Roubust Learning

Amir H. Payberah
payberah@kth.se
2020-11-09

The Course Web Page

https://fid3024.github.io

Adversarial Goals

Adversarial Goals

- Confidentiality and privacy
  - Confidentiality of the model itself (e.g., intellectual property)
  - Privacy of the training or test data (e.g., medical records)

Adversarial Goals

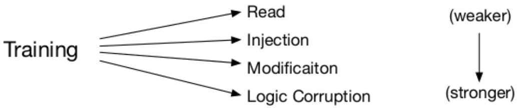
- Confidentiality and privacy
  - Confidentiality of the model itself (e.g., intellectual property)
  - Privacy of the training or test data (e.g., medical records)
- Integrity
  - Integrity of the predictions

Adversarial Goals

- Confidentiality and privacy
  - Confidentiality of the model itself (e.g., intellectual property)
  - Privacy of the training or test data (e.g., medical records)
- Integrity
  - Integrity of the predictions
- Availability
  - Availability of the system deploying machine learning

Adversarial Capabilities for Integrity Attacks

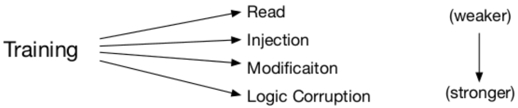
- Training phase



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

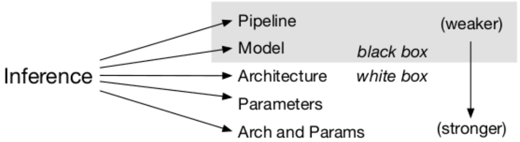
Adversarial Capabilities for Integrity Attacks

- Training phase



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

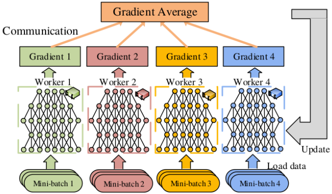
- Inference phase
  - White box
  - Black box



[Papernot et al., SoK: Security and Privacy in Machine Learning, 2018]

Our Focus and Goal

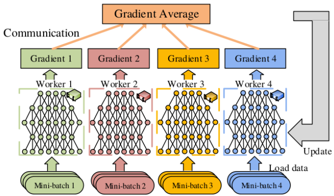
- Data parallelization



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

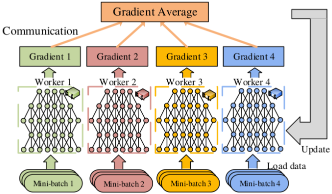
- Data parallelization
- Each worker is prone to adversarial attack.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

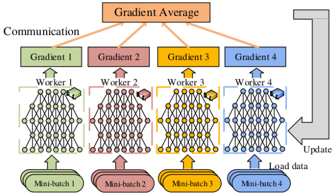
- Data parallelization
- Each worker is prone to adversarial attack.
- Adversarial attacks: some unknown subset of computing devices are compromised and behave adversarially (e.g., sending out malicious messages)



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Our Focus and Goal

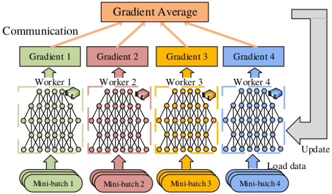
- Data parallelization
- Each worker is prone to adversarial attack.
- Adversarial attacks: some unknown subset of computing devices are compromised and behave adversarially (e.g., sending out malicious messages)
- Our goal: integrity of the model in the training phase



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

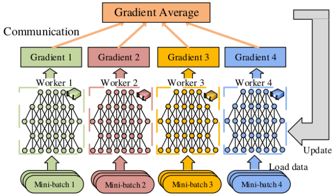
- One parameter server, and n workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

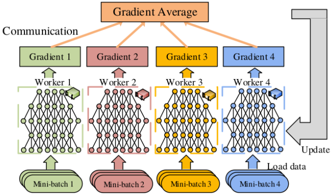
- One parameter server, and n workers.
- Computation is divided into synchronous rounds.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (1/3)

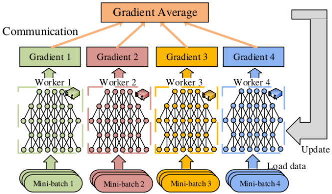
- One parameter server, and $\mathtt{n}$ workers.
- Computation is divided into synchronous rounds.
- During round $\mathtt{t}$, the parameter server broadcasts its parameter vector $\mathbf{w} \in \mathbb{R}^{\mathtt{d}}$ to all the workers.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

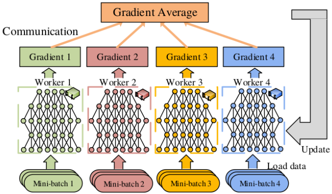
- At each round $\mathtt{t}$, each correct worker $\mathtt{i}$ computes $\mathtt{G_i}(\mathbf{w_t}, \beta)$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

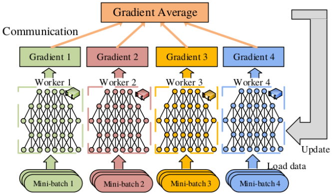
- At each round $\mathtt{t}$, each correct worker $\mathtt{i}$ computes $\mathtt{G_i}(\mathbf{w_t}, \beta)$.
- $\mathtt{G_i}(\mathbf{w_t}, \beta)$: the local estimate of the gradient of the loss function $\nabla \mathtt{J}(\mathbf{w_t})$.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

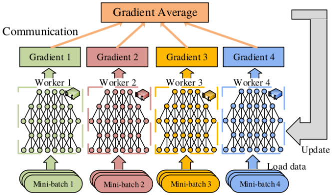
- At each round $\mathtt{t}$, each correct worker $\mathtt{i}$ computes $\mathtt{G_i}(\mathbf{w_t}, \beta)$.
- $\mathtt{G_i}(\mathbf{w_t}, \beta)$: the local estimate of the gradient of the loss function $\nabla \mathtt{J}(\mathbf{w_t})$.
- β: a mini-batch of i.i.d. samples drawn from the dataset.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (2/3)

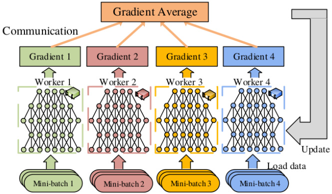
- At each round $\mathtt{t}$, each correct worker $\mathtt{i}$ computes $\mathtt{G_i}(\mathbf{w}_\mathtt{t}, \beta)$.
- $\mathtt{G_i}(\mathbf{w}_\mathtt{t}, \beta)$: the local estimate of the gradient of the loss function $\nabla \mathtt{J}(\mathbf{w}_\mathtt{t})$.
- β: a mini-batch of i.i.d. samples drawn from the dataset.
- $\mathtt{G_i}(\mathbf{w}_\mathtt{t}, \beta) = \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l_i}(\mathbf{w}_\mathtt{t}, \mathbf{x})$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

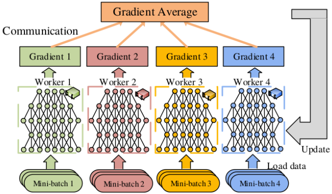
- The parameter server computes $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

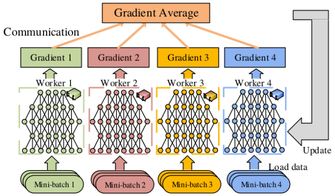
- The parameter server computes $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$
- Gradient Aggregation Rule (GAR): $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathtt{G}_i$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Stochastic Gradient Descent (3/3)

