北京大學
PEKING UNIVERSITY

Distributed Machine Learning (Part I)

Kun Yuan

Center for Machine Learning Research @ Peking University

MLSS, Okinawa Mar. 6, 2024

About me

- 2014 – 2019, Ph.D. in ECE@UCLA

- 2019 – 2022, Algorithm engineer in Alibaba Group

- 2022 – Present, Assistant professor, Peking University

- Research interests:

  Fast, **scalable**, **reliable**, and **distributed** algorithms for large-scale machine learning

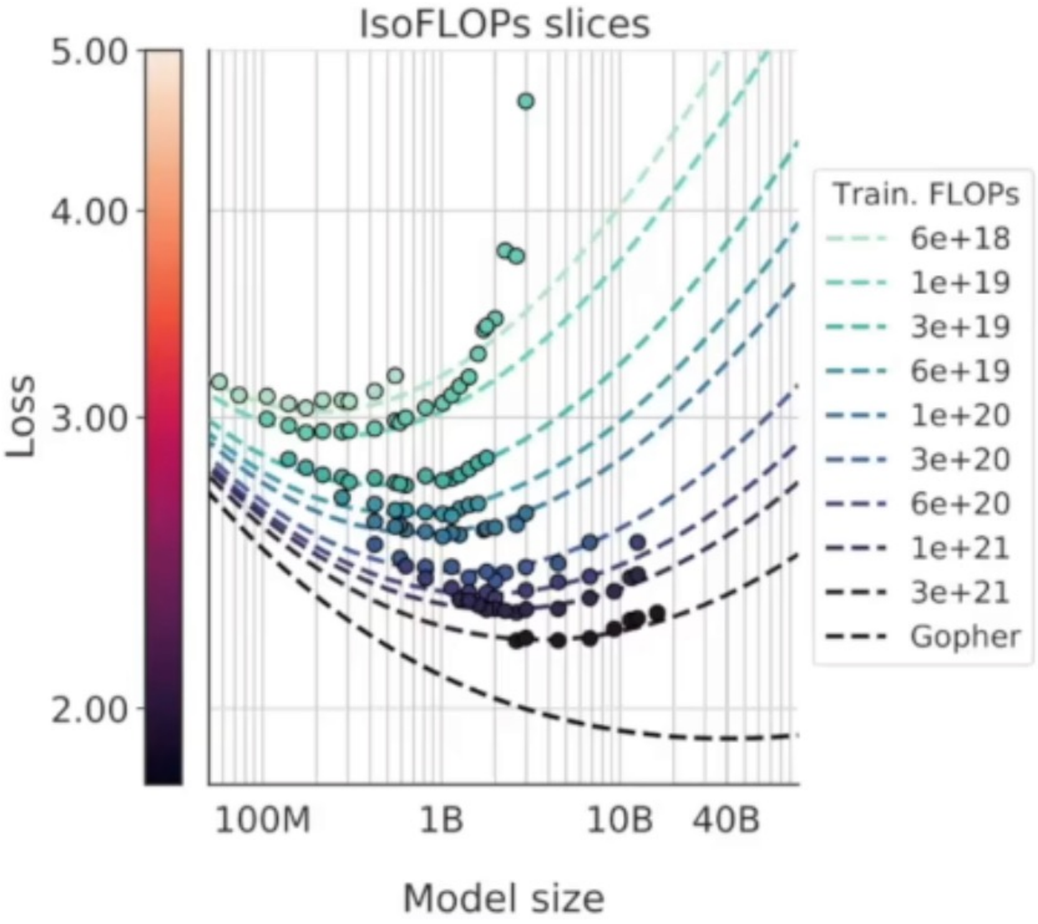
- Recipient of 2017 IEEE Signal Processing Society Young Author Best Paper Award (joint with Dr. Wei Shi)

Preface

Motivations

Chinchilla law

[Training Compute-Optimal Large Language Models, 2022]



The "RGB" elements in LLM:

Larger dataset

+

Bigger model

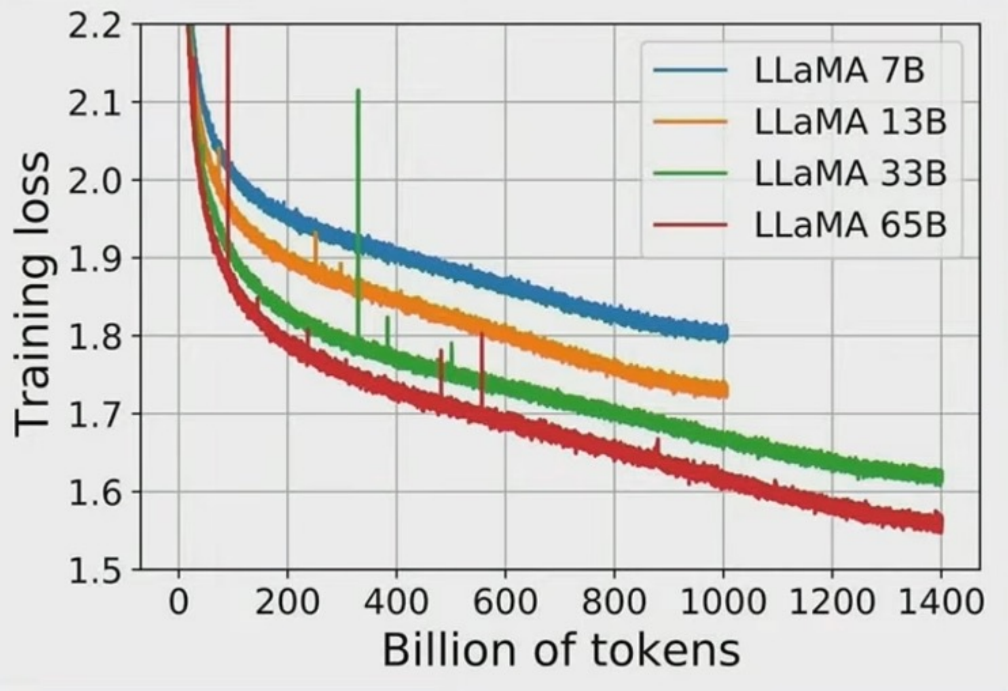
+

Longer training

= Better LLM

LLaMA follows Chinchilla law

[LLaMA: Open and Efficient Foundation Language Models,2023]

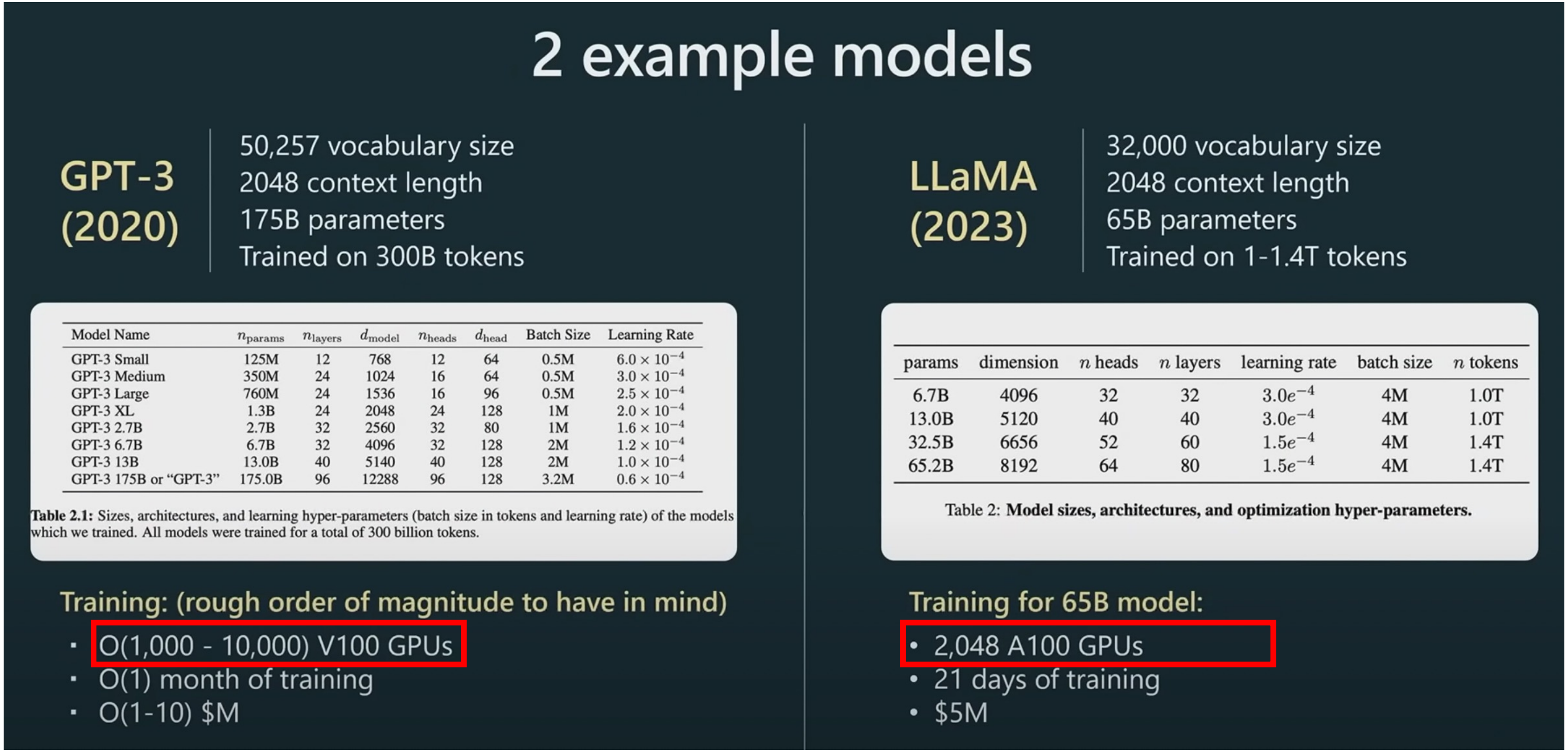


According to Chinchilla law:

- LLM model gets increasingly bigger
- Dataset gets increasingly larger
- Distributed training gets increasingly important

Thousands of GPUs are needed to train LLM

[State of GPT, 2023]



2 example models

GPT-3 (2020)

50,257 vocabulary size
2048 context length
175B parameters
Trained on 300B tokens

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

Training: (rough order of magnitude to have in mind)

- O(1,000 - 10,000) V100 GPUs
- O(1) month of training
- O(1-10) $M

LLaMA (2023)

32,000 vocabulary size
2048 context length
65B parameters
Trained on 1-1.4T tokens

params	dimension	n heads	n layers	learning rate	batch size	n tokens
6.7B	4096	32	32	$3.0e^{-4}$	4M	1.0T
13.0B	5120	40	40	$3.0e^{-4}$	4M	1.0T
32.5B	6656	52	60	$1.5e^{-4}$	4M	1.4T
65.2B	8192	64	80	$1.5e^{-4}$	4M	1.4T

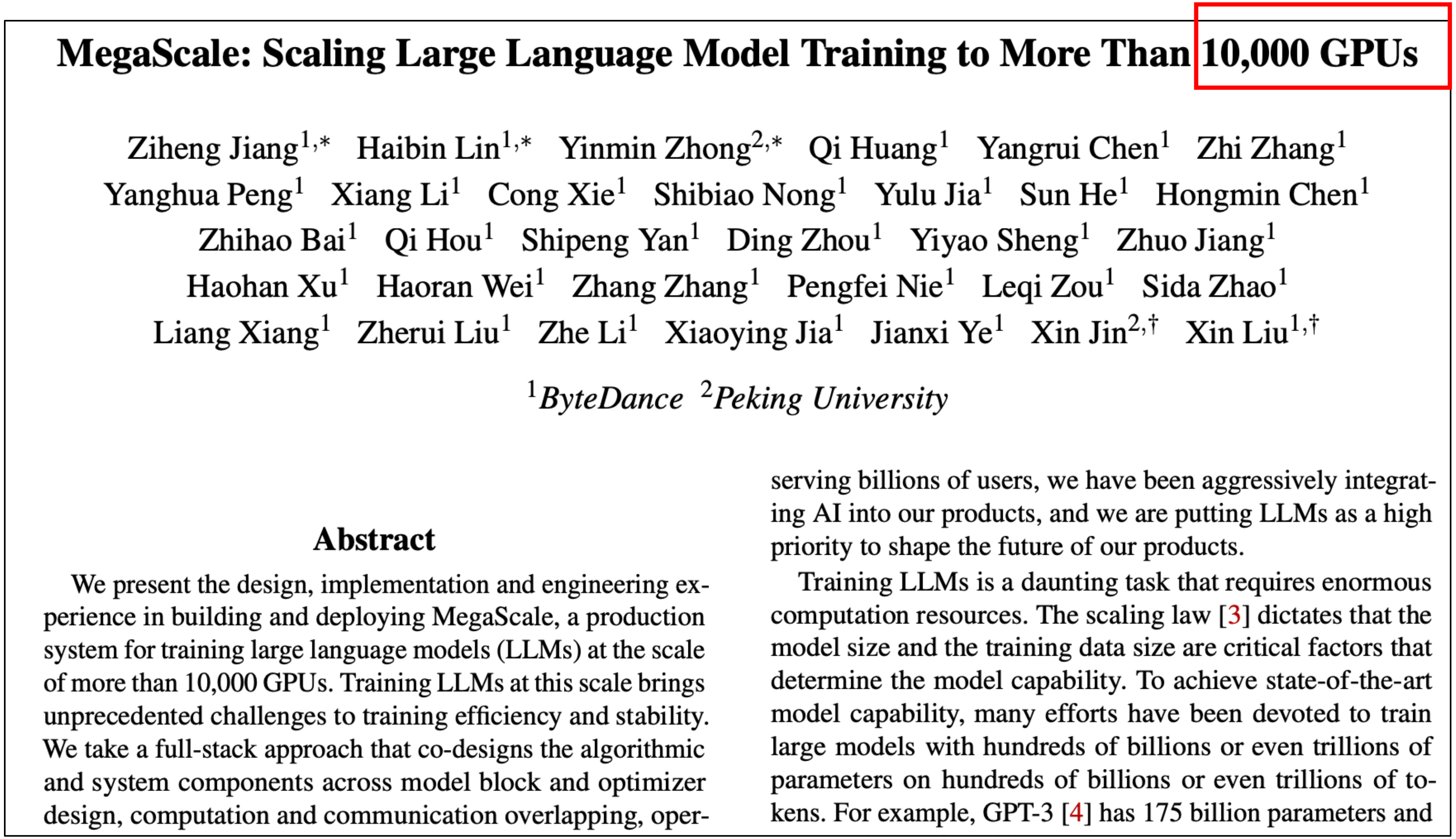
Table 2: **Model sizes, architectures, and optimization hyper-parameters.**

Training for 65B model:

- 2,048 A100 GPUs
- 21 days of training
- $5M

Recent GPU clusters scale with more than 10000 GPUs

Feb. 23, 2024



MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs

Ziheng Jiang[1,*] Haibin Lin[1,*] Yinmin Zhong[2,*] Qi Huang[1] Yangrui Chen[1] Zhi Zhang[1]
Yanghua Peng[1] Xiang Li[1] Cong Xie[1] Shibiao Nong[1] Yulu Jia[1] Sun He[1] Hongmin Chen[1]
Zhihao Bai[1] Qi Hou[1] Shipeng Yan[1] Ding Zhou[1] Yiyao Sheng[1] Zhuo Jiang[1]
Haohan Xu[1] Haoran Wei[1] Zhang Zhang[1] Pengfei Nie[1] Leqi Zou[1] Sida Zhao[1]
Liang Xiang[1] Zherui Liu[1] Zhe Li[1] Xiaoying Jia[1] Jianxi Ye[1] Xin Jin[2,†] Xin Liu[1,†]

[1]*ByteDance* [2]*Peking University*

Abstract

We present the design, implementation and engineering experience in building and deploying MegaScale, a production system for training large language models (LLMs) at the scale of more than 10,000 GPUs. Training LLMs at this scale brings unprecedented challenges to training efficiency and stability. We take a full-stack approach that co-designs the algorithmic and system components across model block and optimizer design, computation and communication overlapping, oper-

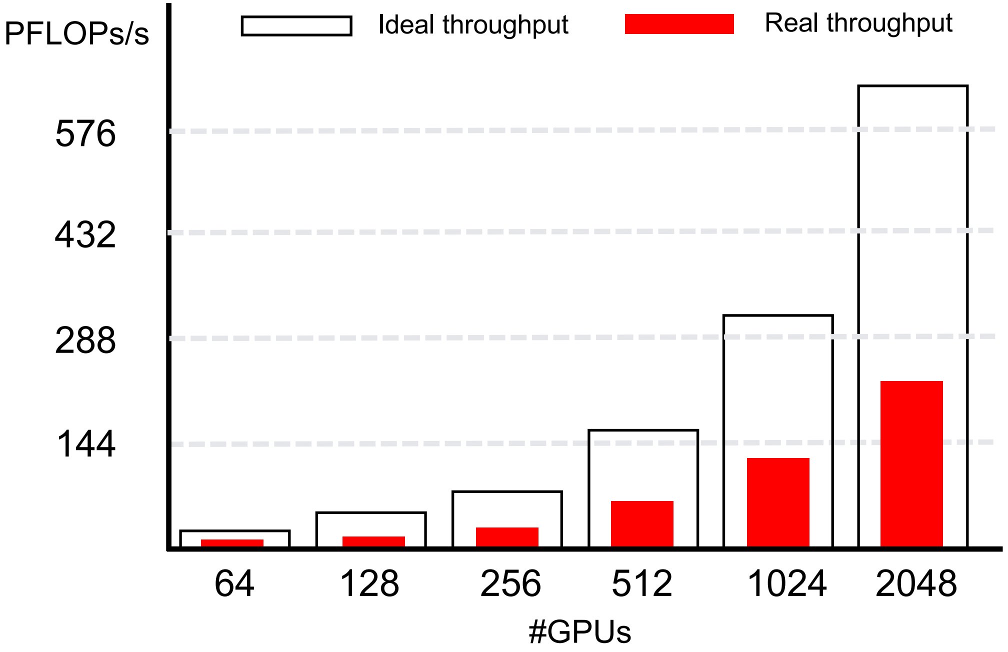
serving billions of users, we have been aggressively integrating AI into our products, and we are putting LLMs as a high priority to shape the future of our products.

Training LLMs is a daunting task that requires enormous computation resources. The scaling law [3] dictates that the model size and the training data size are critical factors that determine the model capability. To achieve state-of-the-art model capability, many efforts have been devoted to train large models with hundreds of billions or even trillions of parameters on hundreds of billions or even trillions of tokens. For example, GPT-3 [4] has 175 billion parameters and

Distributed training over massive GPUs is extremely challenging

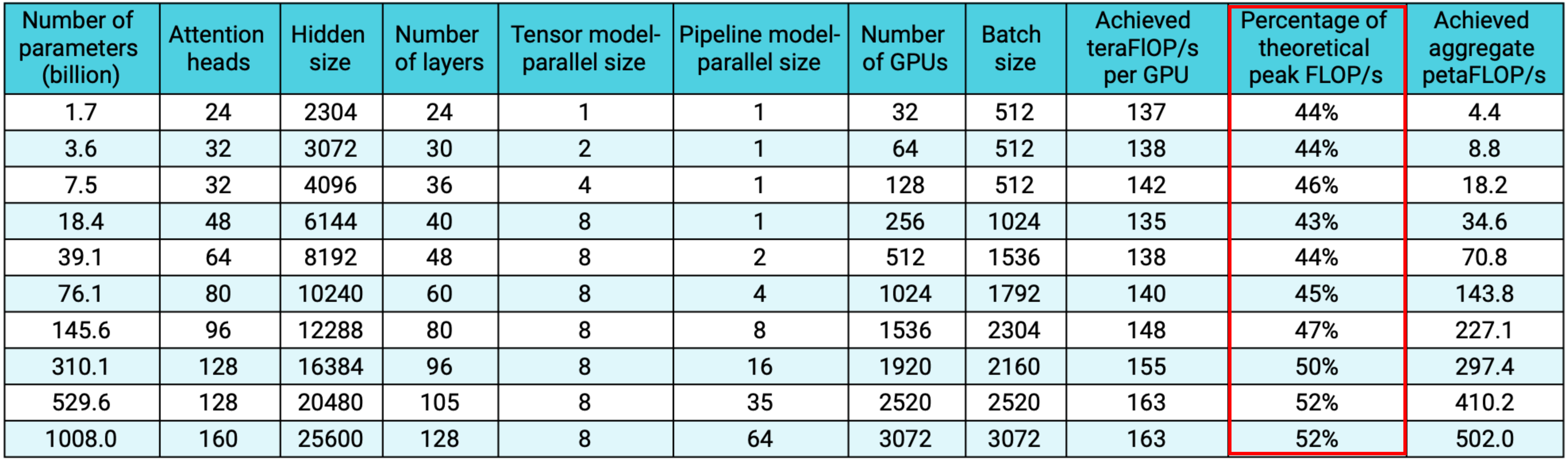
- The **communication overhead** and **GPU idle time** severely hamper the scalability
- Each GPU can only achieve **30%~55%** of its peak FLOPs/s during LLM training
- When GPU achieves 30% of peak FLOP/s, we say the system achieves 30% scalability

30% of its peak FLOPs/s visualization



Existing systems suffer from severe scalability issue

[Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021]



Number of parameters (billion)	Attention heads	Hidden size	Number of layers	Tensor model-parallel size	Pipeline model-parallel size	Number of GPUs	Batch size	Achieved teraFlOP/s per GPU	Percentage of theoretical peak FLOP/s	Achieved aggregate petaFLOP/s
1.7	24	2304	24	1	1	32	512	137	44%	4.4
3.6	32	3072	30	2	1	64	512	138	44%	8.8
7.5	32	4096	36	4	1	128	512	142	46%	18.2
18.4	48	6144	40	8	1	256	1024	135	43%	34.6
39.1	64	8192	48	8	2	512	1536	138	44%	70.8
76.1	80	10240	60	8	4	1024	1792	140	45%	143.8
145.6	96	12288	80	8	8	1536	2304	148	47%	227.1
310.1	128	16384	96	8	16	1920	2160	155	50%	297.4
529.6	128	20480	105	8	35	2520	2520	163	52%	410.2
1008.0	160	25600	128	8	64	3072	3072	163	52%	502.0

