Byzantine-Resilient Model Training

FID3024 - Module 4

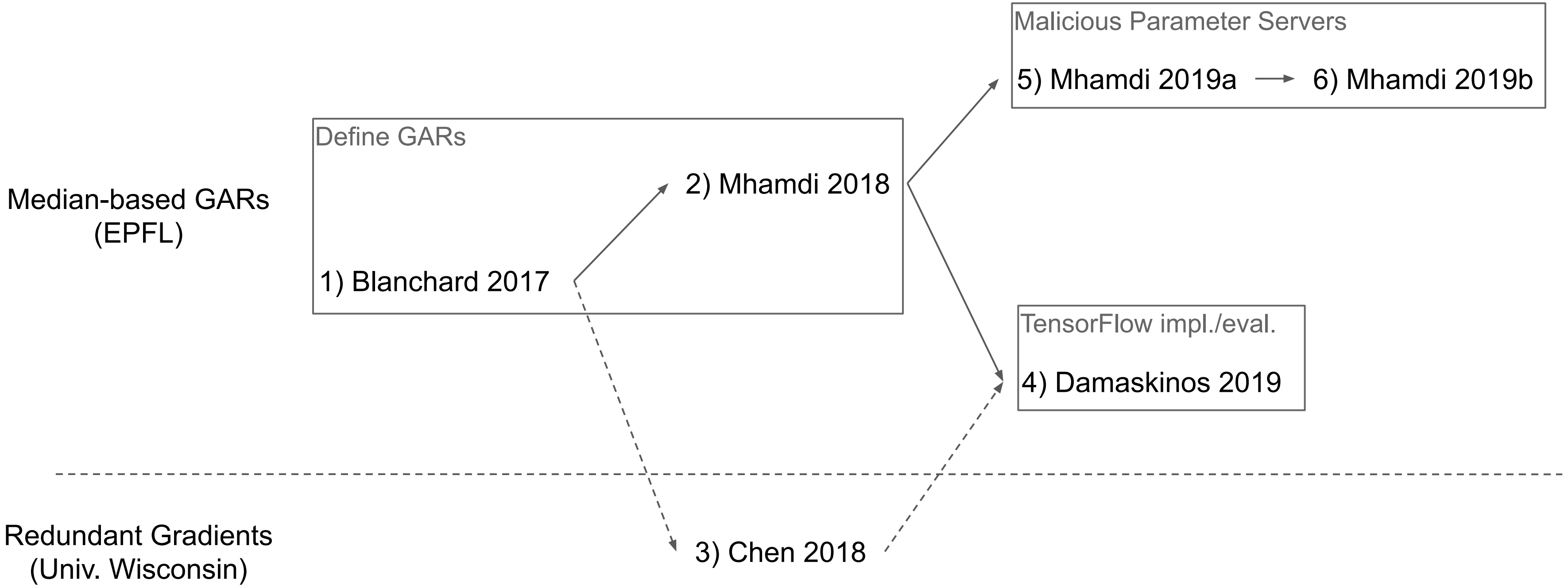
Mandi Chen, Lodovico Giaretta,
Daniel F. Perez-Ramirez

Robust Machine Learning

- Availability attacks
    - Prevent the inference system from working
- Confidentiality attacks
    - Extract sensitive information from the model
- Integrity attacks
    - Compromise the quality of the trained model

Omniscient malicious devices
within our data-parallel training environment

Papers Timeline



Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent

P. Blanchard, E. Mhamdi, R. Guerraoui, J. Steiner

NIPS 2017

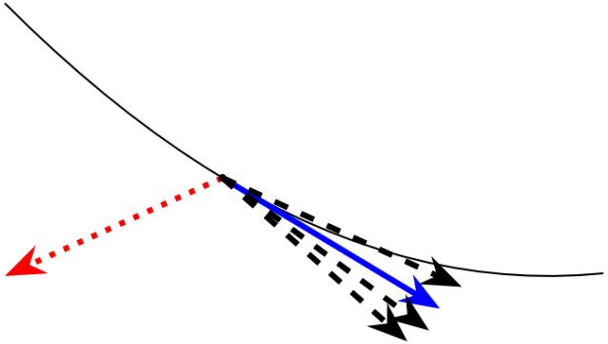
The Problem with SGD

- Data-parallel SGD aggregation is a linear combination of all gradients:

$$F(G_1, \ldots, G_n) = \sum_{i=1}^{N} \lambda_i G_i \qquad \forall i\ \lambda_i \neq 0$$

- A single malicious gradient $\boldsymbol{G}_n$ can undo all other gradients and replace them with a target gradient $\boldsymbol{U}$:

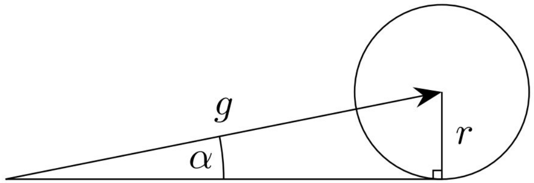
$$G_n = \frac{1}{\lambda_n}\left(U - \sum_{i=1}^{N-1} \lambda_i G_i\right) \Rightarrow F(G_1, \ldots, G_n) = U$$



We need a new Gradient Aggregation Rule (GAR)

A Definition of Byzantine Resilience

- A GAR is *(α, f)-Byzantine Resilient* iff:
  - Given f byzantine gradients
  - Outputs a gradient that deviates from the correct one ($\boldsymbol{g}$) by at most an angle α
  - Outputs a gradient whose moments are bound by those of the correct gradient $\boldsymbol{g}$



We need an (α, f)-Byzantine Resilient GAR

Krum: an (α, f)-Byzantine Resilient GAR

- Idea:
  - The $n - f$ non-byzantine gradients should form a tightly-packed cluster
  - Find a tightly-packed cluster of $n - f - 1$ gradients
  - Output the gradient that is closest to all others in this cluster

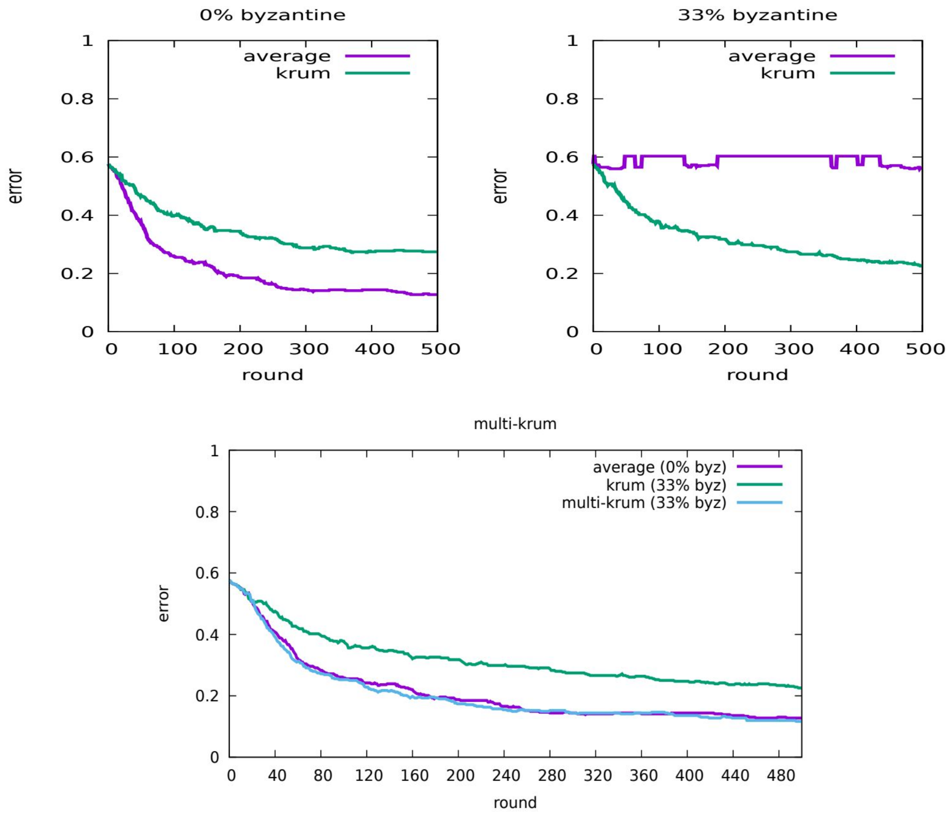
- Implementation:
  - Find the $n - f - 2$ closest $\boldsymbol{G_j}$ for every $\boldsymbol{G_i}$ (forming the $n - f - 1$ tightest cluster around $\boldsymbol{G_i}$)
  - Find the $\boldsymbol{G_i}$ with the tightest overall cluster by minimizing

$$s(G_i) = \sum_{i \to j} ||G_i - G_j||^2$$

  - Output $\boldsymbol{G_i}$

MultiKrum + Evaluation

- MultiKrum optimization:
  - Select k gradients instead of 1
  - Tradeoff between resiliency and convergence speed



Issues / Questions

- Why $n - f - 1$ gradients per cluster, instead of $n - f$?
- Why the moments of the output of the GAR must be bounded by those of the real gradient, up to the 4th order?
- How are resiliency and convergence speed affected by different choices of k in MultiKrum?