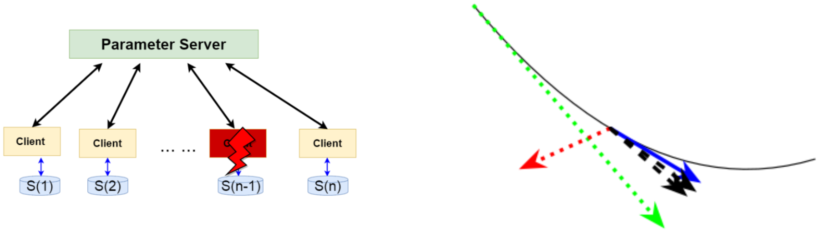
- The parameter server computes $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$
- Gradient Aggregation Rule (GAR): $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathtt{G}_i$
- The parameter server updates the parameter vector $\mathbf{w} \leftarrow \mathbf{w} - \gamma \mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed SGD with Byzantine Workers

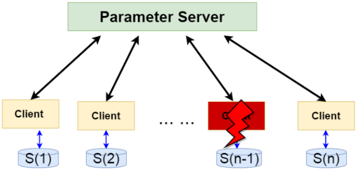
- Among the $\mathtt{n}$ workers, $\mathtt{f}$ of them are possibly Byzantine (behaving arbitrarily).

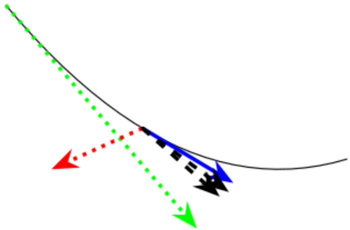


[El-Mhamdi et al., Fast and Secure Distributed Learning in High Dimension, 2019]

Distributed SGD with Byzantine Workers

- Among the $\mathtt{n}$ workers, $\mathtt{f}$ of them are possibly Byzantine (behaving arbitrarily).
- A Byzantine worker $\mathtt{b}$ proposes a vector $\mathtt{G}_{\mathtt{b}}$ that can deviate arbitrarily from the vector it is supposed.





[El-Mhamdi et al., Fast and Secure Distributed Learning in High Dimension, 2019]

Averaging GAR and Byzantine Workers

- Averaging GAR: $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathtt{G}_i$
- $\mathbf{w} \leftarrow \mathbf{w} - \gamma \mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$

Averaging GAR and Byzantine Workers

- Averaging GAR: $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathtt{G}_i$
- $\mathbf{w} \leftarrow \mathbf{w} - \gamma \mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$
- Even a single Byzantine worker can prevent convergence.

Averaging GAR and Byzantine Workers

- Averaging GAR: $\mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n) = \frac{1}{n}\sum_{i=1}^{n} \mathtt{G}_i$
- $\mathbf{w} \leftarrow \mathbf{w} - \gamma \mathtt{F}(\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_n)$
- Even a single Byzantine worker can prevent convergence.
- Proof: if the Byzantine worker proposes $\mathtt{G}_n = n\mathtt{U} - \sum_{i=1}^{n-1} \mathtt{G}_i$, then $\mathtt{F} = \mathtt{U}$.

(α, f)-Byzantine-Resilience (1/2)

- Assume `n` workers, where `f` of them are Byzantine workers.

(α, f)-Byzantine-Resilience (1/2)

- Assume $\mathtt{n}$ workers, where $\mathtt{f}$ of them are Byzantine workers.
- $\alpha \in [0, \pi/2]$ and $\mathtt{f} \in \{0, \cdots, \mathtt{n}\}$.

(α, f)-Byzantine-Resilience (1/2)

- Assume $\mathtt{n}$ workers, where $\mathtt{f}$ of them are Byzantine workers.
- $\alpha \in [0, \pi/2]$ and $\mathtt{f} \in \{0, \cdots, \mathtt{n}\}$.
- $(\mathtt{G}_1, \cdots, \mathtt{G}_{\mathtt{n}-\mathtt{f}}) \in (\mathbb{R}^{\mathtt{d}})^{\mathtt{n}-\mathtt{f}}$ are i.i.d. random vectors

(α, f)-Byzantine-Resilience (1/2)

- Assume $\mathtt{n}$ workers, where $\mathtt{f}$ of them are Byzantine workers.
- $\alpha \in [0, \pi/2]$ and $\mathtt{f} \in \{0, \cdots, \mathtt{n}\}$.
- $(\mathtt{G}_1, \cdots, \mathtt{G}_{\mathtt{n}-\mathtt{f}}) \in (\mathbb{R}^{\mathtt{d}})^{\mathtt{n}-\mathtt{f}}$ are i.i.d. random vectors
  - $\mathtt{G}_\mathtt{i} \sim \mathtt{g}$
  - $\mathbb{E}[\mathtt{g}] = \mathcal{J}$, where $\mathcal{J} = \nabla \mathtt{J}(\mathbf{w})$

(α, f)-Byzantine-Resilience (1/2)

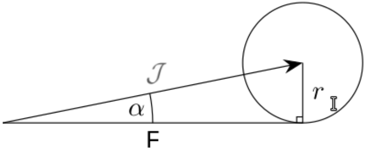
- Assume $\mathtt{n}$ workers, where $\mathtt{f}$ of them are Byzantine workers.
- $\alpha \in [0, \pi/2]$ and $\mathtt{f} \in \{0, \cdots, \mathtt{n}\}$.
- $(\mathtt{G}_1, \cdots, \mathtt{G}_{\mathtt{n}-\mathtt{f}}) \in (\mathbb{R}^{\mathtt{d}})^{\mathtt{n}-\mathtt{f}}$ are i.i.d. random vectors
  - $\mathtt{G}_\mathtt{i} \sim \mathtt{g}$
  - $\mathbb{E}[\mathtt{g}] = \mathcal{J}$, where $\mathcal{J} = \nabla \mathtt{J}(\mathbf{w})$
- $(\mathtt{B}_1, \cdots, \mathtt{B}_\mathtt{f}) \in (\mathbb{R}^{\mathtt{d}})^{\mathtt{f}}$ are random vectors, possibly dependent between them and the vectors $(\mathtt{G}_1, \cdots, \mathtt{G}_{\mathtt{n}-\mathtt{f}})$

(α, f)-Byzantine-Resilience (2/2)

- A GAR $\mathtt{F}$ is said to be (α, f)-Byzantine-resilient if, for any $\mathtt{1} \leq \mathtt{j_1} < \cdots < \mathtt{j_f} \leq \mathtt{n}$, the vector $\mathtt{F}(\mathtt{G_1}, \cdots, \mathtt{B_1}, \cdots, \mathtt{B_f}, \cdots, \mathtt{G_n})$ satisfies:

(α, f)-Byzantine-Resilience (2/2)

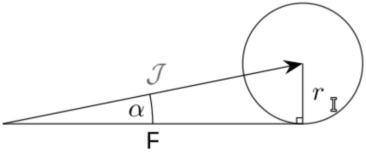
- A GAR $\mathtt{F}$ is said to be (α, f)-Byzantine-resilient if, for any $\mathtt{1} \leq \mathtt{j_1} < \cdots < \mathtt{j_f} \leq \mathtt{n}$, the vector $\mathtt{F}(\mathtt{G_1}, \cdots, \mathtt{B_1}, \cdots, \mathtt{B_f}, \cdots, \mathtt{G_n})$ satisfies:
  1. Vector $\mathtt{F}$ that is not too far from the real gradient $\mathcal{J}$, i.e., $||\mathbb{E}[\mathtt{F}] - \mathcal{J}|| \leq \mathtt{r}$.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]