Nvidia Megatron: the most popular distributed training framework

Existing systems suffer from severe scalability issue

- Open AI and Meta does not disclose its training scalability, but we can infer them

- End-to-end training time can be estimated by

$$\text{Training time} = \frac{\text{Total FLOPS to train the model}}{\text{FLOPs/s of the GPU cluster}}$$

$$\approx \frac{8\,T\,P}{N\,F\,U}$$

T: the number of tokens

P: the number of parameters

N: the number of GPUs

F: Peak FLOP/s per GPU

U: Utilization

[Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021]

Existing systems suffer from severe scalability issue

- **Open AI GPT3-175B**: 1024 A100 GPUs, 300B tokens, training time is 35 days

$$\mathrm{U} \approx \frac{8 \times (300 \times 10^9) \times (175 \times 10^9)}{1024 \times (312 \times 10^{12}) \times (35 \times 24 \times 3600)} = \mathbf{0.44}$$

The scalability is around 0.44

- **Meta LLaMA-65B**: 2048 A100 GPUs, 1.4TB tokens, training time is 21 days

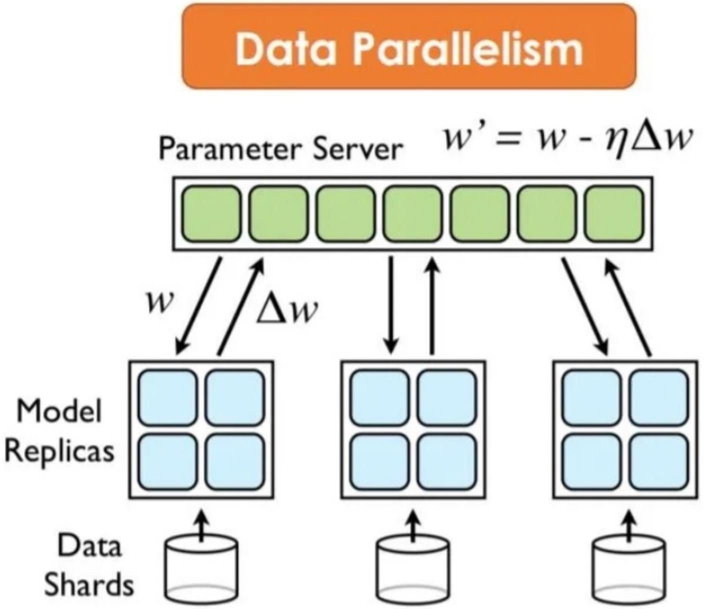
$$\mathrm{U} \approx \mathbf{0.3}$$

The scalability is around 0.3

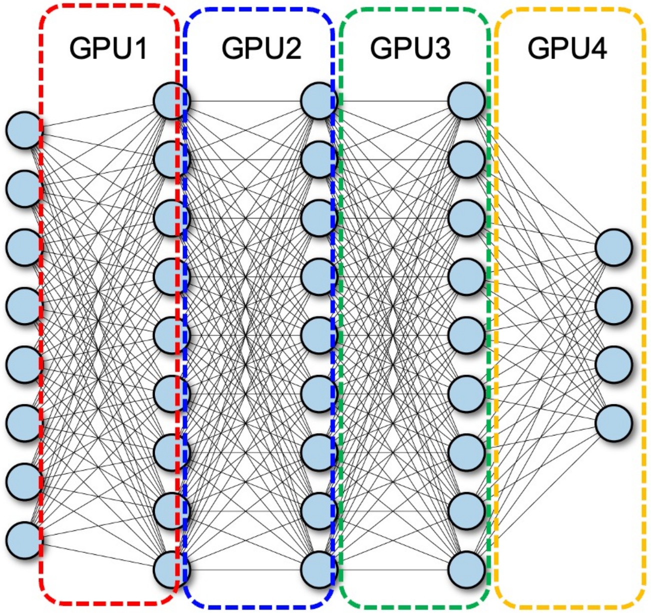
- We find even Nvidia, Open AI, and Meta **cannot** achieve strong scalability

Communication overhead is the top factor hampering the scalability

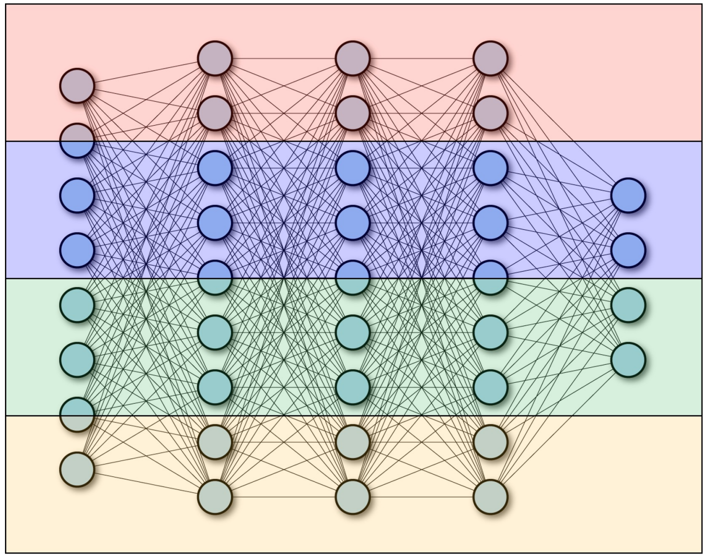
- 3D parallelism indicates 3 orthogonal parallel techniques used to train LLM



Data parallelism



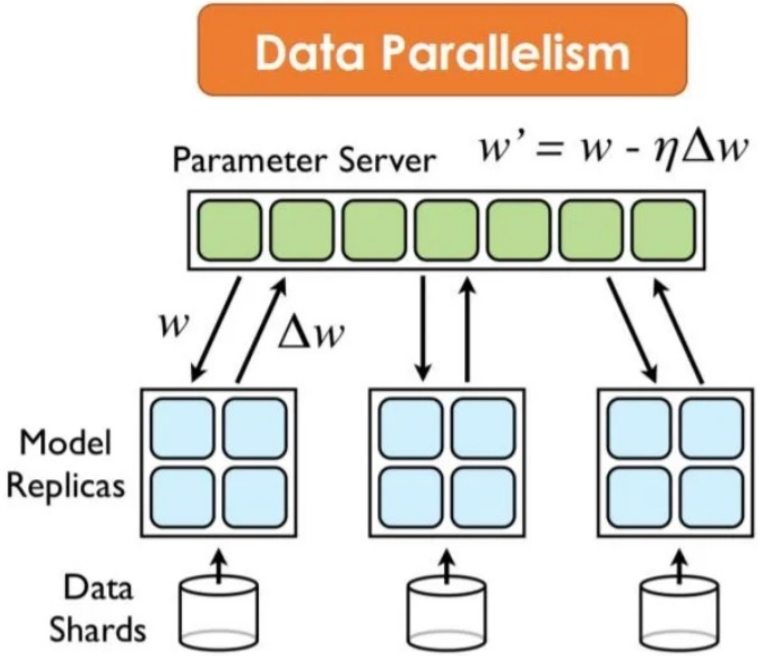
Pipeline parallelism



Tensor parallelism

Communication overhead is the top factor hampering the scalability

- Each parallel technique will introduce significant communication overhead
- Data parallelism introduce **gradient communication**



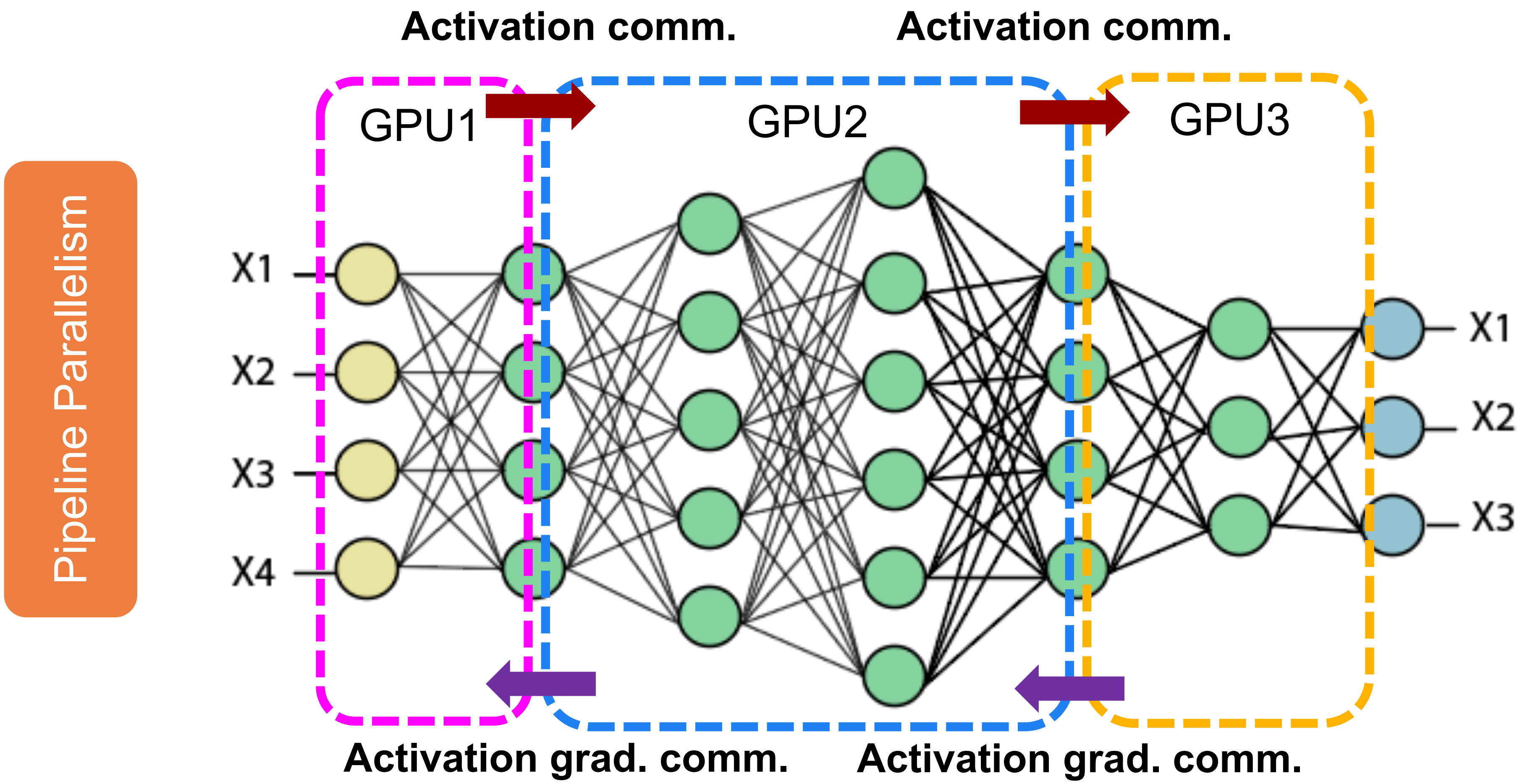
$$g_i^{(k)} = \nabla F(x^{(k)}; \xi_i^{(k)}) \qquad \text{(Local compt.)}$$

$$x^{(k+1)} = x^{(k)} - \frac{\gamma}{n}\sum_{i=1}^{n} g_i^{(k)} \qquad \text{(Global comm.)}$$

- Gradient is of the same dimension as the model (very big)

Communication overhead is the top factor hampering the scalability

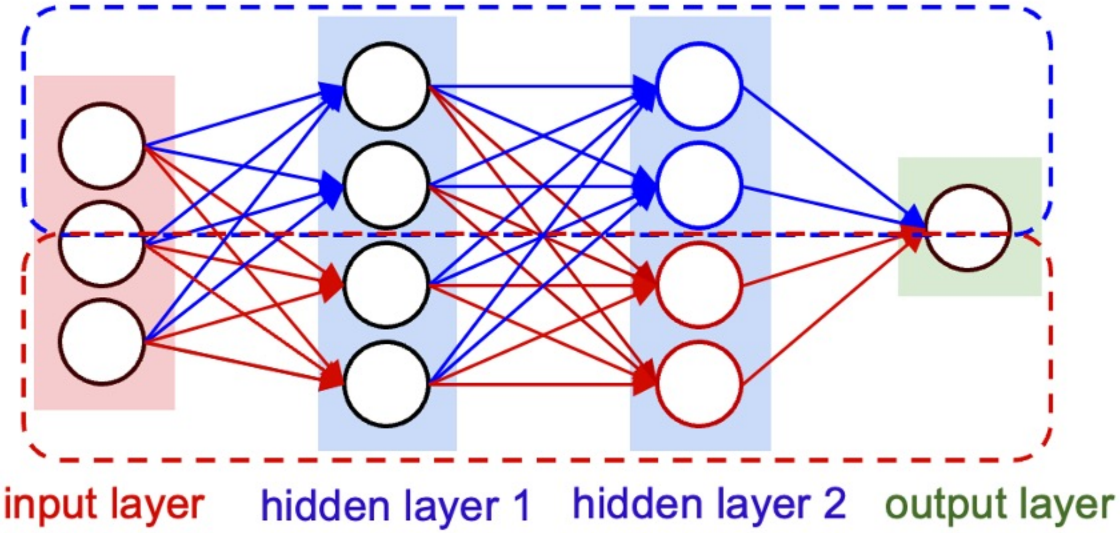
- Pipeline parallelism introduces massive **activation** communication as well as bubble time



Communication overhead is the top factor hampering the scalability

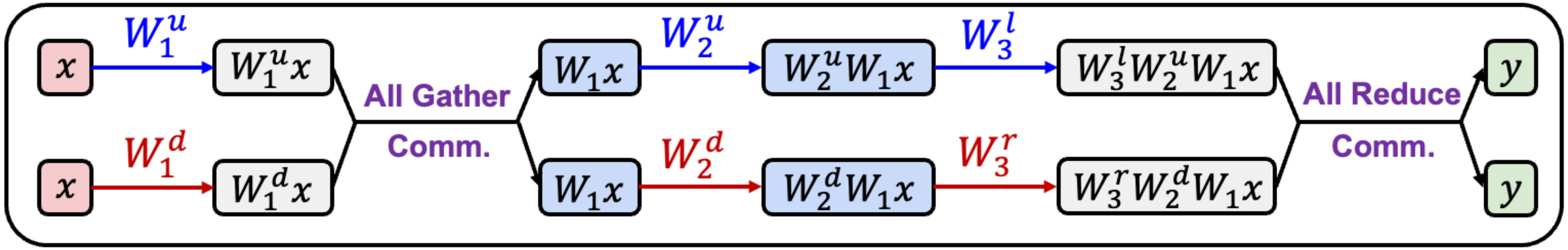
- Tensor parallelism introduces **split** or **merge** communication in almost every activation layer

Tensor Parallelism



$$y = W_3 W_2 W_1 x$$

Computational graph



Communication overhead is the top factor hampering the scalability

- Various useful system techniques are used to reduce comm. overhead and GPU idle time
  - Comm.-Compt. overlap [MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs, 2024]
  - 1F1B pipeline parallelism [Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021]
  - Automated 3D parallelism [Galvatron: Efficient Transformer Training over Multiple GPUs Using Automatic Parallelism, 2022]
- These techniques can improve the scalability to **60%** [MegaScale]; further improvement is very difficult
- **Communication-efficient algorithms will provide new powers to improve scalability**

Collaborative learning with edge devices

- Besides training large models in data-centers, distributed machine learning also applies to collaborative learning with edge devices, e.g., **federated learning**

- Each device collects local dataset; small, valuable and sensitive; cannot be shared (data silos)

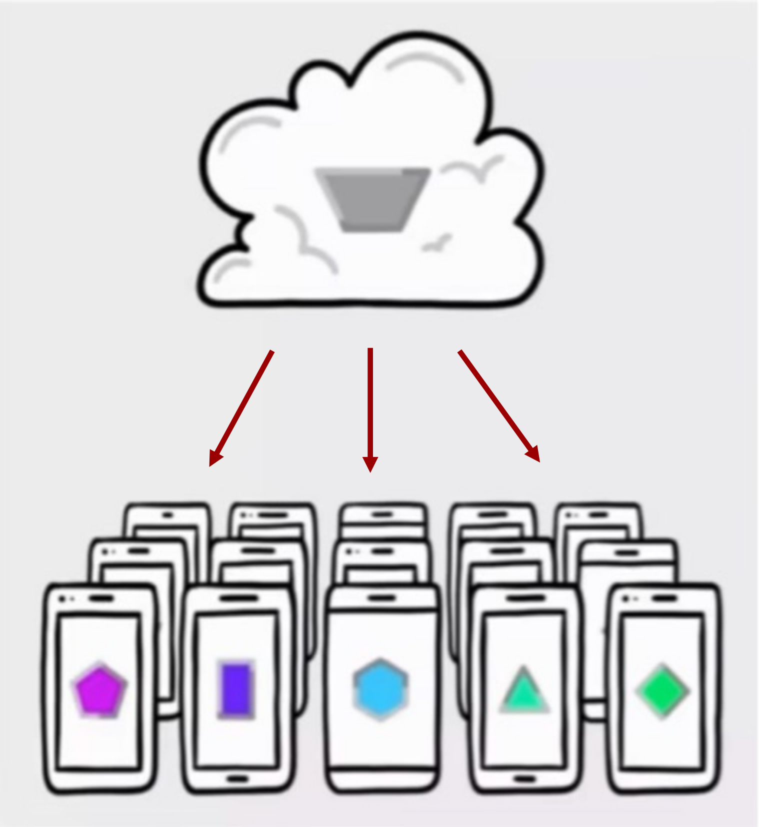
- Collaborate to train a global model without sharing data (break the data silos)

Data silos in edge computing

[https://m.thepaper.cn/newsDetail_forward_25528954]

Edge computing brings more challenges to distributed learning

- Server cannot control user’s device and data. Users join and leave the learning process at their will
- Communication is way more expensive than computation (cannot overlap comm. and compt.)
- Clients are not stable; not available when out of power
- Data distributions are highly non-iid
- Trade-off between fairness and incentive



Require new algorithms to resolve these challenges

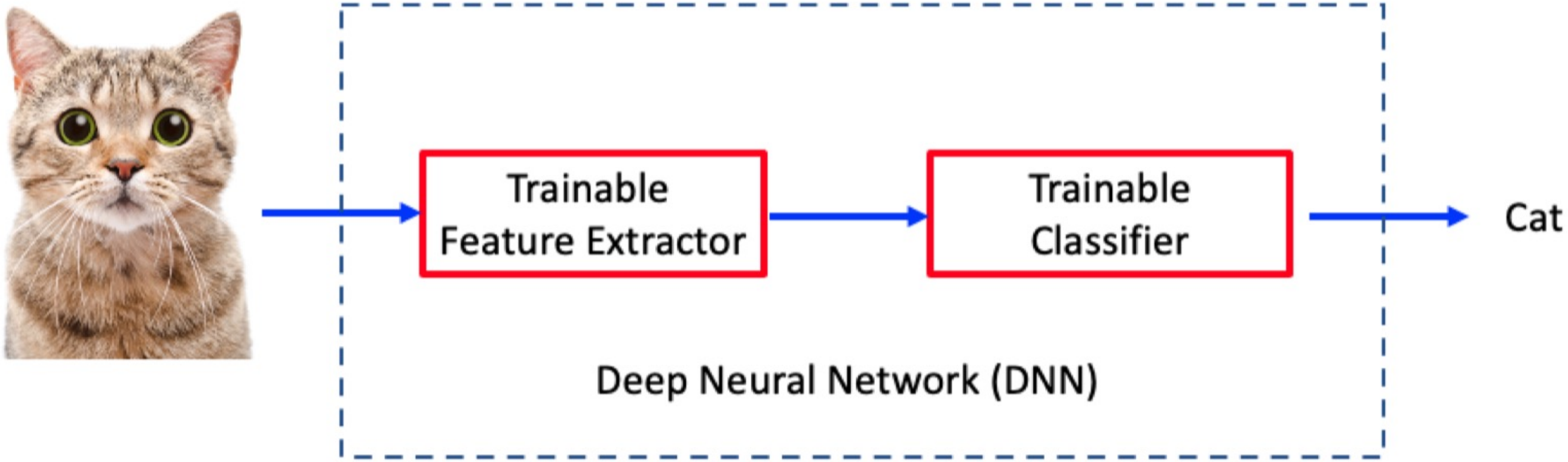
Part I

Distributed Machine Learning and Parallel SGD

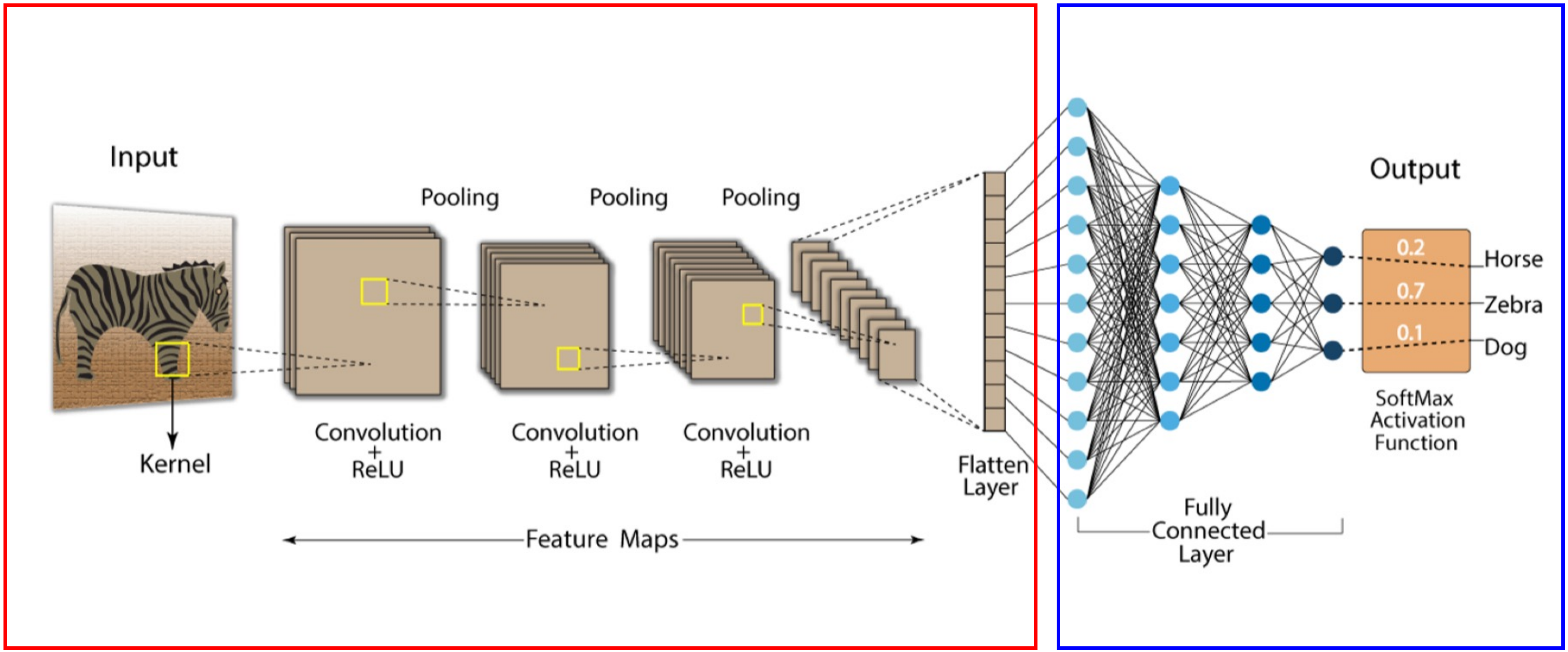
Deep neural network (DNN)

