Faster Learning over Networks and BlueFog[1]

Bicheng Ying (Google), Kun Yuan (Alibaba), Hanbin Hu (UCSB)
Ji Liu (Baidu), **Wotao Yin** (UCLA)

The 18th China Symposium on Machine Learning and Applications

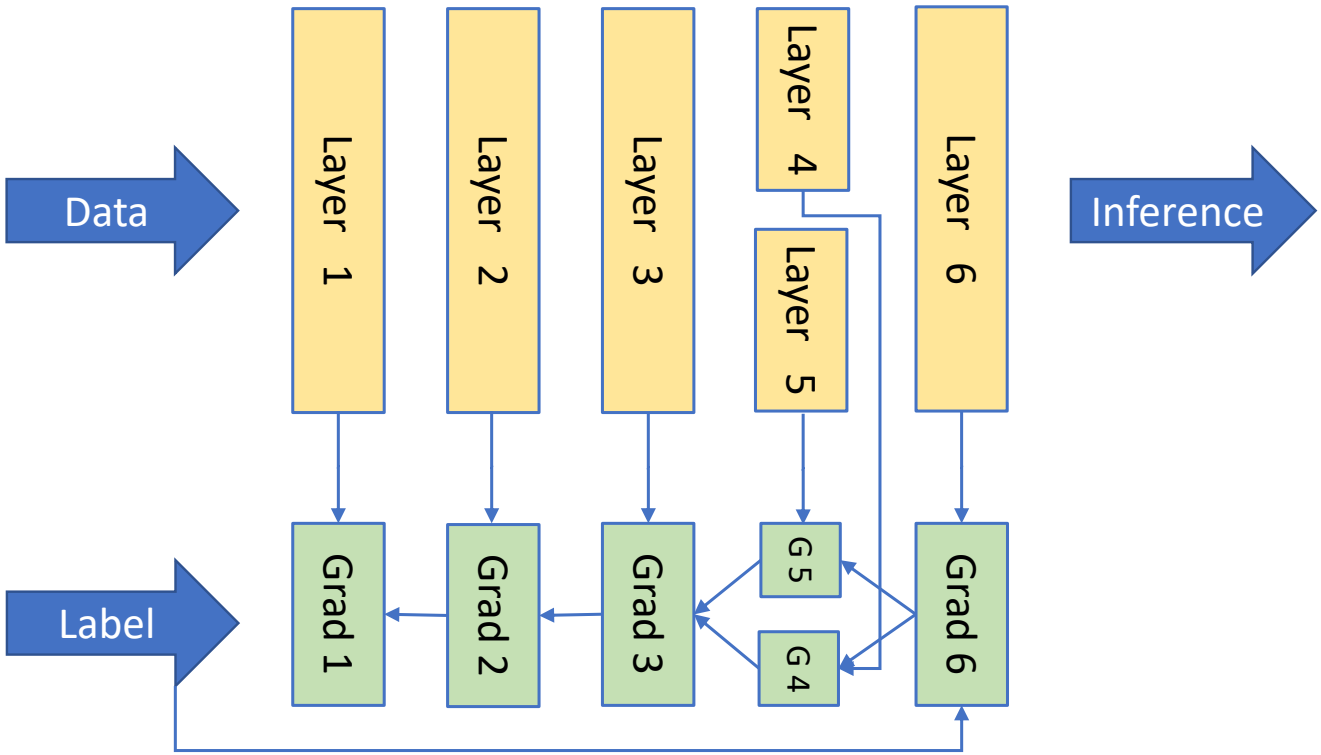
November 7, 2020

[1] Open source project `https://github.com/Bluefog-Lib/bluefog`

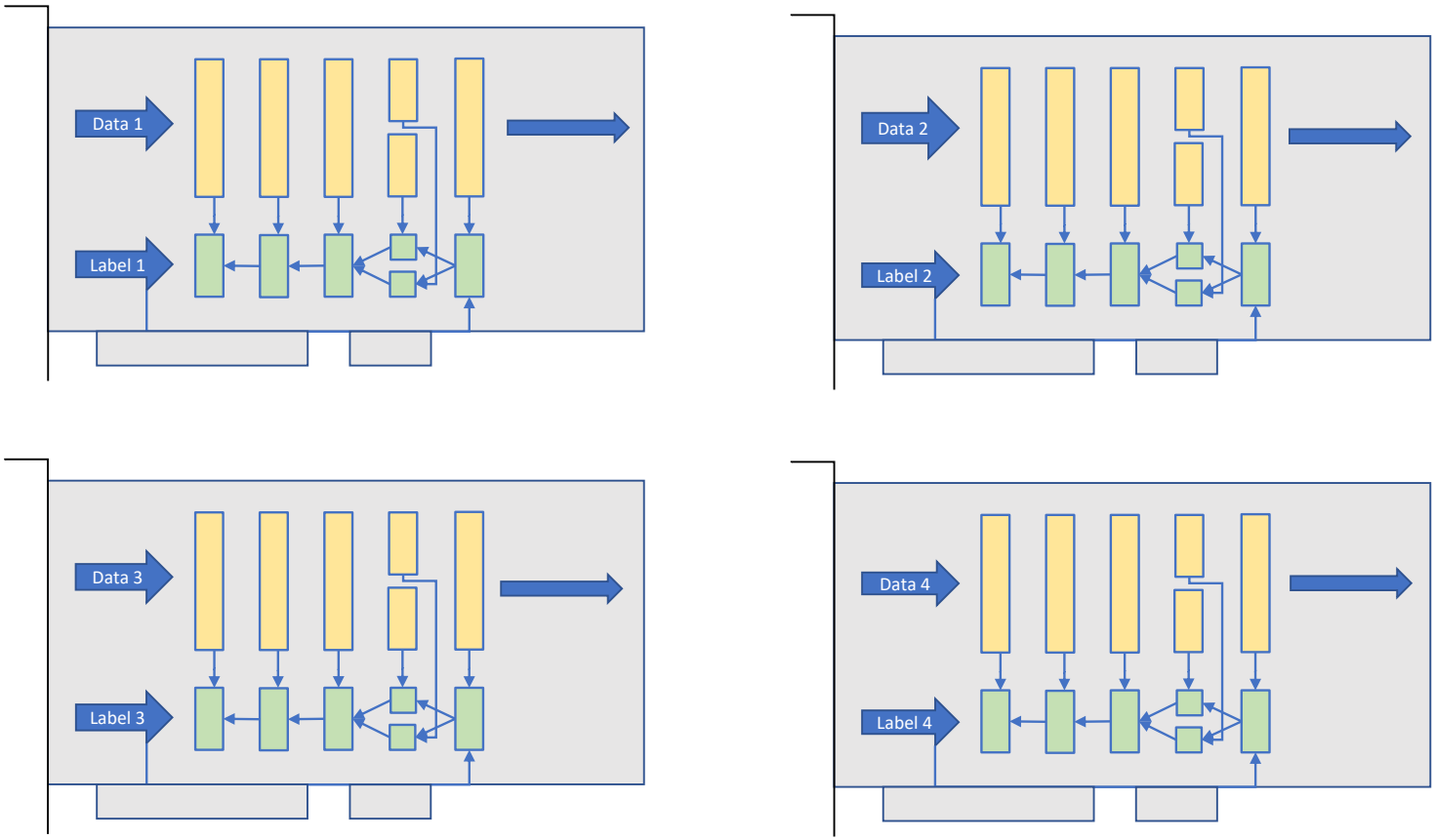
Among the biggest issues of DL research and applications