(α, f)-Byzantine-Resilience (2/2)

- A GAR $\mathtt{F}$ is said to be (α, f)-Byzantine-resilient if, for any $1 \leq \mathtt{j}_1 < \cdots < \mathtt{j}_f \leq \mathtt{n}$, the vector $\mathtt{F}(\mathtt{G}_1, \cdots, \mathtt{B}_1, \cdots, \mathtt{B}_f, \cdots, \mathtt{G}_n)$ satisfies:
  1. Vector $\mathtt{F}$ that is not too far from the real gradient $\mathcal{J}$, i.e., $||\mathbb{E}[\mathtt{F}] - \mathcal{J}|| \leq \mathtt{r}$.
  2. Moments of $\mathtt{F}$ should be controlled by the moments of the (correct) gradient estimator $\mathtt{g}$, where $\mathbb{E}[\mathtt{g}] = \mathcal{J}$.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]

Byzantine-Resilience GAR

- Median
- Krum
- Multi-Krum
- Brute

Median

- $\mathtt{n} \geq 2\mathtt{f} + 1$

Median

- $n \geq 2f + 1$
- $\text{median}(x_1, \cdots, x_n) = \arg\min_{x \in \mathbb{R}} \sum_{i=1}^{n} |x_i - x|$

Median

- $n \geq 2f + 1$
- $\mathtt{median}(x_1, \cdots, x_n) = \arg\min_{x \in \mathbb{R}} \sum_{i=1}^{n} |x_i - x|$
- d: the gradient vectors dimension.
- Geometric median
$$F = \mathtt{GeoMed}(G_1, \cdots, G_n) = \arg\min_{G \in \mathbb{R}^d} \sum_{i=1}^{n} ||G_i - G||$$

Median

- $n \geq 2f + 1$
- $\mathtt{median}(x_1, \cdots, x_n) = \arg\min_{x \in \mathbb{R}} \sum_{i=1}^{n} |x_i - x|$
- d: the gradient vectors dimension.
- Geometric median
$$F = \mathtt{GeoMed}(G_1, \cdots, G_n) = \arg\min_{G \in \mathbb{R}^d} \sum_{i=1}^{n} ||G_i - G||$$
- Marginal median
$$F = \mathtt{MarMed}(G_1, \cdots, G_n) = \begin{pmatrix} \mathtt{median}(G_1[1], \cdots, G_n[1]) \\ \vdots \\ \mathtt{median}(G_1[d], \cdots, G_n[d]) \end{pmatrix} \tag{1}$$

Krum

- $\mathtt{n} \geq 2\mathtt{f} + 3$

Krum

- $\mathtt{n} \geq 2\mathtt{f} + 3$
- Idea: to preclude the vectors that are too far away.

Krum

- $\mathtt{n} \geq 2\mathtt{f} + 3$
- Idea: to preclude the vectors that are too far away.
- $\mathtt{s(i)} = \sum_{\mathtt{i} \rightarrow \mathtt{j}} ||\mathtt{G_i} - \mathtt{G_j}||^2$, the score of the worker $\mathtt{i}$.

Krum

- $n \geq 2f + 3$
- Idea: to preclude the vectors that are too far away.
- $s(i) = \sum_{i \to j} ||G_i - G_j||^2$, the score of the worker i.
- $i \to j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i.

Krum

- $n \geq 2f + 3$
- Idea: to preclude the vectors that are too far away.
- $s(i) = \sum_{i \to j} ||G_i - G_j||^2$, the score of the worker i.
- $i \to j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i.
- $F(G_1, \cdots, G_n) = G_{i_*}$

Krum

- $n \geq 2f + 3$
- Idea: to preclude the vectors that are too far away.
- $s(i) = \sum_{i \to j} ||G_i - G_j||^2$, the score of the worker i.
- $i \to j$ denotes that G_j belongs to the $n - f - 2$ closest vectors to G_i.
- $F(G_1, \cdots, G_n) = G_{i_*}$
- G_{i_*} refers to the worker minimizing the score, $s(i_*) \leq s(i)$ for all i.

Multi-Krum

- Multi-Krum computes the score for each vector proposed (as in Krum).

Multi-Krum

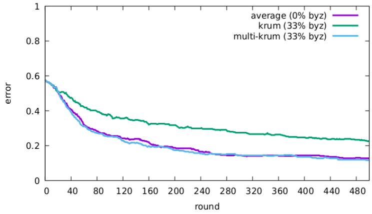
- Multi-Krum computes the score for each vector proposed (as in Krum).
- It selects m vectores $G_{1_*}, \cdots, G_{m_*}$, which score the best $(1 \leq m \leq n - f - 2)$.

Multi-Krum

- Multi-Krum computes the score for each vector proposed (as in Krum).
- It selects $\mathtt{m}$ vectores $\mathtt{G}_{1_*}, \cdots, \mathtt{G}_{\mathtt{m}_*}$, which score the best $(1 \leq \mathtt{m} \leq \mathtt{n} - \mathtt{f} - 2)$.
- It outputs their average $\frac{1}{\mathtt{m}} \sum_{\mathtt{i}} \mathtt{G}_{\mathtt{i}_*}$.

Multi-Krum

- Multi-Krum computes the score for each vector proposed (as in Krum).
- It selects m vectores $G_{1_*}, \cdots, G_{m_*}$, which score the best $(1 \leq m \leq n - f - 2)$.
- It outputs their average $\frac{1}{m} \sum_i G_{i_*}$.
- The cases $m = 1$ and $m = n$ correspond to Krum and averaging, respectively.



[Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017]

Brute

- $\mathtt{n} \geq \mathtt{2f} + \mathtt{1}$

Brute

- $\mathtt{n} \geq 2\mathtt{f} + 1$
- $\mathcal{Q} = \{\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_\mathtt{n}\}$

Brute

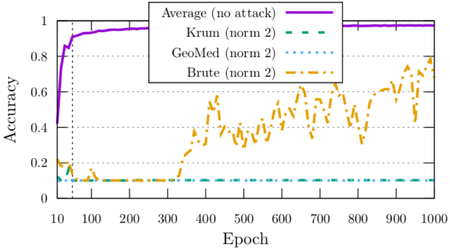
- $\mathtt{n} \geq 2\mathtt{f} + 1$
- $\mathcal{Q} = \{\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_\mathtt{n}\}$
- $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = \mathtt{n} - \mathtt{f}\}$
    - The set of all the subsets of $\mathtt{n} - \mathtt{f}$

Brute

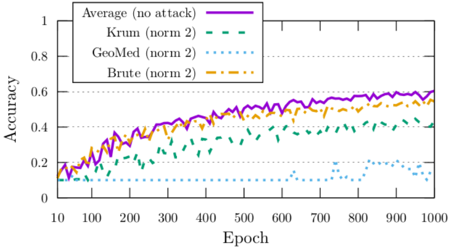
- $\mathtt{n} \geq 2\mathtt{f} + 1$
- $\mathcal{Q} = \{\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_\mathtt{n}\}$
- $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = \mathtt{n} - \mathtt{f}\}$
  - The set of all the subsets of $\mathtt{n} - \mathtt{f}$
- $\mathcal{S} = \arg\min_{\mathcal{X} \in \mathcal{R}} (\max_{(\mathtt{G}_\mathtt{i}, \mathtt{G}_\mathtt{j}) \in \mathcal{X}^2} (||\mathtt{G}_\mathtt{i} - \mathtt{G}_\mathtt{j}||))$
  - Selects the $\mathtt{n} - \mathtt{f}$ most clumped gradients among the submitted ones.