- A deep neural network (DNN) typically includes a **feature extractor** and a **classifier**

- Well-trained DNN can make precise predictions



Example: convolutional neural network

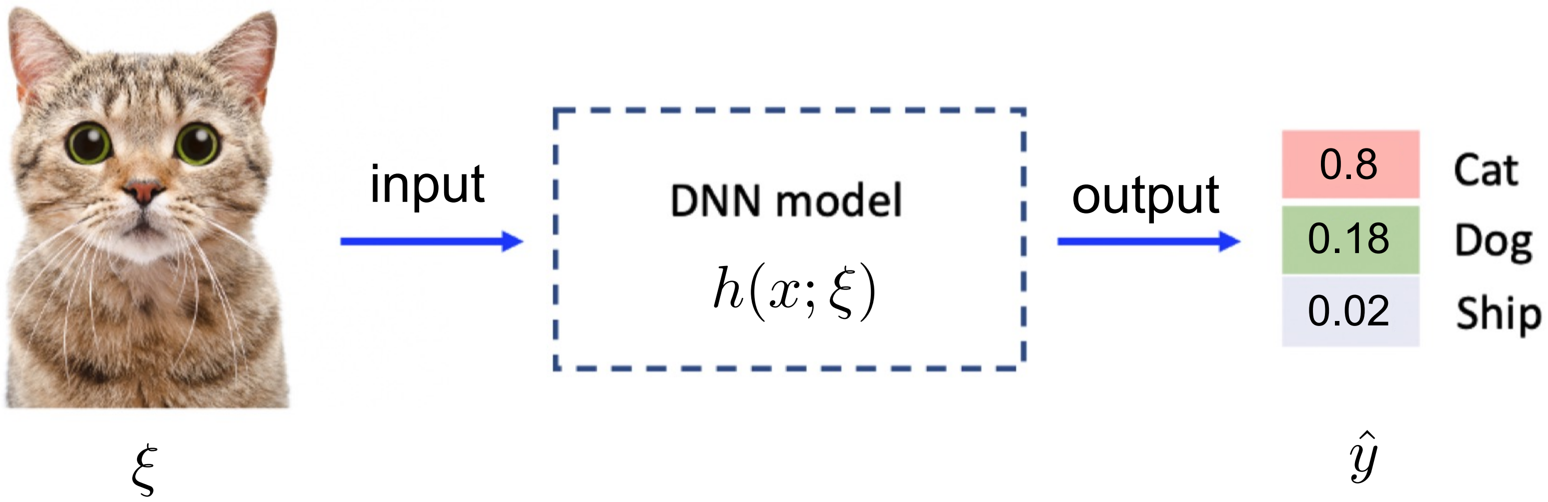


Feature extractor

Classifier

DNN model

- We model DNN as $h(x;\xi):\mathbb{R}^d \to \mathbb{R}^c$ that maps input data ξ to a probability $\hat{y}$
  - $x \in \mathbb{R}^d$ is the DNN model parameter to be trained
  - ξ is a random input data sample
  - c is the number of classes



input

DNN model

$h(x;\xi)$

output

0.8	Cat
0.18	Dog
0.02	Ship

ξ

$\hat{y}$

Training DNN can be formulated into an optimization problem

- Define $L(\hat{y}, y) = -\sum_{j=1}^{d} y_{[j]} \log(\hat{y}_{[j]})$ as the loss function to measure the difference between predictions and the ground-truth label, where $y_{[j]}$ is the j-th element in y

- The model parameter $x^\star$ can be achieved by solving the following optimization problem

$$x^\star = \arg\min_{x \in \mathbb{R}^d} \left\{ \mathbb{E}_{(\xi,y)\sim\underbrace{\mathcal{D}}_{\text{data distribution}}} \left[L\Big(\underbrace{h(x;\xi)}_{\text{prediction}}, \underbrace{y}_{\text{real label}}\Big) \right] \right\}$$

Training DNN can be formulated into an optimization problem

- The model parameter $x^\star$ can be achieved by solving the following optimization problem

$$x^\star = \arg\min_{x\in\mathbb{R}^d} \left\{ \mathbb{E}_{(\xi,y)\sim \underbrace{\mathcal{D}}_{\text{data distribution}}} \left[L\Big(\underbrace{h(x;\xi)}_{\text{prediction}}, \underbrace{y}_{\text{real label}}\Big) \right] \right\}$$

- If we define $\boldsymbol{\xi} = (\xi, y)$ and $F(x;\boldsymbol{\xi}) = L(h(x;\xi), y)$, the above problem becomes

$$x^\star = \arg\min_{x\in\mathbb{R}^d} \{\mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{D}}[F(x;\boldsymbol{\xi})]\}$$

- Most optimization researchers use the second formulation as the staring point to develop algorithms

DNN model is notoriously difficult to train

- Highly-nonconvex cost functions; cannot find global minima; trapped into local minimum

- The model size is large, i.e., $x \in \mathbb{R}^d$ is of extremely high dimensions

- The size of the dataset is huge

DNN training = Non-convex training + Huge dimensions + Huge dataset

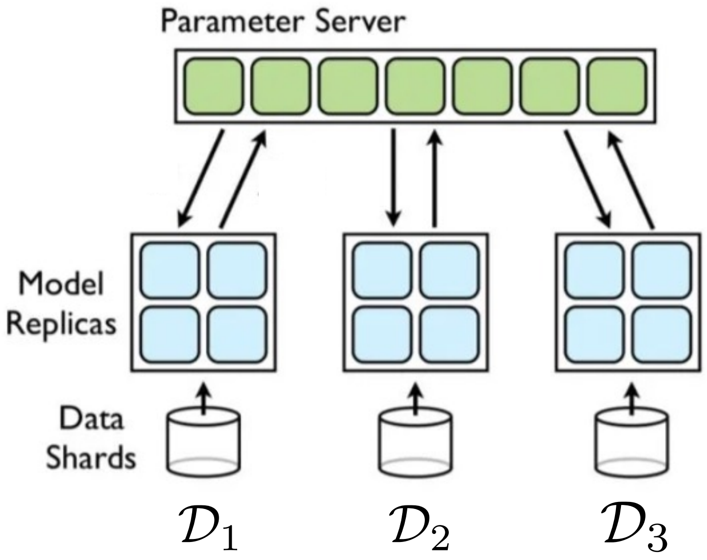
- **Efficient** and **scalable** distributed learning approaches are in urgent need

Distributed machine learning

A network of n nodes (devices such as GPUs) collaborate to solve the problem:

$$\min_{x\in\mathbb{R}^d} \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \quad \text{where} \quad \boxed{f_i(x)} = \boxed{\mathbb{E}_{\xi_i\sim D_i} F(x;\xi_i)}$$

- Each component $f_i : \mathbb{R}^d \to \mathbb{R}$ is local and private to node i
- Random variable ξ_i denotes local data that follows distribution D_i
- Each local distribution D_i is different; data heterogeneity exists
- This talk will sorely focus on the **data parallelism** problem



Vanilla parallel stochastic gradient descent (PSGD)

$$\min_{x\in\mathbb{R}^d} \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \quad \text{where} \quad f_i(x) = \mathbb{E}_{\xi_i\sim D_i} F(x;\xi_i).$$

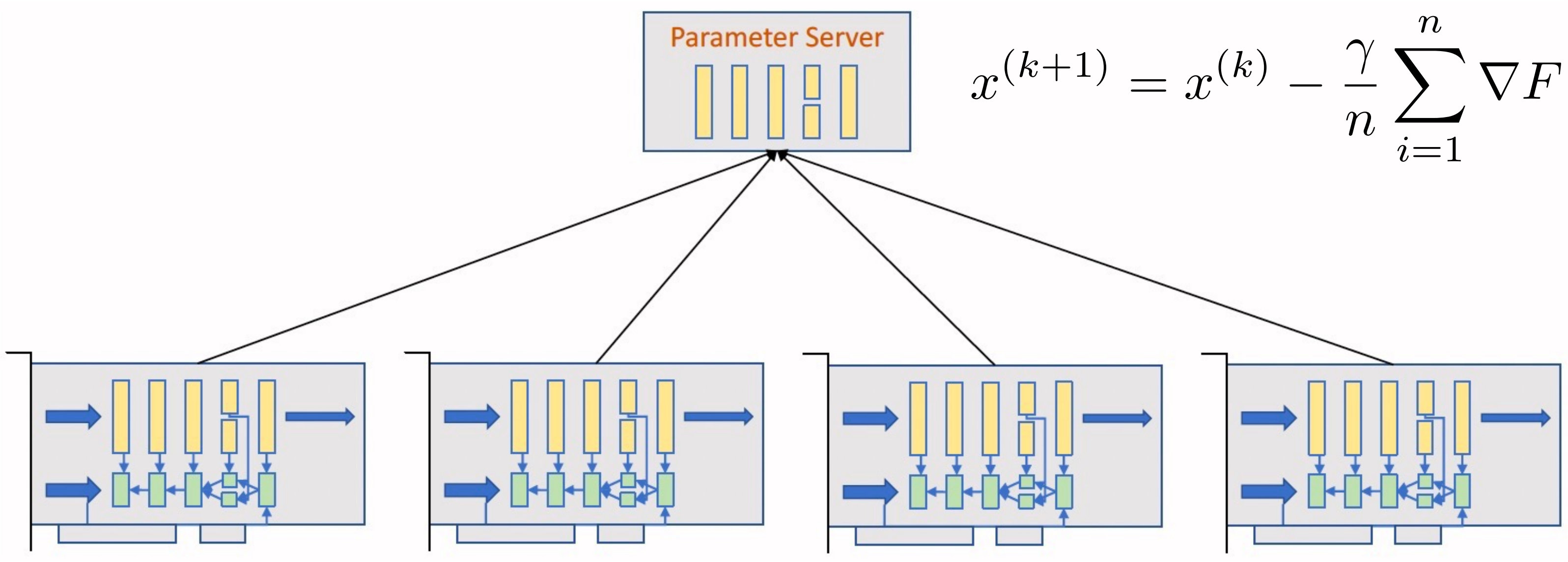
PSGD

$$\boxed{\begin{aligned} g_i^{(k)} &= \nabla F(x^{(k)};\xi_i^{(k)}) && \text{(Local compt.)} \\ x^{(k+1)} &= x^{(k)} - \frac{\gamma}{n}\sum_{i=1}^{n} g_i^{(k)} && \text{(Global comm.)} \end{aligned}}$$

- Each node i samples data $\xi_i^{(k)}$ and computes gradient $\nabla F(x^{(k)};\xi_i^{(k)})$
- All nodes synchronize (i.e. globally average) to update model x per iteration

Vanilla parallel stochastic gradient descent (PSGD)

- Federated learning typically implements PSGD using parameter server



$$x^{(k+1)} = x^{(k)} - \frac{\gamma}{n}\sum_{i=1}^{n}\nabla F(x^{(k)};\xi_i^{(k)})$$

$\nabla F(x^{(k)};\xi_1^{(k)})$ $\nabla F(x^{(k)};\xi_2^{(k)})$ $\nabla F(x^{(k)};\xi_3^{(k)})$ $\nabla F(x^{(k)};\xi_4^{(k)})$

- LLM training within data-centers implements PSGD using Ring-Allreduce

PSGD convergence rate

Assumption [PSGD assumption]

(1) Each $f_i(x)$ is L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x-y\|$ for any x, y;

(2) Each stochastic gradient is unbiased and has bounded variance, i.e.,

$$\mathbb{E}[\nabla F(x;\xi_i)] = \nabla f_i(x), \quad \mathbb{E}\|\nabla F(x;\xi_i) - \nabla f_i(x)\|^2 \le \sigma^2$$

Theorem [PSGD convergence]

Under the above assumptions and with proper γ, PSGD converges as follows

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(x^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}}\right)$$

[K. Yuan, Lecture 6: stochastic gradient descent, PKU Class “Optimization for deep learning”, check Kun Yuan’s website]

Linear speedup in PSGD

Theorem [PSGD convergence]

Under the above assumptions and with proper γ, PSGD converges as follows

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(x^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}}\right)$$

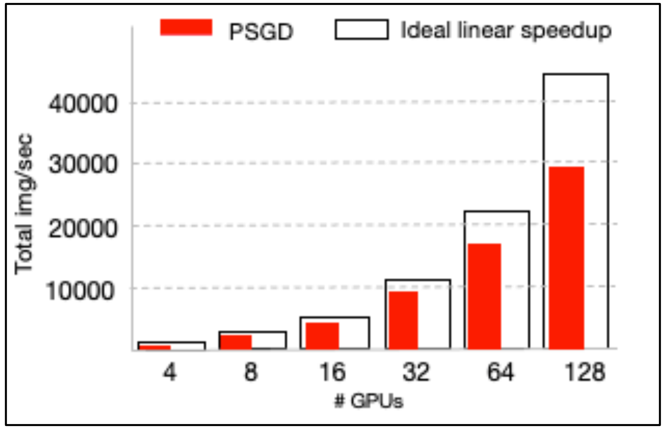
- This implies that to achieve an ϵ-accurate solution, PSGD needs

$$\frac{\sigma}{\sqrt{nT}} \le \epsilon \quad \Longrightarrow \quad T \ge \frac{\sigma^2}{n\epsilon^2}$$

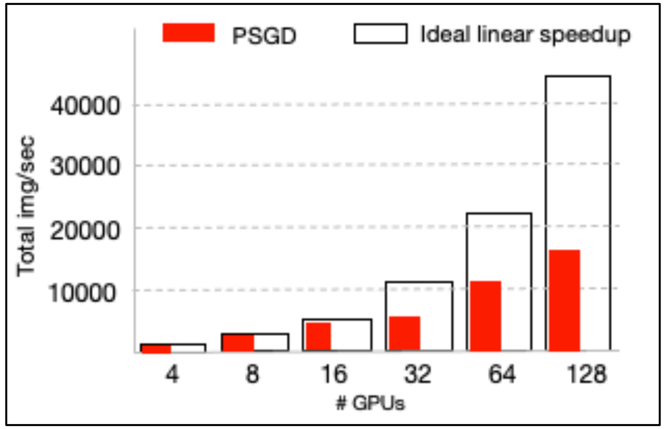
iterations, which decreases linearly with number of nodes n; this is called **linear speedup**

PSGD cannot achieve ideal scalability due to comm. overhead

- PSGD **cannot** achieve ideal linear speedup in throughput due to comm. overhead
- Larger comm-to-compt ratio leads to worse performance in PSGD



Small comm.-to-compt. ratio



Large comm.-to-compt. ratio

Ring-Allreduce is used in each experiment

- How can we reduce the communication overhead in PSGD?

B. Ying, K. Yuan, H. Hu, Y. Chen and W. Yin, “BlueFog: Make decentralized algorithms practical for optimization and deep learning”, arXiv: 2111. 04287, 2021

Methodologies to save communication

- Global average incurs $O(n)$ comm. overhead; proportional to network size n

  [Decentralized communication]

- Each node interacts with the server at every iteration; proportional to iteration numbers

  [Lazy communication]

- Each node sends a full model (or gradient) to the server; proportional to dimension d

  [Compressed communication]

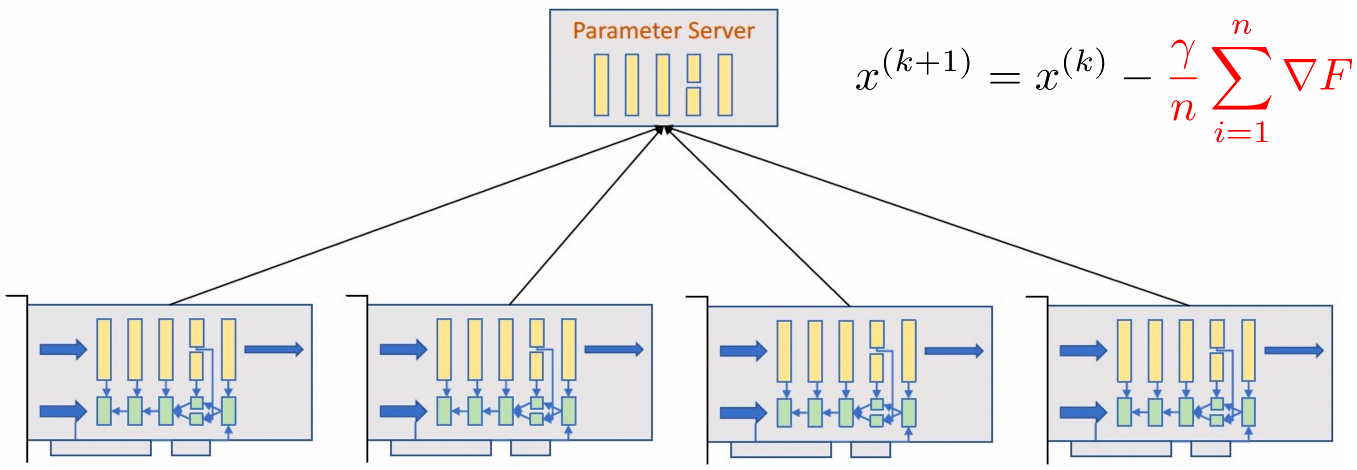
- and more (asynchronous communication; robust communication against Byzantine nodes, etc.)

PART 03-1

Decentralized SGD: communication efficiency

Parallel SGD review

Parameter-server architecture



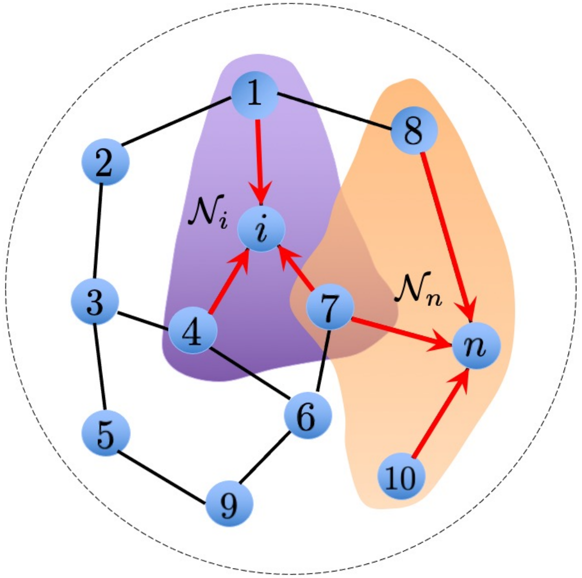
$$x^{(k+1)} = x^{(k)} - \frac{\gamma}{n} \sum_{i=1}^{n} \nabla F(x^{(k)}; \xi_i^{(k)})$$

- Global average using PS incurs $O(n)$ comm. overhead; **proportional to network size n**
- Global average using Ring-Allreduce incurs $O(n)$ latency; **proportional to network size n**
- When network size n is large (say n >= 10000), PSGD suffers severe communication overhead

Decentralized SGD (DSGD)

- To break $O(n)$ comm. overhead, we replace global average with partial average

$$\boxed{\begin{aligned} x_i^{(k+\frac{1}{2})} &= x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad &\text{(Local update)} \\ x_i^{(k+1)} &= \sum_{\boxed{j \in \mathcal{N}_i}} \boxed{w_{ij}} x_j^{(k+\frac{1}{2})} \quad &\text{(Partial averaging)} \end{aligned}}$$



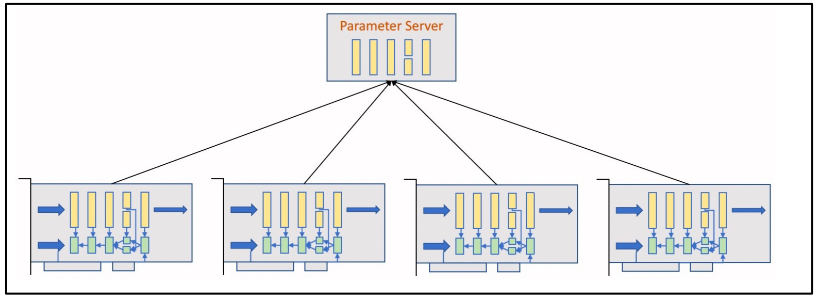
- DSGD = local SGD update + partial averaging [1, 2]
- $\mathcal{N}_i$ is the set of neighbors at node i ; w_{ij} scales information from j to i
- Incurs $O(d_{\max})$ comm. overhead per iteration where $d_{\max} = \max_i \{|\mathcal{N}_i|\}$ is the graph maximum degree

[1] C. G. Lopes and A. H. Sayed, ”Diffusion least-mean-squares over adaptive networks”, ICASSP, 2007

[2] A. Nedich and A. Ozdaglar, ” Distributed subgradient methods for multi-agent optimization”, IEEE TAC, 2009