The Hidden Vulnerability of Distributed Learning in Byzantium

E. Mhamdi, R. Guerraoui, S. Rouault

ICML 2018

Brute: another (α, f)-Byzantine Resilient GAR

- Idea:
  - The n - f non-byzantine gradients should form a tightly-packed cluster
  - List all possible clusters of n - f gradients:

$$\mathcal{R} = \{\mathcal{X} \mid \mathcal{X} \subset \mathcal{G} \wedge |\mathcal{X}| = n - f\}$$

  - Find the most tightly-packed cluster:

$$S = \arg\min_{\mathcal{X} \in \mathcal{R}} \left(\max_{(V_1, V_2) \in \mathcal{X}^2} \left(||V_1 - V_2||_p \right) \right)$$

  - Average the elements of the cluster

A very expensive GAR...

The Problem with GARs

- Models are typically large: the dimensionality of the gradients is $d \gg 1$
- When $d \gg 1$, the l_p norms can hardly distinguish:
  - A small difference on each dimension
  - A large difference in a single dimension
- A malicious gradient can be very close to all good gradients according to the norm, but still have a very bad entry in one dimension
- If it gets selected, it is hard for SGD to converge to a good solution

A stronger resiliency guarantee is needed

The Solution: Bulyan

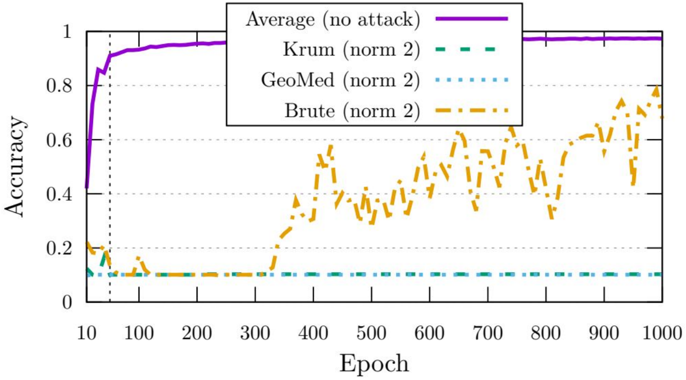
- Idea:
  - Act on each dimension independently
  - For each dimension, average β gradients that are around the median
  - With enough gradients, the median is bound by non-byzantine gradients

- Implementation:
  - Given $\theta \geq 2f + 3$ gradients, perform the following for each dimension
  - Select the $\beta = \theta - 2f \geq 3$ values closest to the median
  - Return their average

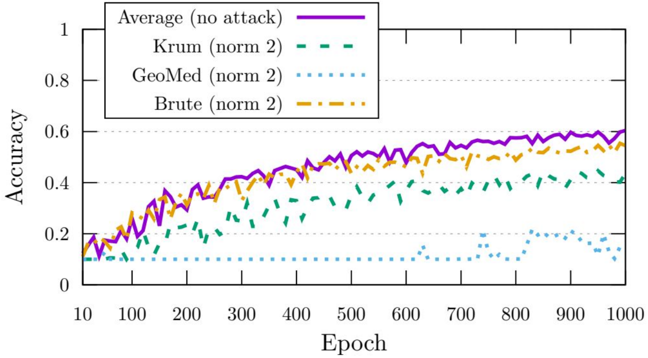
Bulyan: selecting θ gradients

- Bulyan
  - Requires $n \geq 4f + 3$ gradients
  - Requires an (α, f)-Byzantine Resilient GAR
  - Uses the GAR to iteratively select $\theta = n - 2f \geq 2f + 3$ gradients
- Why?
  - It seems that the quorum requirement would hold without this selection
  - Without this selection, a larger percentage of byzantine nodes can be tolerated
- Possible Reasons
  - (α, f)-Byzantine Resilient GAR guarantees that Bulyan is (α, f)-Byzantine Resilient ?
  - To speed up the computation? But is it better than random sampling?
  - Does it provide better results than Bulyan without any selection?

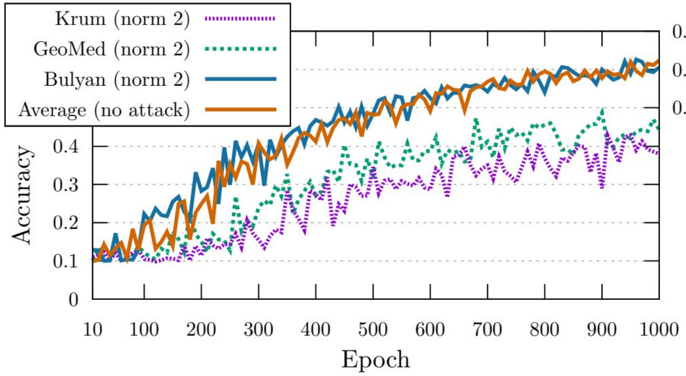
Evaluation



Average (no attack)
Krum (norm 2)
GeoMed (norm 2)
Brute (norm 2)
Accuracy
Epoch
0
0.2
0.4
0.6
0.8
1
10
100
200
300
400
500
600
700
800
900
1000



Average (no attack)
Krum (norm 2)
GeoMed (norm 2)
Brute (norm 2)
Accuracy
Epoch
0
0.2
0.4
0.6
0.8
1
10
100
200
300
400
500
600
700
800
900
1000



Krum (norm 2)
GeoMed (norm 2)
Bulyan (norm 2)
Average (no attack)
Accuracy
Epoch
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
10
100
200
300
400
500
600
700
800
900
1000

DRACO: Byzantine-resilient Distributed Training via Redundant Gradients

L.Chen, H.Wang, Z.Charles, D.Papailiopoulos

The Objective

We consider how to compute

$$\sum_{i=1}^{B} \mathbf{g}_i$$

in a distributed and *adversary-resistant* manner, assuming that adversarial nodes

- have access to infinite computational power, the entire data set, the training algorithm
- have knowledge of any defenses present in the system.
- may collaborate with each other.

Median-based approaches

- Pros: they can be robust to up to a constant fraction of the compute nodes being adversarial
- Cons:
  - convergence for such systems require restrictive assumptions such as convexity
  - need to be re-tailored to each different training algorithm
  - the geometric median aggregation may dominate the training time in large-scale settings.

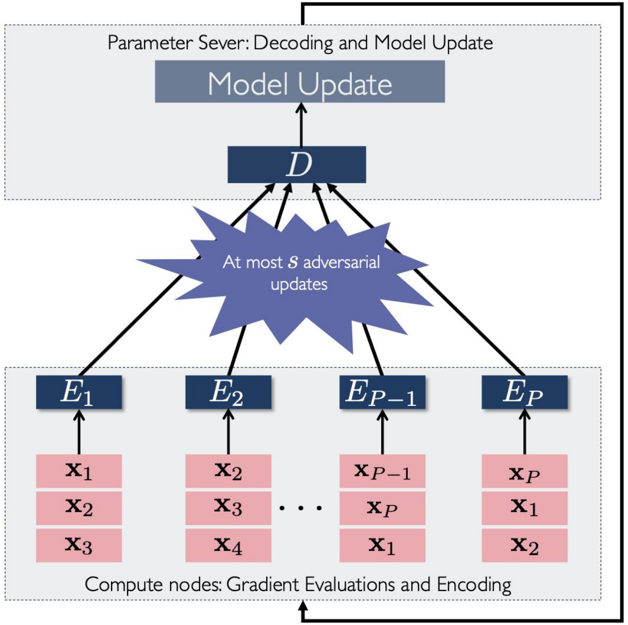
Solution: DRACO

Idea: use redundancy to guard against failures

Allocate **B gradients** to the **P compute nodes** using a P × B *allocation matrix* **A.**
the redundancy ratio $r \triangleq \frac{1}{P}\|\mathbf{A}\|_0$

To guarantee convergence, r must satisfy $r \geq 2s + 1$, where s is the number of adversarial nodes

1. Each worker processes **rB/P** gradients and sends an **encoded** linear combination of those to the PS.
2. After receiving the P gradient sums, the PS uses a **decoding function** to remove the effect of the adversarial nodes and reconstruct the original desired sum of the B gradients.



How to design A, E and D?