- Scale to larger models and bigger data
- Bring down training time from days to hours
- Separate low-level system implementations from ML modeling

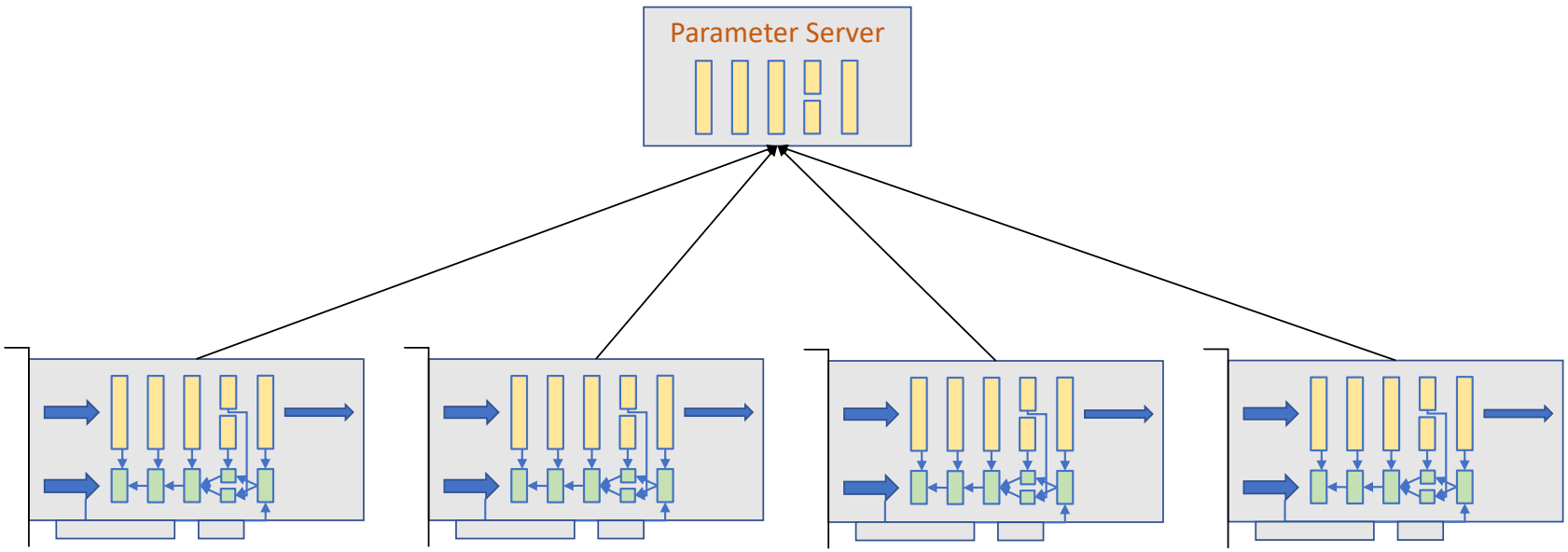
DNN training



Data parallel training



Parameter server approach [Li et.al. 2014]



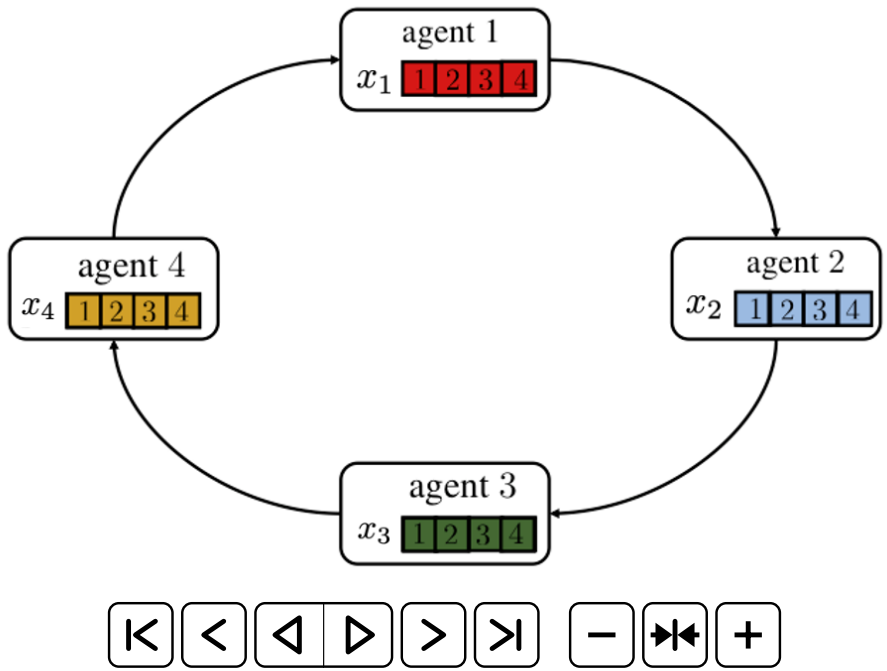
Pros: mature implementation (2015–), fault tolerance