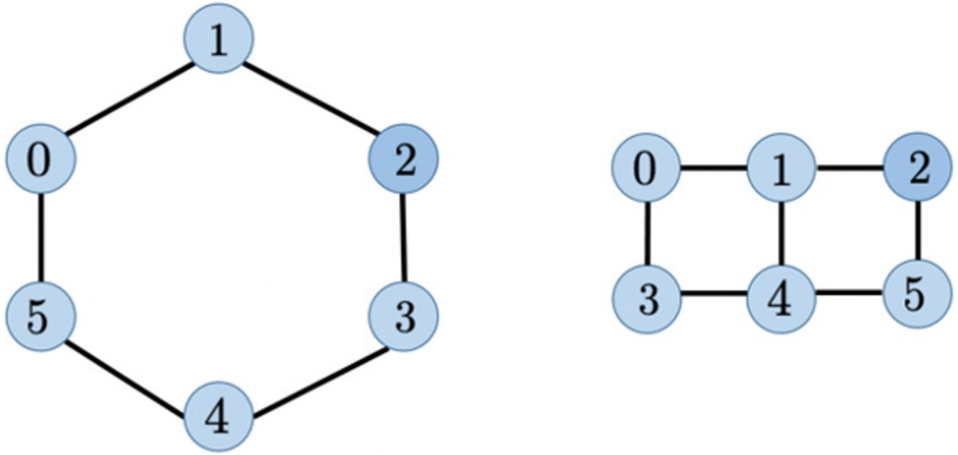
DSGD is more communication-efficient than PSGD

- Incurs $O(1)$ comm. overhead on **sparse** topologies; much less than global average $O(n)$



Global averaging over centralized network

comm. overhead $O(n)$

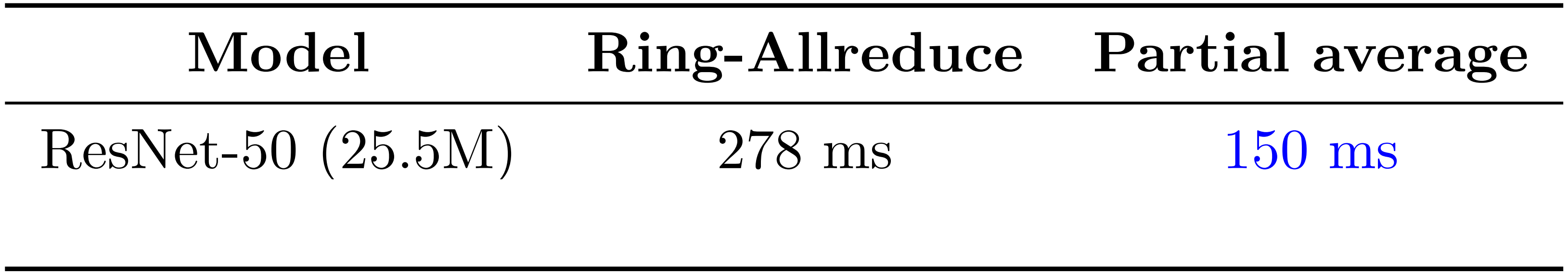


Partial averaging over ring or grid

comm. overhead $O(1)$

DSGD is more communication-efficient than PSGD

- A real experiment on a 256-GPUs cluster [1]



Model	Ring-Allreduce	Partial average
ResNet-50 (25.5M)	278 ms	150 ms

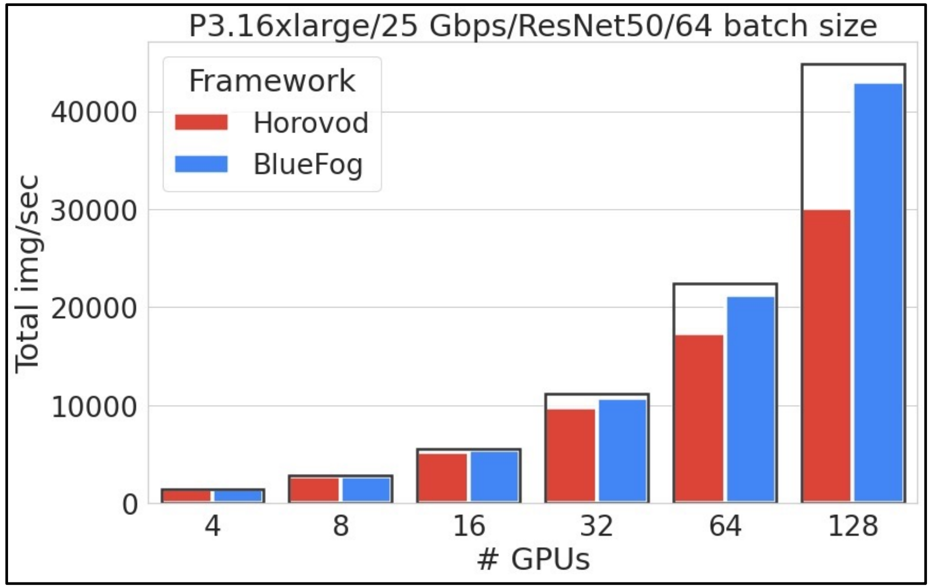
Table. Comparison of per-iter comm. time in terms of runtime with 256 GPUs

- DSGD saves more communications per iteration for larger models

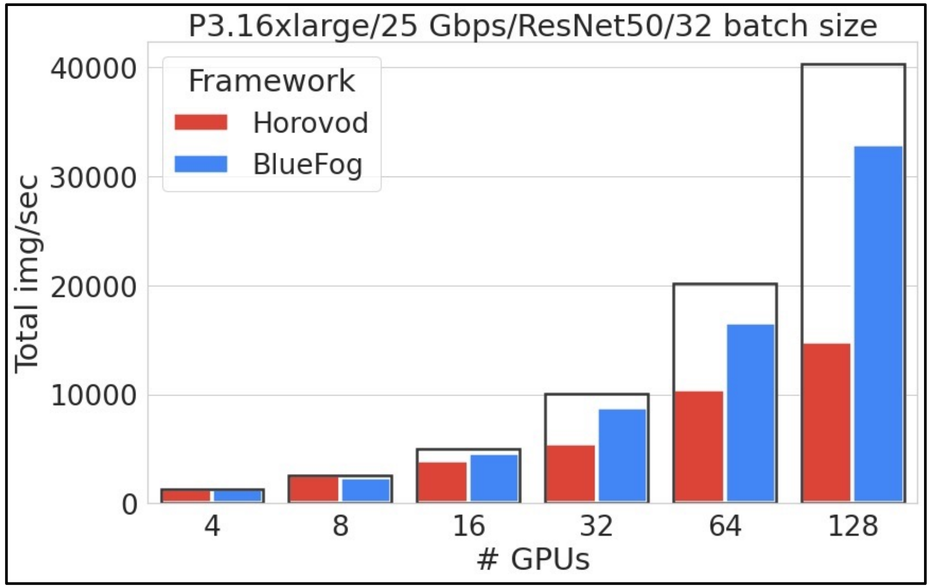
[1] Y. Chen, K. Yuan, Y. Zhang, P. Pan, Y. Xu, and W. Yin, ``Accelerating Gossip SGD with Periodic Global Averaging”, ICML 2021

DSGD is more communication-efficient than PSGD

- DSGD (BlueFog) has **better scalability** than PSGD (Horovod) due to its small comm. overhead



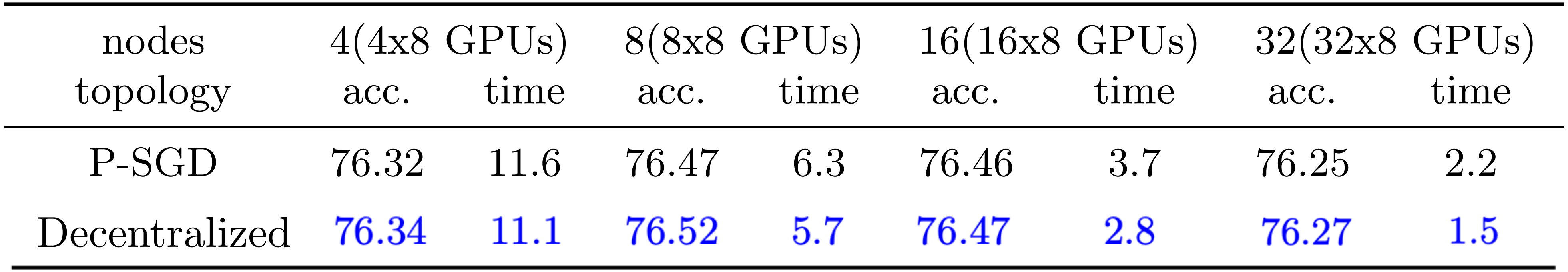
Small comm.-to-compt. ratio



Large comm.-to-compt. ratio

B. Ying, K. Yuan, H. Hu, Y. Chen and W. Yin, "BlueFog: Make decentralized algorithms practical for optimization and deep learning", arXiv: 2111. 04287, 2021

DSGD saves more wall-clock time without severely hurting performance

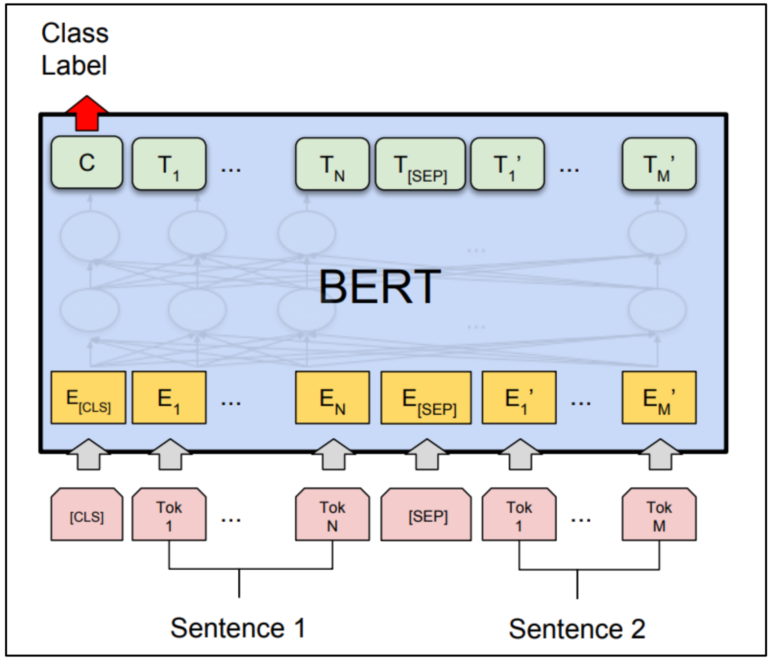


nodes	4(4x8 GPUs)		8(8x8 GPUs)		16(16x8 GPUs)		32(32x8 GPUs)	
topology	acc.	time	acc.	time	acc.	time	acc.	time
P-SGD	76.32	11.6	76.47	6.3	76.46	3.7	76.25	2.2
Decentralized	76.34	11.1	76.52	5.7	76.47	2.8	76.27	1.5

DSGD shows very impressive linear speedup performance and saves more time than PSGD!

B. Ying, K. Yuan, Y. Chen, H. Hu, and W. Yin, “Exponential Graph is Provably Efficient for Decentralized Deep Training”, NeurIPS 2021

DSGD saves more wall-clock time without severely hurting performance



Model: BERT-Large (330M parameters)

Dataset: Wikipedia (2500M words) and

BookCorpus (800M words)

Hardware: 64 GPUs

Table. Comparison in loss and training time [1]

Method	Final Loss	Wall-clock Time (hrs)
P-SGD	1.75	59.02
D-SGD	1.77	30.4

[1] Y. Chen, K. Yuan, Y. Zhang, P. Pan, Y. Xu, and W. Yin, ``Accelerating Gossip SGD with Periodic Global Averaging", ICML 2021

The impressive performance of DSGD requires many efforts

- Vanilla DSGD cannot achieve strong performance against PSGD

- This lecture will highlight the efforts we devoted to making DSGD practical for real applications

PART 03-2

Decentralized SGD: convergence property

Does decentralized SGD converge? Yes!

- Recall the PSGD and DSGD recursions

Parallel SGD

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Local update)}$$

$$x_i^{(k+1)} = \frac{1}{n}\sum_{j=1}^{n} x_j^{(k+\frac{1}{2})} \quad \text{(Global averaging)}$$

Decentralized SGD

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Local update)}$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

- DSGD will converge if the **partial average** asymptotically converge to the **global average**

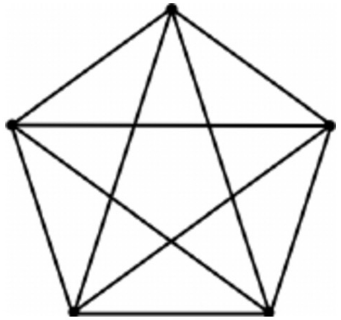
- This argument is true if the weight matrix $W = [w_{ij}]_{i=1,j=1}^{n} \in \mathbb{R}^{n\times n}$ is doubly-stochastic

$$W\mathbf{1}_n = \mathbf{1}_n \quad \text{and} \quad \mathbf{1}_n^T W = \mathbf{1}_n^T$$

Doubly-stochastic weight matrix is easy to construct

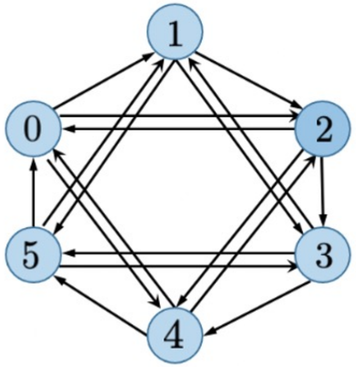
- Doubly-stochastic weight matrix is easy to construct over all undirected graphs and some directed graphs

Fully-connected weight matrix



$$W = \frac{1}{5}\mathbb{1}\mathbb{1}^T = \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}$$

Exponential weight matrix



$$W = \begin{bmatrix} \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

A numerical illustration

- We use examples to test whether partial average will converge to the global average

- Partial average: $x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k)}$ for k = 0, 1, …, T

Partial average over **ring** graph

Iter T	x1	x2	x3	x4	x5
0	0	1	2	3	4
1	1.67	1	2	3	2.33
5	1.95	1.93	2.00	2.07	2.05
10	1.998	1.996	2.000	2.000	2.002
20	1.999	1.999	2.000	2.000	2.000
50	**2.000**	**2.000**	**2.000**	**2.000**	**2.000**

Global average

Partial average over **exponential** graph

Iter T	x1	x2	x3	x4	x5
0	0	1	2	3	4
1	1.75	1.5	2.5	2.25	2.0
5	2.002	2.001	2.00	1.999	1.998
10	1.999	1.999	2.000	2.000	2.002
20	**2.000**	**2.000**	**2.000**	**2.000**	**2.000**

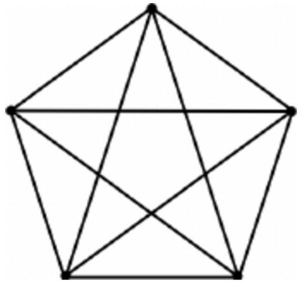
Global average

Network topology determines the consensus rate

- The above numerical example implies that
  - Partial average **asymptotically converges** to global average
  - **Network topology determines how fast** that partial average will converge to global average

- We introduce quantity ρ to gauge the graph connectivity

$$\rho = \|W - \frac{1}{n}\mathbb{1}\mathbb{1}^T\|_2 \in (0,1) \ \text{ where } W = [w_{ij}] \in \mathbb{R}^{n\times n}$$



$$W = \frac{1}{5}\mathbb{1}\mathbb{1}^T = \begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}$$

  - Well-connected topology has $\rho \to 0$, e.g. fully-connected topology
  - Sparsely-connected topology has $\rho \to 1$, e.g. ring has $\rho = O(1 - \frac{1}{n^2})$

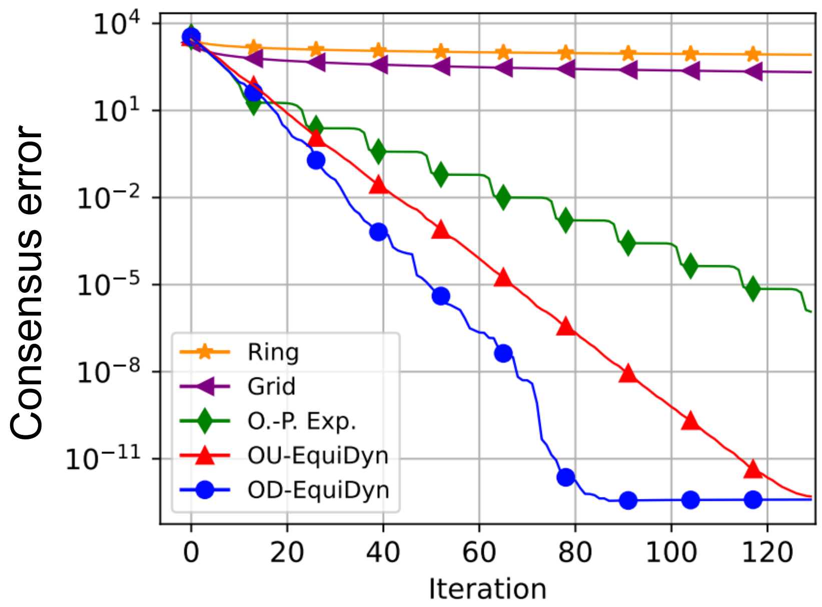
Network topology determines the consensus rate

- With graph connectivity indicator ρ, it can be proved that

$$\|k\text{-th partial average} - \text{global average}\| \leq C\rho^k$$

Based on the above fact, ρ is also referred to as **consensus rate**

Network topology	Consensus rate ρ
Ring	$O(1-\frac{1}{n^2})$
Grid	$O(1-\frac{1}{n\ln(n)})$
Torus	$O(1-\frac{1}{n})$
ExpoGraph	$O(1-\frac{1}{\ln(n)})$
GeoMedian	$O(1-\frac{\ln(n)}{n})$
Erdos-Renyi	$O(1)$
EquiGraph	$O(1)$



Song et. al., "Communication-Efficient Topologies for Decentralized Learning with O(1) Consensus Rate", NeurIPS 2022

DSGD convergence

Recall $\nabla f_i(x) = \mathbb{E}_{\xi_i \sim \mathcal{D}_i}[\nabla F(x;\xi_i)]$

Assumption [DSGD assumption]

(1) Each $f_i(x)$ is L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x-y\|$ for any x, y;

(2) Each stochastic gradient is unbiased and has bounded variance, i.e.,

$$\mathbb{E}[\nabla F(x;\xi_i)] = \nabla f_i(x), \quad \mathbb{E}\|\nabla F(x;\xi_i) - \nabla f_i(x)\|^2 \le \sigma^2$$

Same assumptions as PSGD

(3) Gradient dissimilarity can be upper bounded, i.e.,

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(x) - \nabla f(x)\|^2 \le b^2$$

Data heterogeneity assumption

- Quantity b^2 gauges the magnitude of data heterogeneity; Larger b^2 implies worse heterogeneity
- If $b^2 = 0$, we have $\nabla f_i(x) = \nabla f(x)$ for any i, implying that all distributions $\mathcal{D}_i$ are **homogeneous**

DSGD convergence

Theorem [DSGD convergence]

Under the above assumptions and with proper γ, DSGD converges as follows [1]

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} + \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}\boxed{(1-\rho)^{1/3}}} + \frac{\rho^{2/3}\boxed{b^{2/3}}}{T^{2/3}\boxed{(1-\rho)^{2/3}}}\right)$$

where $\bar{x}^k = \frac{1}{n}\sum_{i=1}^{n} x_i^{(k)}$.