Brute

- $\mathtt{n} \geq 2\mathtt{f} + 1$
- $\mathcal{Q} = \{\mathtt{G}_1, \mathtt{G}_2, \cdots, \mathtt{G}_\mathtt{n}\}$
- $\mathcal{R} = \{\mathcal{X} | \mathcal{X} \subset \mathcal{Q}, |\mathcal{X}| = \mathtt{n} - \mathtt{f}\}$
  - The set of all the subsets of $\mathtt{n} - \mathtt{f}$
- $\mathcal{S} = \arg\min_{\mathcal{X} \in \mathcal{R}} (\max_{(\mathtt{G}_\mathtt{i}, \mathtt{G}_\mathtt{j}) \in \mathcal{X}^2} (||\mathtt{G}_\mathtt{i} - \mathtt{G}_\mathtt{j}||))$
  - Selects the $\mathtt{n} - \mathtt{f}$ most clumped gradients among the submitted ones.
- $\mathtt{F}(\mathtt{G}_1, \cdots, \mathtt{G}_\mathtt{n}) = \frac{1}{\mathtt{n}-\mathtt{f}} \sum_{\mathtt{G} \in \mathcal{S}} \mathtt{G}$



MNIST



CIFAR-10

[El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018]

Weak Byzantine Resilience

- Limitation of previous aggregation methods.
- If gradient dimension $d \gg 1$, then the distance function between two vectors $||X - Y||_p$, cannot distinguish these two cases:

Weak Byzantine Resilience

- Limitation of previous aggregation methods.
- If gradient dimension $\mathtt{d} \gg \mathtt{1}$, then the distance function between two vectors $||\mathtt{X}-\mathtt{Y}||_{\mathtt{p}}$, cannot distinguish these two cases:
- 1. Does $\mathtt{X}$ and $\mathtt{Y}$ disagree a bit on each coordinate?

Weak Byzantine Resilience

- Limitation of previous aggregation methods.
- If gradient dimension $d \gg 1$, then the distance function between two vectors $||X - Y||_p$, cannot distinguish these two cases:
- 1. Does X and Y disagree a bit on each coordinate?
- 2. Does X and Y disagree a lot on only one?

Strong Byzantine Resilience

- Ensuring convergence (as in weak Byzantine resilience functions).

Strong Byzantine Resilience

- Ensuring convergence (as in weak Byzantine resilience functions).
- Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule $\mathtt{A}$.

Strong Byzantine Resilience

- Ensuring convergence (as in weak Byzantine resilience functions).
- Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule `A`.
- `A` can be Brute, Krum, Median, etc.

Strong Byzantine Resilience

- Ensuring convergence (as in weak Byzantine resilience functions).
- Ensures that each coordinate is agreed on by a majority of vectors that were selected by a Byzantine resilient aggregation rule `A`.
- `A` can be Brute, Krum, Median, etc.
- Bulyan is a strong Byzantine-resilience algorithm.

The Hidden Vulnerability of Distributed Learning in Byzantium

Bulyan - Step One (1/2)

- $\mathtt{n} \geq \mathtt{4f} + 3$
- A two step process.

Bulyan - Step One (1/2)

- $\mathtt{n} \geq 4\mathtt{f} + 3$
- A two step process.
- The first one is to recursively use $\mathtt{A}$ to select $\theta = \mathtt{n} - 2\mathtt{f}$ gradients:

Bulyan - Step One (1/2)

- $\mathtt{n} \geq 4\mathtt{f} + 3$
- A two step process.
- The first one is to recursively use $\mathtt{A}$ to select $\theta = \mathtt{n} - 2\mathtt{f}$ gradients:
  1. With $\mathtt{A}$, choose, among the proposed vectors, the closest one to $\mathtt{A}$'s output (for Krum this would be the exact output of $\mathtt{A}$).

Bulyan - Step One (1/2)

- $\mathtt{n} \geq 4\mathtt{f} + 3$
- A two step process.
- The first one is to recursively use $\mathtt{A}$ to select $\theta = \mathtt{n} - 2\mathtt{f}$ gradients:
  1. With $\mathtt{A}$, choose, among the proposed vectors, the closest one to $\mathtt{A}$'s output (for Krum this would be the exact output of $\mathtt{A}$).
  2. Remove the chosen gradient from the received set and add it to the selection set $\mathtt{S}$.

Bulyan - Step One (1/2)

- $\mathtt{n} \geq \mathtt{4f} + \mathtt{3}$
- A two step process.
- The first one is to recursively use $\mathtt{A}$ to select $\theta = \mathtt{n} - \mathtt{2f}$ gradients:
  1. With $\mathtt{A}$, choose, among the proposed vectors, the closest one to $\mathtt{A}$'s output (for Krum this would be the exact output of $\mathtt{A}$).
  2. Remove the chosen gradient from the received set and add it to the selection set $\mathtt{S}$.
  3. Loop back to step 1 if $|\mathtt{S}| < \theta$.

Bulyan - Step One (2/2)

- $\theta = \mathtt{n} - 2\mathtt{f} \geq 2\mathtt{f} + 3$, thus $\mathtt{S} = (\mathtt{S}_1, \cdots, \mathtt{S}_\theta)$ contains a majority of non-Byzantine gradients.

Bulyan - Step One (2/2)

- $\theta = \mathtt{n} - 2\mathtt{f} \geq 2\mathtt{f} + 3$, thus $\mathtt{S} = (\mathtt{S}_1, \cdots, \mathtt{S}_\theta)$ contains a majority of non-Byzantine gradients.
- For each $\mathtt{i} \in [1..\mathtt{d}]$, the median of the θ coordinates $\mathtt{i}$ of the selected gradients is always bounded by coordinates from non-Byzantine submissions.

Bulyan - Step Two

- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[\mathtt{d}])$.

Bulyan - Step Two

- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[d])$.
- $\forall \mathtt{i} \in [1..\mathtt{d}], \mathtt{F}[\mathtt{i}] = \frac{1}{\beta} \sum_{\mathtt{X} \in \mathtt{M}[\mathtt{i}]} \mathtt{X}[\mathtt{i}]$

Bulyan - Step Two

- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[\mathtt{d}])$.
- $\forall \mathtt{i} \in [1..\mathtt{d}], \mathtt{F}[\mathtt{i}] = \frac{1}{\beta} \sum_{\mathtt{X} \in \mathtt{M}[\mathtt{i}]} \mathtt{X}[\mathtt{i}]$
- $\beta = \theta - 2\mathtt{f} \geq 3$