Cons: many-to-one communication is no scalable

Ring Allreduce [Patarasuk and Yuan 2009]

Started by Distributed PaddlePaddle [Gibiansky 2017] (Baidu)

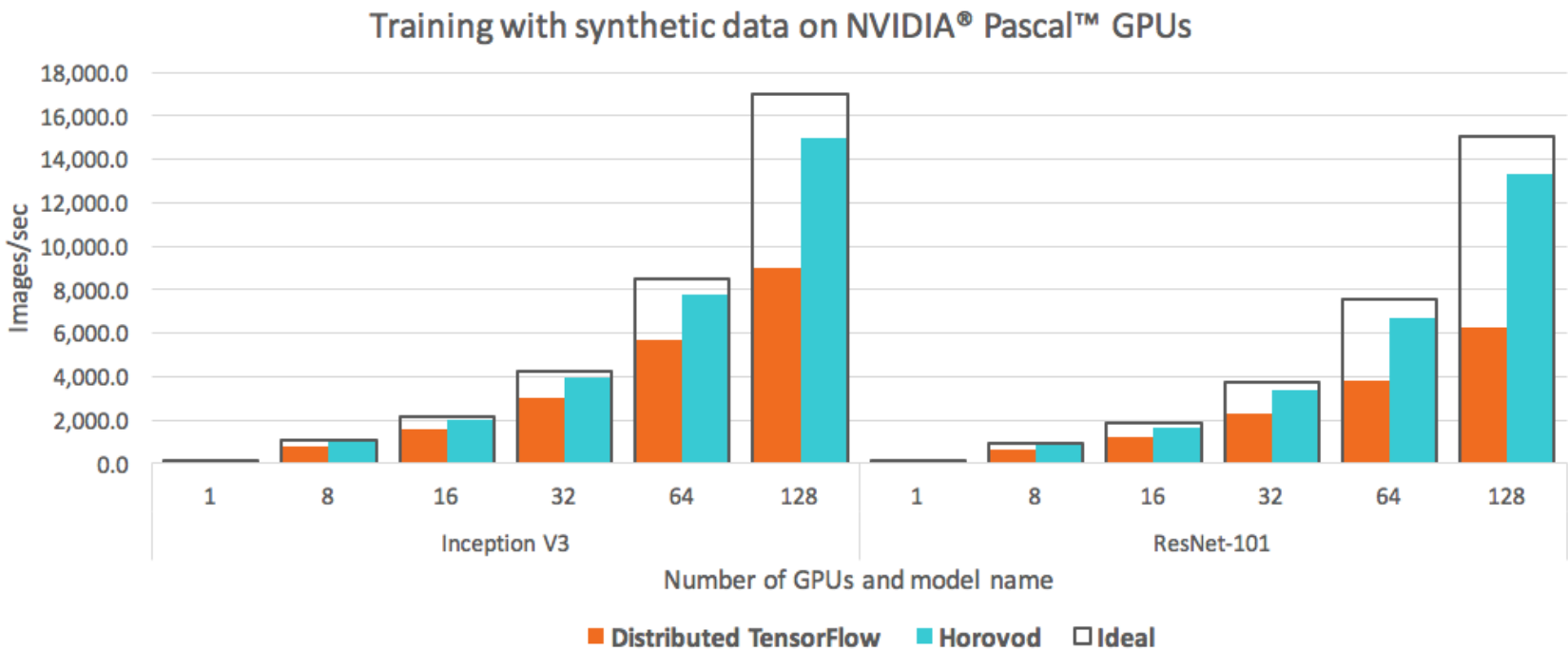
Popularized by Horovod [Sergeev and Del Balso 2018] (Linux Foundation AI)



Pros: mature implementation (2018–), bandwidth optimality

Cons: total latency grows linearly

Distributed Tensorflow vs Horovod

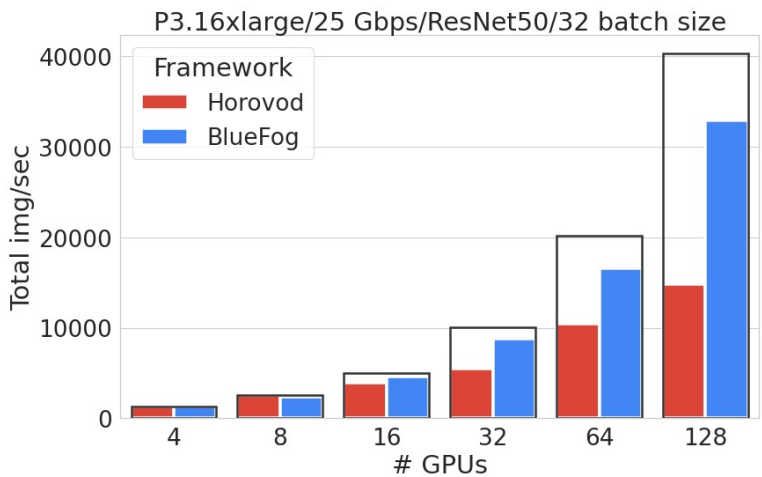


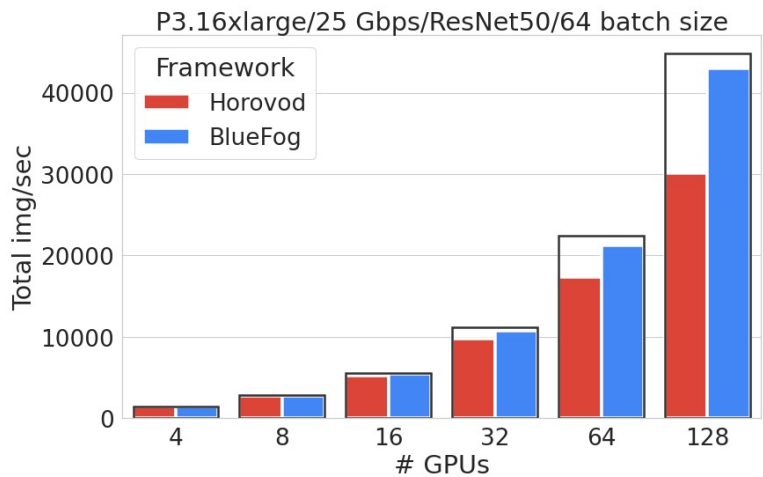
Result is from Horovod GitHub homepage.