Encoding-decoding gradients

- The encoding schemes are based on the **fractional repetition code** and **cyclic repetition code**
- The decoding schemes utilize an efficient **majority vote decoder** and a novel **Fourier decoder**

Fractional repetition code with **majority vote decoder** $(\mathbf{A}^{Rep}, E^{Rep}, D^{Rep})$

Encoding stage:

1. $$\mathbf{A}^{Rep} = \begin{bmatrix} \mathbf{1}_{r\times r} & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} \\ \mathbf{0}_{r\times r} & \mathbf{1}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \mathbf{0}_{r\times r} & \cdots & \mathbf{0}_{r\times r} & \mathbf{1}_{r\times r} \end{bmatrix}.$$

2. $$\mathbf{Y}_j^{Rep} = \left(\mathbf{1}_d \mathbf{A}_{j,\cdot}^{Rep}\right) \odot \mathbf{G}.$$

3. $$E_j^{Rep}(\mathbf{Y}_j^{Rep}) = \mathbf{Y}_j^{Rep} \mathbf{1}_P.$$

4. $$\mathbf{z}_j = E_j^{Rep}(\mathbf{Y}_j^{Rep}) \Longrightarrow$$ Send to the PS

Decoding stage:

$$D^{Rep}(\mathbf{R}) = \sum_{\ell=1}^{\frac{P}{r}} Maj\left(\mathbf{R}_{\cdot,(\ell\cdot(r-1)+1):(\ell\cdot r)}\right).$$

Encoding-decoding gradients

Cyclic Code with **Fourier decoding** $(\mathbf{A}^{Cyc}, E^{Cyc}, D^{Cyc})$

Let **C** be a P × P inverse discrete Fourier transformation (IDFT) matrix

$$\mathbf{C}_{jk} = \frac{1}{\sqrt{P}} \exp\left(\frac{2\pi i}{P}(j-1)(k-1)\right), \quad j,k = 1,2,\cdots,P.$$

Let $\mathbf{C}_L$ be the first P−2s rows of **C** and $\mathbf{C}_R$ be the last 2s rows

Encoding stage:

$$\alpha_j = \{k : \mathbf{A}^{Cyc}_{j,k} = 0\}$$

$$\mathbf{0} = \begin{bmatrix}\mathbf{q}_j & 1\end{bmatrix} \cdot [\mathbf{C}_L]_{\cdot,\alpha_j}$$

$$\mathbf{Q} \triangleq \begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_P\end{bmatrix}$$

$$\mathbf{W} \triangleq \begin{bmatrix}\mathbf{Q} & \mathbf{1}_P\end{bmatrix} \cdot \mathbf{C}_L$$

$$\mathbf{Y}^{Cyc}_j = \left(\mathbf{1}_d \mathbf{A}^{Cyc}_{j,\cdot}\right) \odot \mathbf{G}$$

$$E^{Cyc}_j(\mathbf{Y}^{Cyc}_j) = \mathbf{G}\mathbf{W}_{\cdot,j}$$

$$\mathbf{z}^{Cyc}_j \triangleq E^{Cyc}_j(\mathbf{Y}^{Cyc}_j) \implies \text{Send to the PS}$$

Encoding-decoding gradients

Cyclic Code with **Fourier decoding** $(\mathbf{A}^{Cyc}, E^{Cyc}, D^{Cyc})$

Suppose there is a function *φ(·)* that can compute the adversarial node index set V

Decoding Stage:

1. $V = \phi(\mathbf{R})$
2. $U = \{1, 2, \cdots, P\} - V$
3. Find $\mathbf{b}$ by solving $\mathbf{W}_{\cdot,U}\mathbf{b} = \mathbf{1}_P$
4. $\mathbf{u}^{Cyc} = \mathbf{R}_{\cdot,U}\mathbf{b}$

This approach has linear-time in encoding and decoding

Experiments

Adversarial Attack Models:

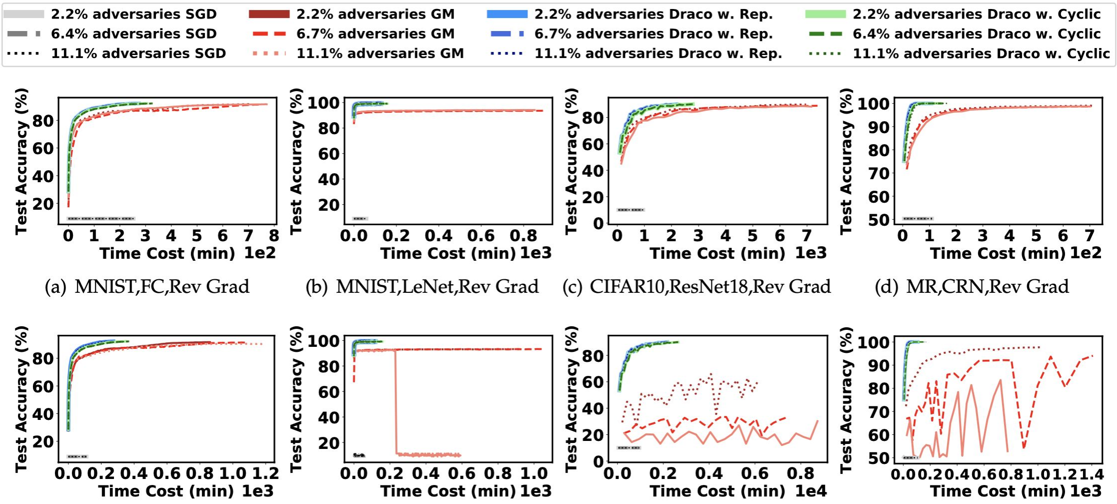
1. **Reversed gradient adversary** send $-c\mathbf{g}$ to PS, for some $c > 0$
2. **Constant adversary** send $\kappa = -100$

In either setup, at each iteration, s nodes are randomly selected to act as adversaries.

Compare DRACO against SGD and a GM approach(chen et. al 2017).

DRACO converges several times faster than the GM approach, using both the repetition and cyclic codes.

End-to-end Convergence Performance



(a) MNIST,FC,Rev Grad (b) MNIST,LeNet,Rev Grad (c) CIFAR10,ResNet18,Rev Grad (d) MR,CRN,Rev Grad

(e) MNIST,FC,Const (f) MNIST,LeNet,Const (g) CIFAR10,ResNet18,Const (h) MR,CRN,Const

Experiments

Per iteration cost of DRACO

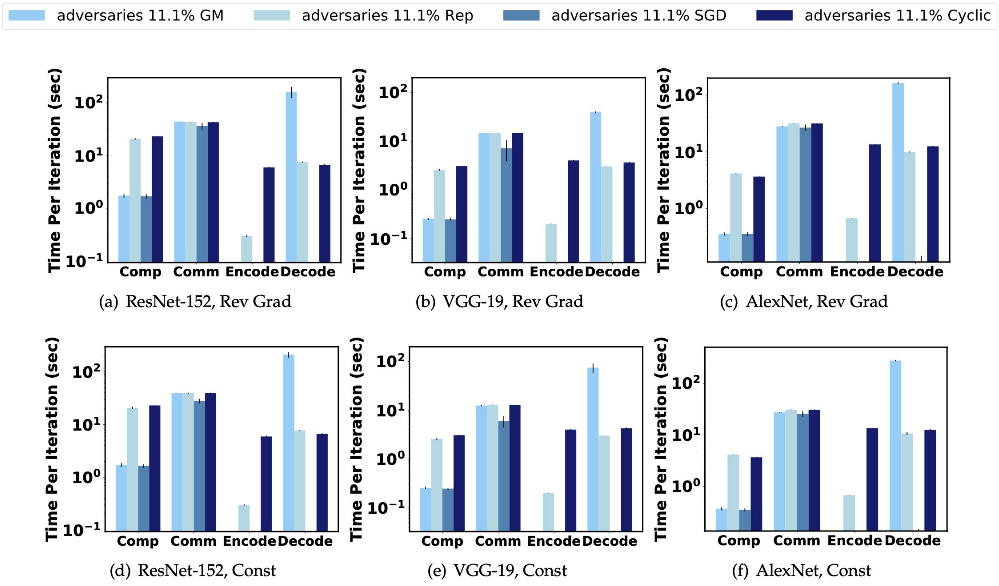
On ResNet-152, VGG-19, and AlexNet

Table 5: Averaged Per Iteration Time Costs on ResNet-152 with 11.1% adversary

Time Cost (sec)	Comp	Comm	Encode	Decode
GM const	1.72	39.74	0	212.31
Rep const	20.81	39.36	0.24	7.74
SGD const	1.64	27.99	0	0.09
Cyclic const	23.08	39.36	5.94	6.64
GM rev grad	1.73	43.98	0	161.29
Rep rev grad	20.71	42.86	0.29	7.54
SGD rev grad	1.69	36.27	0	0.09
Cyclic rev grad	23.08	42.86	5.95	6.65

Table 6: Averaged Per Iteration Time Costs on VGG-19 with 11.1% adversary

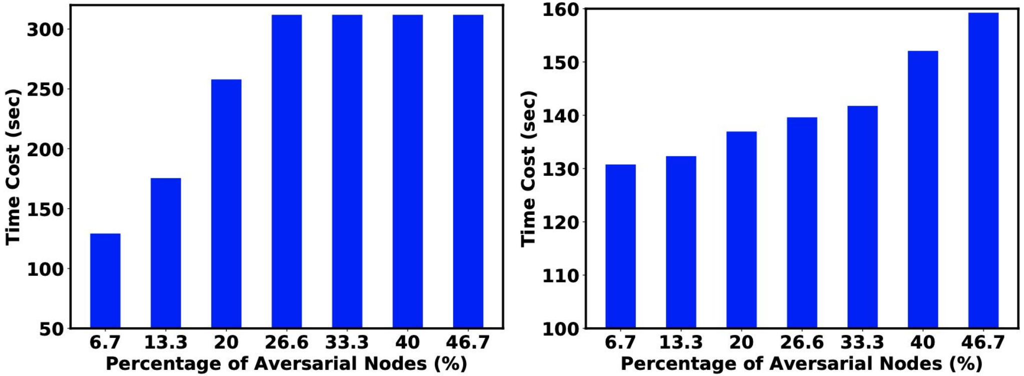
Time Cost (sec)	Comp	Comm	Encode	Decode
GM const	0.26	12.47	0	74.63
Rep const	2.59	12.91	0.20	3.03
SGD const	0.25	6.9	0	0.03
Cyclic const	3.08	12.91	4.01	4.30
GM rev grad	0.26	14.57	0	39.02
Rep rev grad	2.55	14.66	0.20	3.04
SGD rev grad	0.25	7.15	0	0.03
Cyclic rev grad	3.07	14.66	4.02	3.65



(a) ResNet-152, Rev Grad
(b) VGG-19, Rev Grad
(c) AlexNet, Rev Grad
(d) ResNet-152, Const
(e) VGG-19, Const
(f) AlexNet, Const

Experiments

Effects of number of adversaries



(a) Repetition Code (b) Cyclic Code

Summary

- DRACO can resist any s adversarial compute nodes during training and returns a model identical to the one trained in the adversary-free setup.
- In DRACO, most of the computational effort is carried through by the compute nodes. This allows the framework to offer up to orders of magnitude faster convergence in real distributed setups.
- With redundancy ratio r, DRACO can tolerate up to $(r - 1)/2$ adversaries, which is information–theoretically *tight*. Since in realistic regimes, only a constant number of nodes are malicious, DRACO is in general a fast approach.
- DRACO can be applied to any first-order methods, including gradient descent, SVRG, coordinate descent, and projected or accelerated versions of these algorithms.

