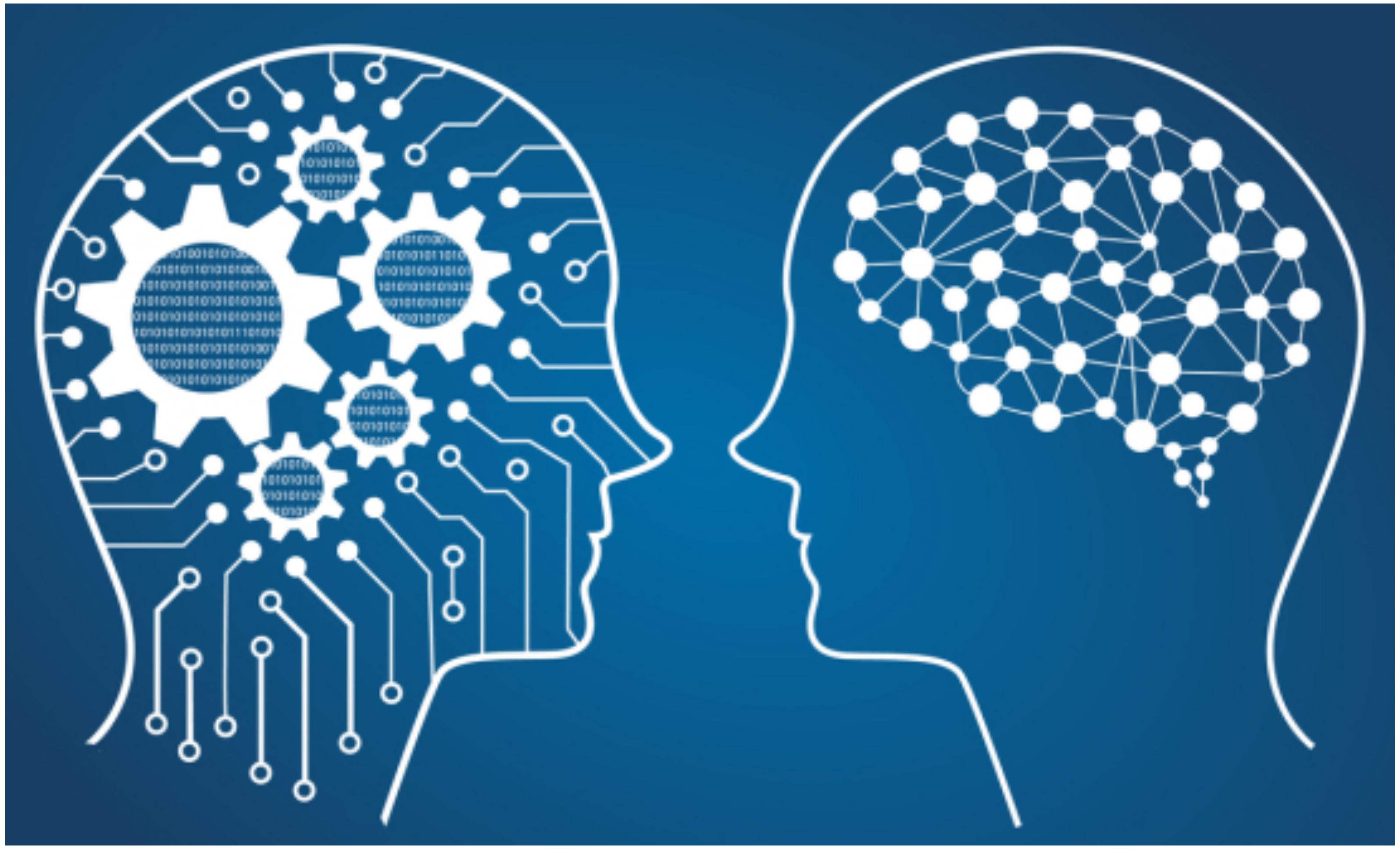


Distributed Machine Learning

Basics and Recent Work

Ashkan Yousefpour

Computer Science
University of Texas at Dallas
CS7301-003   Fall 2018

October, 2018

Image borrowed form https://news.sophos.com/en-us/2017/07/24/5-questions-to-ask-about-machine-learning/

Outline

- Intro: Distributed Machine Learning (DML)
  - Based on *Tiffany Tuor et al. “Distributed Machine Learning in Coalition Environments: Overview of Techniques”*
  - 1- Training
  - 2- Test/Inference
- Recent work in DML
  - Paper review
  - *Chen, Lingjiao, Hongyi Wang, and Dimitris Papailiopoulos. "Draco: Robust Distributed Training against Adversaries." (SysML 2018)*.

Part 1

Intro: Distributed Machine Learning

Introduction

- Machine Learning
    - Benefits
    - Applications
- ML is resource-intensive
    - Distributed Machine Learning (DML)

Distributed Machine Learning

- 1- Distributed training
  - Scalability
  - Improve performance
- 2- Distributed test/inference
  - Reduce bandwidth
  - Reduce delay
  - E.g. in edge computing and IoT

Distributed Machine Learning

- 1- Distributed training
  - Scalability
  - Improve performance
- 2- Distributed test/inference
  - Reduce bandwidth
  - Reduce delay
  - E.g. in edge computing and IoT

Training

- Loss function
- Supervised vs. unsupervised
- Loss function:
  - Soft-SVM $l(w, x, y) = \lambda\,|\,w\,|_2 + \max(0,1 - y(w\,.\,x))$
  - Logistic regression $l(w, x, y) = log(1 + exp(-y(w\,.\,x)))$
  - K-means $l(w, x) = min_k\,|\,x - w(k)\,|^2$

Training

- Training Data D
  - Tuples (x, y)
- Overall Loss $L(w) = \frac{1}{|D|} \sum_{(x,y) \in D} l(w, x, y)$