Bulyan - Step Two

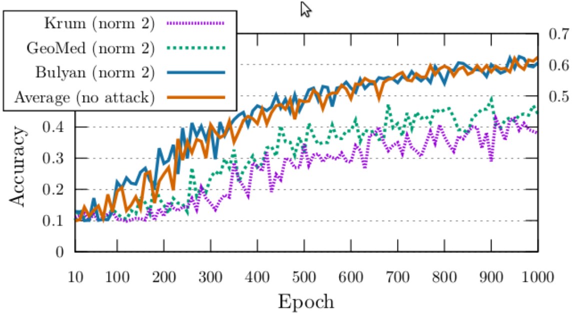
- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[\mathtt{d}])$.
- $\forall \mathtt{i} \in [1..\mathtt{d}], \mathtt{F}[\mathtt{i}] = \frac{1}{\beta} \sum_{\mathtt{X} \in \mathtt{M}[\mathtt{i}]} \mathtt{X}[\mathtt{i}]$
- $\beta = \theta - 2\mathtt{f} \geq 3$
- $\mathtt{M}[\mathtt{i}] = \arg\min_{\mathtt{R} \subset \mathtt{S}, |\mathtt{R}| = \beta} (\sum_{\mathtt{X} \in \mathtt{R}} |\mathtt{X}[\mathtt{i}] - \mathtt{median}[\mathtt{i}]|)$

Bulyan - Step Two

- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[d])$.
- $\forall \mathtt{i} \in [1..d], \mathtt{F}[\mathtt{i}] = \frac{1}{\beta} \sum_{\mathtt{X} \in \mathtt{M}[\mathtt{i}]} \mathtt{X}[\mathtt{i}]$
- $\beta = \theta - 2\mathtt{f} \geq 3$
- $\mathtt{M}[\mathtt{i}] = \arg\min_{\mathtt{R} \subset \mathtt{S}, |\mathtt{R}| = \beta} (\sum_{\mathtt{X} \in \mathtt{R}} |\mathtt{X}[\mathtt{i}] - \mathtt{median}[\mathtt{i}]|)$
- $\mathtt{median}[\mathtt{i}] = \arg\min_{\mathtt{m} = \mathtt{Y}[\mathtt{i}], \mathtt{Y} \in \mathtt{S}} (\sum_{\mathtt{Z} \in \mathtt{S}} |\mathtt{Z}[\mathtt{i}] - \mathtt{m}|)$

Bulyan - Step Two

- The second step is to generate the resulting gradient $\mathtt{F} = (\mathtt{F}[1], \cdots, \mathtt{F}[d])$.
- $\forall \mathtt{i} \in [1..d], \mathtt{F}[\mathtt{i}] = \frac{1}{\beta} \sum_{\mathtt{X} \in \mathtt{M}[\mathtt{i}]} \mathtt{X}[\mathtt{i}]$
- $\beta = \theta - 2\mathtt{f} \geq 3$
- $\mathtt{M}[\mathtt{i}] = \arg\min_{\mathtt{R} \subset \mathtt{S}, |\mathtt{R}| = \beta} (\sum_{\mathtt{X} \in \mathtt{R}} |\mathtt{X}[\mathtt{i}] - \mathtt{median}[\mathtt{i}]|)$
- $\mathtt{median}[\mathtt{i}] = \arg\min_{\mathtt{m} = \mathtt{Y}[\mathtt{i}], \mathtt{Y} \in \mathtt{S}} (\sum_{\mathtt{Z} \in \mathtt{S}} |\mathtt{Z}[\mathtt{i}] - \mathtt{m}|)$
- Each $\mathtt{ith}$ coordinate of $\mathtt{F}$ is equal to the average of the β closest $\mathtt{ith}$ coordinates to the median $\mathtt{ith}$ coordinate of the θ selected gradients.

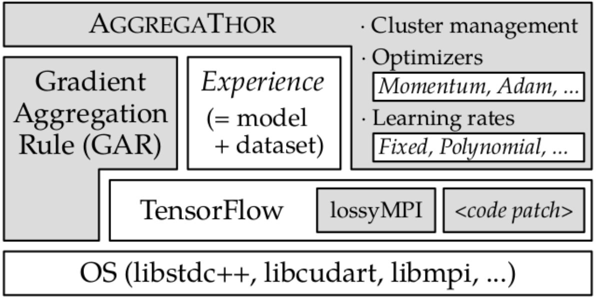


[El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018]

AggregaThor: Byzantine Machine Learning via Robust Gradient Aggregation

AggregaThor (1/2)

- A framework that handles the distribution of the training of a TensorFlow neural network graph over a cluster of machines.
- This distribution is robust to Byzantine cluster nodes.



[Damaskinos et al., AggregaThor: Byzantine Machine Learning via Robust Gradient Aggregation, 2019]

AggregaThor (2/2)

- Relies on Multi-Krum and Bulyan.

AggregaThor (2/2)

- Relies on Multi-Krum and Bulyan.
- Multi-Krum selects `m` gradients that deviate less from the majority
  - Based on their relative distances.

AggregaThor (2/2)

- Relies on Multi-Krum and Bulyan.
- Multi-Krum selects `m` gradients that deviate less from the majority
  - Based on their relative distances.
- Bulyan takes the aforementioned `m` vectors.

AggregaThor (2/2)

- Relies on Multi-Krum and Bulyan.
- Multi-Krum selects `m` gradients that deviate less from the majority
  - Based on their relative distances.
- Bulyan takes the aforementioned `m` vectors.
  - Computes their coordinate-wise median.

AggregaThor (2/2)

- Relies on Multi-Krum and Bulyan.
- Multi-Krum selects `m` gradients that deviate less from the majority
  - Based on their relative distances.
- Bulyan takes the aforementioned `m` vectors.
  - Computes their coordinate-wise median.
  - Produces a gradient that coordinates are the average of the $\mathtt{m} - 2\mathtt{f}$ closest values to the median.

TensorFlow Limitation

- In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR.

TensorFlow Limitation

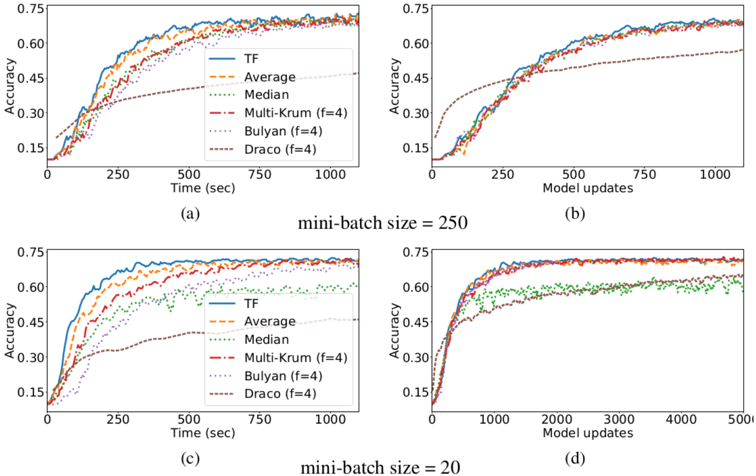
- In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR.
- TensorFlow allows any node in the cluster to execute arbitrary operations anywhere in the cluster.

TensorFlow Limitation

- In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR.
- TensorFlow allows any node in the cluster to execute arbitrary operations anywhere in the cluster.
- A single Byzantine worker could continually overwrite the shared parameters with arbitrary values.