2018 ACM Gordon Bell Prize

- Awarded to NERSC-led team at ORNL and LBNL
- Exascale deep learning for climate analysis
- Running **Horovod** over 27k+ V100 GPUs, achieving 90.7% scaling efficiency, 1.13 exaflops peak

- Communication framework for PyTorch
- Just a few lines of Python
- Supports MPI and NCCL
- Higher throughput than Hovovod





Exact vs approximate SGD

Data-parallel formulation: Let D_i be agent i's local training data,

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^{n} \mathbb{E}_{\xi_i \sim D_i} F(x; \xi_i).$$

Mini-batch SGD: Let B_i^k be the mini-batch of agent i at iteration k,

$$x^{k+1} = x^k - \frac{\alpha^k}{n} \sum_{i=1}^{n} \underbrace{\frac{1}{|B_i^k|} \sum_{\xi_i \in B_i^k} \nabla F(x^k; \xi_i)}_{\text{mini-batch grad at } i}.$$

Neighbor-averaging SGD [Cattivelli et.al. 2008, Nedic and Ozdaglar 2009]: Let x_i be agent i's local copy, W be a weight matrix, for $i = 1, \ldots, n$,

$$x_i^{k+1} = \sum_{j=1}^{n} W_{ij} \left(x_j^k - \alpha^k (\text{mini-batch grad at } j) \right).$$

Goal: design W_{ij} that leads to cheap communication and very close approximation to mini-batch SGD. (No actual matrix-vector multiplication needed.)

Weight matrix W

Given $y_1, \ldots, y_n$ from n different nodes, return $x_i = \sum_j W_{ij} y_j$ to node i; write this as

$$\mathbf{x} = W\mathbf{y} = W \begin{bmatrix} — & y_1^T & — \\ & \cdots & \\ — & y_n^T & — \end{bmatrix}.$$

Sparser $W \Rightarrow$ less (thus faster) communication.

Smaller $\rho := \|W - \frac{1}{n}\mathbf{1}\mathbf{1}^T\| \Rightarrow$ better approximation to exact averaging.

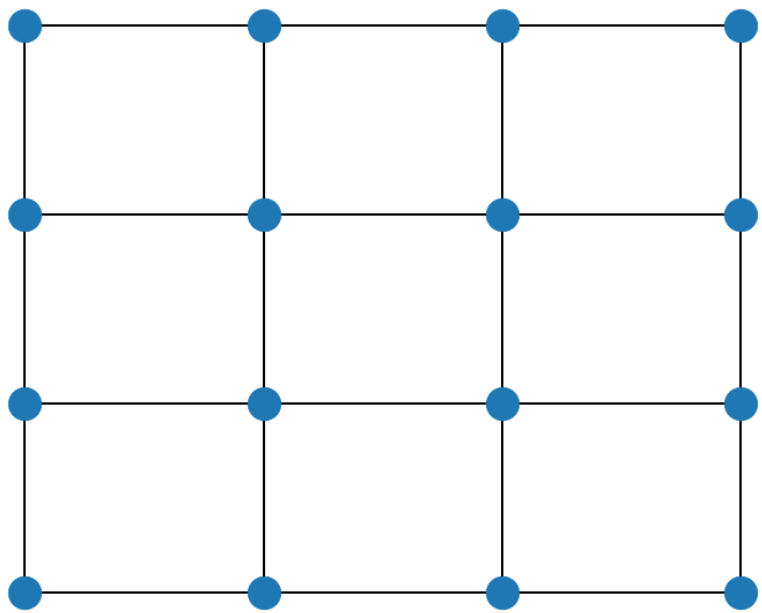
We also require: $W\mathbf{1} = \mathbf{1}$, $\mathbf{1}^T W = \mathbf{1}^T$, and W has eigenvalues:

$$\lambda_1 = 1 > |\lambda_2| \geq \cdots \geq |\lambda_n| > -1.$$

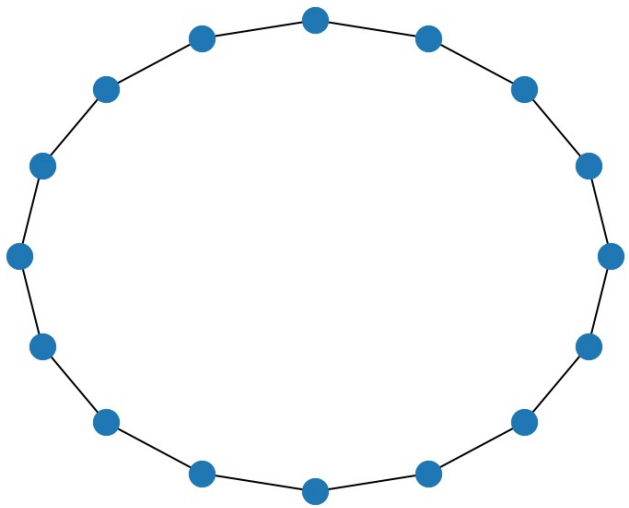
We have $\rho = \max(|\lambda_2|, |\lambda_n|)$.

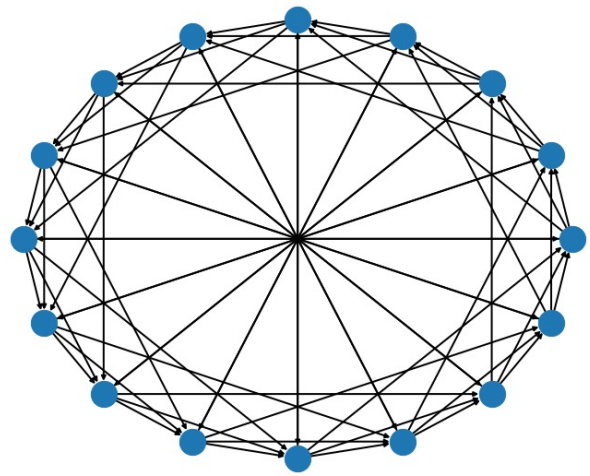
Examples

- $W = \frac{1}{n}\mathbf{1}\mathbf{1}^T$ has $\rho = 0$, but every communicates from and to all other nodes; worst choice!
- Grid $W = \mathrm{Conv2D}\left(\begin{bmatrix} & 1/5 & \\ 1/5 & 1/5 & 1/5 \\ & 1/5 & \end{bmatrix}\right)$ has $\rho \approx 0.868$. Every node connects to four other nodes. But ρ approaches 1 quickly as n increases. Poor global information mixing.