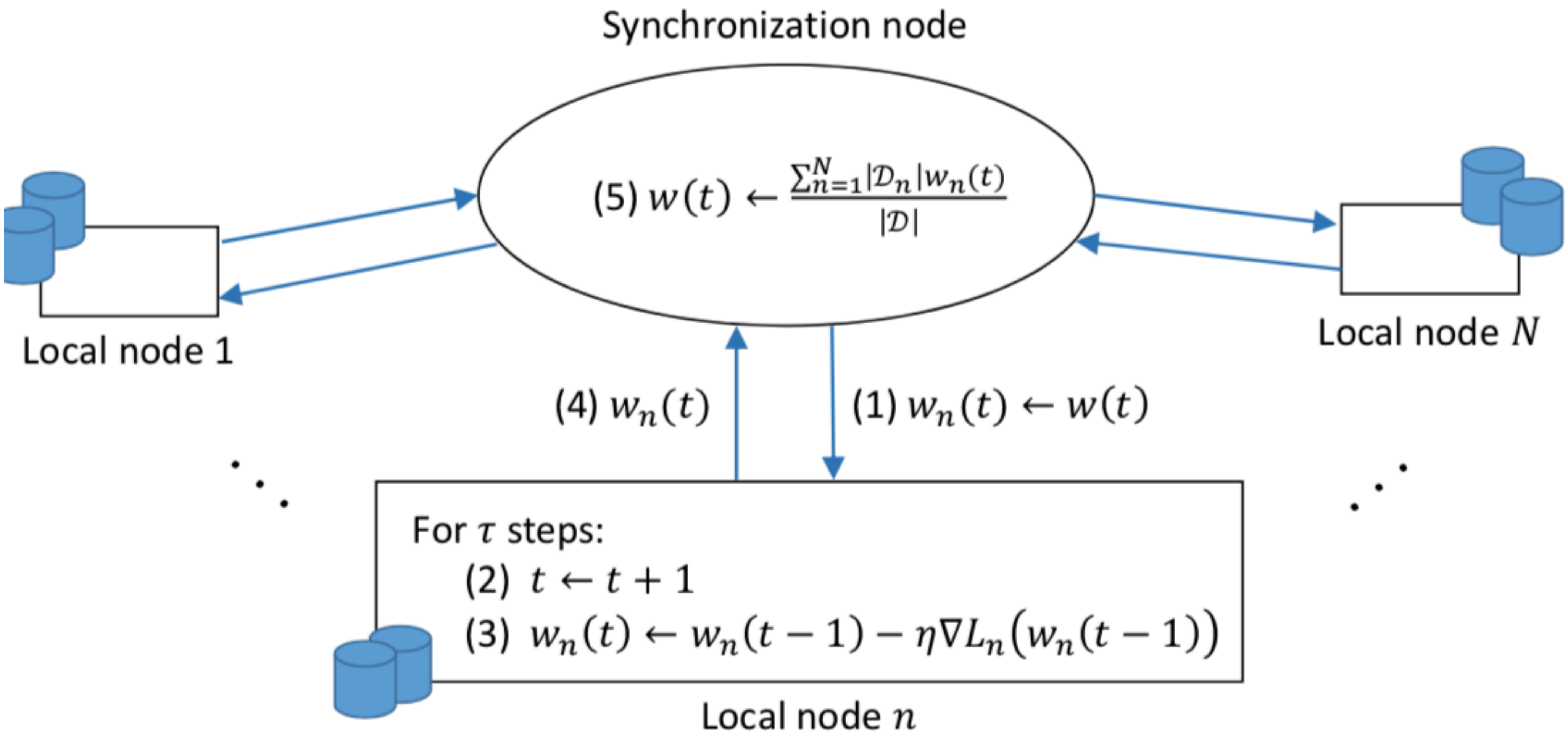
Gradient Descent

$$w(t) = w(t-1) - \eta \nabla L(w(t-1))$$

- 1- Deterministic (GD)
  - Over all training data
- 2- Stochastic (SGD)
  - Over (random) subset of training data $\tilde{D}$
  - Size of $\tilde{D}$ -> Mini-batch size
  - $\tilde{L}(w) = \frac{1}{|\tilde{D}|} \sum_{(x,y)\in\tilde{D}} l(w, x, y)$
  - Theorem: When the samples in $\tilde{D}$ are uniformly drawn (without replacement) from D , we have $E(\tilde{L}(w)) = L(w)$ $E(\bigtriangledown \tilde{L}(w)) = \bigtriangledown L(w)$
- Distributed?

Distributed Gradient Descent (DGD)