TensorFlow Limitation

- In TensorFlow, Byzantine resilience cannot be achieved solely through the use of a Byzantine-resilient GAR.
- TensorFlow allows any node in the cluster to execute arbitrary operations anywhere in the cluster.
- A single Byzantine worker could continually overwrite the shared parameters with arbitrary values.
- AggregaThor patches TensorFlow to overcome the above issues.



mini-batch size = 250

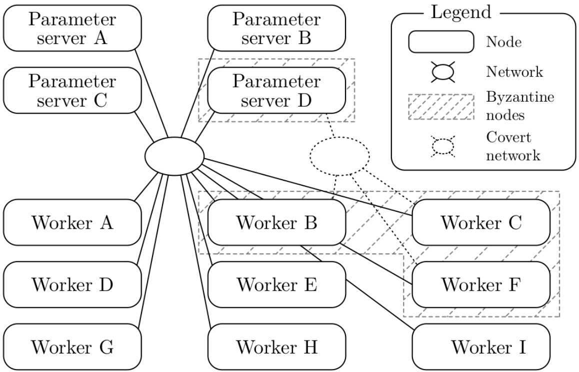
mini-batch size = 20

$$n = 19, f = 4 \ (n \geq 4f + 3)$$

[Damaskinos et al., AggregaThor: Byzantine Machine Learning via Robust Gradient Aggregation, 2019]

What if parameter servers are Byzantine?

SGD: Decentralized Byzantine Resilience



[El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019]

GuanYu

- Byzantine tolerant learning algorithm that is

GuanYu

- Byzantine tolerant learning algorithm that is
  1. Resilience to Byzantine workers.

GuanYu

- Byzantine tolerant learning algorithm that is
  1. Resilience to Byzantine workers.
  2. Resilience to Byzantine parameter servers.

GuanYu

- Byzantine tolerant learning algorithm that is
  1. Resilience to Byzantine workers.
  2. Resilience to Byzantine parameter servers.

- GuanYu tolerates up to $\frac{1}{3}$ Byzantine servers and $\frac{1}{3}$ Byzantine workers.

GuanYu

- Byzantine tolerant learning algorithm that is
  1. Resilience to Byzantine workers.
  2. Resilience to Byzantine parameter servers.

- GuanYu tolerates up to $\frac{1}{3}$ Byzantine servers and $\frac{1}{3}$ Byzantine workers.

- GuanYu uses a GAR for aggregating workers' gradients and Median for aggregating models received from servers.

Assumptions and Notations (1/2)

- Asynchronous network: the lack of any bound on communication delays.

Assumptions and Notations (1/2)

- Asynchronous network: the lack of any bound on communication delays.
- Synchronous training: bulk-synchronous training.

Assumptions and Notations (1/2)

- Asynchronous network: the lack of any bound on communication delays.
- Synchronous training: bulk-synchronous training.
  - The parameter server does not need to wait for all the workers' gradients to make progress, and vice versa.

Assumptions and Notations (1/2)

- Asynchronous network: the lack of any bound on communication delays.
- Synchronous training: bulk-synchronous training.
  - The parameter server does not need to wait for all the workers' gradients to make progress, and vice versa.
  - The quorums indicate the number of messages to wait before aggregating them.

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 3\mathtt{f_{wr}} + 3$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.

Assumptions and Notations (2/2)

- $\mathtt{n}_{\mathtt{ps}} \geq 3\mathtt{f}_{\mathtt{ps}} + 3$ the total number of parameter servers, among which $\mathtt{f}_{\mathtt{ps}}$ are Byzantine.
- $\mathtt{n}_{\mathtt{wr}} \geq 3\mathtt{f}_{\mathtt{wr}} + 3$ the total number of workers, among which $\mathtt{f}_{\mathtt{wr}}$ are Byzantine.
- $\mathtt{M}$ the coordinate-wise median (used in both workers and servers).

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 3\mathtt{f_{wr}} + 3$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.
- $\mathtt{M}$ the coordinate-wise median (used in both workers and servers).
- $\mathtt{F}$ the GAR function (used in the servers)

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 3\mathtt{f_{wr}} + 3$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.
- $\mathtt{M}$ the coordinate-wise median (used in both workers and servers).
- $\mathtt{F}$ the GAR function (used in the servers)
- $2\mathtt{f_{ps}} + 3 \leq \mathtt{q_{ps}} \leq \mathtt{n_{ps}} - \mathtt{f_{ps}}$ the quorum used for $\mathtt{M}$.

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 3\mathtt{f_{wr}} + 3$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.
- $\mathtt{M}$ the coordinate-wise median (used in both workers and servers).
- $\mathtt{F}$ the GAR function (used in the servers)
- $2\mathtt{f_{ps}} + 3 \leq \mathtt{q_{ps}} \leq \mathtt{n_{ps}} - \mathtt{f_{ps}}$ the quorum used for $\mathtt{M}$.
- $2\mathtt{f_{wr}} + 3 \leq \mathtt{q_{wr}} \leq \mathtt{n_{wr}} - \mathtt{f_{wr}}$ the quorum used for $\mathtt{F}$.

Assumptions and Notations (2/2)

- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+3$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 3\mathtt{f_{wr}} + 3$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.
- $\mathtt{M}$ the coordinate-wise median (used in both workers and servers).
- $\mathtt{F}$ the GAR function (used in the servers)
- $2\mathtt{f_{ps}} + 3 \leq \mathtt{q_{ps}} \leq \mathtt{n_{ps}} - \mathtt{f_{ps}}$ the quorum used for $\mathtt{M}$.
- $2\mathtt{f_{wr}} + 3 \leq \mathtt{q_{wr}} \leq \mathtt{n_{wr}} - \mathtt{f_{wr}}$ the quorum used for $\mathtt{F}$.
- $\mathtt{d}$ the dimension of the parameter space $\mathbb{R}^{\mathtt{d}}$.

GuanYu Algorithm - Step 1

- At each step $\mathtt{t}$, each non-Byzantine server $\mathtt{i}$ broadcasts its current parameter vector $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$ to every worker.

GuanYu Algorithm - Step 1

- At each step $\mathtt{t}$, each non-Byzantine server $\mathtt{i}$ broadcasts its current parameter vector $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$ to every worker.
- Each non-Byzantine worker $\mathtt{j}$ aggregates with $\mathtt{M}$ the $\mathtt{q}_{\mathtt{ps}}$ first received $\mathbf{w}^{\mathtt{t}}$.

GuanYu Algorithm - Step 1

- At each step $\mathtt{t}$, each non-Byzantine server $\mathtt{i}$ broadcasts its current parameter vector $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$ to every worker.
- Each non-Byzantine worker $\mathtt{j}$ aggregates with $\mathtt{M}$ the $\mathtt{q}_{\mathtt{ps}}$ first received $\mathbf{w}^{\mathtt{t}}$.
- And computes an estimate $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ of the gradient at the aggregated parameters.

GuanYu Algorithm - Step 2

- Each non-Byzantine worker $\mathtt{j}$ broadcasts its computed gradient estimation $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ to every parameter server.

GuanYu Algorithm - Step 2

- Each non-Byzantine worker $\mathtt{j}$ broadcasts its computed gradient estimation $\mathtt{G}_\mathtt{j}^\mathtt{t}$ to every parameter server.
- Each non-Byzantine parameter server $\mathtt{i}$ aggregates with $\mathtt{F}$ the $\mathtt{q}_{\mathtt{wr}}$ first received $\mathtt{G}^\mathtt{t}$.

GuanYu Algorithm - Step 2

- Each non-Byzantine worker $\mathtt{j}$ broadcasts its computed gradient estimation $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ to every parameter server.
- Each non-Byzantine parameter server $\mathtt{i}$ aggregates with $\mathtt{F}$ the $\mathtt{q}_{\mathtt{wr}}$ first received $\mathtt{G}^{\mathtt{t}}$.
- And performs a local parameter update with the aggregated gradient, resulting in $\overline{\mathbf{w}}_{\mathtt{i}}^{\mathtt{t}}$.

