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

Median-based GARs (EPFL)

1) Blanchard 2017

2) Mhamdi 2018

3) Chen 2018

Define GARs

4) Damaskinos 2019

TensorFlow impl./eval.

Malicious Parameter Servers

5) Mhamdi 2019a → 6) Mhamdi 2019b

Redundant Gradients (Univ. Wisconsin)
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:
  \[ F(G_1, \ldots, G_n) = \sum_{i=1}^{N} \lambda_i G_i \quad \forall i \lambda_i \neq 0 \]

- A single malicious gradient \( G_n \) can undo all other gradients and replace them with a target gradient \( U \):
  \[
  G_n = \frac{1}{\lambda_n} \left( U - \sum_{i=1}^{N-1} \lambda_i G_i \right) \Rightarrow F(G_1, \ldots, G_n) = U
  \]

We need a new Gradient Aggregation Rule (GAR)
A Definition of Byzantine Resilience

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

We need an $(\alpha, f)$-Byzantine Resilient GAR
Krum: an \((\alpha, f)\)-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 \(G_j\) for every \(G_i\) (forming the \(n-f-1\) tightest cluster around \(G_i\))
  - Find the \(G_i\) with the tightest overall cluster by minimizing
    \[
    s(G_i) = \sum_{i \rightarrow j} \|G_i - G_j\|^2
    \]
  - Output \(G_i\)
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 \((\alpha, f)\)-Byzantine Resilient GAR

- **Idea:**
  - The \(n - f\) non-byzantine gradients should form a tightly-packed cluster
  - List all possible clusters of \(n - f\) gradients:
    \[
    \mathcal{R} = \{ \mathcal{X} \mid \mathcal{X} \subset \mathcal{G} \land |\mathcal{X}| = n - f \}
    \]
  - 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 $\beta$ 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 $\theta$ gradients

- **Bulyan**
  - Requires $n \geq 4f + 3$ gradients
  - Requires an $(\alpha, f)$-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**
  - $(\alpha, f)$-Byzantine Resilient GAR guarantees that Bulyan is $(\alpha, f)$-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
DRACO: Byzantine-resilient Distributed Training via Redundant Gradients

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

We consider how to compute
\[ \sum_{i=1}^{B} g_i \]
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 $r \triangleq \frac{1}{P} \|A\|_0$

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** \((A^{Rep}, E^{Rep}, D^{Rep})\)

**Encoding stage:**

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

2. \(Y_j^{Rep} = (1_d A_j^{Rep}) \odot G.\)

3. \(E_j^{Rep}(Y_j^{Rep}) = Y_j^{Rep} 1_P.\)

4. \(z_j = E_j^{Rep}(Y_j^{Rep}) \quad \text{Send to the PS}\)

**Decoding stage:**

\[D^{Rep}(R) = \sum_{\ell=1}^{P} \text{Maj}(R_{,((\ell-1)+1):\ell\cdot r}).\]
Encoding-decoding gradients

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

Let \(C\) be a \(P \times P\) inverse discrete Fourier transformation (IDFT) matrix

\[
C_{jk} = \frac{1}{\sqrt{P}} \exp\left(\frac{2\pi i}{P} (j - 1)(k - 1)\right), \quad j, k = 1, 2, \ldots, P.
\]

Let \(C_L\) be the first \(P-2s\) rows of \(C\) and \(C_R\) be the last \(2s\) rows

Encoding stage:

\[
\alpha_j = \{k : A^{Cyc}_{j,k} = 0\}
\]

\[
0 = [q_j, 1] \cdot [C_L]_{,\alpha_j}
\]

\[
Q \triangleq [q_1, q_2, \ldots, q_P]
\]

\[
W \triangleq [Q, 1_P] \cdot C_L
\]

\[
Y^{Cyc}_j = (1_d A^{Cyc}_{j,\cdot}) \odot G
\]

\[
E^{Cyc}_j(Y^{Cyc}_j) = GW_{,j}
\]

\[
Z^{Cyc}_j \triangleq E^{Cyc}_j(Y^{Cyc}_j)
\]