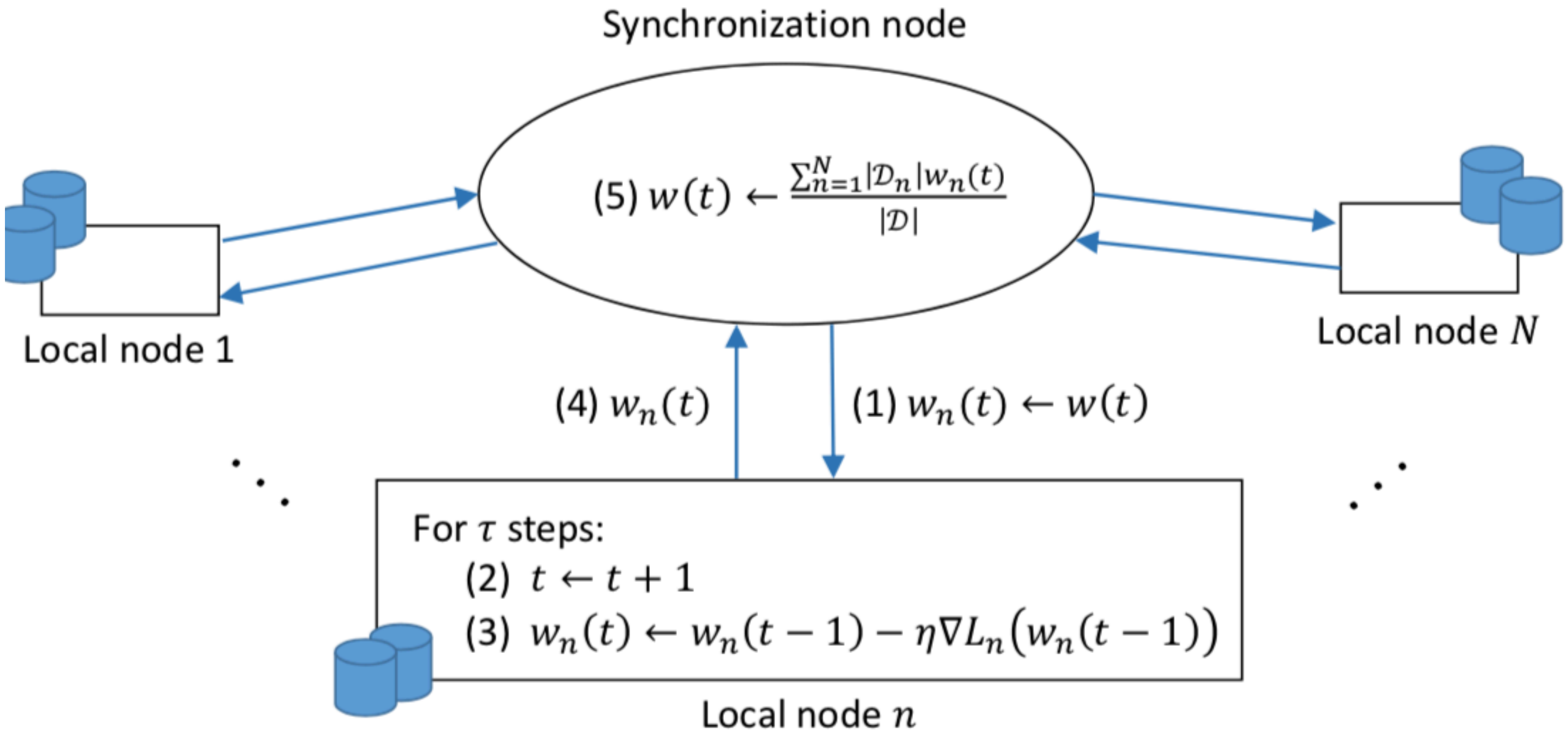
- There are N different nodes
- Each node n has a local training dataset D_n


Synchronization node
(5) $w(t) \leftarrow \frac{\sum_{n=1}^{N} |\mathcal{D}_n| w_n(t)}{|\mathcal{D}|}$
Local node 1
Local node N
(4) $w_n(t)$
(1) $w_n(t) \leftarrow w(t)$
For τ steps:
(2) $t \leftarrow t + 1$
(3) $w_n(t) \leftarrow w_n(t-1) - \eta \nabla L_n\big(w_n(t-1)\big)$
Local node n

Distributed Gradient Descent (DGD) - steps

- No need to send all of the date to all nodes
- Just need gradients locally
  - Deterministic $w_n(t) = w_n(t-1) - \eta \nabla L_n(w(t-1))$ $D = \bigcup_n D_n$
  - Stochastic $w_n(t) = w_n(t-1) - \eta \nabla \tilde{L}_n(w(t-1))$ $\tilde{D} = \bigcup_n \tilde{D}_n$
- Synchronization: weighted average of local weights
  - weight: size of local data

DGD is same as GD



Synchronization node
(5) $w(t) \leftarrow \frac{\sum_{n=1}^{N}|\mathcal{D}_n| w_n(t)}{|\mathcal{D}|}$
Local node 1
Local node N
(4) $w_n(t)$
(1) $w_n(t) \leftarrow w(t)$
For τ steps:
(2) $t \leftarrow t+1$
(3) $w_n(t) \leftarrow w_n(t-1) - \eta \nabla L_n\big(w_n(t-1)\big)$
Local node n

- Theorem 2. When $w_n(t-1) = w(t-1)$ for all n, after performing local deterministic gradient descent according to step(3) and computing $\mathrm{w(t)}$ according to step(5), we have: $w(t) = w(t-1) - \eta \nabla L(w(t-1))$