Comments

- Comparison with Krum or Bulyan?
- Even for GM approach there is only one example

AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation

G. Damaskinos, E. Mhamdi, R. Guerraoui, A. Guirguis, S. Rouault

SysML 2019

Types of Byzantine Resilience

Weak BR

Any form of GAR that *almost surely* converges around a minima, despite the presence of ***f*** Byzantine workers. Ensures $\nabla Q(\boldsymbol{x}^*) = 0$ to some extent

(Multi-)Krum

Allowed dimensional leeway (in *d>>1*-dimensional vector space):

$$\|\breve{X} - Y\|_p = \boxed{\mathcal{O}(\sqrt[p]{d})}$$

Strong BR

Weak BR + reliable against the dimensional leeway.
Ensures not ending at a ‘bad’ optimum.

Bulyan

DRACO

Allowed dimensional leeway (in *d>>1*-dimensional vector space):

$$\boxed{\frac{1}{\sqrt{d}}}$$

Characteristics of GARs so far

	Required workers	Method	Privacy issues	Comparative Performance
m-Multi-Krum	$2f + 3$ With $m \leq n - f - 2$	“Median” (total squared distance)	+	?
Bulyan	$4f + 3$	Median (coordinate-wise)	+	?
DRACO	$2f + 1$	Gradient replication & coding scheme	+/-	?

Motivation for AggregaThor

Explicitly stated in the paper:

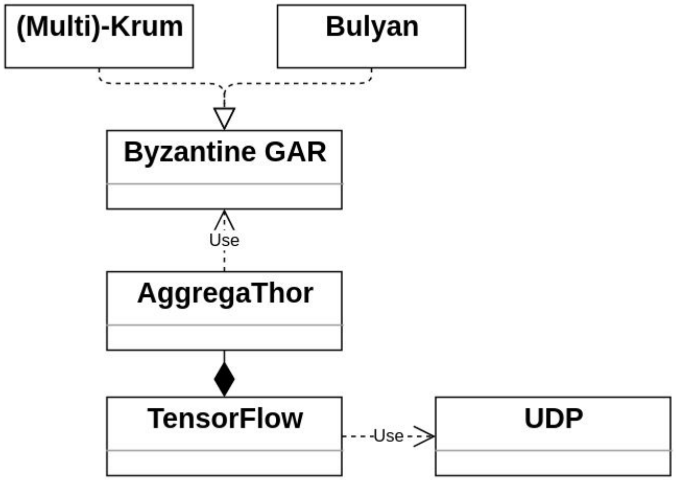
Implement previously proposed GAR in a **realistic** environment to test their practical **scalability**.

AggregaThor true (implicit) motivation:

The people from DRACO argue that their "*framework offers up to orders of magnitude faster convergence in real distributed setups*" compared to Median-based methods… Lets see if this holds true.

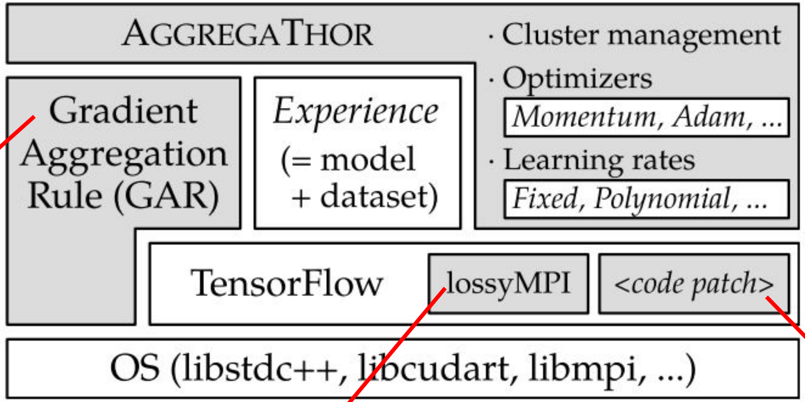
AggregaThor

Framework built on top of TensorFlow to implement state-of-the-art Byzantine resilience algorithms.



- Parameter server model
    - Assumes **correct** parameter server
- AggregaThor manages the deployment and execution of a model training session over a cluster of machines.
- Uses (unreliable) UDP for faster transfers

AggregaThor Design specifics



Modular integration of GARs

Other GARs can be used

Modify MPI communication points to employ UDP sockets

Only tf.train.Server instances can create and modify the graph

Instead of any node in the cluster.

Evaluation

- CIFAR-10 Dataset
- CNN with 1.75M parameters
- Metrics:
  - **Throughput**: total gradients received per second
  - **Classification accuracy**
- 19 workers and 1 PS

Non-Byzantine Env.

- Baseline: vanilla TF
- Against: AggregaThor (with Multi-Krum, Bulyan, Median method*, simple average) and DRACO.
- Includes scalability eval. on ResNet-50

Byzantine Env.

- Baseline: vanilla TF
- Against: AggregaThor
- Corrupt data
- Dropped packets

* Xie et al. Generalized Byzantine-tolerant SGD. 2018. Arxiv 1802.10116

Evaluation: Non-Byzantine Environment

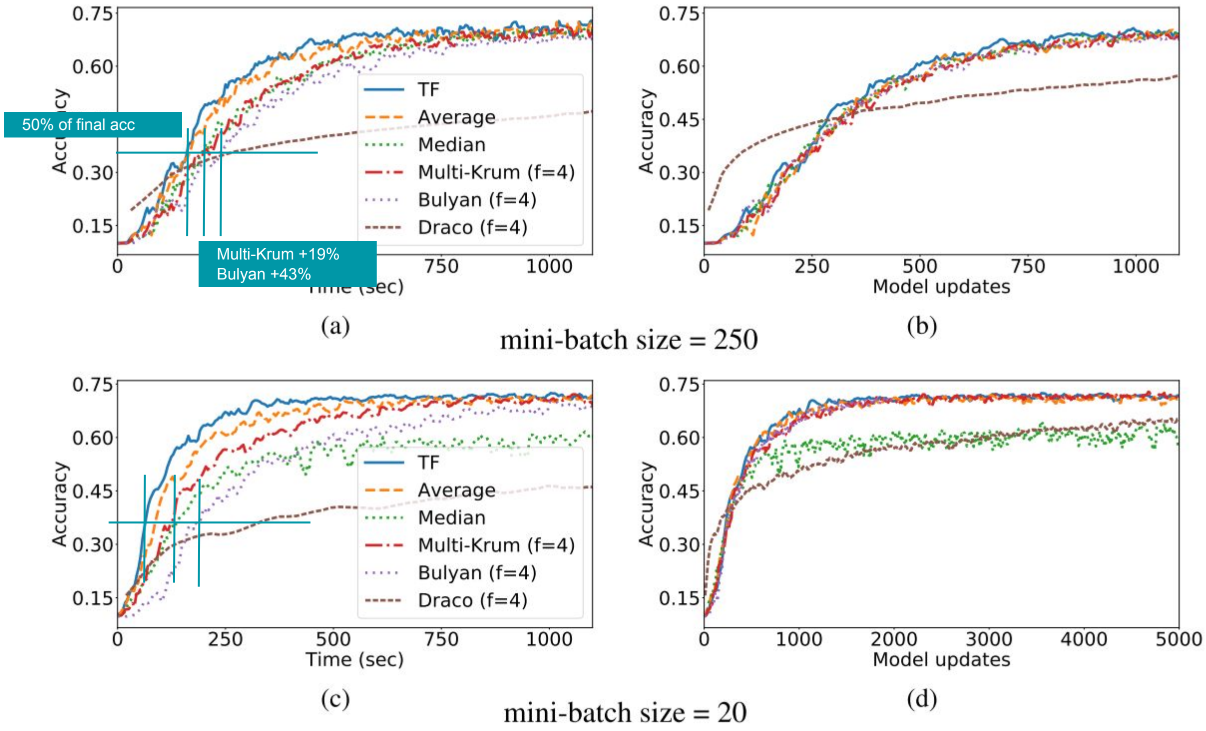


Figure 3. Overhead of AGGREGATHOR in a non-Byzantine environment.