- **Left:** bilateral ring $W = \text{circ}(1/3, 1/3, 1/3, \dots)$ has $\rho = \frac{1}{3} + \frac{2}{3}\cos(2\pi/n)$. Every node connects directly to two other nodes. Poor global information mixing.





- **Right:** exp2 ring W has $\rho = 1 - 2/(2 + \lfloor \log_2(n-1) \rfloor)$ for even n. Every node connects to $\lfloor \log_2(n-1) \rfloor$ other nodes. Both sparse and good mixing!

Fixed vs dynamic neighbor averaging

Fixed Neighbor-averaging SGD:

$$x_i^{k+1} = \sum_{j=1}^{n} W_{ij} \left(x_j^k - \alpha^k (\text{mini-batch grad at } j) \right).$$

Dynamic Neighbor-averaging SGD:

$$x_i^{k+1} = \sum_{j=1}^{n} W_{ij}^{(k)} \left(x_j^k - \alpha^k (\text{mini-batch grad at } j) \right).$$

Each round uses a different W.

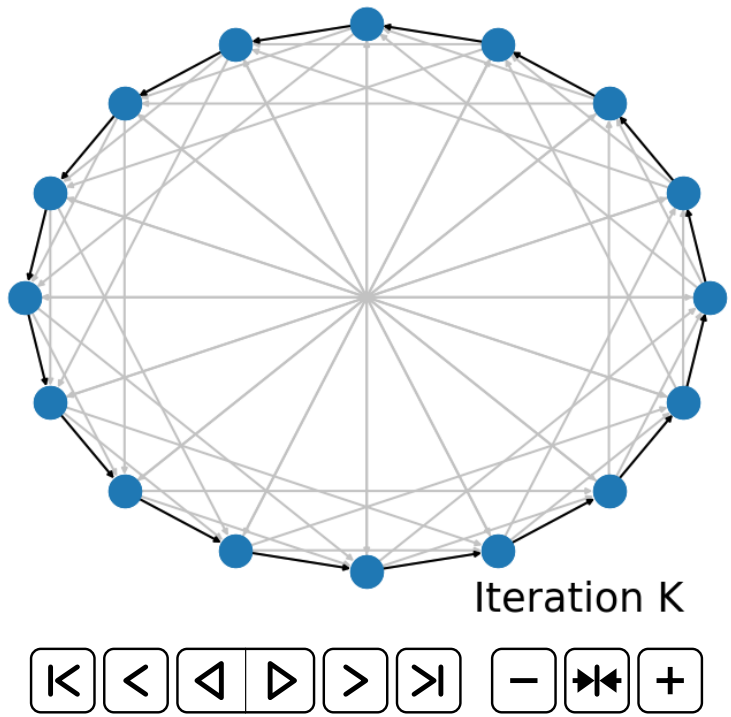
Further generalization:

1. if communication is faster, apply multiple W per mini-batch gradient
2. if communication is slower, apply multiple mini-batch gradients per W

For simplicity, assume one W per mini-batch gradient

Dynamic exp2-ring [Assran et.al. 2019]

Take $n = 16$ for example. Break a 16-node exp2-graph into four subgraphs. To each subgraph, assign a unique W with weights $1/2, 1/2$ for the active nodes.



In every subgraph, every node communicates one other node. Computing $W\mathbf{y}$ takes $O(1)$ time.

8-node example

$$W^{(1)} = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ & & & \cdots & \cdots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.5 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ & & & \cdots & \cdots & & & \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

$$W^{(3)} = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ & & & \cdots & \cdots & & & \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0.5 \end{bmatrix}$$

Exact averaging achieved by finite dynamic neighbor averaging

Theorem: When $n = 2^\tau$ for $\tau \in \mathbb{Z}$, dynamic exp-2 averaging satisfies

$$W^{(\tau)} W^{(\tau-1)} \cdots W^{(1)} = \frac{1}{n} \mathbf{1}\mathbf{1}^T$$

Furthermore, for any $p = 2, \ldots, \tau$,

$$W^{(p-1)} \cdots W^{(1)} W^{(\tau)} \cdots W^{(p)} = \frac{1}{n} \mathbf{1}\mathbf{1}^T.$$

This W-sequence is communication optimal among all averaging matrices.

Higher throughput

Define: n nodes, M-sized message, B bandwidth, L latency.

	Bandwidth Cost	**Latency**	**Total Cost**
Parameter server	$O(nM/B)$	$O(L)$	$O(n+1)$
Ring allreduce	$O(2M/B)$	$O(2nL)$	$O(1+n)$
Static exp2 averaging	$O(\log(n)M/B)$	$O(\log(n)L)^2$	$\tilde{O}(1+1)$
Dynamic exp2 averaging	$O(M/B)$	$O(L)$	$O(1+1)$

Neighbor averaging is **much cheaper** than any allreduce per round.

[2] Assume no conflict or racing when receiving messages from neighbors.

Training convergence rate

Let: σ^2 be variance of gradient noise

	Rate for non-convex loss, iid data
Allreduce SGD	$O\left(\frac{\sigma}{\sqrt{nT}} + \frac{1}{T}\right)$
Neighbor-averaging SGD[3]	$O\left(\frac{\sigma}{\sqrt{nT}} + \frac{\sigma^{2/3}\rho^{2/3}}{T^{2/3}(1-\rho)^{1/3}} + \frac{1}{(1-\rho)T}\right)$

[3] First proved in [Lian et al. 2017] and later improved in [Koloskova et al. 2020]

Large-scale training for image classification

- Model: ResNet-50 ($\sim$25.5M parameters)
- Dataset: ImageNet-1K (1000 classes)
- Size: 1,281,167 training images and 50,000 validation images
- GPUs: 8×8

Method	Epochs/Hours to 76%.
Allreduce SGD	68 / 5.57
Neighbor-averaging SGD	76 / 4.23

Periodic allreduce [Chen et.al. 2020]

$$\boldsymbol{y}_i^{(k)} = \boldsymbol{x}_i^{(k)} - \gamma \nabla F_i(\boldsymbol{x}_i^{(k)}; \boldsymbol{\xi}_i^{(k+1)})$$

$$\boldsymbol{x}_i^{(k+1)} = \begin{cases} \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{y}_j^{(k)} & \text{If mod}(k+1, H) = 0 \\ \sum_j W_{ij} \boldsymbol{y}_j^{(k)} & \text{If mod}(k+1, H) \neq 0 \end{cases}$$

Selecting $H < \frac{1}{1-\rho}$ can provably accelerate Neighbor-averaging SGD.