Notes on DGD

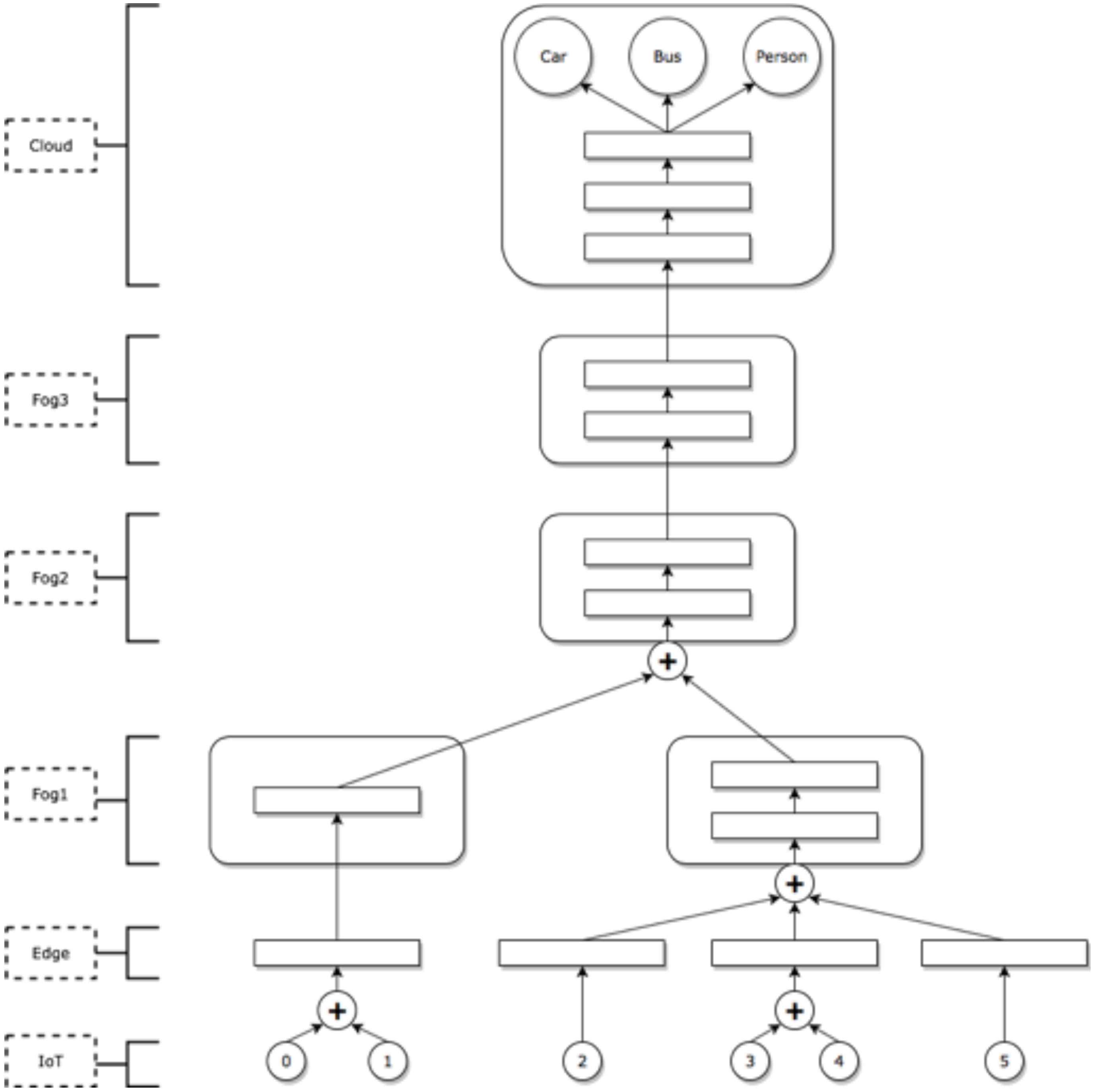
- Data samples in $\tilde{D}_n$ is drawn at each node independently, from its local dataset D_n
- Theorem2 holds for stochastic GD if $\frac{|\tilde{D}_n|}{|D_n|} = \frac{|\tilde{D}|}{|D|}$
- In general, distributed and centralized gradient descents are *not equivalent* when there are multiple steps of local iteration between synchronization $\tau > 1$
- Large value of τ saves bandwidth (but impacts learning accuracy) (comm vs. comp tradeoff: [1])

[1] S. Wang, et al. "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. of IEEE INFOCOM*, 2018

Distributed Machine Learning

- 1- Distributed training
  - Scalability
  - Improve performance
- 2- Distributed test/inference
  - Reduce bandwidth
  - Reduce delay
  - E.g. in edge computing and IoT

2- Distributed Test



2- Distributed Test

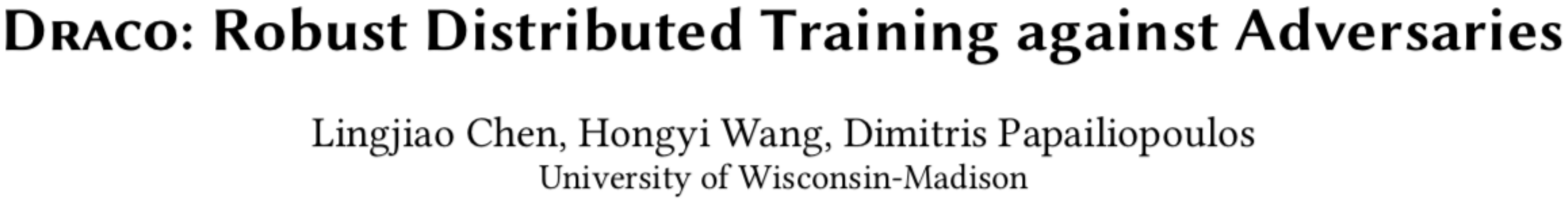
- Pipeline of modules/functions
- Modules may run on separate physical nodes
- E.g. run few layers on edge, more layers in cloud
  - Data size gets smaller as it traverses to cloud
  - Saves bandwidth, delay. Improves privacy
- Optimization: How to partition the model onto nodes
  - For faster test (minimize test time)