* AggregaThor reaches (at some point) baseline acc

* DRACO as well, but takes longer time

 2f+1 more gradients required

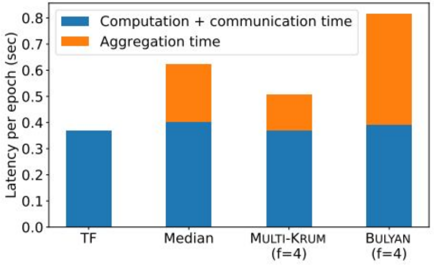
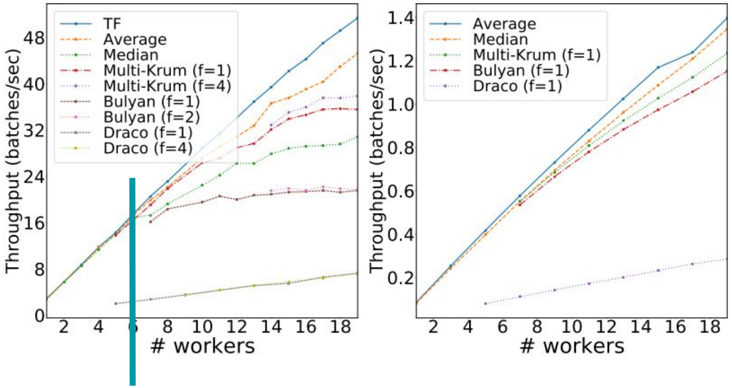


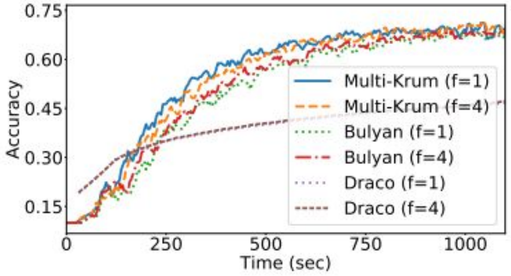
Figure 4. Latency breakdown.

Evaluation: Non-Byzantine Environment

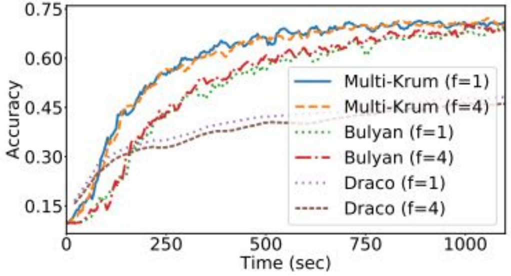


(a) CNN (b) ResNet50

Figure 5. Throughput comparison.



(a) mini-batch size = 250



(b) mini-batch size = 20

Figure 6. Impact of f on convergence.

Evaluation: Byzantine Environment

Corrupted Data

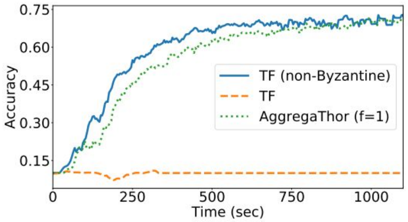


Figure 7. Impact of malformed input on convergence.

Mini-batch 250
(seems like same picture as 5.a)

Dropped packets

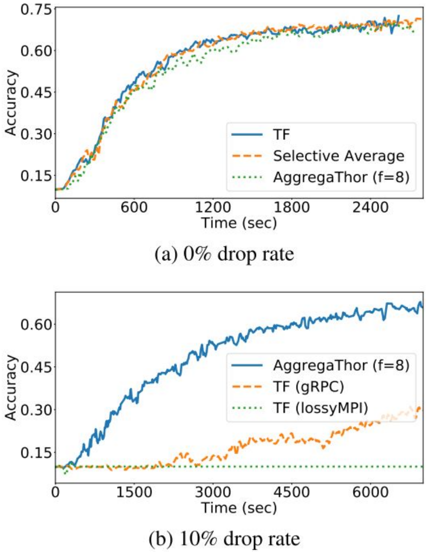


Figure 8. Impact of dropped packets on convergence.

Max. # of attackers (f = 8)

Concluding Remarks

- Authors argue that in practice, a weak Byzantine attack already requires a prohibitively large cost.
  - ≈ 10^{20} operations for 100 workers and vector precision of 10^{-9}.

    → Practitioners can use AggregaThor with just Multi-Krum in most cases

  - AggregaThor employs multi-aggregation rule: enable the server to leverage m > 1 workers in each step.
- BR against parameter server still an open issue

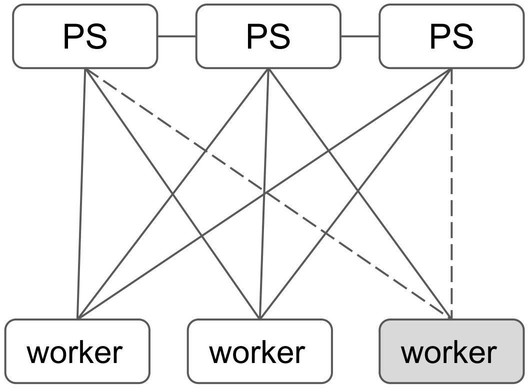
SGD: Decentralized Byzantine Resilience

E.Mhamdi, R.Guerraoui, A.Guirgui, S.Rouault

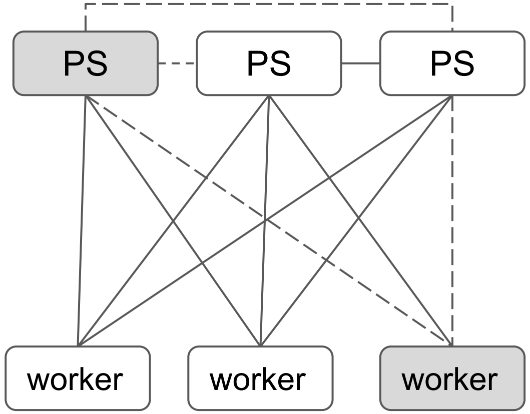
Motivation

Previous work assume the parameter server is free from malicious behavior, which is not necessary true.

Networks with Byzantine workers



Networks with Byzantine workers and parameter servers



GuanYu algorithm

F : Multi–Krum
M : coordinate–wise median
$2f + 3 \leq q \leq n - f$: the quorum used for M
$2\overline{f} + 3 \leq \overline{q} \leq \overline{n} - \overline{f}$: the quorum used for F

$\Longrightarrow$ GuanYu does not wait for all *n* nodes to start aggregation

Parameter server $\boldsymbol{i \in [1 \,..\, n - f]}$		**Worker** $\boldsymbol{j \in [1 \,..\, \overline{n} - \overline{f}]}$
$\theta_t^{(i)}$	$\xrightarrow{q}$	
		$g_t^{(j)} = \widehat{\nabla L}\left(M\left(\theta_t^{(a)} \ldots \theta_t^{(b)}\right)\right) \sim G_t^{(j)}$
Local update	$\xleftarrow{\overline{q}}$	
$\theta_t^{(i)} - \eta_t\, F\left(g_t^{(x)} \ldots g_t^{(y)}\right) = \underline{\overline{\theta}_{t+1}^{(i)}}$		
	$\circlearrowright_q$	
$M\left(\overline{\theta}_{t+1}^{(z)} \ldots \overline{\theta}_{t+1}^{(w)}\right) = \theta_{t+1}^{(i)}$	$\xrightarrow{q}$	
		....

Proof of convergence

Assumptions: on top of the case with one trusted parameter server, GuanYu assumes

1. ***L*** is Lipschitz continuous.
2. After some step $t_s \in N$, all the non–Byzantine parameter vectors are roughly aligned.

Intuitions:

1. Non–Byzantine parameter vectors gets almost–surely arbitrary close to each other after some step $t \in N$.
2. By the *contraction effect* of the **median** and assumption 1, if one non–Byzantine parameter vector converges, the others will get close to it.
3. Learning rate η_t converging toward 0.

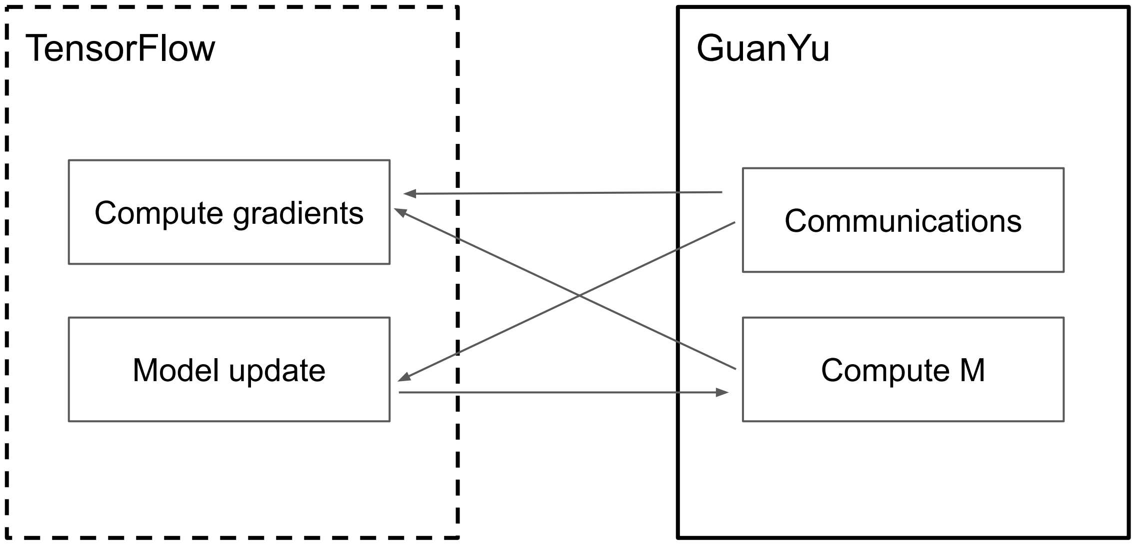
Stage 1 (before t_{inflex}):

- Byzantine parameter vectors (noises) pushes non–Byzantine parameter vectors away from each other.