GuanYu Algorithm - Step 3

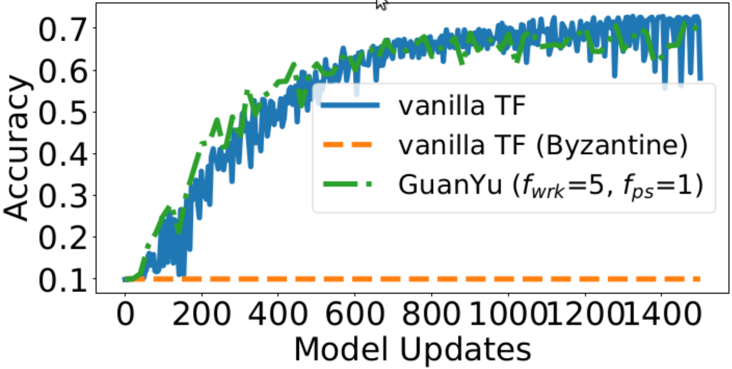
- Each non-Byzantine parameter server $\mathtt{i}$ broadcasts $\overline{\mathbf{w}}_{\mathtt{i}}^{\mathtt{t+1}}$ to every other parameter servers.

GuanYu Algorithm - Step 3

- Each non-Byzantine parameter server $\mathtt{i}$ broadcasts $\overline{\mathbf{w}}_{\mathtt{i}}^{\mathtt{t+1}}$ to every other parameter servers.
- They aggregate with $\mathtt{M}$ the $\mathtt{q_{ps}}$ first received $\overline{\mathbf{w}}_{\mathtt{k}}^{\mathtt{t+1}}$.

GuanYu Algorithm - Step 3

- Each non-Byzantine parameter server $\mathtt{i}$ broadcasts $\overline{\mathbf{w}}_{\mathtt{i}}^{\mathtt{t+1}}$ to every other parameter servers.
- They aggregate with $\mathtt{M}$ the $\mathtt{q_{ps}}$ first received $\overline{\mathbf{w}}_{\mathtt{k}}^{\mathtt{t+1}}$.
- This aggregated parameter vector is $\overline{\mathbf{w}}_{\mathtt{i}}^{\mathtt{t+1}}$.



[El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019]

GuanYu Limitations

- Network asynchrony assumtion is costly:
  - It requires three communication rounds, instead of two in the vanilla case.

GuanYu Limitations

- Network asynchrony assumtion is costly:
  - It requires three communication rounds, instead of two in the vanilla case.
- It requires a large number of compute nodes and server replicas to work, as one cannot differentiate between a Byzantine machine and a slow one in such network.

Fast Machine Learning with Byzantine Workers and Servers

LiuBei

- Uses a GAR to aggregate workers' gradients.

LiuBei

- Uses a GAR to aggregate workers' gradients.
- Tolerating Byzantine servers using a filtering technique and the scatter/gather protocol (both assume network synchrony).

LiuBei

- Uses a GAR to aggregate workers' gradients.
- Tolerating Byzantine servers using a filtering technique and the scatter/gather protocol (both assume network synchrony).
  - Scatter phase: servers work independently and do not communicate with each other.

LiuBei

- Uses a GAR to aggregate workers' gradients.
- Tolerating Byzantine servers using a filtering technique and the scatter/gather protocol (both assume network synchrony).
  - Scatter phase: servers work independently and do not communicate with each other.
  - Gather phase: correct servers communicate to bring their view of models back close to each other.

LiuBei

- Uses a GAR to aggregate workers' gradients.
- Tolerating Byzantine servers using a filtering technique and the scatter/gather protocol (both assume network synchrony).
  - Scatter phase: servers work independently and do not communicate with each other.
  - Gather phase: correct servers communicate to bring their view of models back close to each other.
  - The number of gather steps is usually very small and hence, their overhead is insignificant.

Assumptions and Notations

- Network synchrony: an upper bound on communication machines.
- $\mathtt{n_{ps}} \geq 3\mathtt{f_{ps}}+1$ the total number of parameter servers, among which $\mathtt{f_{ps}}$ are Byzantine.
- $\mathtt{n_{wr}} \geq 2\mathtt{f_{wr}} + 1$ the total number of workers, among which $\mathtt{f_{wr}}$ are Byzantine.

LiuBei - Byzantine Workers

- LiuBei can use any existing synchronous GAR that follows the robustness definition (α, f)-Byzantine-Resilience.

LiuBei - Byzantine Servers (1/2)

- Tolerating Byzantine servers using robust aggregation requires communication with all servers in each round: big communication overhead.

LiuBei - Byzantine Servers (1/2)

- Tolerating Byzantine servers using robust aggregation requires communication with all servers in each round: big communication overhead.
- LiuBei lets each worker pull only one model from any of the server replicas and then checks if the pulled model is suspicious or not.

LiuBei - Byzantine Servers (2/2)

- A worker does this check by applying two filters on the pulled model: the Lipschitz filter and the models filter.

LiuBei - Byzantine Servers (2/2)

- A worker does this check by applying two filters on the pulled model: the Lipschitz filter and the models filter.
- If the model is suspicious, the worker discards it and pulls a new model from another parameter server.

LiuBei - Byzantine Servers (2/2)

- A worker does this check by applying two filters on the pulled model: the Lipschitz filter and the models filter.
- If the model is suspicious, the worker discards it and pulls a new model from another parameter server.
- The maximum number of models that can be pulled by a worker in one iteration is $\mathtt{f_{ps}} + 1$.

Lipschitz Filter

- Assume at time $\mathtt{t}$ worker $\mathtt{j}$ owns a model $\mathbf{w}_{\mathtt{j}}^{\mathtt{t}}$ and computes gradient $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ based on that model.

Lipschitz Filter

- Assume at time $\mathtt{t}$ worker $\mathtt{j}$ owns a model $\mathbf{w}_{\mathtt{j}}^{\mathtt{t}}$ and computes gradient $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ based on that model.
- A correct server $\mathtt{i}$ should include $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ while updating its model $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$, given network synchrony.

Lipschitz Filter

- Assume at time t worker j owns a model $\mathbf{w}_j^t$ and computes gradient G_j^t based on that model.
- A correct server i should include G_j^t while updating its model $\mathbf{w}_i^t$, given network synchrony.
- The worker j then does two steps in parallel: $\mathbf{w}_i^{t+1}$

Lipschitz Filter

- Assume at time $\mathtt{t}$ worker $\mathtt{j}$ owns a model $\mathbf{w}_{\mathtt{j}}^{\mathtt{t}}$ and computes gradient $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ based on that model.
- A correct server $\mathtt{i}$ should include $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ while updating its model $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$, given network synchrony.
- The worker $\mathtt{j}$ then does two steps in parallel: $\mathbf{w}_{\mathtt{i}}^{\mathtt{t+1}}$
  1. Estimates the updated model locally based on its own gradient: $\mathbf{w}_{\mathtt{j(l)}}^{\mathtt{t+1}}$

Lipschitz Filter

- Assume at time t worker j owns a model $\mathbf{w}_j^t$ and computes gradient G_j^t based on that model.
- A correct server i should include G_j^t while updating its model $\mathbf{w}_i^t$, given network synchrony.
- The worker j then does two steps in parallel: $\mathbf{w}_i^{t+1}$
  1. Estimates the updated model locally based on its own gradient: $\mathbf{w}_{j(i)}^{t+1}$
  2. Pulls a model $\mathbf{w}_i^{t+1}$ from a parameter server i.

