\tag{6}$$

Then, using k and τ from (6), it holds:

$$\begin{aligned}u_{\tau+1} &\leq m\,u_\tau + \left(l\,c\,m^2\right)\eta_\tau\,u_\tau + \eta_\tau\\ &\leq k\,u_\tau + \eta_\tau\end{aligned}$$

Similarly for step $\tau + t > \tau + t_s$:

$$\begin{aligned}u_{\tau+t} &\leq k\,u_{\tau+t-1} + \eta_{\tau+t-1}\\ u_{\tau+t} &\leq k\,u_{\tau+t-2} + k\,\eta_{\tau+t-2} + \eta_{\tau+t-1}\\ &\leq k^t\,u_\tau + \sum_{i=1}^{t} k^{i-1}\,\eta_{\tau+t-i}\end{aligned}$$

Finally, using Lemma (3), we conclude that:

$$\lim_{t\to+\infty}\mathbb{E}\left(\max_{(a,b)\in[1\,..\,n-f]^2}\left\|\theta_t^{(a)}-\theta_t^{(b)}\right\|\right) = \lim_{t\to+\infty} u_t = \lim_{t\to+\infty} u_{\tau+t} = 0$$

9.3.2 Almost–sure convergence argument of the loss function's gradient

A direct consequence of Subsection 9.3.1 is that beyond a certain time, the standard deviation of the gradient estimators, as well as the *drift* between the (non–Byzantine) parameter vectors can be bounded arbitrarily close to a single server situation. More precisely, let $\varepsilon > 0$ be any positive real number.

The following holds, and is a direct consequence of the limits stated above:

$$\exists t_\varepsilon \geq 0,\ \forall t \geq t_\varepsilon,\ \begin{cases} \mathbb{E}\left(\max\limits_{i \in \left[1\,..\,\overline{n}-\overline{f}\right]} \left\| g_t^{(i)} - \mathbb{E}\, g_t^{(1)} \right\|\right) \leq \sigma + \varepsilon \\ \mathbb{E}\left(\max\limits_{(a,b)\in[1\,..\,n-f]^2} \left\| \theta_t^{(a)} - \theta_t^{(b)} \right\|\right) < \frac{\varepsilon}{l} \end{cases}$$

The first inequality provides the bounded variance guarantee needed to plug GuanYu into the convergence proof of [10], the second inequality provides the remaining requirement, i.e. the bound on the statistical higher moments of the gradient estimator as if there was a *single* parameter. More precisely, we have the following, let (a, b) be any two non-Byzantine parameter servers, using a triangle inequality, the second inequality above and Assumption 8 (bounded moments) we have:

$$\begin{aligned} &\forall t > t_\varepsilon \forall r \in [2\,..\,4]\,,\ \exists\,(A_r, B_r) \in \mathbb{R}^2, \\ &\forall\,(i,t,\theta) \in \left[1\,..\,\overline{n}-\overline{f}\right] \times \mathbb{N} \times \mathbb{R}^d,\ \ \mathbb{E}\left\| g_t^{(a)} \right\| \leq A'_r + B'_r \left\| \theta_t^{(b)} \right\|^r \end{aligned}$$

where A'_r and B'_r are positive constants, depending polynomially (at most with degree r, by using the second inequality above, the triangle inequality and a binomial expansion) on ε, σ and (A_r, B_r) from Assumption 8, allowing us to use the convergence proof of [10] (Proposition 2) regardless of the identity of the (non–Byzantine) parameter server.

9.4 Practical evidence

In order to validate our assumptions in the practice, we did some micro–measurements to observe the alignment of the parameter vectors. In this section, we describe our methodology and results.

Methodology. We consider the quantities $\theta_t^{(i)}$, $\forall i \in [1\,..\,n-f]$ and $t > t_s$ (our assumption must hold *eventually*, e.g. after some large number of steps t_s). First, we calculate the differences between all parameter vectors (we name this *difference vectors*) and kept ones with the k highest norms. Then, we calculate the angle between these *difference vectors* to see how they are aligned. We do that every 20 steps, to empirically check whether our assumption keeps holding.

Results. Generally, we find that the angle between differences (from *difference vectors*) of the highest norms is always close to 0°. We give here the values of $\cos(\phi)$ where, ϕ is the angle between two difference vectors which are a and b. This is given by $\frac{a \cdot b}{\|a\|\|b\|}$. Thus, the closer this value to 1, the closer the angle to 0°. Sample results are given in Table 2. This is just a snippet from the results we found.

Table 2: This table depicts the experimental results we have while studying contraction of parameter vectors. It gives (at some steps) the biggest 2 norms of differences between parameter vectors collected by non–Byzantine parameter servers along with $\cos(\phi)$ where ϕ is the angle between these two difference vectors. This is recorded after some large step number.

Step	$\cos(\phi)$	max_diff_1	max_diff_2
1340	0.9822574257850647	1.4122562	1.4163861
1380	0.9926297664642334	1.3927394	1.3937825
1400	0.9881128072738647	1.493591	1.5035304
1440	0.9863847494125366	1.345111	1.3537675
1480	0.9819352030754089	1.3270174	1.3435347