Part 2

Paper Review

Recent Paper Review

- DRACO
  - SysML 2018
  - Extended: ICML 2018

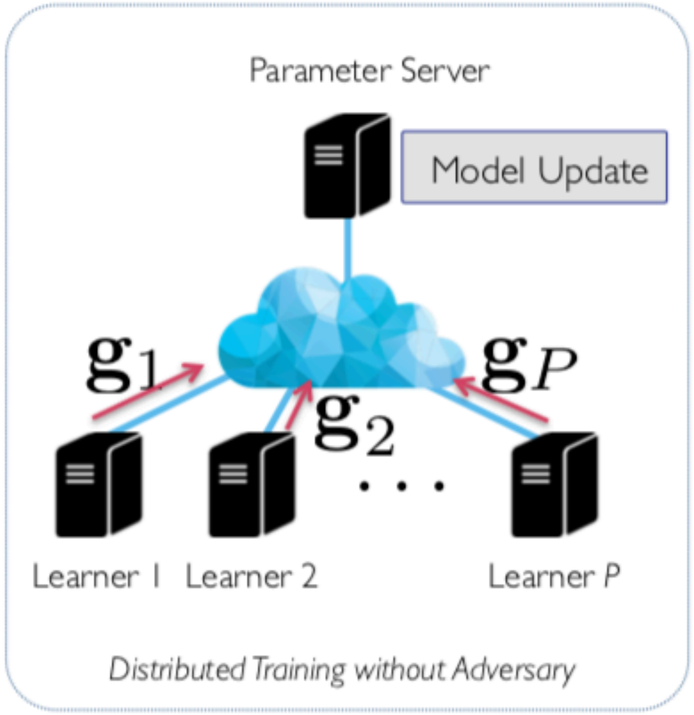


DRACO: Robust Distributed Training against Adversaries

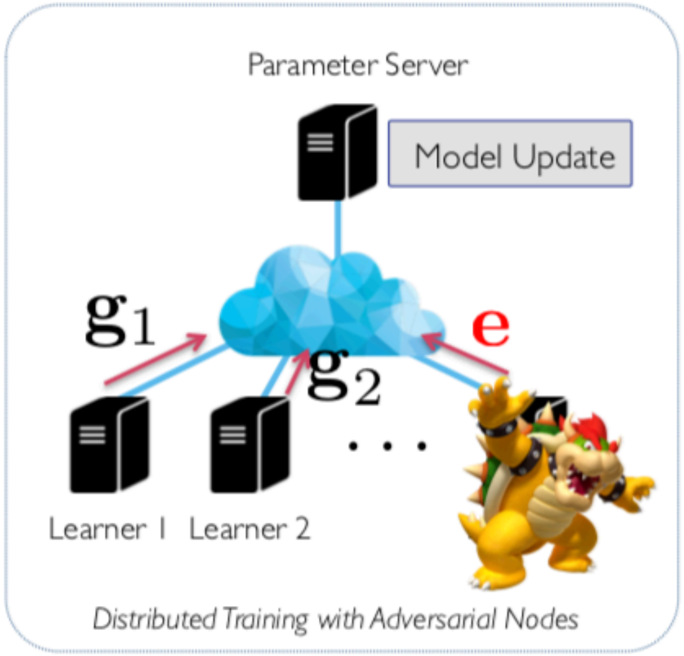
Lingjiao Chen, Hongyi Wang, Dimitris Papailiopoulos
University of Wisconsin-Madison

Draco Introduction

- Challenge: Robustness in DML
  - Failure of nodes
  - Adversarial nodes <—
- In both cases:
  - During training <—
  - During test



Distributed Training without Adversary



Distributed Training with Adversarial Nodes

Draco Motivation

- State of the art
    - Synchronized training
    - Geometric median, rather than average
        - (addresses up to 1/2 adversaries)
- Shortcoming
    - Calculating geometric median is computationally intensive

Draco Idea

- Idea: redundancy!
- Allow node to evaluate redundant gradients
- Parameter Server (PS) can
  - Detect adversaries (defend via majority)
  - Recover correct average from non-adversaries
- More redundancy, more robustness

Draco Framework

- Uses mini-batch (distributed) SGD

$$w(t) = w(t-1) - \frac{\eta}{|D_k|} \sum_{(x,y)\in D_k} \nabla L(w(t-1))$$

  - PS stores a global model
  - Data is distributed among P nodes
  - During computation phase on mini batch, each node samples B/P data points from its local subset of data
  - Ship gradients to PS. Then PS applies to global model. Sends update back to workers