Lipschitz Filter

- Assume at time $\mathtt{t}$ worker $\mathtt{j}$ owns a model $\mathbf{w}_{\mathtt{j}}^{\mathtt{t}}$ and computes gradient $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ based on that model.
- A correct server $\mathtt{i}$ should include $\mathtt{G}_{\mathtt{j}}^{\mathtt{t}}$ while updating its model $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}}$, given network synchrony.
- The worker $\mathtt{j}$ then does two steps in parallel: $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}+1}$
  1. Estimates the updated model locally based on its own gradient: $\mathbf{w}_{\mathtt{j}(\mathtt{l})}^{\mathtt{t}+1}$
  2. Pulls a model $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}+1}$ from a parameter server $\mathtt{i}$.
- If server $\mathtt{i}$ is correct, then, the growth of the pulled model $\mathbf{w}_{\mathtt{i}}^{\mathtt{t}+1}$ should be close to that of the estimated local model $\mathbf{w}_{\mathtt{j}(\mathtt{l})}^{\mathtt{t}+1}$.

Model Filter

- LiuBei uses the model filter to bound the distance between models in any two successive iterations.

Model Filter

- LiuBei uses the model filter to bound the distance between models in any two successive iterations.
- We assume all correct machines initialize models with the same state.

Model Filter

- LiuBei uses the model filter to bound the distance between models in any two successive iterations.
- We assume all correct machines initialize models with the same state.
- Building upon the guarantees given by the used GAR, at iteration t, a worker can estimate an upper bound on the distance between two successive states of a correct model.

LiuBei Algorithm (1/2)

- LiuBei operates in two phases: scatter and gather.

Algorithm 1 Worker Algorithm

```
Calculate the value of T and a value for seed
model ← init_model(seed)
r ← random_int(1,n_ps)
t ← 0
grad ← model.backprop()
repeat
    local_model ← apply_grad(model,grad)
    if t mod T = 0 then
        models ← read_models()
        model ← MeaMed(models)
    else
        i ← 0
        repeat
            new_model ← read_model(
                (r + t + i) mod n_ps)
            new_grad ← new_model.backprop()
            i ← i + 1
        until pass_filters(new_model)
        model ← new_model
        grad ← new_grad
    end if
    t ← t + 1
until t > num_iterations
```

LiuBei Algorithm (1/2)

- LiuBei operates in two phases: scatter and gather.
- One gather step is executed every T iterations (line 8 to 11).

Algorithm 1 Worker Algorithm

```
1:  Calculate the value of T and a value for seed
2:  model ← init_model(seed)
3:  r ← random_int(1,n_ps)
4:  t ← 0
5:  grad ← model.backprop()
6:  repeat
7:      local_model ← apply_grad(model,grad)
8:      if t mod T = 0 then
9:          models ← read_models()
10:         model ← MeaMed(models)
11:     else
12:         i ← 0
13:         repeat
14:             new_model ← read_model(
                    (r + t + i) mod n_ps)
15:             new_grad ← new_model.backprop()
16:             i ← i + 1
17:         until pass_filters(new_model)
18:         model ← new_model
19:         grad ← new_grad
20:     end if
21:     t ← t + 1
22: until t > num_iterations
```

LiuBei Algorithm (1/2)

- LiuBei operates in two phases: scatter and gather.
- One gather step is executed every T iterations (line 8 to 11).
- We call the whole T iterations a scatter step.

Algorithm 1 Worker Algorithm

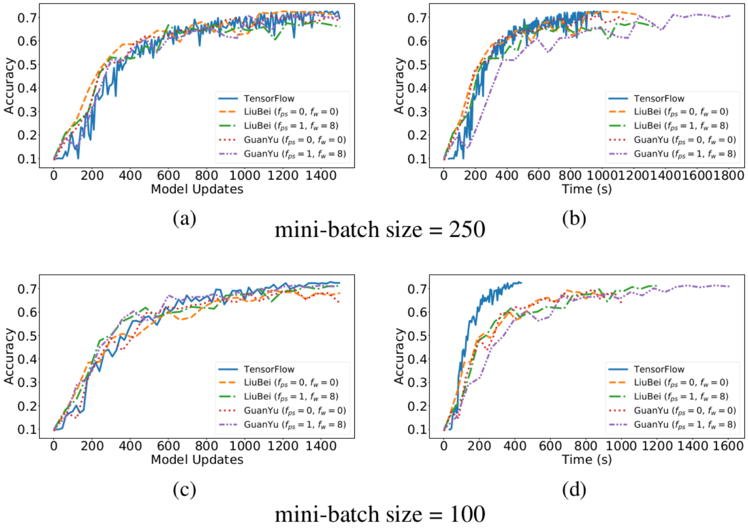
1: Calculate the value of T and a value for *seed*
2: model ← init_model(seed)
3: r ← random_int(1,n_{ps})
4: $t \leftarrow 0$
5: grad ← model.backprop()
6: **repeat**
7:     local_model ← apply_grad(model,grad)
8:     **if** $t \mod T = 0$ **then**
9:         models ← read_models()
10:         model ← MeaMed(models)
11:     **else**
12:         $i \leftarrow 0$
13:         **repeat**
14:             new_model ← read_model($(r + t + i) \mod n_{ps}$)
15:             new_grad ← new_model.backprop()
16:             $i \leftarrow i + 1$
17:         **until** pass_filters(new_model)
18:         model ← new_model
19:         grad ← new_grad
20:     **end if**
21:     $t \leftarrow t + 1$
22: **until** $t >$ num_iterations

LiuBei Algorithm (2/2)

- LiuBei operates in two phases: scatter and gather.
- One gather step is executed every `T` iterations (line 8 to 11).
- We call the whole `T` iterations a scatter step.

Algorithm 2 Parameter Server Algorithm

1: Calculate the value of T and a value for *seed*
2: model $\leftarrow$ init_model(seed)
3: $t \leftarrow 0$
4: **repeat**
5:     grads $\leftarrow$ read_gradients()
6:     grad $\leftarrow$ MDA(grads)
7:     model.update(grad)
8:     **if** $t \mod T = 0$ **then**
9:         model $\leftarrow$ read_models()
10:         model $\leftarrow$ MeaMed(models)
11:     **end if**
12:     $t \leftarrow t + 1$
13: **until** $t >$ num_iterations



[El Mhamdi et al., Fast Mahcine Learning with Byzantine Workers and Servers, 2019]

Summary

Summary

- Integrity in data-parallel learning
- Weak Byzantine resilience
- Strong Byzantine resilience
- Byzantine parameter servers

Reference

- Xie et al., Generalized Byzantine-tolerant SGD, 2018
- Blanchard et al., Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent, 2017
- El Mhamdi et al., The Hidden Vulnerability of Distributed Learning in Byzantium, 2018
- Damaskinos et al., AGGREGATHOR: Byzantine Machine Learning via Robust Gradient Aggregation, 2019
- El Mhamdi et al., SGD: Decentralized Byzantine Resilience, 2019
- El Mhamdi et al., Fast Machine Learning with Byzantine Workers and Servers, 2019

Questions?