- As iteration $T \to \infty$, DSGD converges to a stationary solution
- **Network topology** influences the convergence; Sparse topology ($\rho \to 1$) results in slower convergence
- **Data heterogeneity** influences the convergence; large hegero. b^2 results in slower convergence

[1] A. Koloskova, et. al., "A unified theory of decentralized sgd with changing topology and local updates", ICML 2020

PSGD v.s. DSGD

PSGD $$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(x^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}}\right)$$

DSGD $$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} \qquad\qquad\qquad\qquad\qquad\qquad\right)$$

- DSGD convergence is tight; utilizing fully-connected topology ($\rho = 0$) reduces to PSGD

PSGD v.s. DSGD

asymptotic rate

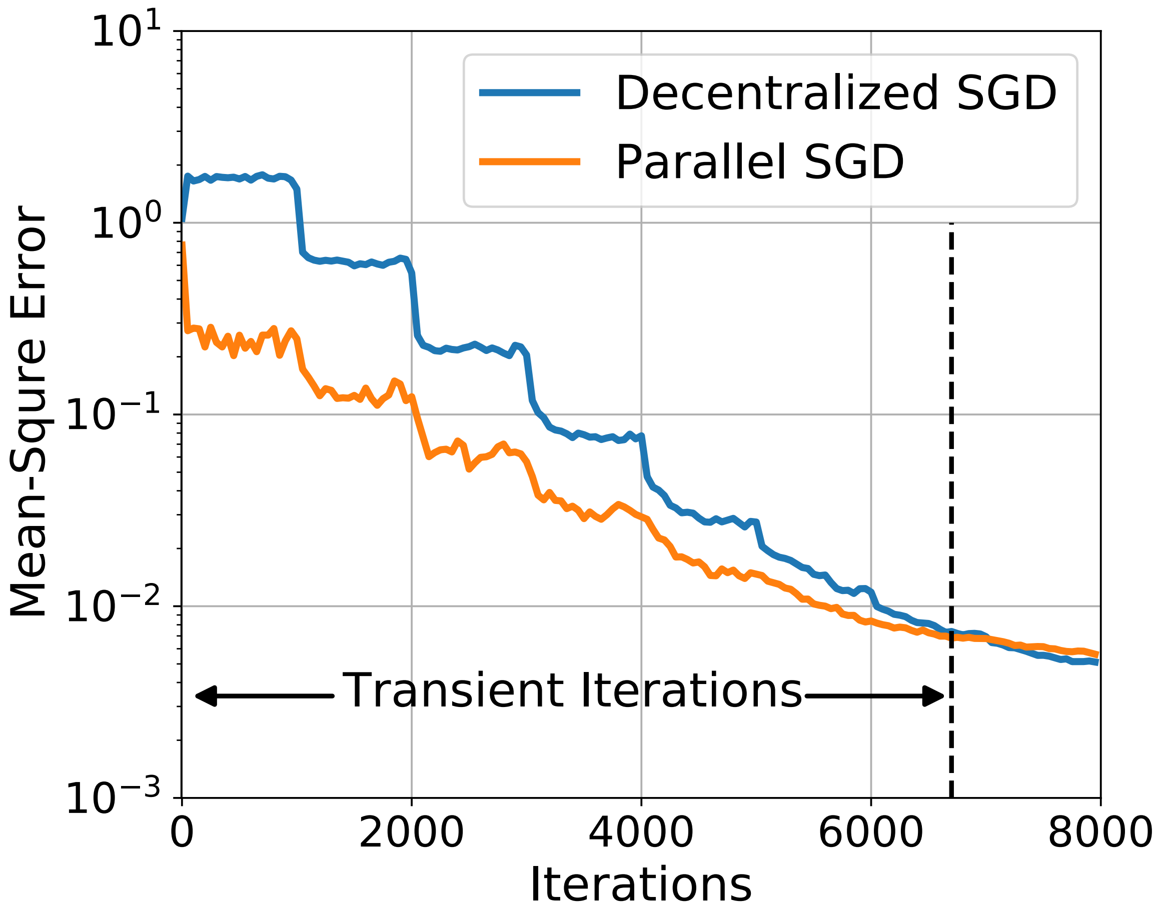
PSGD $$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(x^{(k)})\|^2 = \mathcal{O}\left(\boxed{\frac{\sigma}{\sqrt{nT}}}\right)$$

Extra overhead

DSGD $$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\boxed{\frac{\sigma}{\sqrt{nT}}} + \boxed{\frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}} + \frac{\rho^{2/3}b^{2/3}}{T^{2/3}(1-\rho)^{2/3}}}\right)$$

- DSGD convergence is tight; utilizing fully-connected topology ($\rho = 0$) reduces to PSGD
- D-SGD can asymptotically converge as fast as P-SGD when $T \to \infty$; the first term dominates; reach **linear speedup** asymptotically
- But DSGD **requires more iterations** to reach that stage due to the overhead caused by partial average

PSGD v.s. DSGD



- DSGD converges asymptotically as fast as PSGD when T is sufficiently large
- DSGD has to experience transient iterations to catch up with PSGD
- In real practice, we may not be able to run very large iterations due to resource or time constraints
- We need to accelerate DSGD and reduce its transient iterations

Transient iteration complexity

- Transient iteration complexity gauges the non-asymptotic stage
- **Definition**: number of iterations before a distributed algorithm achieves its linear speedup stage

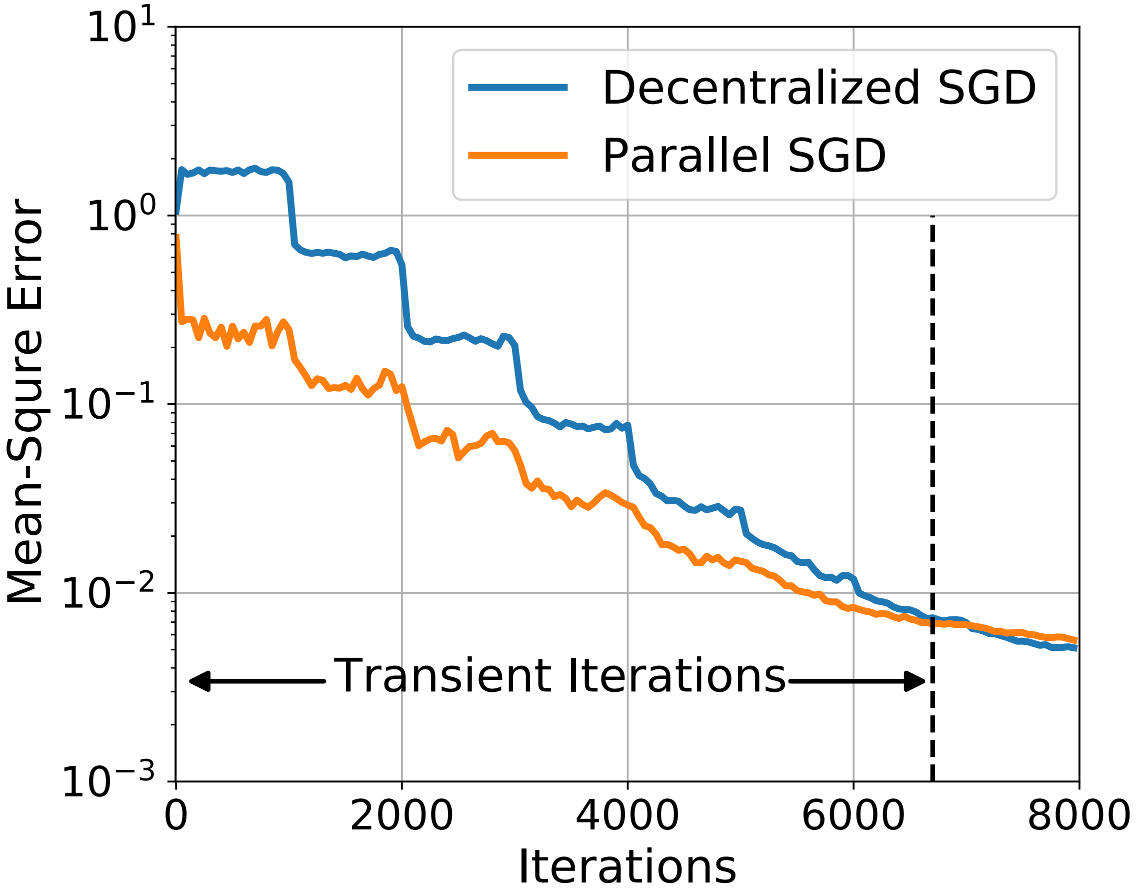
DSGD convergence:

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} + \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}} + \frac{\rho^{2/3}b^{2/3}}{T^{2/3}(1-\rho)^{2/3}}\right)$$

Transient iterations:

$$\left.\begin{aligned} \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}} &\le \frac{\sigma}{\sqrt{nT}} \\[2ex] \frac{\rho^{2/3}b^{2/3}}{T^{2/3}(1-\rho)^{2/3}} &\le \frac{\sigma}{\sqrt{nT}} \end{aligned}\right\} \quad \mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2} + \frac{\rho^4 n^3 b^4}{\sigma^6(1-\rho)^4}\right)$$

Major factors that influence the transient iteration complexity



DSGD tran. iters. $\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2 \boxed{(1-\rho)^2}} + \frac{\rho^4 n^3 \boxed{b^4}}{\sigma^6 \boxed{(1-\rho)^4}}\right)$

- **Large data heterogeneity** b^2 will significantly enlarge the transient iterations
- **Sparsely-connected network** $(\rho \to 1)$ will significantly enlarge the transient iterations
- Data heterogeneity and network topology will amplify the influence of each other

A brief summary

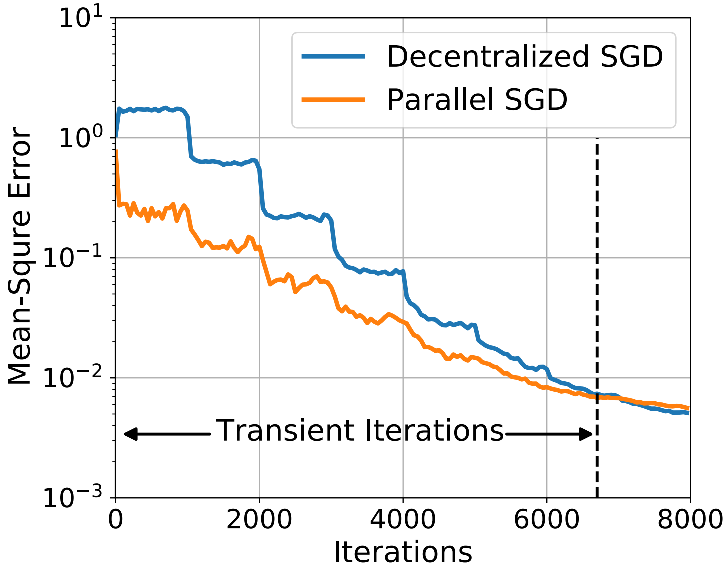
- DSGD utilizes **partial average** in model updates

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \quad \text{(Local update)}$$
$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} \quad \text{(Partial averaging)}$$

- Partial average asymptotically converges to the global average; makes DSGD converge

- Achieve asymptotical linear speedup rate, but needs transient iterations

$$\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2} + \frac{\rho^4 n^3 b^4}{\sigma^6(1-\rho)^4}\right)$$



- In next few slides, we will reduce transient iterations by improving data heterogeneity and network topology

PART 03-3

Decentralized SGD: removing data heterogeneity

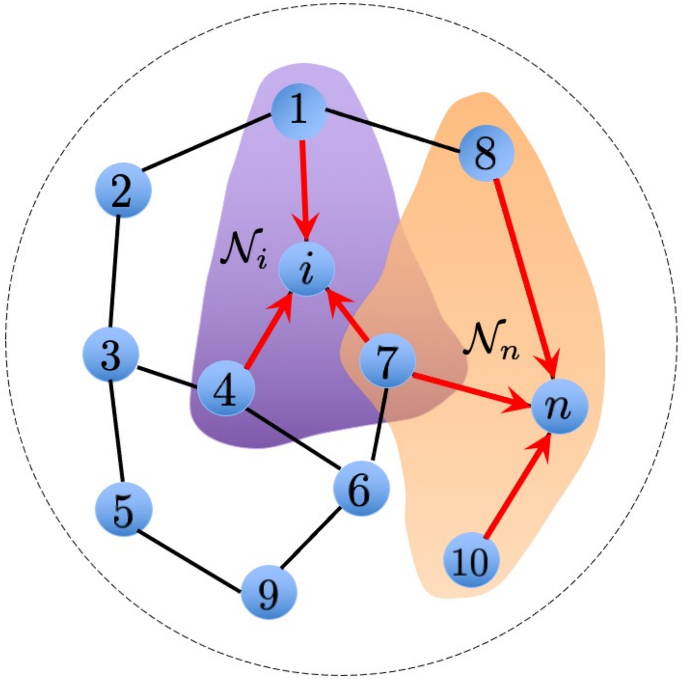
D-SGD suffers from data heterogeneity

- Recall the distributed stochastic optimization problem

$$\min_{x\in\mathbb{R}^d} \quad f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x), \quad \text{where} \quad f_i(x) = \mathbb{E}_{\xi_i\sim D_i} F(x;\xi_i).$$

- The D-SGD algorithm iterates as follows

$$\boxed{\begin{aligned} x_i^{(k+\frac{1}{2})} &= x_i^{(k)} - \gamma\nabla F(x_i^{(k)};\xi_i^{(k)}) && \text{(Local update)}\\ x_i^{(k+1)} &= \sum_{j\in\mathcal{N}_i} w_{ij} x_j^{(k+\frac{1}{2})} && \text{(Partial averaging)}\end{aligned}}$$



- We can write the above algorithm into one line

$$x_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(x_j^{(k)} - \gamma\nabla F(x_j^{(k)};\xi_j^{(k)})\right)$$

D-SGD suffers from data heterogeneity

- Suppose no gradient noise exists and all nodes stay at the stationary solution, i.e.,

$$\nabla F(x_i^{(k)};\xi_i^{(k)}) = \nabla f(x_i^{(k)}) \quad \text{and} \quad x_1^{(k)} = x_2^{(k)} = \cdots = x_n^{(k)} = x^\star$$

- The D-SGD algorithm becomes

$$x_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(x^\star - \gamma\nabla f_j(x^\star)\right) = x^\star - \gamma\sum_{j\in\mathcal{N}_i} w_{ij}\nabla f_j(x^\star)$$

- In homogeneous-data scenario where $f_i(x) = f(x)$ for any node i, we have $\nabla f_i(x^\star) = \nabla f(x^\star) = 0$ and

$$x_i^{(k+1)} = x^\star - \gamma\sum_{j\in\mathcal{N}_i} w_{ij}\nabla f_j(x^\star) = x^\star$$

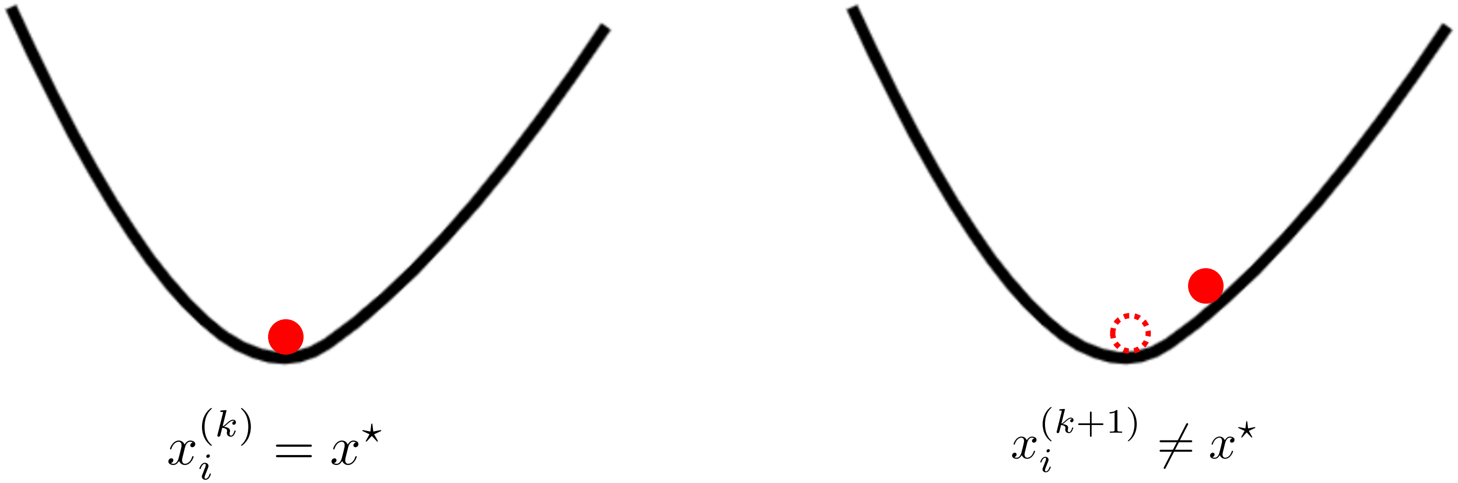
Therefore, the stationary solution is stable in homogeneous-data scenario

D-SGD suffers from data heterogeneity

- In heterogeneous-data scenario where $\nabla f_i(x) \neq \nabla f(x)$ for any node i, we have $\nabla f_i(x^\star) \neq \nabla f(x^\star)$ and

$$x_i^{(k+1)} = x^\star - \gamma \sum_{j\in\mathcal{N}_i} w_{ij}\nabla f_j(x^\star) \neq x^\star - \gamma\nabla f(x^\star) = x^\star$$

Therefore, the stationary solution is NOT stable in heterogeneous-data scenario



- This bias is caused by the algorithmic structure, not the gradient noise

A crude idea to remove data heterogeneity

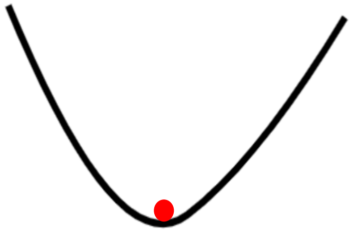
- The fundamental reason for the bias is $\nabla f_i(x) \neq (1/n)\sum_{i=1}^{n}\nabla f_i(x)$ due to data heterogeneity

- To remove the bias caused by data heterogeneity, an impractical but intuitive algorithm is

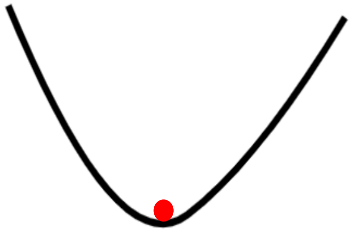
$$x_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(x_j^{(k)} - \gamma\nabla f_j(x_j^{(k)})\right) \longrightarrow x_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(x_j^{(k)} - \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x_j^{(k)})\right)$$

- Assume all nodes initialize at $x_i^{(k)} = x^\star$, we find $x_i^{(k+1)}$ still remains at $x^\star$

$$x_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(x^\star - \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x^\star)\right)$$
$$= x^\star - \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x^\star) = x^\star$$



$x_i^{(k)} = x^\star$



$x_i^{(k+1)} = x^\star$

Track the global gradient

- How to get the dynamic global average $\frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x_j^{(k)})$ in a decentralized manner ?

- **Decentralized tracking** (also known as dynamic average consensus [1])

$$y_i^{(k+1)} = \sum_{j\in\mathcal{N}_i} w_{ij}\left(y_j^{(k)} + \nabla f_j(x_j^{(k+1)}) - \nabla f_j(x_j^{(k)})\right) \qquad \text{where} \qquad y_i^{(0)} = \nabla f_i(x_i^{(0)})$$

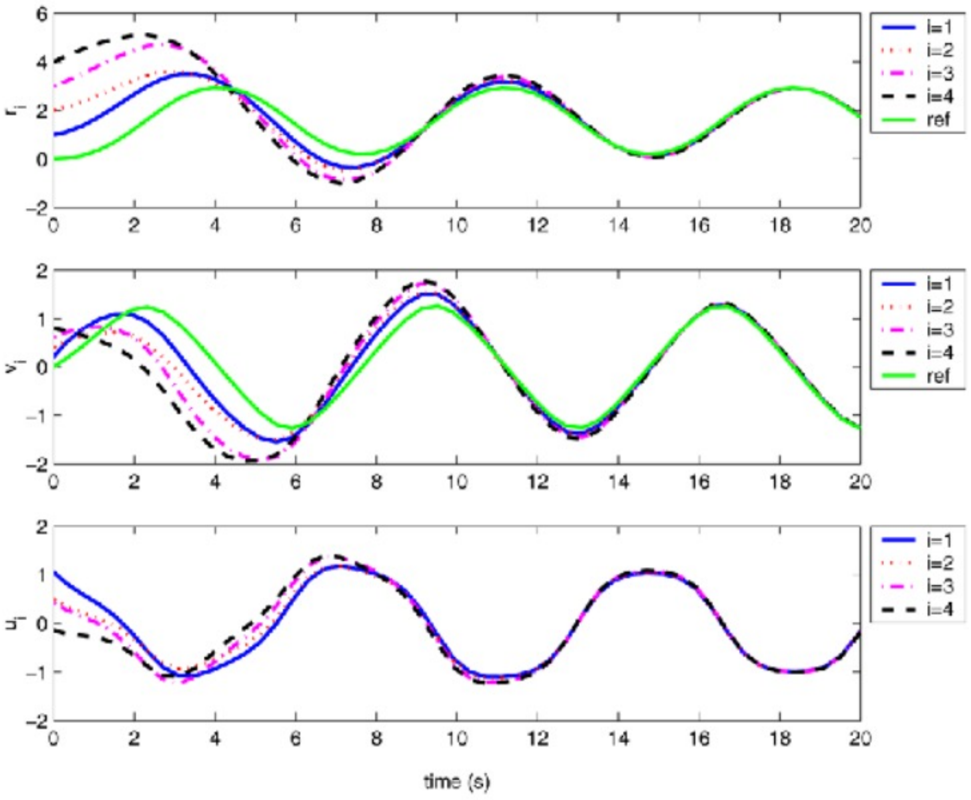
- When $\nabla f_i(x_i^{(k)})$ converges to a fixed point (such as $\nabla f_i(x^\star)$) or oscillates slowly, we have [1]

$$y_i^{(k)} \to \frac{1}{n}\sum_{j=1}^{n}\nabla f_j(x_j^{(k)}), \qquad \forall i \in [n]$$

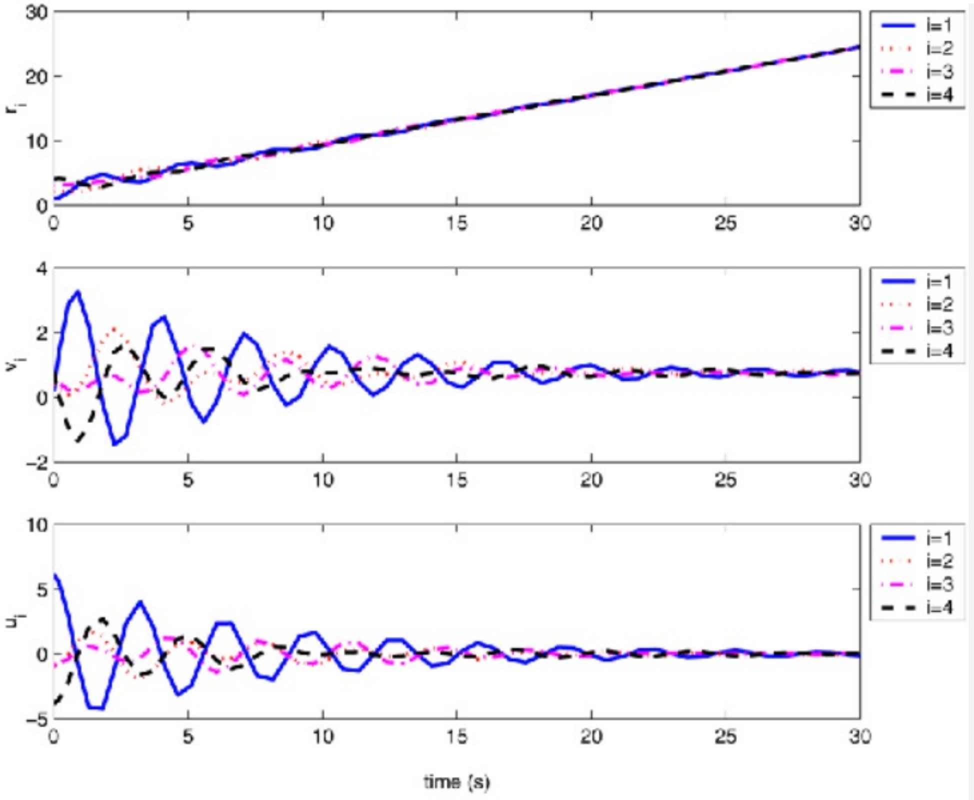
[1] M. Zhu and S. Martinez, “Discrete-time dynamic average consensus”, Automatica, 2010

Track the global gradient: illustration

Dynamic gradient has small oscillation



Dynamic gradient converges to a stationary point



Green: globally averaged gradient

Simulation results are from [Ren IEEE TAC 2007]

Gradient tracking algorithm

- By dynamically track the globally-averaged gradient, we can achieve a new decentralized algorithm [1-3]