Large-scale training for image classification

- Model: ResNet-50 ($\sim$25.5M parameters)
- Dataset: ImageNet-1K (1000 classes)
- Size: 1,281,167 training images and 50,000 validation images
- Hardware: 32×8 GPUs

Method	Epochs/Hours to 76%.
Allreduce SGD	94 / 1.74
Neighbor-averaging SGD	91 / 1.20

Large-scale BERT training for language modeling

- Model: BERT-Large ($\sim$330M parameters)
- Dataset: Wikipedia (2500M words) and BookCorpus (800M words)
- Hardware: 8×8 GPUs

Method	Final Loss	Wall-clock Time (hrs)
Allreduce SGD	1.75	59.02
Neighbor-averaging SGD SGD	1.77	30.4

How to use BlueFog

DNN example

BlueFog has a high-level API that wraps around any torch optimizer.

Example:

```
import torch
import bluefog.torch as bf
bf.init()
...
optimizer = optim.SGD(model.parameters(), lr=lr*bf.size())
optimizer = bf.DistributedNeighborAllreduceOptimzer( \
  optimizer, model=model)
...
# Torch training code
```

BlueFog also provides optimizers: Distributed Allreduce, Distributed Hierarchical Neighbor Allreduce, etc.

SPMD (single program, multiple data)

One code for all nodes; different nodes have different data and unique ranks.

```python
# hello_world.py
import bluefog.torch as bf
bf.init()
print("I am rank {} in size {}".format(bf.rank(), bf.size()))
```

```
> bfrun -np 2 python hello_world.py

I am rank 1 in size 2
I am rank 0 in size 2
```

Neighbor averaging

Example: compute the average of ranks of the nodes

```
import torch
import bluefog.torch as bf
bf.init()

x = torch.Tensor([bf.rank()])

for _ in range(100):
    x = bf.neighbor_allreduce(x)
print("rank {} has x={}".format(bf.rank(), x))
```

Defaults:

- `bf.init()` creates a static exp2 graph
- neighbor-averaging weights are set to $\frac{1}{\text{neighbors}+1}$ for every incoming neighbors and the node itself

```
> bfrun -np 10 python neighbor_avg.py

rank 0 has x=tensor([4.5000])
rank 3 has x=tensor([4.5000])
rank 9 has x=tensor([4.5000])
rank 1 has x=tensor([4.5000])
rank 7 has x=tensor([4.5000])
rank 4 has x=tensor([4.5000])
rank 2 has x=tensor([4.5000])
rank 6 has x=tensor([4.5000])
rank 5 has x=tensor([4.5000])
rank 6 has x=tensor([4.5000])
```

Neighbor averaging using dynamic subgraphs

Example: Default dynamic exp2 averaging

```
dynamic_neighbors = topology_util.GetDynamicSendRecvRanks(
         bf.load_topology(), bf.rank())

for _ in range(maxite):
   to_neighbors, from_neighbors = next(dynamic_neighbors)

   avg_weight = 1/(len(from_neighbors) + 1)

   xi = bf.neighbor_allreduce(xi, name='x',
      self_weight=avg_weight,
      neighbor_weights={r: avg_weight for r in from_neighbors},
      send_neighbors=to_neighbors)
```

You can replace `GetDynamicSendRecvRanks()` with your own.

Decentralized gradient descent [Nedic and Ozdaglar 2009]

To approximate solve

$$\underset{\mathbf{x}}{\text{minimize}} \ \alpha \sum_{i=1}^{n} f_i(x_i) \qquad \text{subject to } x_1 = \cdots = x_n,$$

we can apply *decentralized gradient descent*:

$$\mathbf{x}^{k+1} = W\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k).$$

Implementation using static exp2:

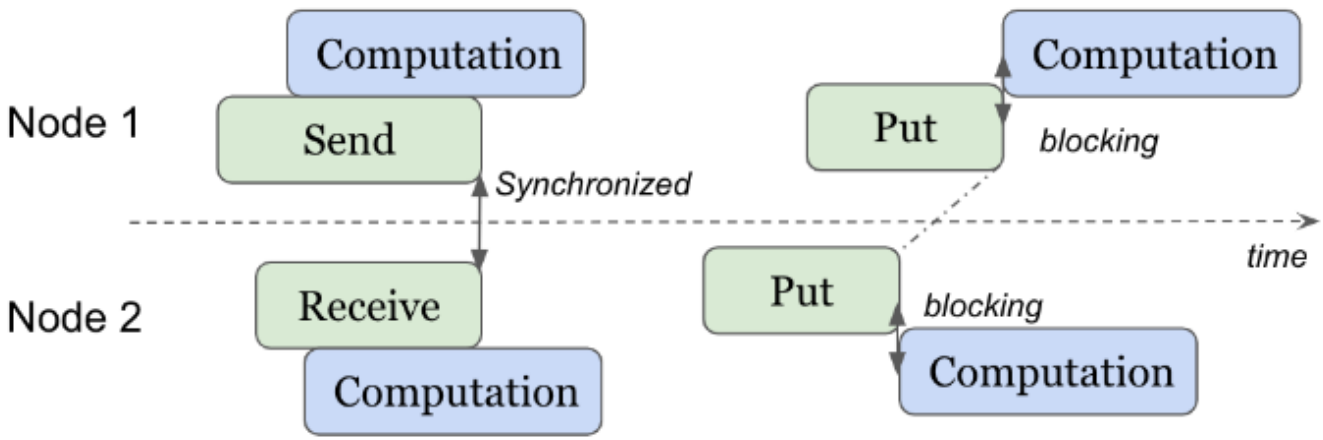
```
# DGD recursion
for k in range(maxite):
   xi = bf.neighbor_allreduce(xi) - alpha*ComputeGrad(fi,xi)
```

Blocking and asynchrony

Each node has two threads: communication thread and computation thread

- **non-blocking:** allow concurrent threads to save time
- blocking: computation starts after communication completes

Synchronization is similar concept but applies to operations across different nodes. All collective communications are synchronous.