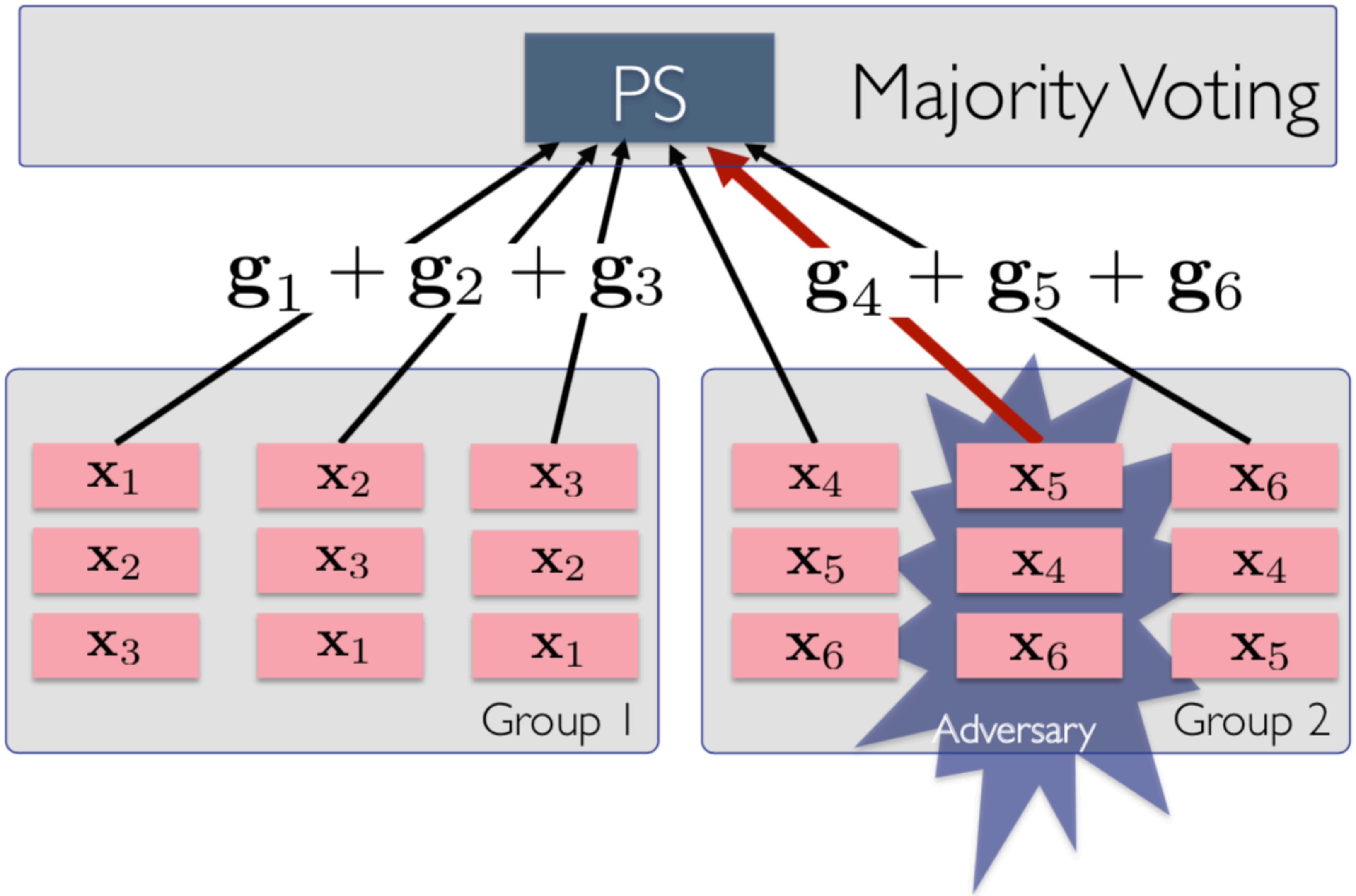
Draco Adversaries

- Adversaries: *A compute node is considered to be an adversarial node, if it does not return the prescribed gradient update given its allocated samples. Such a node can ship back to the PS any arbitrary update of dimension equal to that of the true gradient.*
- Even one adversary can fail the model to converge
- Draco: robustness by algorithmic redundancy while guaranteeing an identical model to that of mini-batch SGD in the adversary-free case

Draco Robustness

- Instead of relying on PS, nodes will do more gradients
  - Instead of B/P gradients, they calc r.B/P
- Up to (r - 1)/2 adversarial (50%) with no effect
- To tolerate s adversaries, r > 2s
- Borrow from coding theory: cyclic code, repetition code

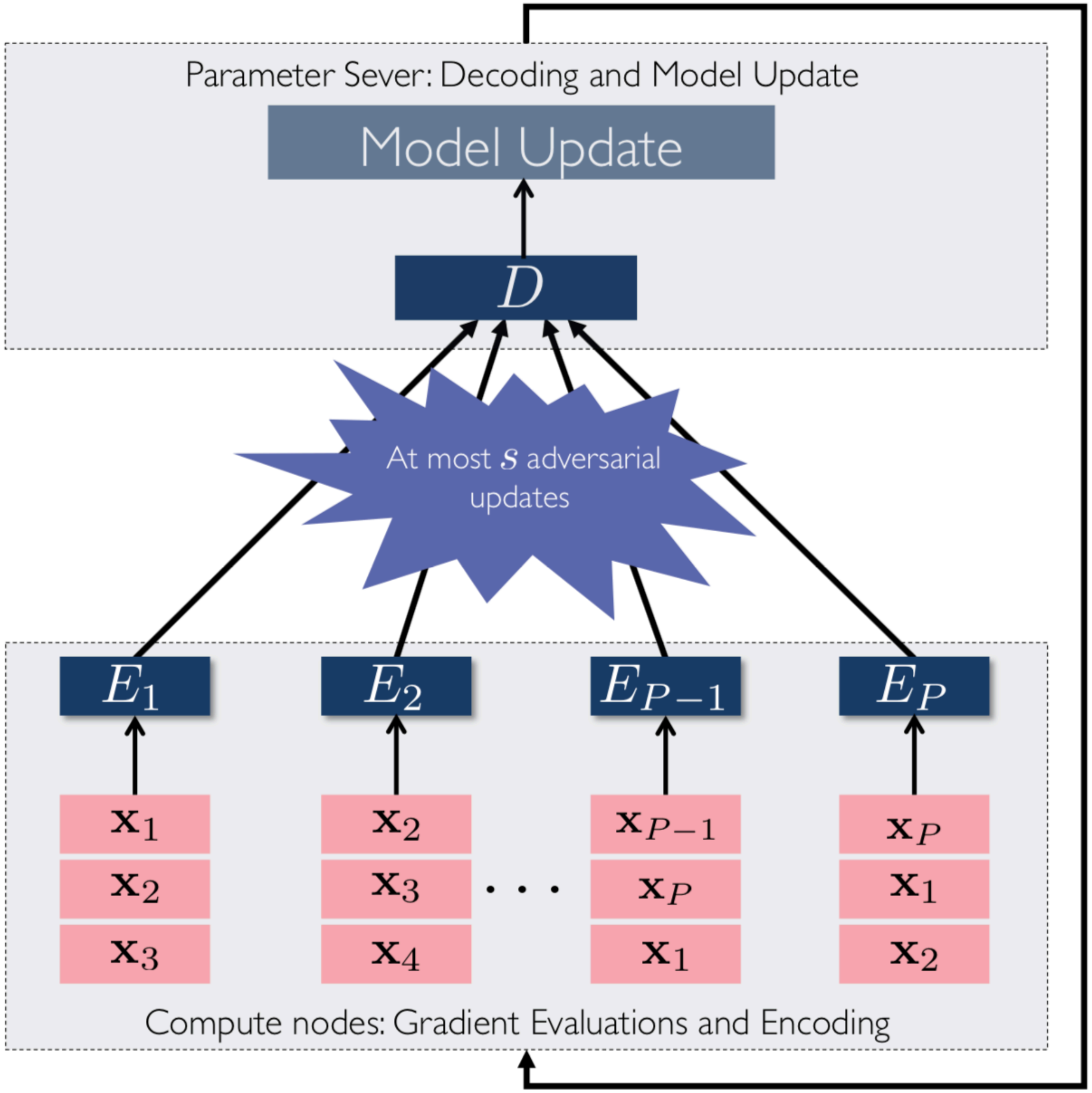
Majority Voting


PS
Majority Voting
$\mathbf{g}_1 + \mathbf{g}_2 + \mathbf{g}_3$
$\mathbf{g}_4 + \mathbf{g}_5 + \mathbf{g}_6$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$
$\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_2$
$\mathbf{x}_3$ $\mathbf{x}_1$ $\mathbf{x}_1$
Group 1
$\mathbf{x}_4$ $\mathbf{x}_5$ $\mathbf{x}_6$
$\mathbf{x}_5$ $\mathbf{x}_4$ $\mathbf{x}_4$
$\mathbf{x}_6$ $\mathbf{x}_6$ $\mathbf{x}_5$
Adversary
Group 2