Stage 2 (after t_{inflex}):

- The learning rate becomes small enough, the *contraction effect* pulls back together the non–Byzantine parameter vectors.

Implementation



Experiments

Setup:

- CIFAR-10 dataset
- CNN with 1.75M parameters, fixed batch size & learning rate
- up to 5/18 Byzantine in workers, ⅙ in parameter servers

Evaluation Metrics

- *Throughput* : measures the total number of updates that the deployed system can do per second.
- *Accuracy*: measures the top–1 cross–accuracy

Non–Byzantine Environment

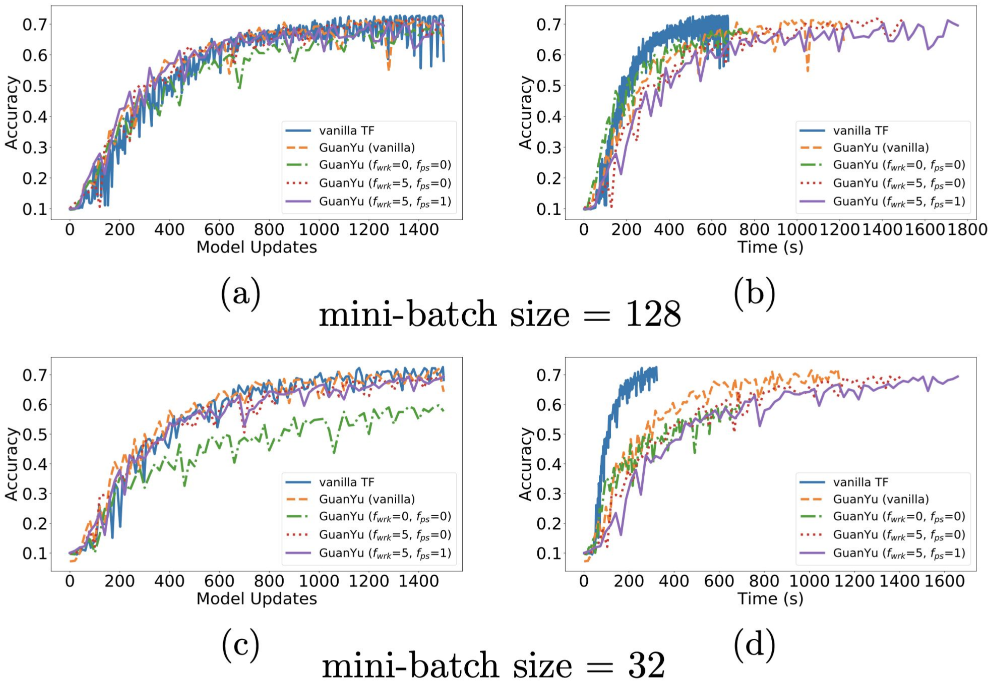


Figure 3: Overhead of GuanYu in a non-Byzantine environment.

More Byzantine players helps achieve a better convergence rate in terms of model updates, because increasing f forces servers to wait for more replies.

Non–Byzantine Environment

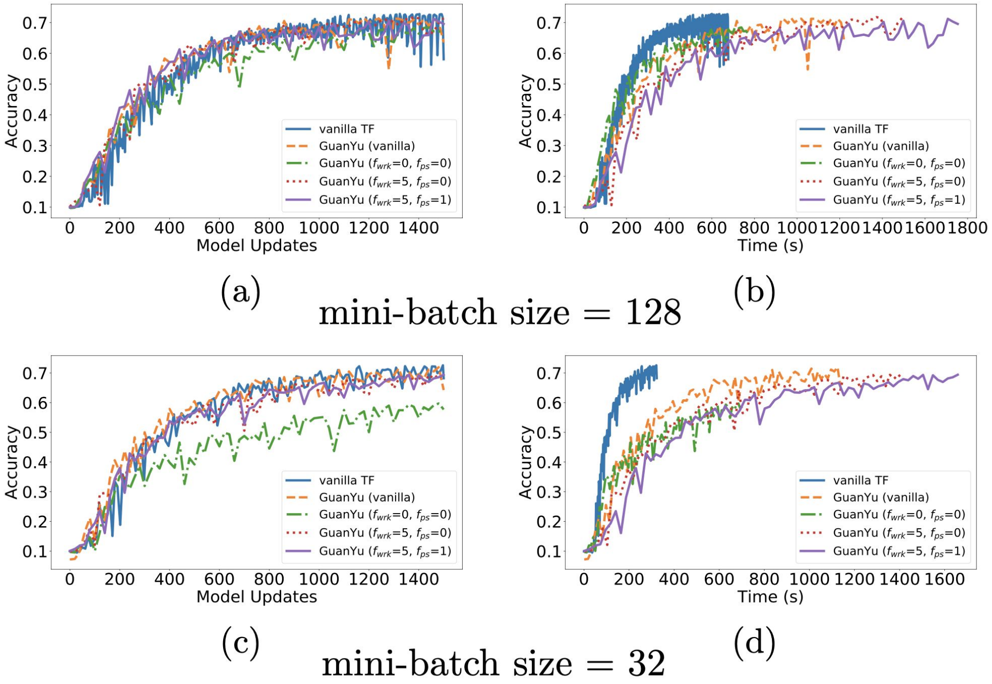


Figure 3: Overhead of GuanYuin a non-Byzantine environment.

Explanation on the overhead:

1. GuanYu uses rather naive implementations comparing to TensorFlow in device placement, communication and calculation operators.

2. Converting tensors to numpy arrays (and vice versa) and feeding tensors to a graph incur a big overhead.

Byzantine Environment

Types of Byzantine attack:

1. send corrupted gradients to parameter servers
2. send corrupted parameter vectors/model to workers
3. send different replies to different participants
4. not responding at all to requests

"We tested different possible Byzantine behaviors and we got approximately similar results"

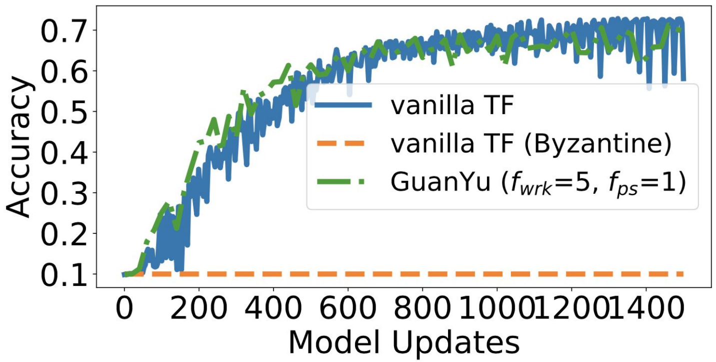


Figure 4: Impact of Byzantine players on convergence.

Conclusions and remarks

- GuanYu is the first approach that combines the resilience to **both Byzantine workers and Byzantine parameter servers**
- GuanYu guarantees convergence in environments up to 1/3 Byzantine servers and 1/3 Byzantine workers, which is **optimal in the asynchronous** setting.
- GuanYu has reasonable overhead compared to a non-Byzantine vanilla TensorFlow

Comments

1. Could have explored more NN architectures in the experiment. (e.g. LeNet, ResNet etc.)
2. GuanYu can tolerant 1/3 Byzantine servers, however, only 1/6 was tested in the experiment.
3. The runtime problem in converting tensor to numpy array might be possibly avoided?
4. Could have used better notations.

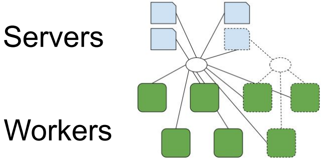
Fast Machine Learning with -Byzantine Workers and Servers

E. Mhamdi, R. Guerraoui, A. Guirguis

ACM Symposium on Principles of Distributed Computing (PODC) 2020

Motivation

GuanYu



Bulyan as GAR for workers' gradients

Median for models' aggregation from servers

- ➢ Each worker requires communicating with **majority** of servers for computing median
- ➢ Assumes **network asynchrony**: no bound on communication delays . But no free lunch…
  - Requires **3** communication rounds

    All-servers → **All**-workers **param**. broadcast → **All**-workers→ **All**-servers **gradient** broadcast → **All**-servers → **All**-servers **param**. broadcast

  - Assumes a **maximum distance** between parameter vectors (min. correct servers)

Desired

Total Byzantine resilience with....