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} \left(x_j^{(k)} - \gamma y_j^{(k)} \right) \qquad \text{(DSGD)}$$

$$y_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} \left(y_j^{(k)} + \nabla F(x_j^{(k+1)}; \xi_j^{(k+1)}) - \nabla F(x_j^{(k+1)}; \xi_j^{(k)}) \right) \qquad \text{(Track gradient)}$$

where $y_i^{(0)} = \nabla F(x_i^{(0)}; \xi_i^{(0)})$ in the initialization

- Since $y_i^{(k)}$ is supposed to converge to $\frac{1}{n}\sum_{i=1}^{n} \nabla F(x_i^{(k)}; \xi_i^{(k)})$, this algorithm is named as **Gradient Tracking**

[1] J. Xu et. al., "Augmented distributed gradient methods for multi-agent optimization under uncoordinated constant stepsizes", IEEE CDC 2015

[2] P Di Lorenzo and G Scutari, "Next: In-network nonconvex optimization", IEEE TSIPN, 2016

[3] A Nedic, A Olshevsky, W Shi, "Achieving geometric convergence for distributed optimization over time-varying graphs", SIOPT, 2017

Gradient tracking: convergence

Assumption [GT assumption]

(1) Each $f_i(x)$ is L-smooth, i.e., $\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x - y\|$ for any x, y;

(2) Each stochastic gradient is unbiased and has bounded variance, i.e.,

$$\mathbb{E}[\nabla F(x;\xi_i)] = \nabla f_i(x), \quad \mathbb{E}\|\nabla F(x;\xi_i) - \nabla f_i(x)\|^2 \le \sigma^2$$

Theorem [GT convergence]

Under the above assumptions and with proper γ, Gradient tracking converges as

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} + \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}}\right)$$

[1] S. Alghunaim and K. Yuan, "A Unified and Refined Convergence Analysis for Non-Convex Decentralized Learning", IEEE TSP 2022

Gradient tracking has improved transient iteration complexity

Theorem [GT convergence]

Under the above assumptions and with proper γ, Gradient tracking converges as

$$\text{(GT)} \qquad \frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} + \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}}\right)$$

- Recall DSGD with heterogeneous data converges as follows

$$\text{(DSGD)} \qquad \frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = \mathcal{O}\left(\frac{\sigma}{\sqrt{nT}} + \frac{\rho^{2/3}\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}} + \frac{\rho^{2/3}b^{2/3}}{T^{2/3}(1-\rho)^{2/3}}\right)$$

- Gradient tracking eliminates the influence of data heterogeneity and shorten the transient stage

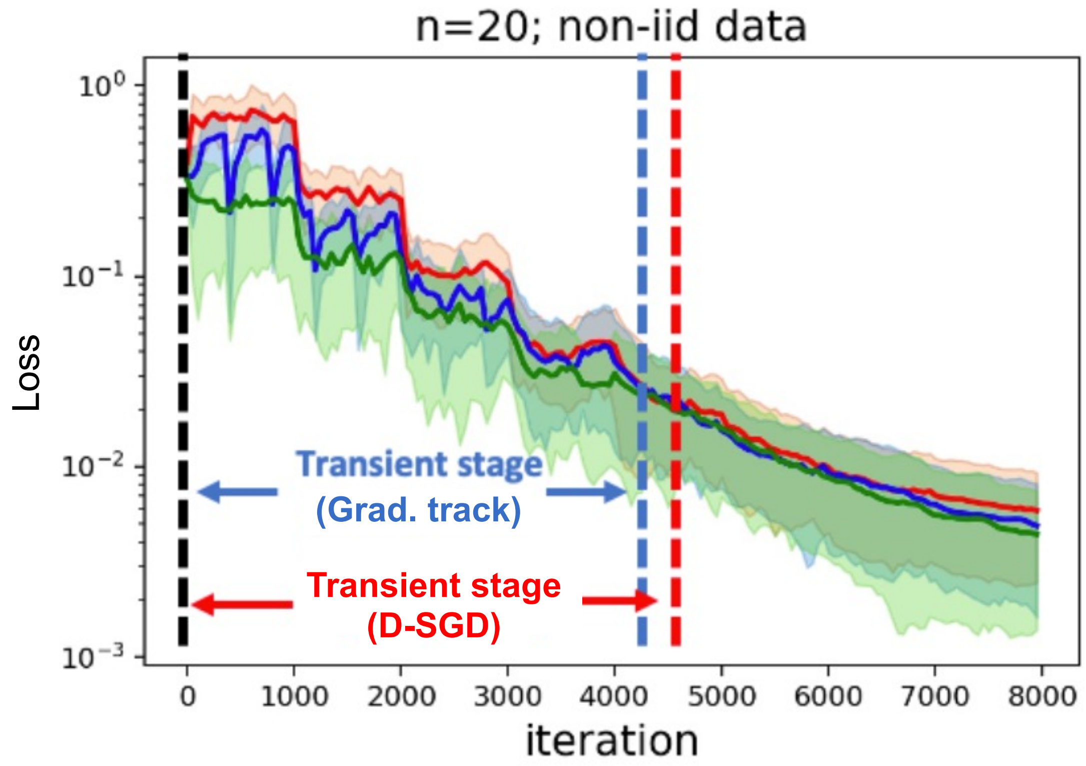
DSGD: $\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2} + \frac{\rho^4 n^3 b^4}{\sigma^6(1-\rho)^4}\right)$ → Gradient tracking: $\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2}\right)$

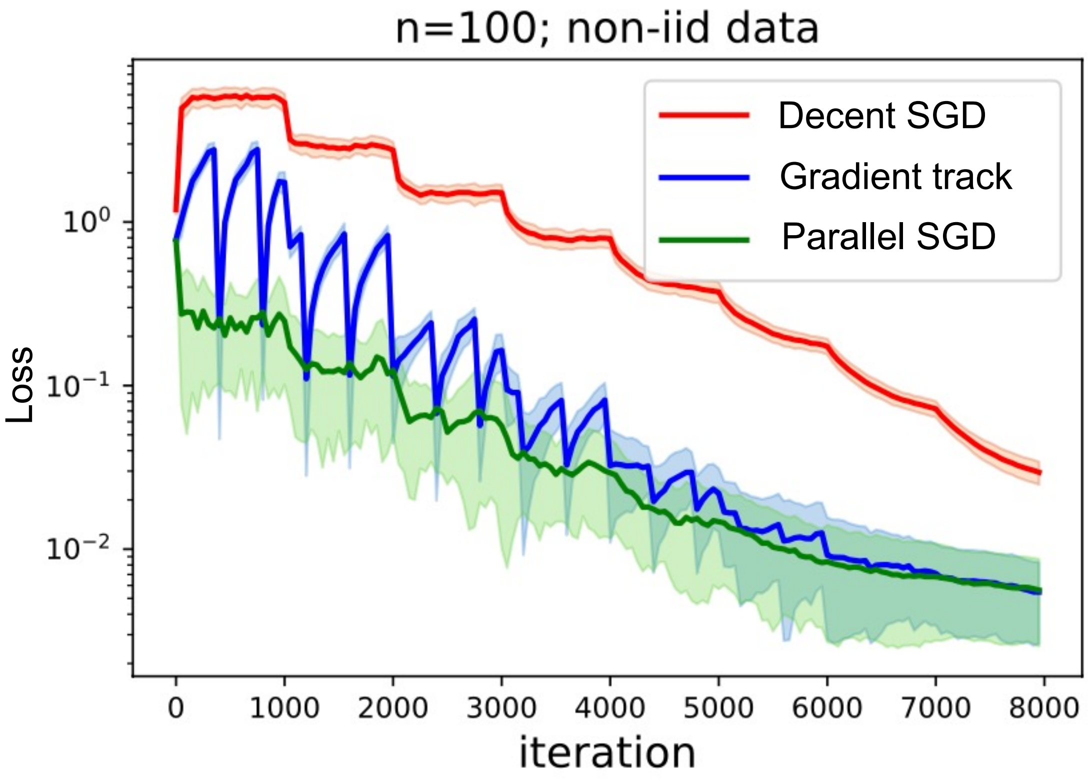
Empirical studies

DSGD: $\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2}+\frac{\rho^4 n^3 b^4}{\sigma^6(1-\rho)^4}\right)$ Gradient tracking: $\mathcal{O}\left(\frac{\rho^4 n^3}{\sigma^2(1-\rho)^2}\right)$

- DNN training over ring topology with **non-iid data**

$$1-\rho=O(n^{-2})$$





- Gradient tracking has much shorter transient stage than D-SGD for sparse topology

PART 03-4

Decentralized SGD: topology design

Trade-off in topology design

- Given a topology, its maximum degree $d_{\max}$ decides communication overhead per iteration

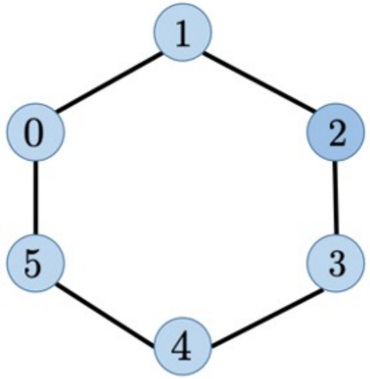
- Its connectivity $\rho = \|W - \frac{1}{n}\mathbb{1}\mathbb{1}^T\|_2 \in (0,1)$ decides the length of transient stage $\mathcal{O}(n^3/(1-\rho)^2)$

- Trade-off between per-iteration communication and transient iteration complexity

	Sparse topology	Dense topology
per-iter comm.	✓	×
trans. iter. complexity	×	✓

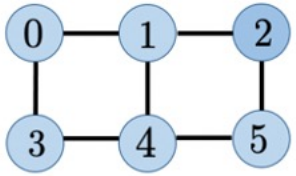
What topology to use?

- Shall we use these common topologies to organize all nodes?



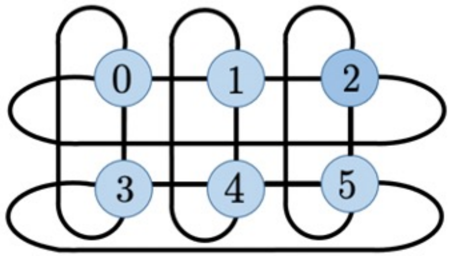
Ring

$$\rho = \mathcal{O}(1 - \frac{1}{n^2})$$



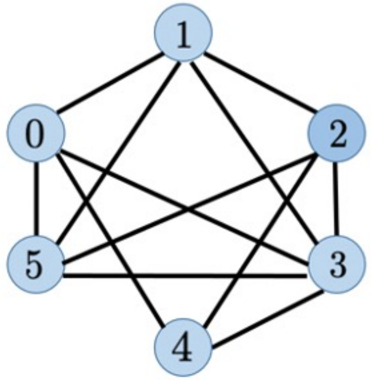
Grid

$$\mathcal{O}(1 - \frac{1}{n\ln(n)})$$



Torus

$$\mathcal{O}(1 - \frac{1}{n})$$



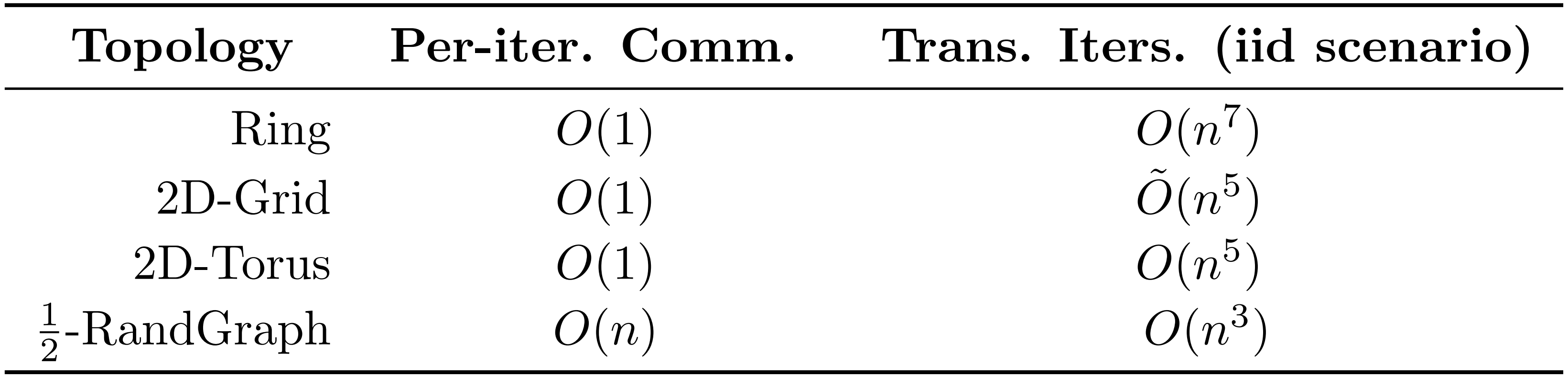
Erdos-Renyi Random

$$\mathcal{O}(\rho)$$

$\rho \in (0, 1)$ independent of n

What topology to use?

- Communication cost v.s. transient iteration complexity in DSGD



Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$O(1)$	$O(n^7)$
2D-Grid	$O(1)$	$\tilde{O}(n^5)$
2D-Torus	$O(1)$	$O(n^5)$
$\frac{1}{2}$-RandGraph	$O(n)$	$O(n^3)$

The smaller both comm. cost and tran. Iters. are, the better

- These topologies either have expensive communication cost or longer transient stage

- **Is there any topology that enables both cheap communication and fast convergence?**

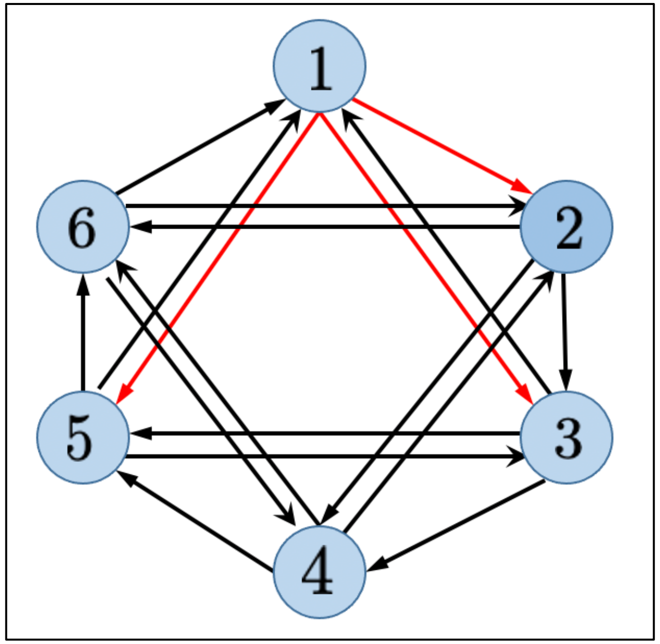
Static exponential graph: topology and per-iteration comm.

- Each node links to neighbors that are $2^0, 2^1, \cdots, 2^{\lfloor \log_2(n-1) \rfloor}$ away [1,2]

- In the figure, node 1 connects to node 2, 3 and 5.

- Each node has $\lceil \log_2(n) \rceil$ neighbors; per-iter comm. cost is $O(\log_2(n))$

- Empirically successful in deep training but less theoretically understood



[1] M. Assran et. al., “Stochastic Gradient Push for Distributed Deep Learning”, ICML 2018

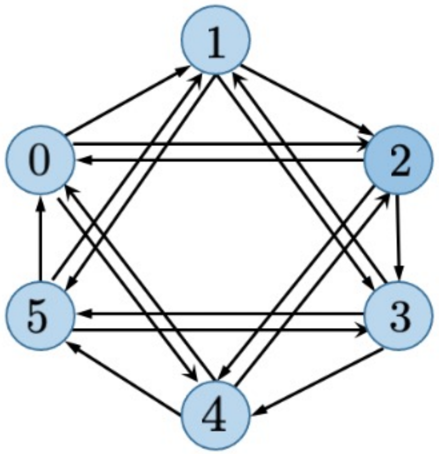
[2] B. Ying, K. Yuan, Y. Chen, H. Hu, P. Pan, W. Yin, Exponential Graph is Provably Efficient for Decentralized Deep Training, NeurIPS 2021

Static exponential graph: weight matrix

- The weight matrix associated with exponential graph is defined as (doubly stochastic)

$$w_{ij}^{\text{exp}} = \begin{cases} \frac{1}{\lceil \log_2(n) \rceil + 1} & \text{if } \log_2(\text{mod}(j-i,n)) \text{ is an integer or } i=j \\ 0 & \text{otherwise.} \end{cases}$$

- An illustrating example:



$$W = \begin{bmatrix} \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$$

Static exponential graph: connectivity

- Is the static exponential graph well connected?

Theorem. Let $\tau = \lceil \log_2(n) \rceil$, and $\rho = \|W - \frac{1}{n}\mathbb{1}\mathbb{1}^T\|_2$ be the spectral gap. It holds that

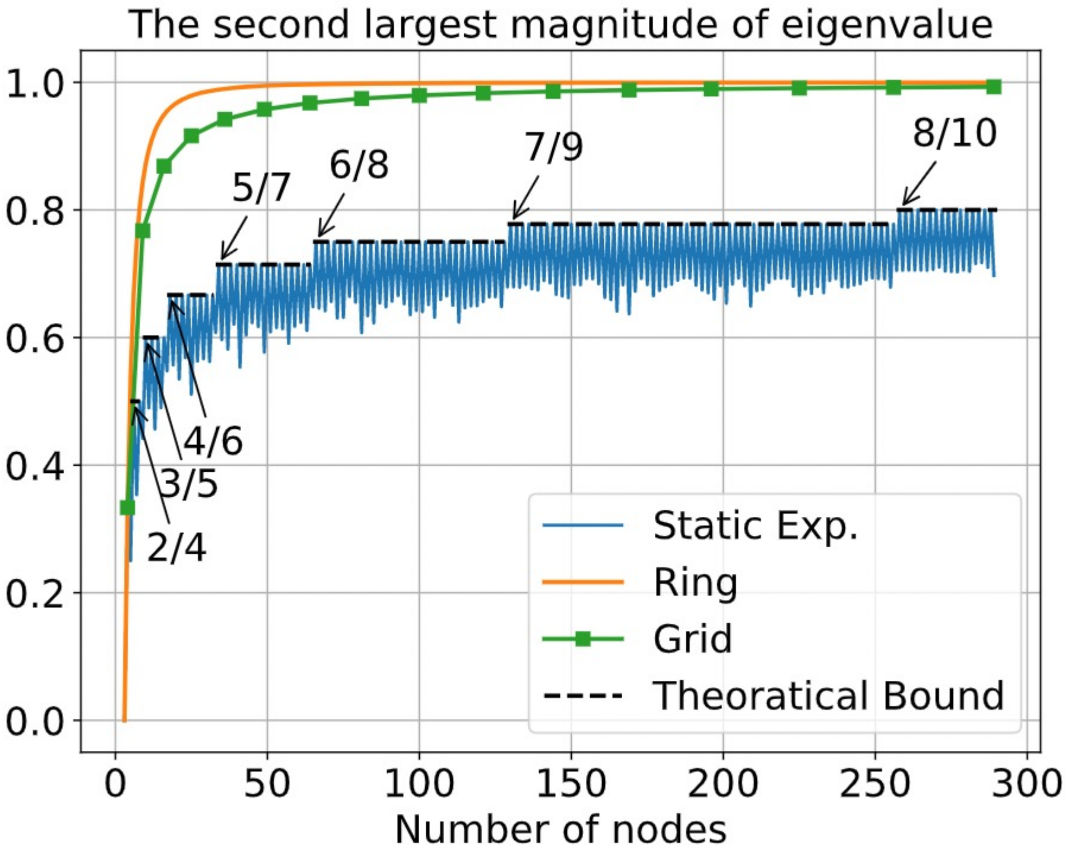
$$\begin{cases} \rho = 1 - \dfrac{2}{\tau + 1}, & \text{when } n \text{ is even} \\ \rho < 1 - \dfrac{2}{\tau + 1}, & \text{when } n \text{ is odd} \end{cases}$$

- This theorem implies that exponential graph has $\rho(W) = O(1 - 1/\log_2(n))$

- Highly non-trivial proofs; requires smart utilization of Fourier transform

B. Ying, K. Yuan, Y. Chen, H. Hu, P. Pan, W. Yin, Exponential Graph is Provably Efficient for Decentralized Deep Training, NeurIPS 2021

Static exponential graph: illustration of the spectral gap



- Our theoretical bound is very tight

- Spectral gap increases slowly when n grows

Static exponential graph: transient iterations in DSGD

- Recall DSGD has transient iteration complexity $O(n^3/(1-\rho)^2)$ in iid scenarios

- With $\rho(W) = O(1 - 1/\log_2(n))$, exponential graphs have tran. iters. as $O(n^3 \log_2^2(n))$

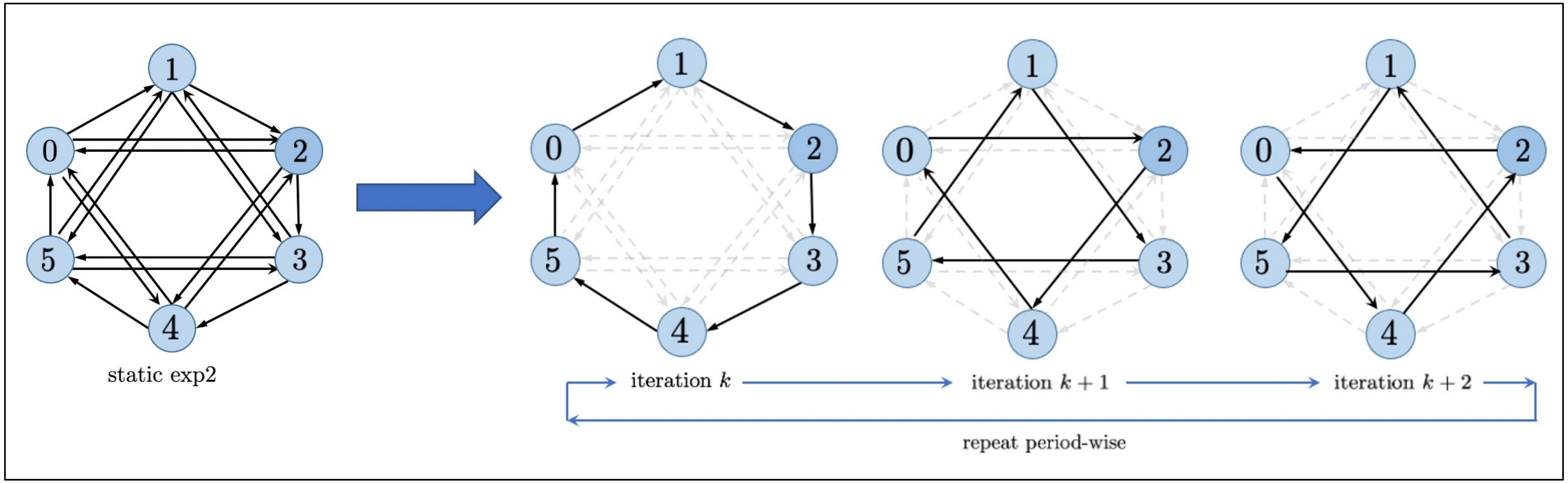
- Per-iteration communication and transient iteration complexity are **nearly the best** (up to $\log_2(n)$)

Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$O(1)$	$O(n^7)$
2D-Grid	$O(1)$	$\tilde{O}(n^5)$
2D-Torus	$O(1)$	$O(n^5)$
$\frac{1}{2}$-RandGraph	$O(n)$	$O(n^3)$
Static Exp	$\tilde{O}(1)$	$\tilde{O}(n^3)$

- **Can we achieve even better topology?**

One-peer exponential graph: topology

- **Split** exponential graph into a sequence of one-peer realizations;
- Each node has **exactly one** neighbor per iteration
- **O(1) per-iteration communication**; cheaper than ring (2 neighbors) or grid (3 or 4 neighbors)

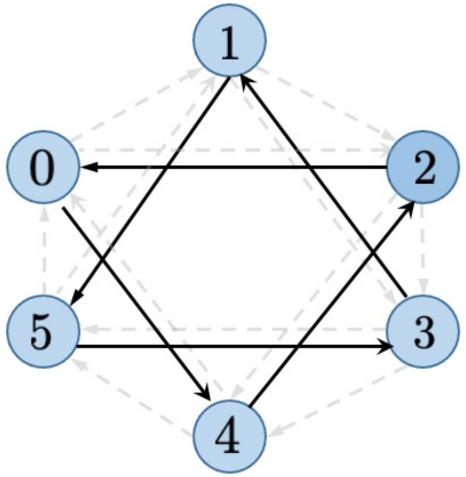


One-peer exponential graph: weight matrix

- We let $\tau = \lceil \log_2(n) \rceil$. The weight matrix $W^{(k)}$ is defined as

$$w_{ij}^{(k)} = \begin{cases} \frac{1}{2} & \text{if } \log_2(\mathrm{mod}(j-i,n)) = \mathrm{mod}(k,\tau) \\ \frac{1}{2} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

- An illustrating example



$$W = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

DSGD over one-peer exponential graph

Sample $W^{(k)}$ over one-peer exponential graph

$$x_i^{(k+\frac{1}{2})} = x_i^{(k)} - \gamma \nabla F(x_i^{(k)}; \xi_i^{(k)}) \qquad \text{(Local update)}$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij}^{(k)} x_j^{(k+\frac{1}{2})} \qquad \text{(Partial averaging)}$$

- DSGD with **time-varying** weight matrix;
- Per-iteration communication cost is **O(1)**; very efficient

One-peer exponential graph: Periodic exact average

- While one-peer exponential graph is **sparse**, it is **effective** to aggregate information

Finite-time exact convergence

Theorem. Suppose $\tau = \log_2(n)$ is a positive integer. It holds that

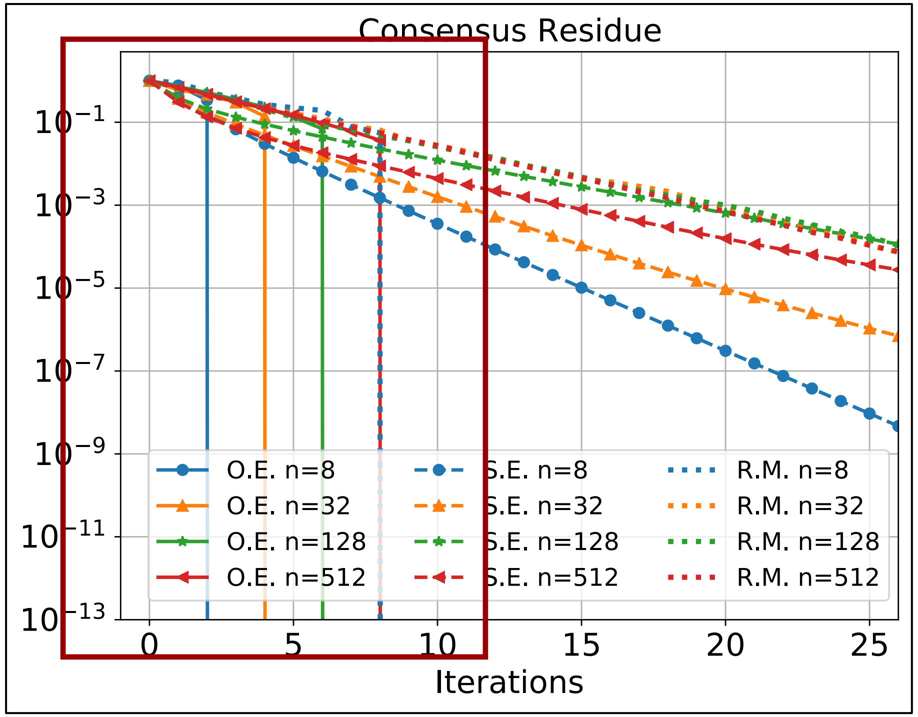
$$W^{(k+\ell)}W^{(k+\ell-1)}\dots W^{(k+1)}W^{(k)} = \frac{1}{n}\mathbb{1}\mathbb{1}^T$$

for any integer $k \geq 0$ and $\ell \geq \tau - 1$.

- While each realization is sparser, a sequence (with length τ) of one-peer graphs will enable effective global averaging

One-peer exponential graph: Periodic exact average

- We examine $\|\overbrace{\frac{1}{n}\mathbb{1}\mathbb{1}^T x}^{\text{Global average}} - \overbrace{\prod_{k=0}^{T} W^{(k)} x}^{\text{Partial average}}\|$ for a vector x
- $\frac{1}{n}\mathbb{1}\mathbb{1}^T x$ is the global average
- $\prod_{k=0}^{T} W^{(k)} x$ is the partial average after T iterations
- One-peer exp. achieves global average after $\log_2(n)$ iters.



Apply one-peer exponential graph to DSGD

Assumption *(1) Each $f_i(x)$ is L-smooth; (2) Each gradient noise is unbiased and has bounded variance σ^2; (3) Each local distribution D_i is identical (iid)*

Theorem *Under the above assumptions and with $\gamma = O(1/\sqrt{T})$, let $\tau = \log_2(n)$ be an integer, DSGD with one-peer exponential graph will converge at*

$$\frac{1}{T}\sum_{k=1}^{T}\mathbb{E}\|\nabla f(\bar{x}^{(k)})\|^2 = O\Big(\frac{\sigma}{\sqrt{nT}} + \underbrace{\frac{\sigma^{2/3}\log_2^{1/3}(n)}{T^{2/3}}}_{\text{extra overhead}}\Big)$$

Novel analysis; require new tricks to utilize periodic exact average to establish tight convergence

Static v.s. one-peer exponential graph (iid scenario)

- Convergence rate of DSGD with iid dataset over static and one-peer exponential graphs are

$$\text{Static exp.} \quad O\Big(\frac{\sigma}{\sqrt{nT}} + \frac{\sigma^{2/3}}{T^{2/3}(1-\rho)^{1/3}}\Big) \quad (\text{where } 1-\rho = O(1/\log_2(n)))$$

$$\text{One-peer exp.} \quad O\Big(\frac{\sigma}{\sqrt{nT}} + \frac{\sigma^{2/3}\log_2^{1/3}(n)}{T^{2/3}}\Big)$$

- DSGD with one-peer exp. converges **as fast as** static exp.; **a surprising result**.

- DSGD with both graphs are with the **same** transient iteration complexity $O(n^3\log_2^2(n))$

- The communication cost saving in one-peer exponential graph is a **free lunch**

Compare one-peer exp. with other topologies

Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$O(1)$	$O(n^7)$
2D-Grid	$O(1)$	$\tilde{O}(n^5)$
2D-Torus	$O(1)$	$O(n^5)$
$\frac{1}{2}$-RandGraph	$O(n)$	$O(n^3)$
Static Exp	$\tilde{O}(1)$	$\tilde{O}(n^3)$
One-Peer Expo	$O(1)$	$\tilde{O}(n^3)$

We recommend using one-peer exponential graph in deep training.

Experiments in deep training (image classification)

ImageNet-1K dataset

1.3M training images

50K test images

1K classes

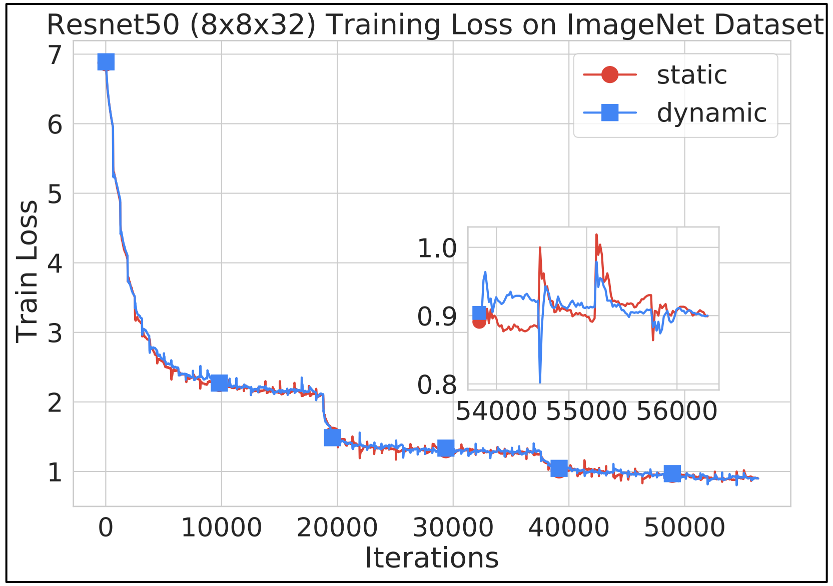
DNN model: ResNet-50 (25.5M parameters)

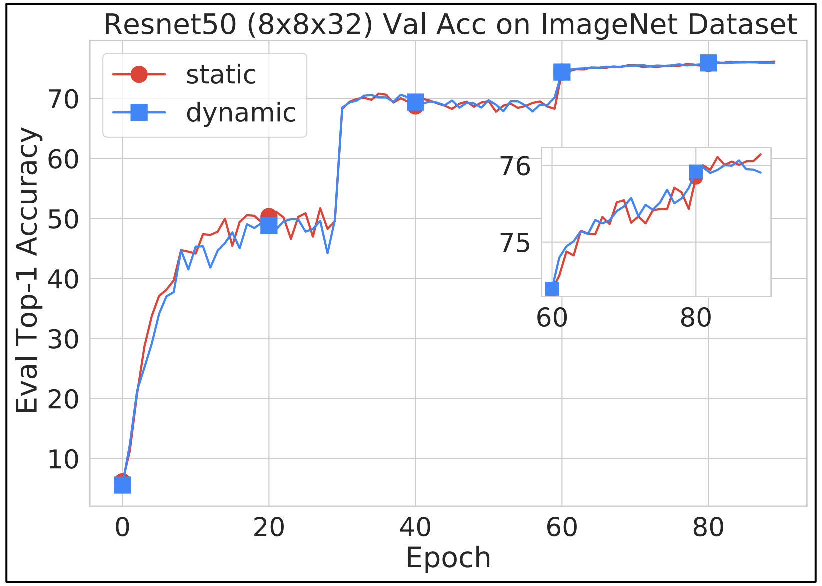
GPU: Up to 256 Tesla V100 GPUs

- **Wall-clock time** to finish 90 epochs of training; measures per-iter communication
- **Validation accuracy** after 90 epochs of training; measures convergence rate

One peer is not slower than static exponential graph

Image classification: ResNet-50 for ImageNet; $8 \times 8 = 64$ GPUs.





One-peer and exponential graphs converge **roughly the same**; but one-peer is more comm. efficient

DSGD over one-peer Exp. achieves better linear speedup

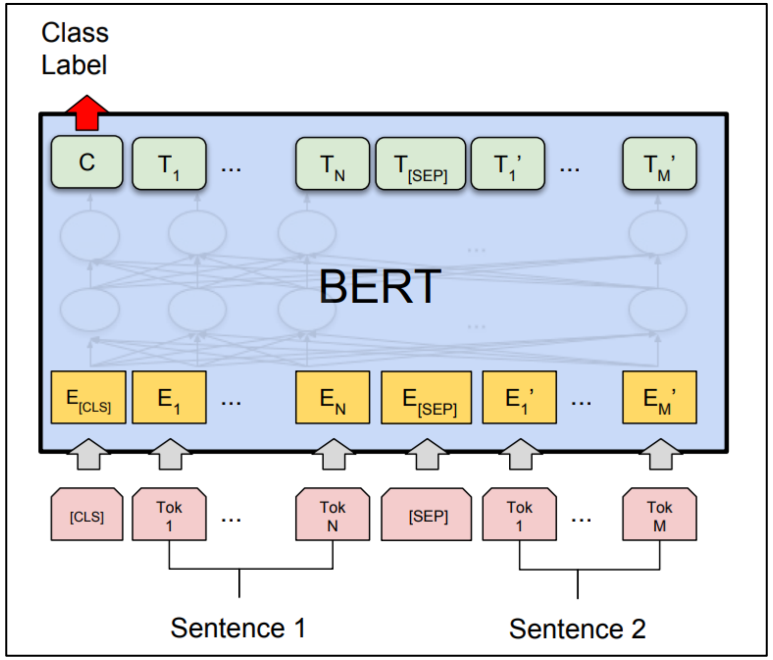
nodes	4(4x8 GPUs)		8(8x8 GPUs)		16(16x8 GPUs)		32(32x8 GPUs)	
topology	acc.	time	acc.	time	acc.	time	acc.	time
P-SGD	76.32	11.6	76.47	6.3	76.46	3.7	76.25	2.2

DSGD over ring has more efficient comm. than PSGD; **suffers from performance degradation**

DSGD over one-peer exp. graph is more comm.-efficient **without performance degradation**

[YYC+21]B. Ying, K. Yuan, Y. Chen, H. Hu, P. Pan, and W. Yin, ``Exponential Graph is Provably Efficient for Deep Training", NeurIPS 2021

Experiments in deep training (language modeling)



Model: BERT-Large (330M parameters)

Dataset: Wikipedia (2500M words) and

BookCorpus (800M words)

Hardware: 64 GPUs

Table. Comparison in loss and training time [CYZ+21]

Method	Final Loss	Wall-clock Time (hrs)
P-SGD	1.75	59.02
D-SGD	1.77	30.4

[CYZ+21] Y. Chen, K. Yuan, Y. Zhang, P. Pan, Y. Xu, and W. Yin, ``Accelerating Gossip SGD with Periodic Global Averaging”, ICML 2021

A brief summary

- Exponential graphs are both sparse and effective. They are nearly best up to logarithm terms
- One-peer exponential graph is even sparser without hurting effectiveness

Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$O(1)$	$O(n^7)$
2D-Grid	$O(1)$	$\tilde{O}(n^5)$
2D-Torus	$O(1)$	$O(n^5)$
$\frac{1}{2}$-RandGraph	$O(n)$	$O(n^3)$
Static Exp	$\tilde{O}(1)$	$\tilde{O}(n^3)$
One-Peer Expo	$O(1)$	$\tilde{O}(n^3)$

However …

- Periodic exact average for one-peer exp. **only holds** when network size **n is a power of 2**

- Not known when one-peer exp. performs well when n is **not** a power of 2

- Not known whether the transient iteration $O(n^3 \log_2^2(n))$ can be further improved to $O(n^3)$

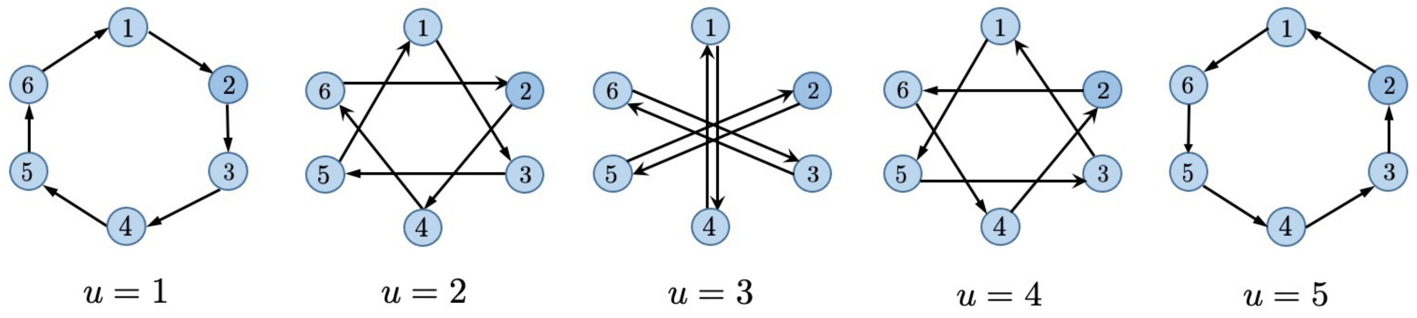
Can we develop topologies that

- have $O(1)$ per-iteration communication cost;

- enable DSGD to converge with $O(n^3)$ transient iteration complexity;

- and are valid for any network size n?

More advanced topologies: EquiTopo graph

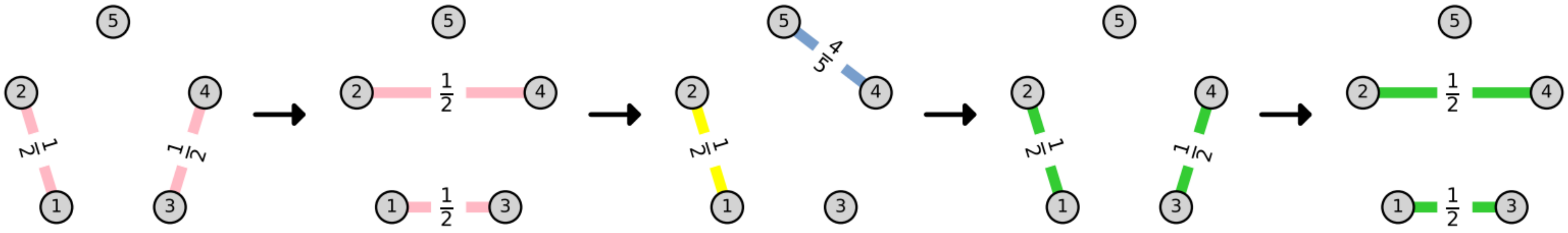


(work for any size)

Topology	Per-iter. Comm.	Trans. Iters. (iid scenario)
Ring	$O(1)$	$O(n^7)$
2D-Grid	$O(1)$	$\tilde{O}(n^5)$
2D-Torus	$O(1)$	$O(n^5)$
$\frac{1}{2}$-RandGraph	$O(n)$	$O(n^3)$
Static Exp	$\tilde{O}(1)$	$\tilde{O}(n^3)$
One-Peer Expo	$O(1)$	$\tilde{O}(n^3)$
O.-P. EquiDyn	$O(1)$	$O(n^3)$ (in expectation)

Z. Song, et. al., “Communication-Efficient Topologies for Decentralized Learning with O(1) Consensus Rate”, NeurIPS 2022

More advanced topologies: Base-k graph



(work for any size)

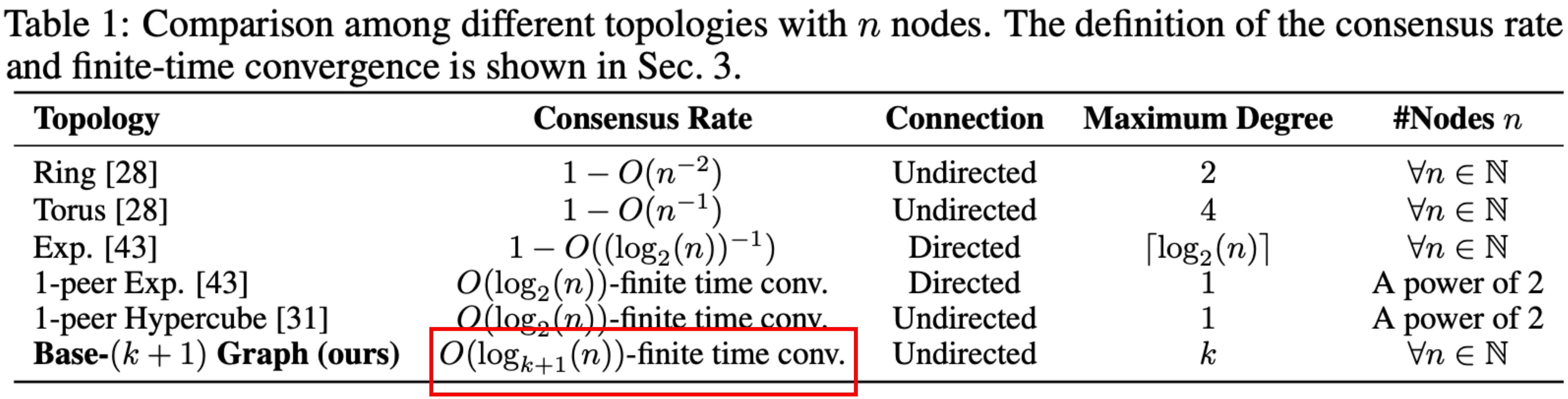


Table 1: Comparison among different topologies with n nodes. The definition of the consensus rate and finite-time convergence is shown in Sec. 3.