Send to the PS
Encoding-decoding gradients

Cyclic Code with Fourier decoding \((A^{Cy}, E^{Cy}, D^{Cy})\)

Suppose there is a function \(\varphi(\cdot)\) that can compute the adversarial node index set \(V\)

Decoding Stage:

1. \(V = \varphi(R)\)
2. \(U = \{1, 2, \ldots, P\} - V\)
3. Find \(b\) by solving \(W_{.U}b = 1_P\)
4. \(u^{Cy} = R_{.U}b\)

This approach has linear-time in encoding and decoding
Experiments

Adversarial Attack Models:

1. **Reversed gradient adversary** send $-cg$ 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.
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

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

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

<table>
<thead>
<tr>
<th>Time Cost (sec)</th>
<th>Comp</th>
<th>Comm</th>
<th>Encode</th>
<th>Decode</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM const</td>
<td>0.26</td>
<td>12.47</td>
<td>0</td>
<td>74.63</td>
</tr>
<tr>
<td>Rep const</td>
<td>2.59</td>
<td>12.91</td>
<td>0.20</td>
<td>3.03</td>
</tr>
<tr>
<td>SGD const</td>
<td>0.25</td>
<td>6.9</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>Cyclic const</td>
<td>3.08</td>
<td>12.91</td>
<td>4.01</td>
<td>4.30</td>
</tr>
<tr>
<td>GM rev grad</td>
<td>0.26</td>
<td>14.57</td>
<td>0</td>
<td>39.02</td>
</tr>
<tr>
<td>Rep rev grad</td>
<td>2.55</td>
<td>14.66</td>
<td>0.20</td>
<td>3.04</td>
</tr>
<tr>
<td>SGD rev grad</td>
<td>0.25</td>
<td>7.15</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>Cyclic rev grad</td>
<td>3.07</td>
<td>14.66</td>
<td>4.02</td>
<td>3.65</td>
</tr>
</tbody>
</table>
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 \textit{almost surely} converges around a minima, despite the presence of $f$ Byzantine workers. Ensures $\nabla Q(x^*) = 0$ to some extent.

\begin{itemize}
  \item (Multi-)Krum
\end{itemize}

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

$$\|\tilde{X} - Y\|_p = \mathcal{O}(\sqrt{d})$$

Strong BR

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

\begin{itemize}
  \item Bulyan
  \item DRACO
\end{itemize}

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

$$\frac{1}{\sqrt{d}}$$
### Characteristics of GARs so far

<table>
<thead>
<tr>
<th></th>
<th>Required workers</th>
<th>Method</th>
<th>Privacy issues</th>
<th>Comparative Performance</th>
</tr>
</thead>
</table>
| \(m\)-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.

AGGREGATHOR

- Cluster management
- Optimizers
  - Momentum, Adam, ...
- Learning rates
  - Fixed, Polynomial, ...

Gradient Aggregation Rule (GAR)

Experience

(= model + dataset)

TensorFlow

lossyMPI

<code patch>

OS (libstdc++, libcudart, libmpi, ...)

[Diagram showing the modular integration of GARs and other components]
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

Evaluation: Non-Byzantine Environment

* AggregaThor reaches (at some point) baseline acc

* DRACO as well, but takes longer time

2f+1 more gradients required

Figure 3. Overhead of AGGREGATHOR in a non-Byzantine environment.
Evaluation: Non-Byzantine Environment

Figure 5. Throughput comparison.

Figure 6. Impact of $f$ on convergence.
Evaluation: Byzantine Environment

Corrupted Data

Dropped packets

Figure 7. Impact of malformed input on convergence.

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

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.
  ○ $\approx 10^{20}$ operations for 100 workers and vector precision of $10^{-9}$.