- ➢ **Reduce** worker-server communication as far as possible.
- ➢ Is having **synchronous** communication too bad?
  → No, most param-server are synchronous
- ➢ Reduce **number** of communication rounds to mimic vanilla Parameter-server approach
  - Vanilla: **2** communication rounds

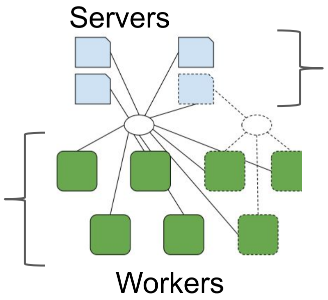
    Server → all-workers param. broadcast → All-workers→ Servers gradient unicast

LiuBei

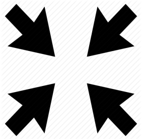
- Does not trust workers nor servers and adds (almost) no communication overhead.

Scatter

Wrokers work independently
and do not communicate

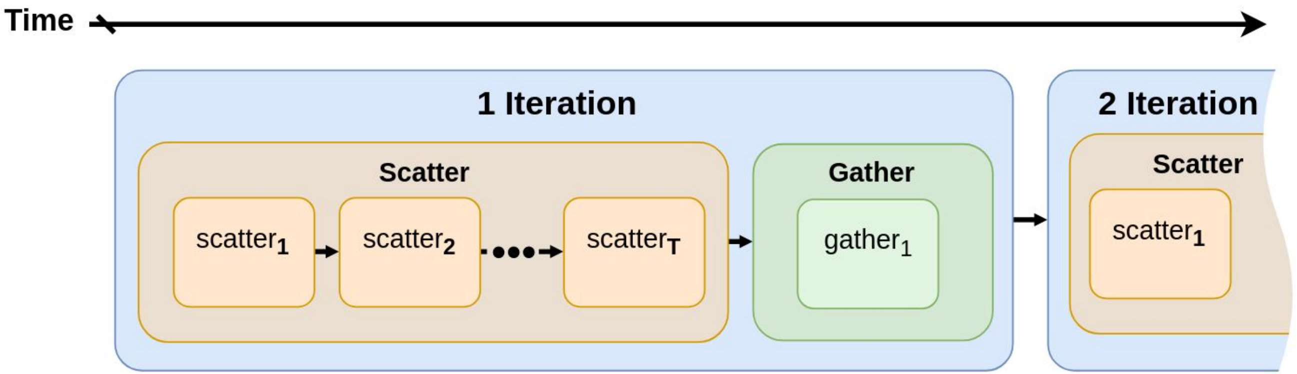


Gather



Correct servers communicate
to bring their view of models
back close to each other.

LiuBei - Steps



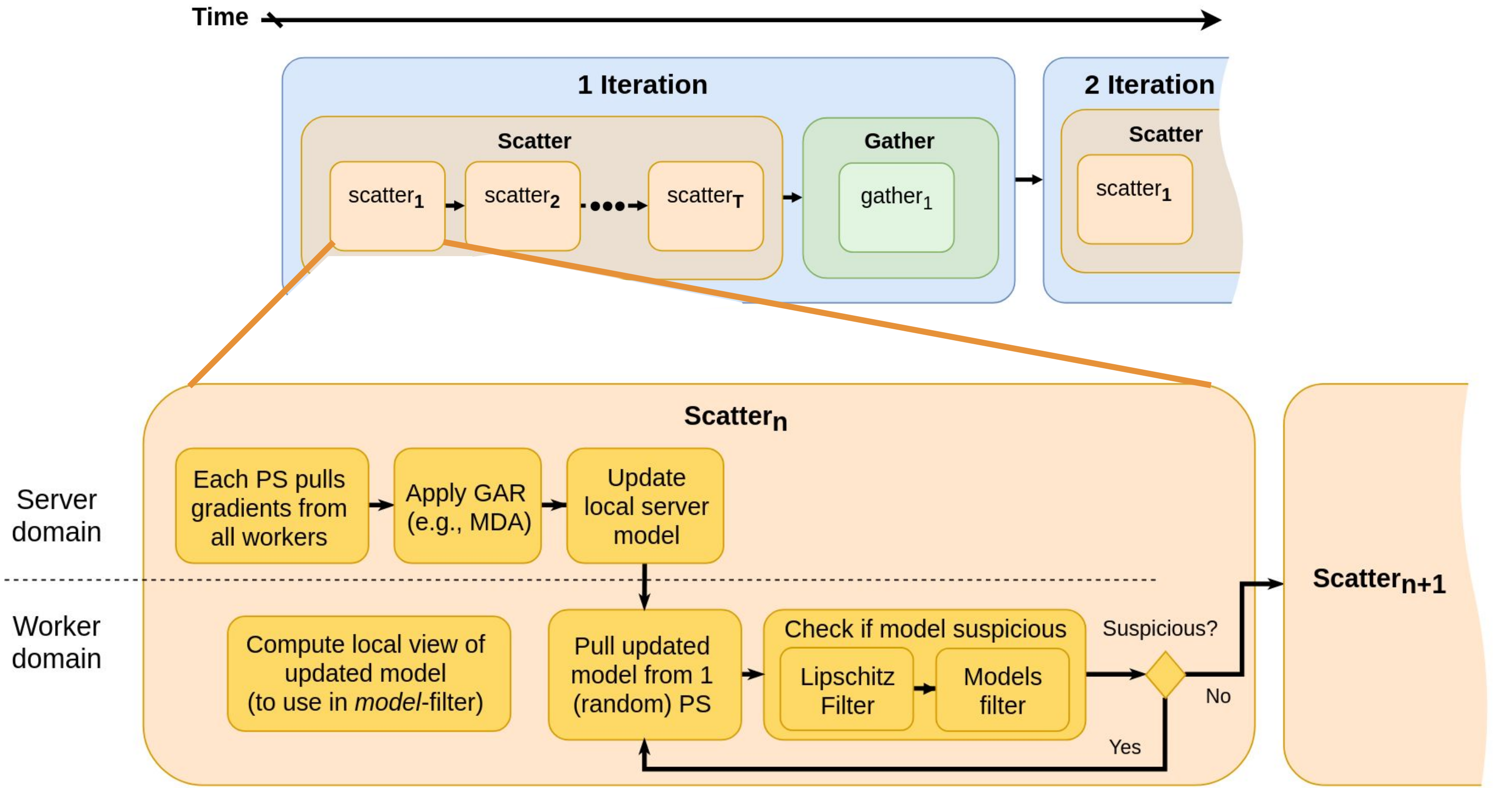
Scatter

Workers work independently and do not communicate

Gather

Correct servers communicate to bring their view of models back close to each other.

LiuBei - Gather



Gather - Lipschitz Filter

Limit the growth of the computed model updates w.r.t. gradients

Worker *j* owns $\theta_t^{(j)}\ g_t^{(j)}$, do 2 things parallely:

1. Locally estimate model: $\theta_{t+1}^{(j(l))}$
2. Pulls **a** model from PS *i*: $\theta_{t+1}^{(i)}$

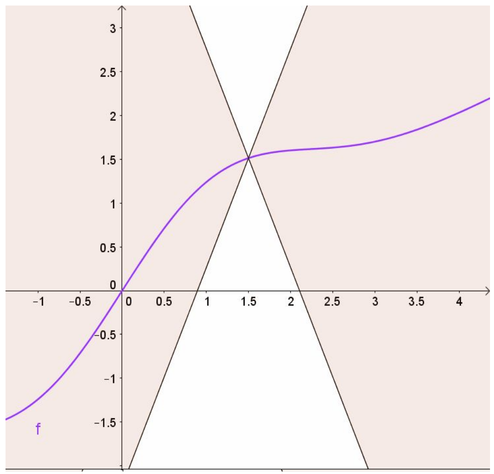
GAR → both should be close

How close? → Lipschitz Coefficient should limit growth of $\theta_{t+1}^{(i)}$

$$k = \left\| g_{t+1}^{(j)} - g_t^{(j)} \right\| / \left\| \theta_{t+1}^{(j(l))} - \theta_t^{(j)} \right\|$$