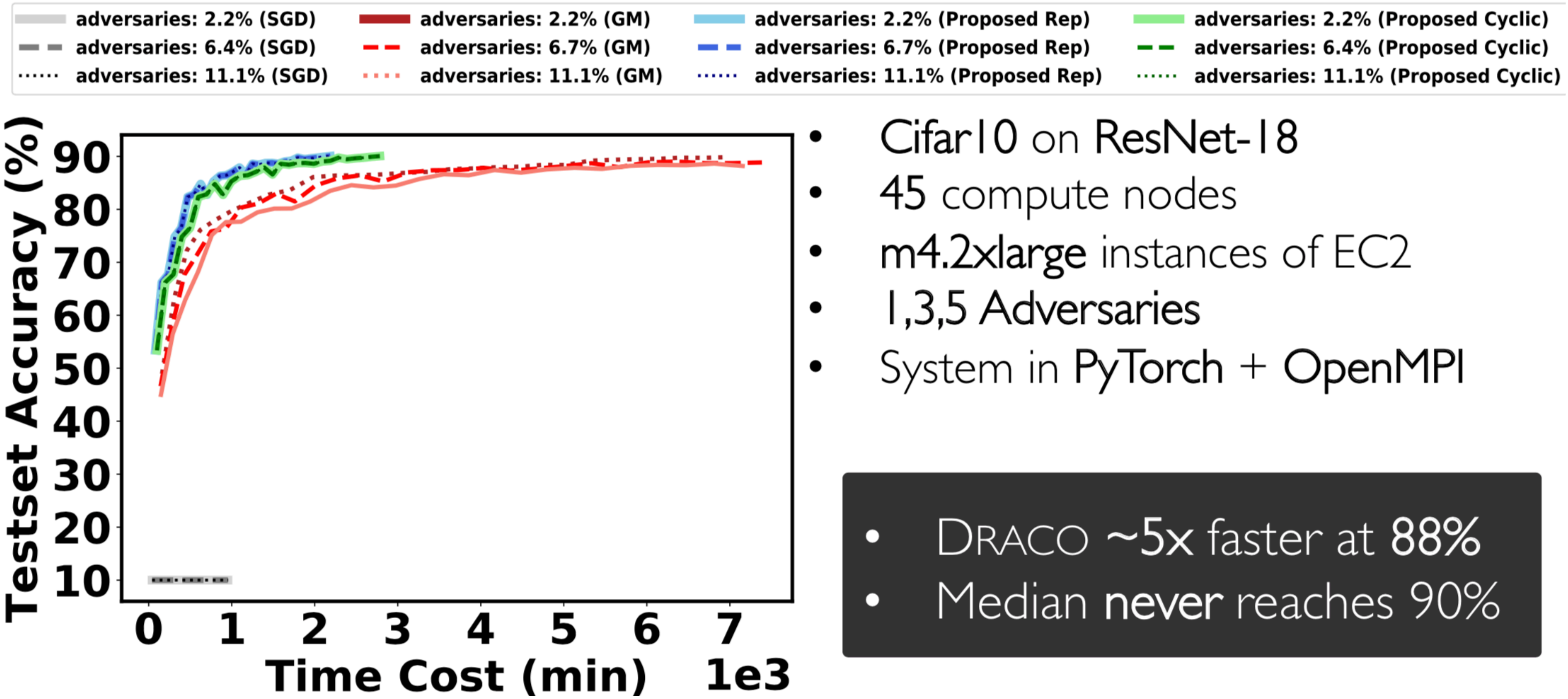
Draco Framework

- Divide compute nodes in (2s+1) groups
- In each group, all nodes compute the average of a set of exactly the same gradients
- Each nodes sends their respective sum back to the PS. The PS aggregates these sums to update the model
- In the same group, PS computes majority