Left: nonblocking but synchronized; **Right:** blocking, may or may not sync'd

By default, BlueFog is blocking and synchronized, but it also supports non-blocking and asynchronous operations

To save time, we ask neighbor allreduce $W\mathbf{x}^k$ not to block computation $\nabla f(\mathbf{x}^k)$, so they can run concurrently.

```
1  for k in range(maxite):
2      handle = bf.neighbor_allreduce_nonblocking(xi)
3      gradi = ComputeGrad(fi, xi)
4      avg_x = bf.wait(handle)
5      xi = avg_x - alpha*gradi
```

Since Line 5 must wait for the result of $W\mathbf{x}^k$.

EXTRA [Shi et.al. 2015]

EXTRA was the first method that solves

$$\underset{x}{\text{minimize}} \sum_{i=1}^{n} f_i(x_i) \qquad \text{subject to } x_1 = \cdots = x_n$$

with a constant α. One form of this method is

$$\begin{cases} \mathbf{x}^1 = W\mathbf{x}^0 - \alpha\nabla f(\mathbf{x}^0), \\ \mathbf{x}^{k+1} = W(2\mathbf{x}^k - \mathbf{x}^{k-1}) - \alpha(\nabla f(\mathbf{x}^k) - \nabla f(\mathbf{x}^{k-1})), \quad k = 1, 2, \cdots \end{cases}$$

The code structure is similar to DGD. Non-blocking communication can accelerate the code.

Tracking

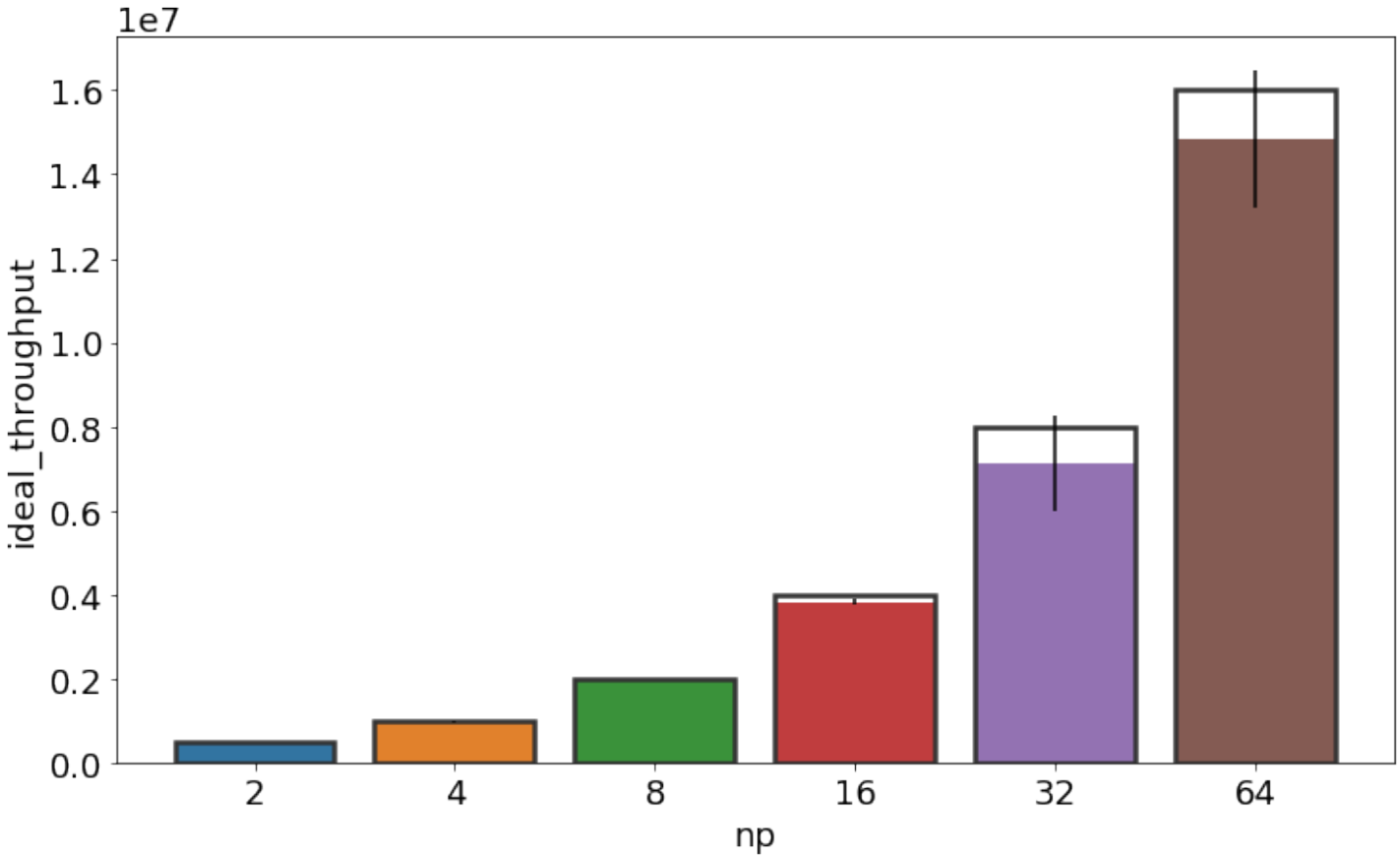
DIGing [Nedic et.al. 2017] is a tracking-based method. For static W, DIGing is a special case of EXTRA. However, DIGing works for dynamic W.

$$\begin{cases} \mathbf{x}^{k+1} = W^{(k)}\mathbf{x}^k - \alpha \mathbf{y}^k \\ \mathbf{y}^{k+1} = W^{(k)}\mathbf{y}^k + \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k) \end{cases}$$