$$k \leq K_p \triangleq \text{quantile}_{\frac{n_{ps}-f_{ps}}{n_{ps}}}\{K\}$$

K: list of all previous L-coefficients



Note: previous K come from other different servers as well

Gather - Models Filter

Bound distance between 2 models in each successive (scatter) iteration

Assumption: all machines initialize models with the same state

GAR guarantees → estimate upper bound on model update

Local estimate model: $\theta_{t+1}^{(j(l))}$

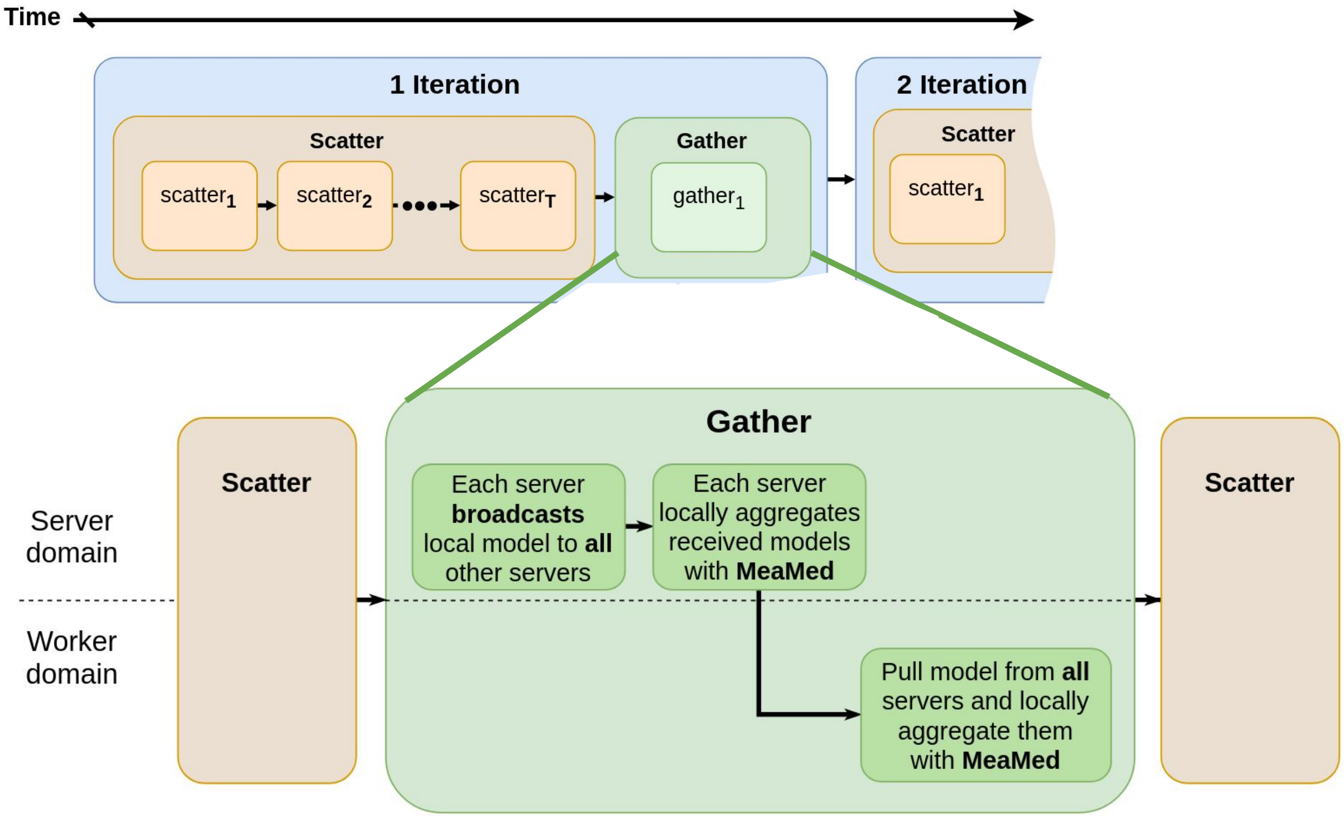
Pulled model from PS: $\theta_{t+1}^{(i)}$

$$\left\|\theta_{t+1}^{(j(l))} - \theta_{t+1}^{(i)}\right\| < \gamma_{T\cdot(t \mod T)} \left\|g_{T\cdot(t \mod T)}\right\| \left({}^{(3T+2)(n_w - f_w)}/{}_{4f_w} + 2\big((t-1) \mod T\big)\right)$$

$$\text{, with } T = {}^{1}/{}_{3l\gamma_1}$$

l: Lipschitz coeff.

LiuBei - Gather



LiuBei Evaluation

- **Datasets**: MNIST and CIFAR-10
- Different neural-network architectures (see table)
- **Baselines**: TensorFlow and GuanYu
- Number of **workers**: 20 (up to 8 Byzantine)
- Number of **servers** varies:
  - TensorFlow: **1** PS
  - LiuBei: **4** servers (tolerates up to 1 Byzantine)
  - GuanYu: **5** servers (tolerates up to 1 Byzantine)
- Metrics:
  - Throughput: number of parameter server updates per second
  - Classification accuracy

Table 1: Models used to evaluate LIUBEI.

Model	# parameters	Size (MB)
MNIST_CNN	79510	0.3
CifarNet	1756426	6.7
Inception	5602874	21.4
ResNet-50	23539850	89.8
ResNet-200	62697610	239.2

LiuBei's Performance

Around 5% loss compared to TF

GuanYu seems to have a better end acc when 1 f_{ps} present (dashed violet line)

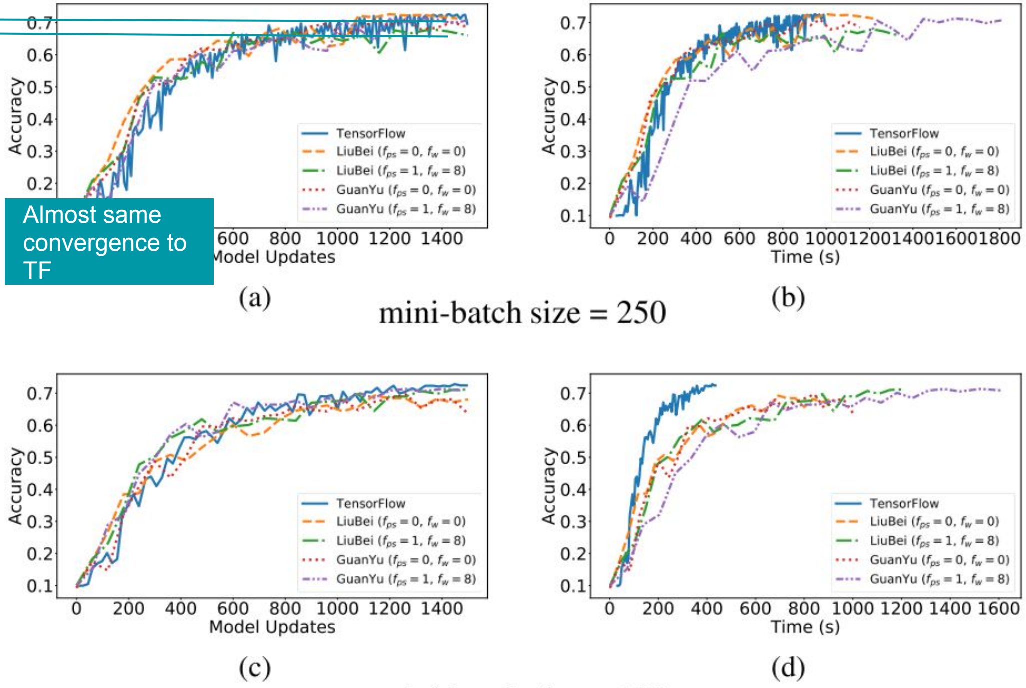
Almost same convergence to TF

24% overhead of LiuBei to TF

70% performance gain of LiuBei compared to GuanYu

User of higher batch-size yields better performance

GuanYu and LiuBei show similar convergence

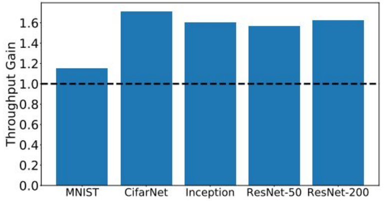


mini-batch size = 100

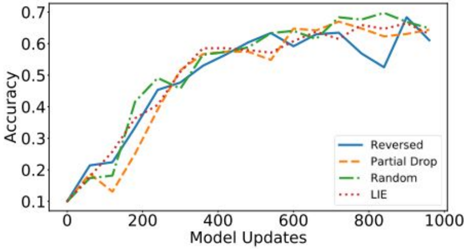
Figure 2: Convergence in a non-Byzantine environment.

LiuBei's Performance

Throughput gain of LiuBei compared to GuanYu

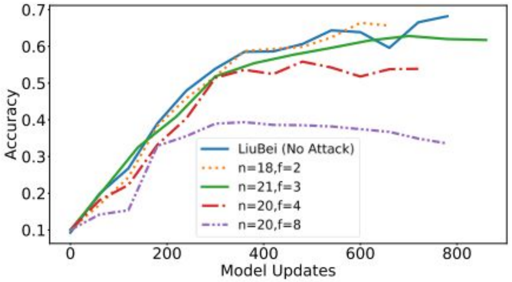


LiuBei response to different server behaviour



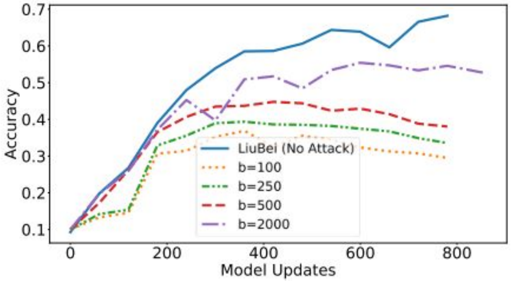
LiuBei response to different number of byzantine workers

Severe degradation starts when 20% of nodes is Byxantine



(a) The ratio $\frac{f_w}{n_w}$

40% acc downgrade (68% compared to TF) when f_{max}=8



(b) Batch size

Using $f = f_{max} = 8$

Increase of batch size has positive impact

Concluding Remarks

Limitation of these Techniques

- Might not work well with Federated Learning environments for...
  - Non i.i.d. data distribution ⇒ correct workers with outlier data are treated as byzantine
  - Draco requires data redundancy ⇒ incompatible with data-privacy guarantees
- Based on a parameter-server architecture
  - What about decentralized approaches (e.g. ring-allreduce, gossip) ?

Any Questions?