Topology	**Consensus Rate**	**Connection**	**Maximum Degree**	**#Nodes n**
Ring [28]	$1 - O(n^{-2})$	Undirected	2	$\forall n \in \mathbb{N}$
Torus [28]	$1 - O(n^{-1})$	Undirected	4	$\forall n \in \mathbb{N}$
Exp. [43]	$1 - O((\log_2(n))^{-1})$	Directed	$\lceil \log_2(n) \rceil$	$\forall n \in \mathbb{N}$
1-peer Exp. [43]	$O(\log_2(n))$-finite time conv.	Directed	1	A power of 2
1-peer Hypercube [31]	$O(\log_2(n))$-finite time conv.	Undirected	1	A power of 2
Base-$(k+1)$ Graph (ours)	$\boxed{O(\log_{k+1}(n))\text{-finite time conv.}}$	Undirected	k	$\forall n \in \mathbb{N}$

Y. Takezawa, et. al., “Beyond Exponential Graph: Communication-Efficient Topologies for Decentralized Learning via Finite-time Convergence”, NeurIPS 2023

PART 03-5

BlueFog: An open-source and high-performance python library

BlueFog

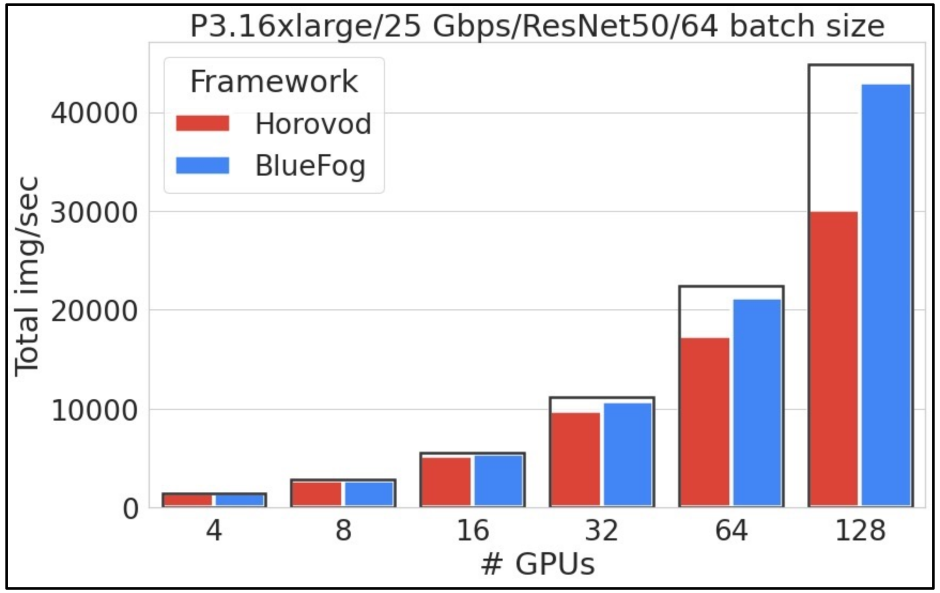
https://github.com/Bluefog-Lib/bluefog

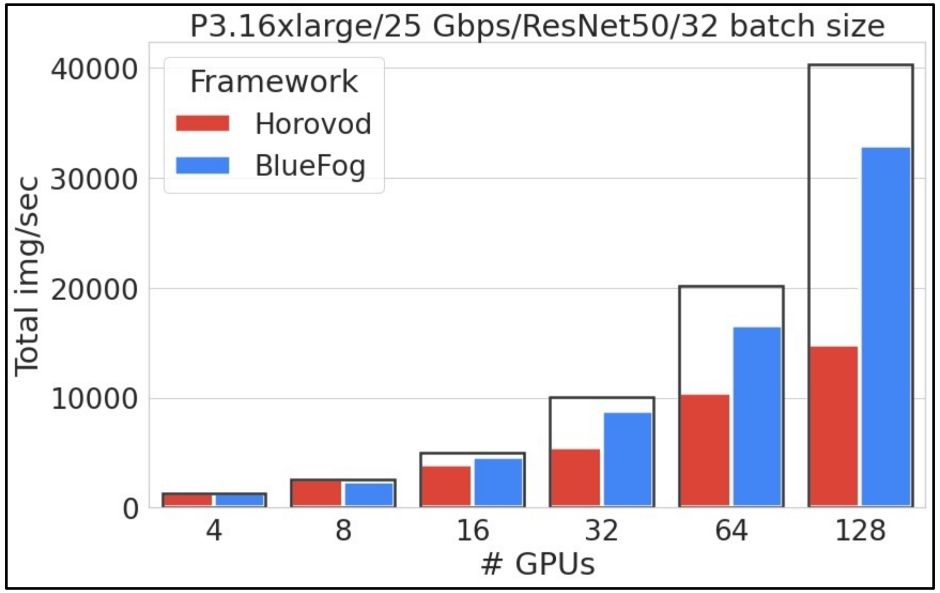
BlueFog

- An open-source library to support decentralized communication in optimization and deep learning
- High-performance
- Easy-to-use

High-performance

- BlueFog has larger throughput than Horovod (the SOTA DL system implementing PSGD) [YYH+21]





- All our research progresses are involved in BlueFog

[YYH+21] B. Ying, K. Yuan, H. Hu, Y. Chen, and W. Yin, ``BlueFog: Make Decentralized Algorithms Practical for Optimization and machine learning", arXiv:2111.04287 [GitHub site: github.com/Bluefog-Lib/bluefog]

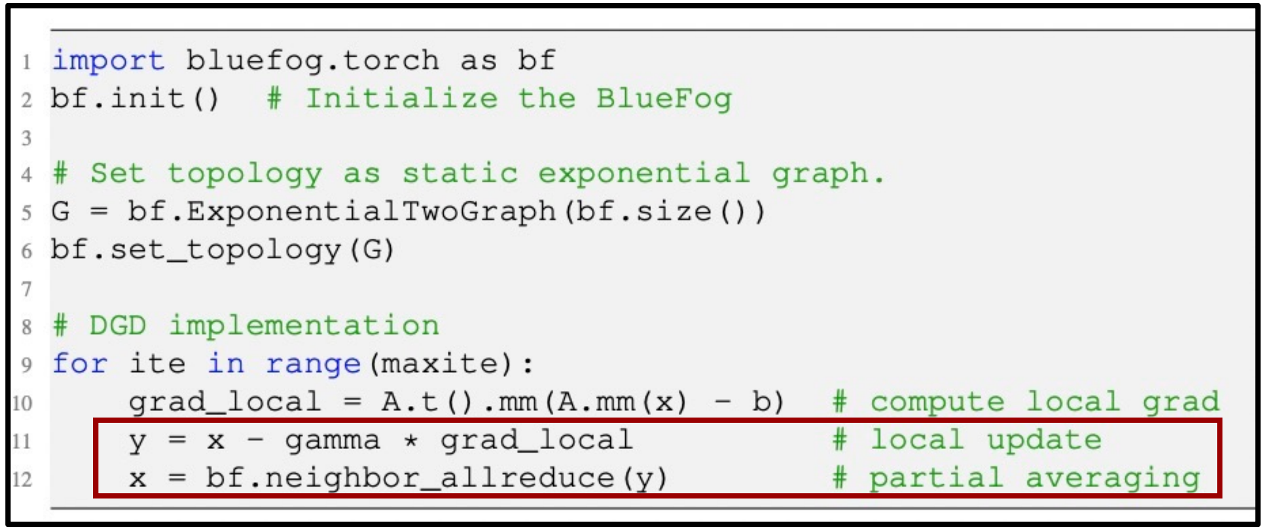
Easy-to-use

- Writing codes for decentralized methods is as easy as writing equations

Decentralized least-square algorithms

$$y_i^{(k)} = x_i^{(k)} - \gamma A_i^T (A_i x_i^{(k)} - b_i)$$

$$x_i^{(k+1)} = \sum_{j \in \mathcal{N}_i} w_{ij} y_j^{(k)}$$



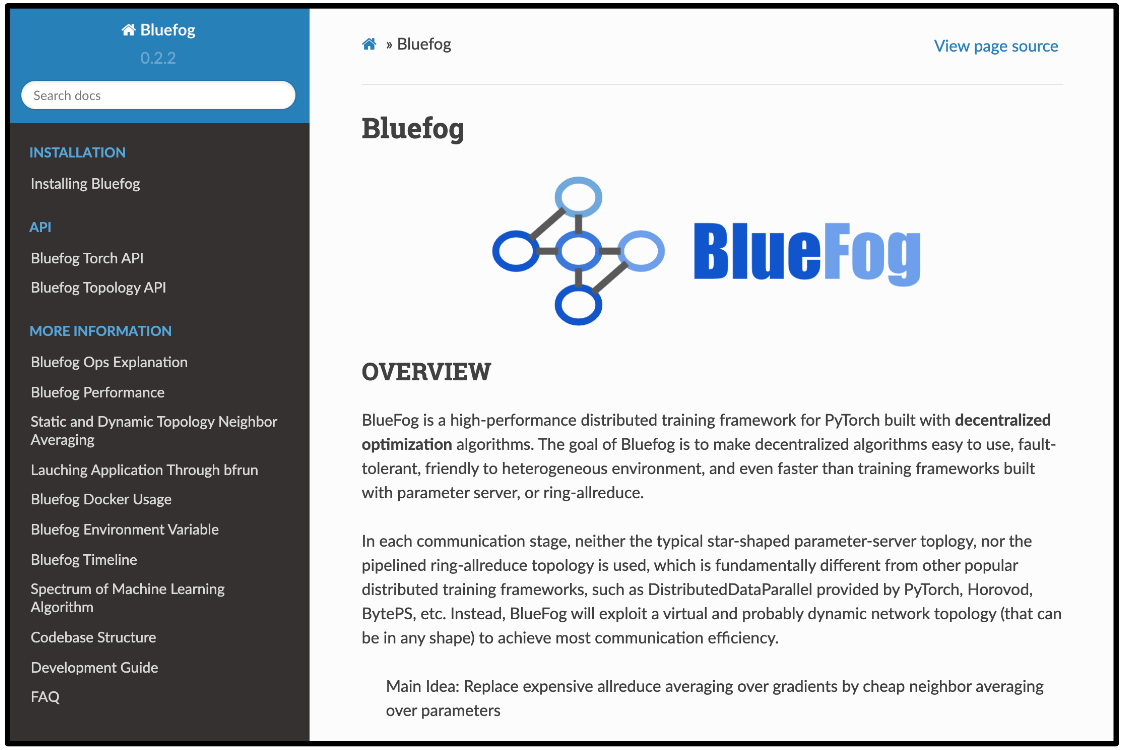
```python
import bluefog.torch as bf
bf.init()   # Initialize the BlueFog

# Set topology as static exponential graph.
G = bf.ExponentialTwoGraph(bf.size())
bf.set_topology(G)

# DGD implementation
for ite in range(maxite):
    grad_local = A.t().mm(A.mm(x) - b)   # compute local grad
    y = x - gamma * grad_local           # local update
    x = bf.neighbor_allreduce(y)         # partial averaging
```

Easy-to-use

Abundant documents



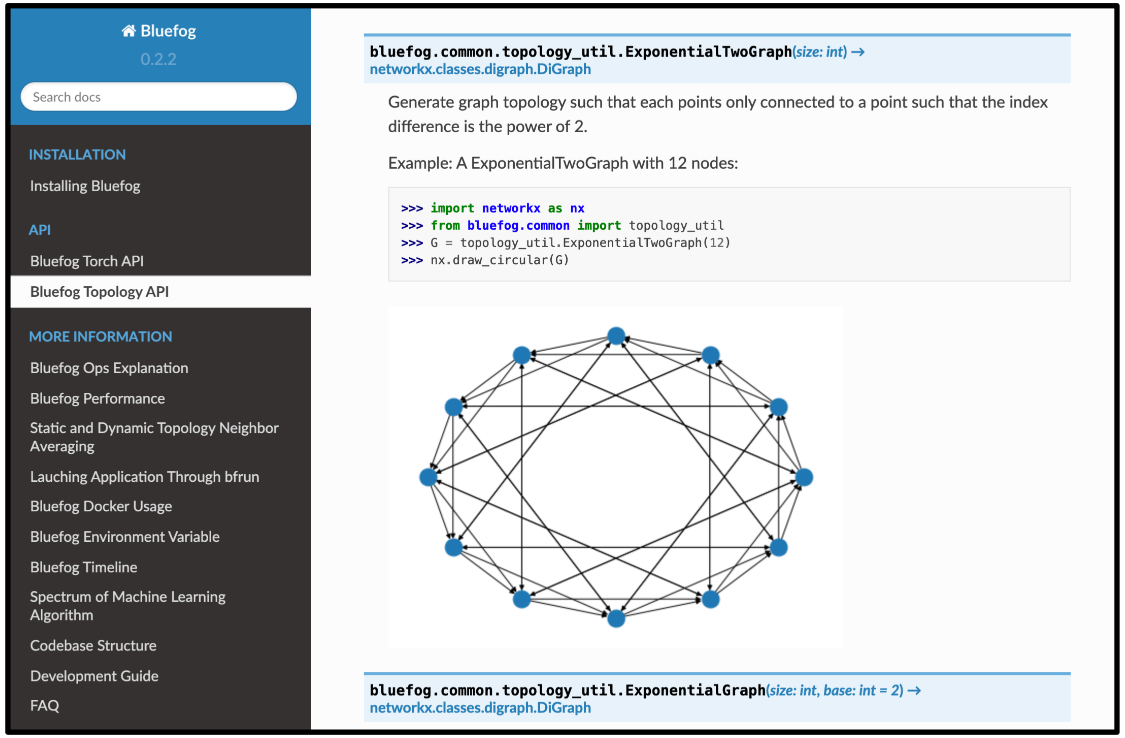
Bluefog

OVERVIEW

BlueFog is a high-performance distributed training framework for PyTorch built with **decentralized optimization** algorithms. The goal of Bluefog is to make decentralized algorithms easy to use, fault-tolerant, friendly to heterogeneous environment, and even faster than training frameworks built with parameter server, or ring-allreduce.

In each communication stage, neither the typical star-shaped parameter-server toplogy, nor the pipelined ring-allreduce topology is used, which is fundamentally different from other popular distributed training frameworks, such as DistributedDataParallel provided by PyTorch, Horovod, BytePS, etc. Instead, BlueFog will exploit a virtual and probably dynamic network topology (that can be in any shape) to achieve most communication efficiency.

> Main Idea: Replace expensive allreduce averaging over gradients by cheap neighbor averaging over parameters



bluefog.common.topology_util.ExponentialTwoGraph(*size: int*) → networkx.classes.digraph.DiGraph

Generate graph topology such that each points only connected to a point such that the index difference is the power of 2.

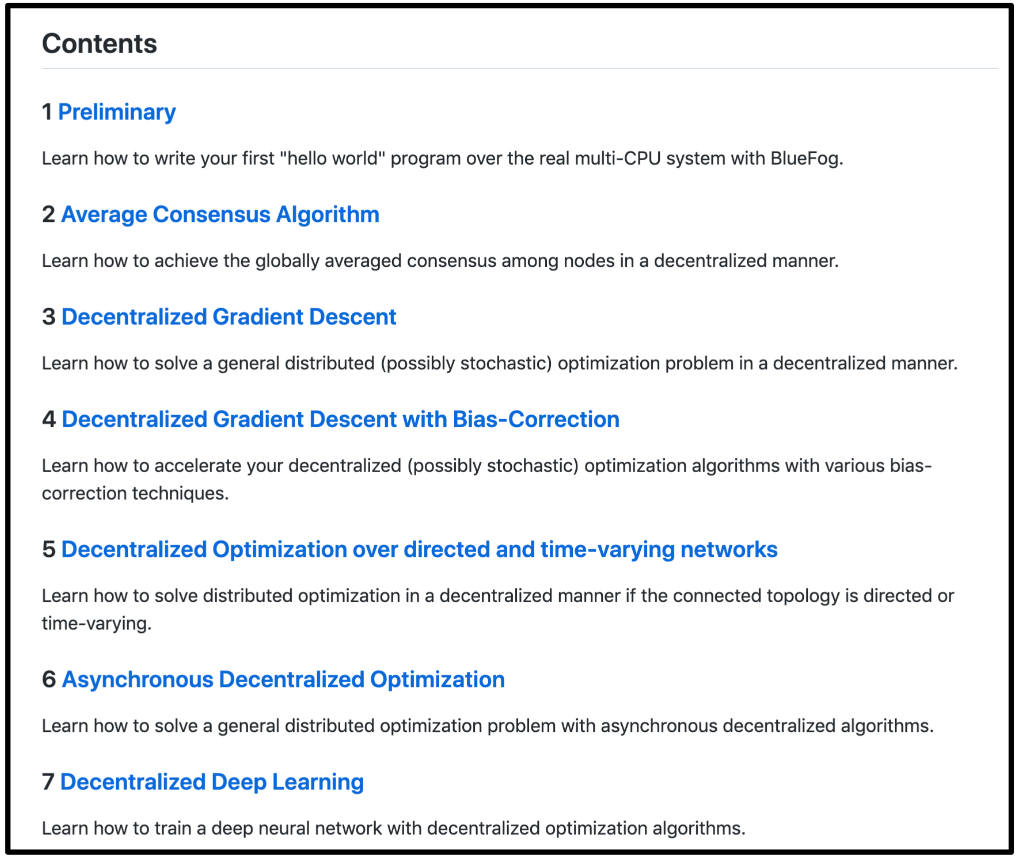
Example: A ExponentialTwoGraph with 12 nodes:

```
>>> import networkx as nx
>>> from bluefog.common import topology_util
>>> G = topology_util.ExponentialTwoGraph(12)
>>> nx.draw_circular(G)
```

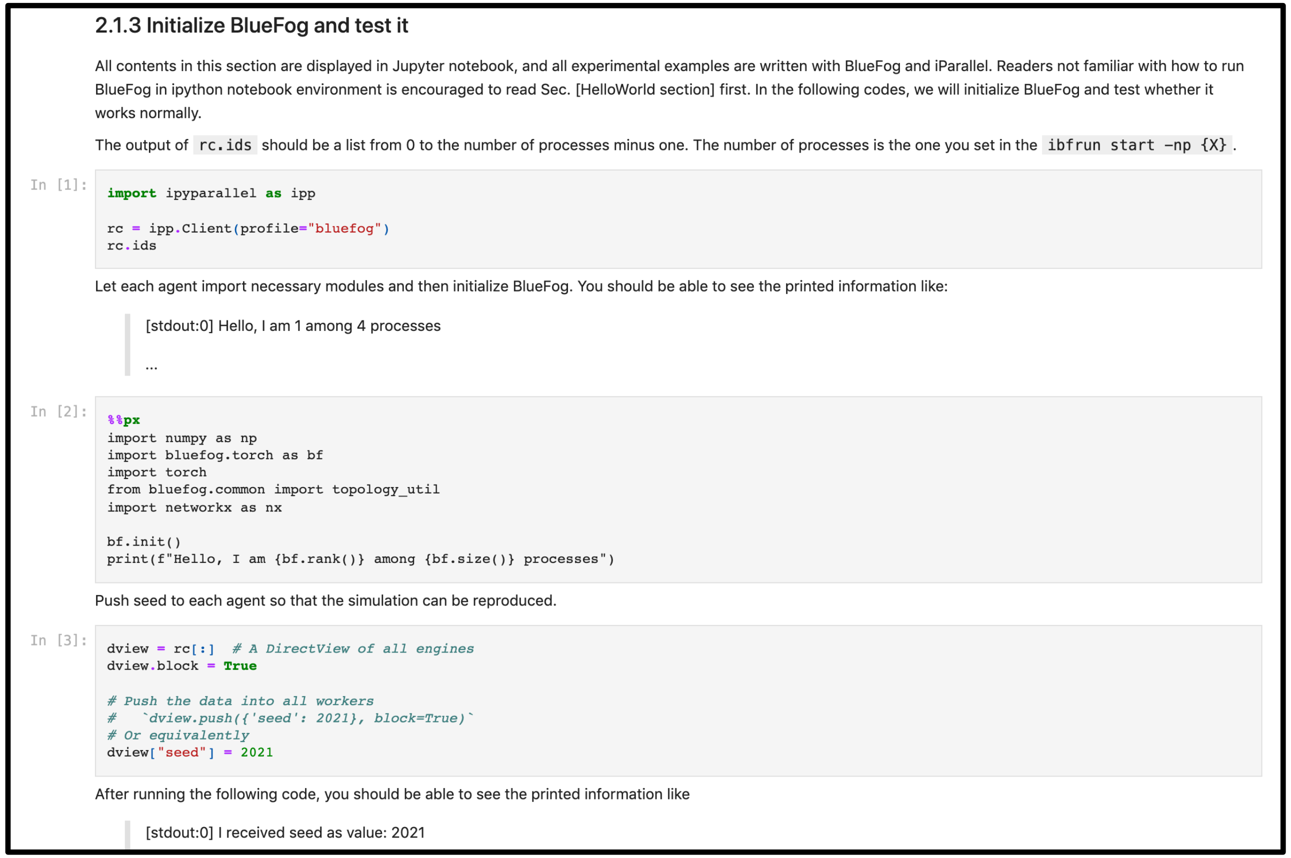
bluefog.common.topology_util.ExponentialGraph(*size: int, base: int = 2*) → networkx.classes.digraph.DiGraph

Easy-to-use

Detailed tutorials



Contents



2.1.3 Initialize BlueFog and test it

All contents in this section are displayed in Jupyter notebook, and all experimental examples are written with BlueFog and iParallel. Readers not familiar with how to run BlueFog in ipython notebook environment is encouraged to read Sec. [HelloWorld section] first. In the following codes, we will initialize BlueFog and test whether it works normally.

The output of `rc.ids` should be a list from 0 to the number of processes minus one. The number of processes is the one you set in the `ibfrun start -np {X}`.

In [1]:
```
import ipyparallel as ipp

rc = ipp.Client(profile="bluefog")
rc.ids
```

Let each agent import necessary modules and then initialize BlueFog. You should be able to see the printed information like:

> [stdout:0] Hello, I am 1 among 4 processes
>
> ...

In [2]:
```
%%px
import numpy as np
import bluefog.torch as bf
import torch
from bluefog.common import topology_util
import networkx as nx

bf.init()
print(f"Hello, I am {bf.rank()} among {bf.size()} processes")
```

Push seed to each agent so that the simulation can be reproduced.

In [3]:
```
dview = rc[:]  # A DirectView of all engines
dview.block = True

# Push the data into all workers
#   `dview.push({'seed': 2021}, block=True)`
# Or equivalently
dview["seed"] = 2021
```

After running the following code, you should be able to see the printed information like

> [stdout:0] I received seed as value: 2021

Decentralized SGD final summary

- Decentralized algorithms save remarkable communication compared to centralized ones
- By removing data heterogeneity, we can significantly improve the transient iterations
- Sparse and effective topologies make decentralized optimization practical for deep training
- In real GPU clusters, decentralized SGD show strong scaling performance without hurting the accuracy performance

Thank you!

Kun Yuan homepage: https://kunyuan827.github.io/

BlueFog homepage: https://github.com/Bluefog-Lib/bluefog