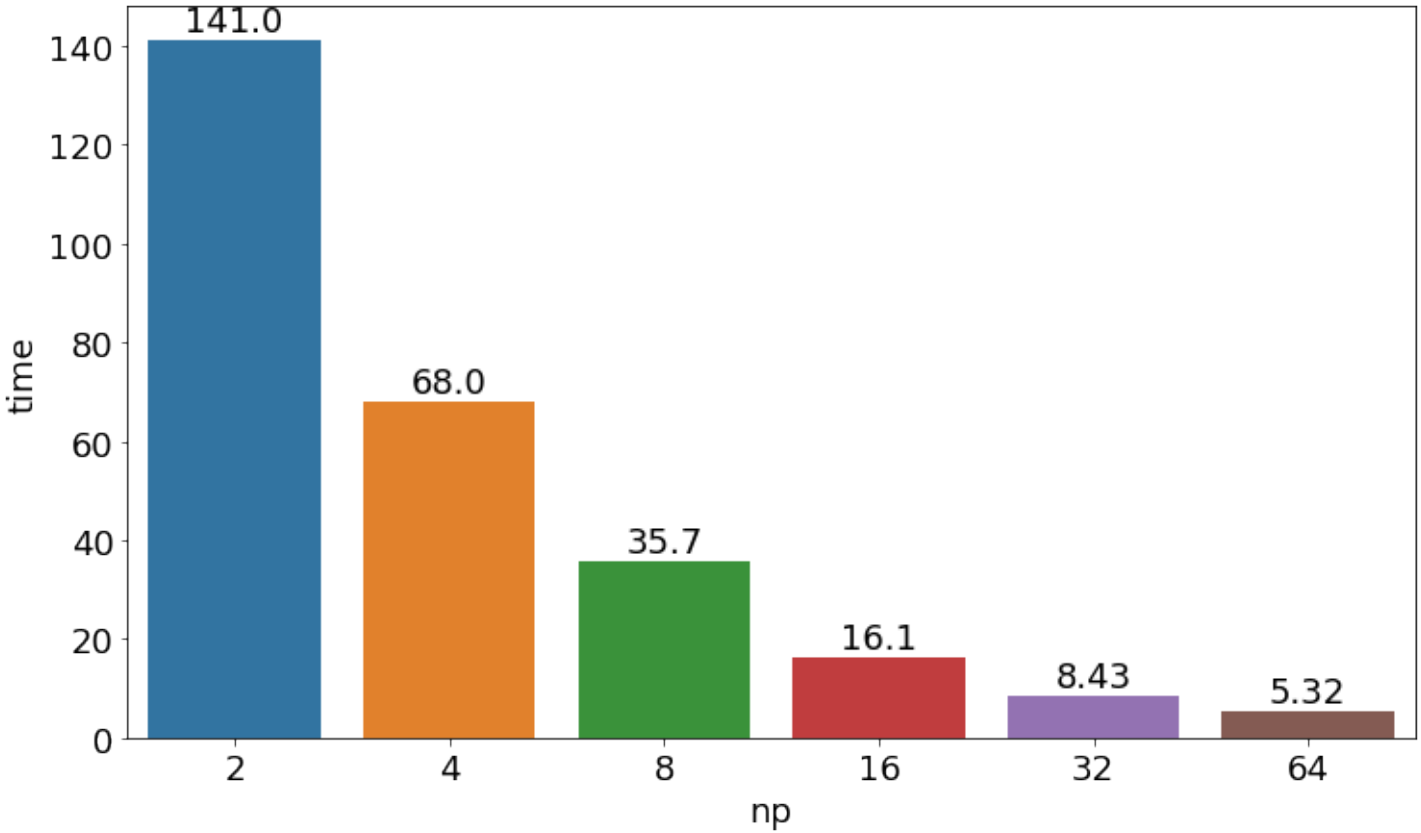
$(\mathbf{y}^k)_k$ a tracking sequence converging to $\lim_k \frac{1}{n}\sum_{i=1}^n \nabla f_i(\mathbf{x}^k)$ if it exists.

```
xi = np.zeros((d,1))
yi = fi_grad_prev = ComputeGrad(fi, xi)
for k in range(maxite):
    self_weight, recv_weights = ComputeWeights(k, bf.rank())
    xi = bf.neighbor_allreduce(xi, self_weight, recv_weights) \
        - alpha*yi
    gi = ComputeGrad(fi, xi)
    yi = bf.neighbor_allreduce(gi, self_weight, recv_weights) \
        + gi - gi_prev
    gi_prev = gi.copy()
```

Linear speedup in throughput on CPU



Linear speedup in running time on CPU



Availability

Open source at `https://github.com/Bluefog-Lib/bluefog`

Contributors: Bicheng Ying, Kun Yuan, Hanbin Hu, Ji Liu, Wotao Yin

Thank you!

References

[Li et.al. 2014] M. Li, et al. "Scaling Distributed Machine Learning with the Parameter Server", OSDI, 2014.

[Patarasuk and Yuan 2009] P. Patarasuk, and X. Yuan, "Bandwidth Optimal All-reduce Algorithms for Clusters of Workstations", Journal of Parallel and Distributed Computing, 2009.

[Gibiansky 2017] A. Gibiansky, "Bringing HPC Techniques to Deep Learning", 2017.

[Sergeev and Del Balso 2018] A. Sergeev, and M. Del Balso, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow", arXiv:1802.05799, 2018.

[Cattivelli et.al. 2008] F. Cattivelli, C. G. Lopes, and A. H. Sayed, "Diffusion Recursive Least-Squares for Distributed Estimation over Adaptive Networks", IEEE TSP, 2008.

[Nedic and Ozdaglar 2009] A. Nedic, and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization", IEEE TAC, 2009.

References

[Assran et.al. 2019] M. Assran, et. al. "Stochastic Gradient Push for Distributed Deep Learning", ICML, 2019.

[Lian et.al. 2017] X. Lian, et al. "Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent", Neurips 2017.

[Koloskova et.al. 2020] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, "A Unified Theory of Decentralized SGD with Changing Topology and Local Updates", ICML 2020.

[Chen et.al. 2020] Y. Chen, et. al., "Accelerating Gossip SGD with Periodic Global Averaging", 2020.

[Shi et.al. 2015] W. Shi, et. al., "EXTRA: An Exact First-Order Algorithm for Decentralized Consensus Optimization", SIAM J. Opt, 2015.

[Nedic et.al. 2017] A., Nedic, A. Olshevsky, and W. Shi, "Achieving Geometric Convergence for Distributed Optimization over Time-varying Graphs." SIAM J. Opt, 2017.