Draco Majority Voting



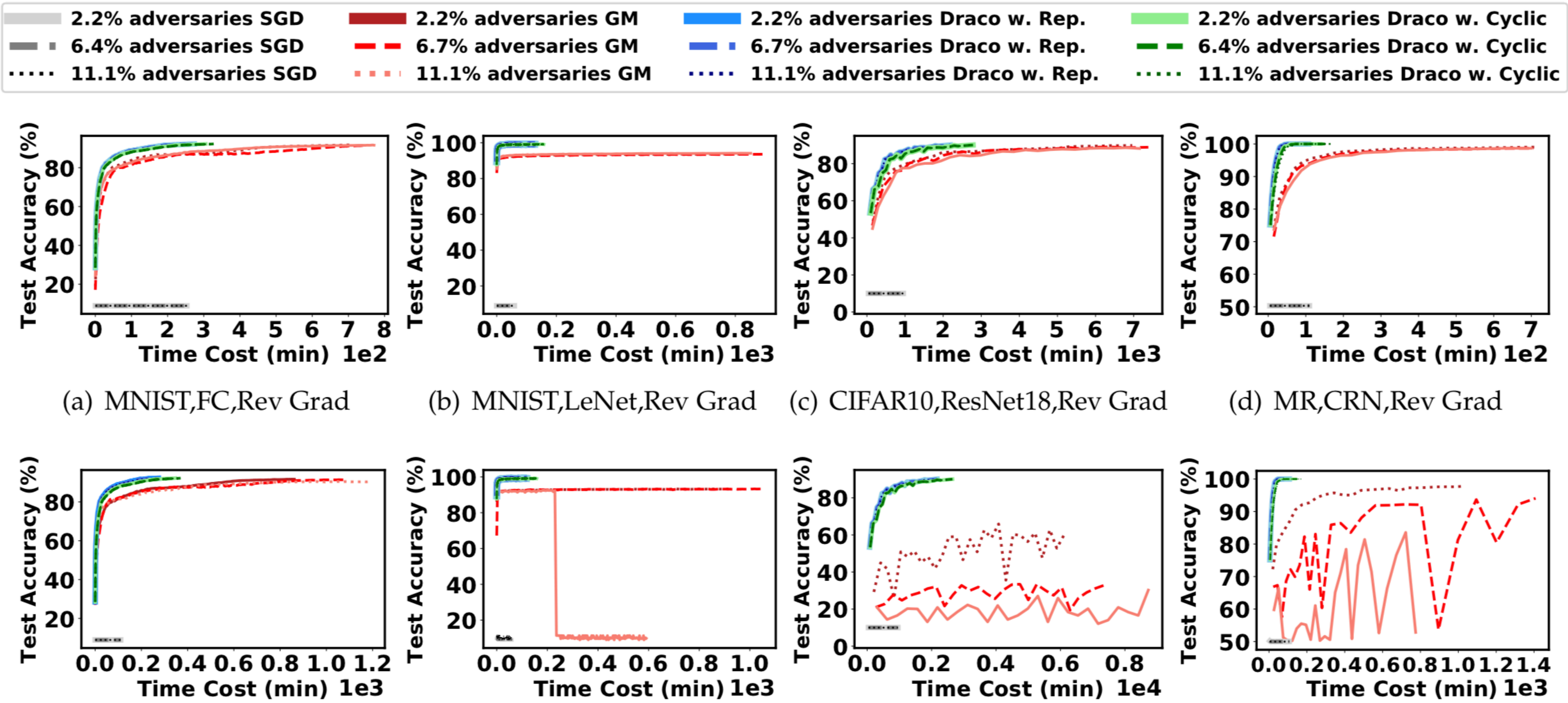
Results



- Cifar10 on ResNet-18
- 45 compute nodes
- m4.2xlarge instances of EC2
- 1,3,5 Adversaries
- System in PyTorch + OpenMPI

- DRACO ~5x faster at 88%
- Median **never** reaches 90%

Results



(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

Results

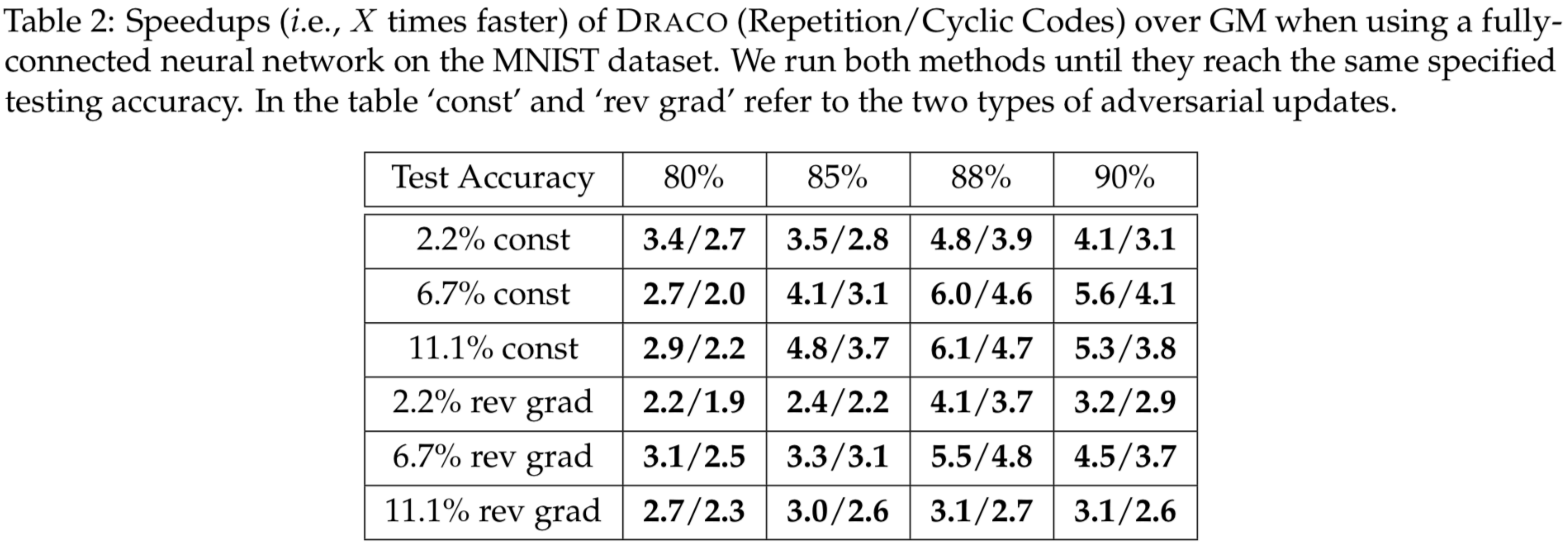


Table 2: Speedups (*i*.e., X times faster) of DRACO (Repetition/Cyclic Codes) over GM when using a fully-connected neural network on the MNIST dataset. We run both methods until they reach the same specified testing accuracy. In the table 'const' and 'rev grad' refer to the two types of adversarial updates.

Test Accuracy	80%	85%	88%	90%
2.2% const	**3.4/2.7**	**3.5/2.8**	**4.8/3.9**	**4.1/3.1**
6.7% const	**2.7/2.0**	**4.1/3.1**	**6.0/4.6**	**5.6/4.1**
11.1% const	**2.9/2.2**	**4.8/3.7**	**6.1/4.7**	**5.3/3.8**
2.2% rev grad	**2.2/1.9**	**2.4/2.2**	**4.1/3.7**	**3.2/2.9**
6.7% rev grad	**3.1/2.5**	**3.3/3.1**	**5.5/4.8**	**4.5/3.7**
11.1% rev grad	**2.7/2.3**	**3.0/2.6**	**3.1/2.7**	**3.1/2.6**