  $\rightarrow$ 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. Guirguui, S. Rouault
Motivation

Previous work assume the parameter server is free from malicious behavior, which is not necessarily true.
### GuanYu algorithm

*F*: Multi–Krum  
*M*: coordinate–wise median  

\[2f + 3 \leq q \leq n - f\]  
the quorum used for *M*  

\[2\bar{f} + 3 \leq \bar{q} \leq n - \bar{f}\]  
the quorum used for *F*  

GuanYu does not wait for all \(n\) nodes to start aggregation

<table>
<thead>
<tr>
<th>Parameter server (i \in [1..n - f])</th>
<th>Worker (j \in [1..n - \bar{f}])</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\theta_t^{(i)}) (\rightarrow q) (\leftarrow \bar{q}) (\rightarrow q)</td>
<td>(g_t^{(j)} = \nabla L (M \left( \theta_t^{(a)} \ldots \theta_t^{(b)} \right)) \sim G_t^{(j)})</td>
</tr>
</tbody>
</table>

Local update  

\[\theta_t^{(i)} = \theta_t^{(i)} - \eta_t F \left( g_t^{(x)} \ldots g_t^{(y)} \right) = \bar{\theta}_{t+1}^{(i)}\]

\[M \left( \bar{\theta}_{t+1}^{(z)} \ldots \bar{\theta}_{t+1}^{(w)} \right) = \theta_t^{(i)}\]
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 \( t_s \in \mathbb{N} \), all the non-Byzantine parameter vectors are roughly aligned.

Intuitions:

1. Non-Byzantine parameter vectors gets \textit{almost-surely arbitrary close} to each other after some step \( t \in \mathbb{N} \).
2. By the \textit{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 \( \eta_t \) 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 \textit{contraction effect} pulls back together the non-Byzantine parameter vectors.
Implementation

TensorFlow

- Compute gradients
- Model update

GuanYu

- Communications
- Compute M
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

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

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

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

Servers

Workers

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
  ○ 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
LiuBei

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

Workers work independently and do not communicate.

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

Workers work independently and do not communicate. Correct servers communicate to bring their view of models back close to each other.
Gather - Lipschitz Filter

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

Worker $j$ owns $\theta_t^{(j)}$, 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 $\rightarrow$ both should be close

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

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

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

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: $\theta_{t+1}^{(i)}$

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

, with $T = \frac{1}{3l \gamma_1}$

I: Lipschitz coeff.
LiuBei - Gather

1 Iteration

Scatter
scatter_1 scatter_2 ... scatter_T

Gather
gather_1

2 Iteration

Scatter
scatter_1

Each server broadcasts local model to all other servers

Each server locally aggregates received models with MeaMed

Pull model from all servers and locally aggregate them with MeaMed

Server domain

Worker domain
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.

<table>
<thead>
<tr>
<th>Model</th>
<th># parameters</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST_CNN</td>
<td>79510</td>
<td>0.3</td>
</tr>
<tr>
<td>CifarNet</td>
<td>1756426</td>
<td>6.7</td>
</tr>
<tr>
<td>Inception</td>
<td>5602874</td>
<td>21.4</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>23539850</td>
<td>89.8</td>
</tr>
<tr>
<td>ResNet-200</td>
<td>62697610</td>
<td>239.2</td>
</tr>
</tbody>
</table>
LiuBei’s Performance

- Around 5% loss compared to TF
- Almost same convergence to TF
- GuanYu seems to have a better end acc when 1 fps present (dashed violet line)
- User of higher batch-size yields better performance
- GuanYu and LiuBei show similar convergence
- 24% overhead of LiuBei to TF
- 70% performance gain of LiuBei compared to GuanYu

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

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

(a) The ratio $\frac{f_{\text{wu}}}{f_{\text{wu}}}$

(b) Batch size Using $f = f_{\text{max}} = 8$

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 $\Rightarrow$ correct workers with outlier data are treated as byzantine
  - Draco requires data redundancy $\Rightarrow$ incompatible with data-privacy guarantees

- Based on a parameter-server architecture
  - What about decentralized approaches (e.g. ring-allreduce, gossip)?
Any Questions?