On the Time Complexity of Common All-Reduce Methods

Kun Yuan

(Internal Reference Only)

September 19, 2019

Outline

1. Communication between machines

2. Distributed deep learning

3. Time complexity for each allreduce method

Outline

1. Communication between machines

2. Distributed deep learning

3. Time complexity for each allreduce method

Communication between two machines

Three major factors that affect the communication rate:

- PCI express (PCIe) throughput: 1 GB/s $\times\, 8$
- Network Interface Card (NIC) throughput: 125 MB/s
- Network Bandwidth (NB): 100 MB/s ~ 1 GB/s

$$\boxed{\text{Communication rate} = \min\{\text{PCIe, NIC, NB}\}}$$

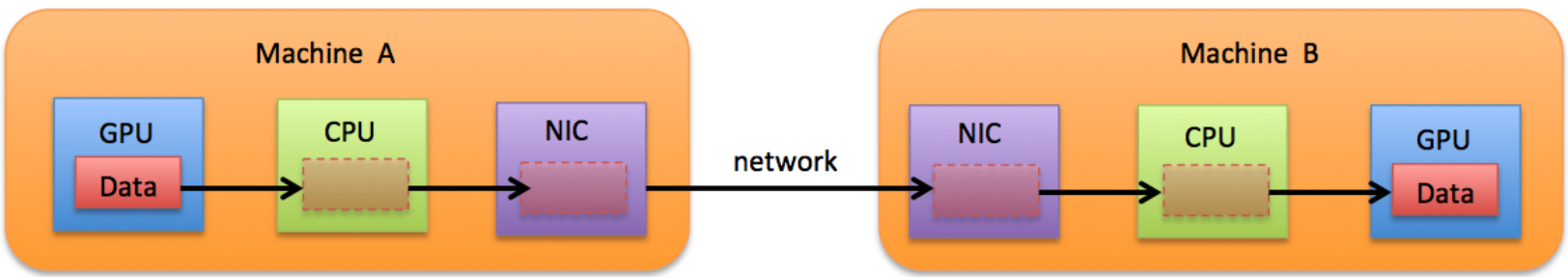

Figure: A simplified communication illustration between different machines.

Communication between two machines

- Each of the three factors can be enhanced with more advanced hardware
  - PCIe: version 1.0 $\sim$ version 4.0; RDMA
  - NIC: Infiniband
  - Network bandwidth: 100 Gigabit Ethernet
- However, such enhancement is far less enough for modern deep learning
- Efficient communication is fundamental to efficient training
- In this talk, we review efficient communication from the alg. perspective

Outline

1. Communication between machines

2. Distributed deep learning

3. Time complexity for each allreduce method

Empirical risk minimization

- The empirical risk minimization (ERM) problem is

$$\min_{w} \; F(w) = \frac{1}{N} \sum_{n=1}^{N} Q(w; x_n)$$

where $\{x_n\}_{n=1}^{N}$ is the dataset; $Q(w; x_n)$ is some cost function

- The above ERM problem can be rewritten as

$$\min_{w} \; F(w) = \frac{1}{P} \sum_{p=1}^{P} F_p(w), \quad \text{where} \quad F_p(w) = \frac{1}{L} \sum_{n=1}^{L} Q(w; x_{p,n})$$

where $\{x_{p,n}\}_{n=1}^{L}$ is the data assigned to node p, and $N = LP$.

Stochastic gradient descent

- Each node computes an approximate gradient for $\nabla F_p(w)$, i.e., $\widehat{\nabla F_p}(w)$:

$$w^{k+1} = w^k - \frac{\alpha}{P} \sum_{p=1}^{P} \widehat{\nabla F_p}(w^k),$$

$$\text{where} \quad \widehat{\nabla F_p}(w^k) = \frac{1}{B} \sum_{n \in \mathcal{B}} \nabla Q(w^k; x_{p,n})$$

  where $\mathcal{B}$ is a small batch of index, and $B = |\mathcal{B}|$.

- SGD is a standard training algorithm for deep learning.

- More advanced training methods: ADAM and its variants.

Communication bottleneck: allreduce

- The core component in distributed SGD is to compute the average

$$\widehat{\nabla F}(w^k) = \frac{1}{P}\sum_{p=1}^{P} \widehat{\nabla F_p}(w^k)$$

  and send it to each local machine.

- This step is called "allreduce" in High-Performance Computing (HPC).

- The time spent in allreduce decides how fast the distributed algorithm is (Given that the computation time of each $\widehat{\nabla F}_p(w^k)$ is fixed).

Major allreduce methods in literature

- There are four major allreduce methods in literature
  - Star-allreduce
  - Tree-allreduce
  - BytePS-allreduce
  - Ring-allreduce

- We will analyze their time complexities one by one.

Outline

1. Communication between machines

2. Distributed deep learning

3. Time complexity for each allreduce method

Basic assumptions

- The topology of the computing nodes is under control
- The gradient is of dimension M; each entry takes B bytes
- The communication time to transmit a unit byte is t_m

$$\boxed{t_m = \frac{1}{\min\{\text{PCIe, NIC, NB}\}}}$$

- The communication rate of each link is the same (regardless of their physical distance)
- The time to establish a connection is t_c
- We consider the synchronized scenario: all computations are finished before the allreduce starts

Star allreduce

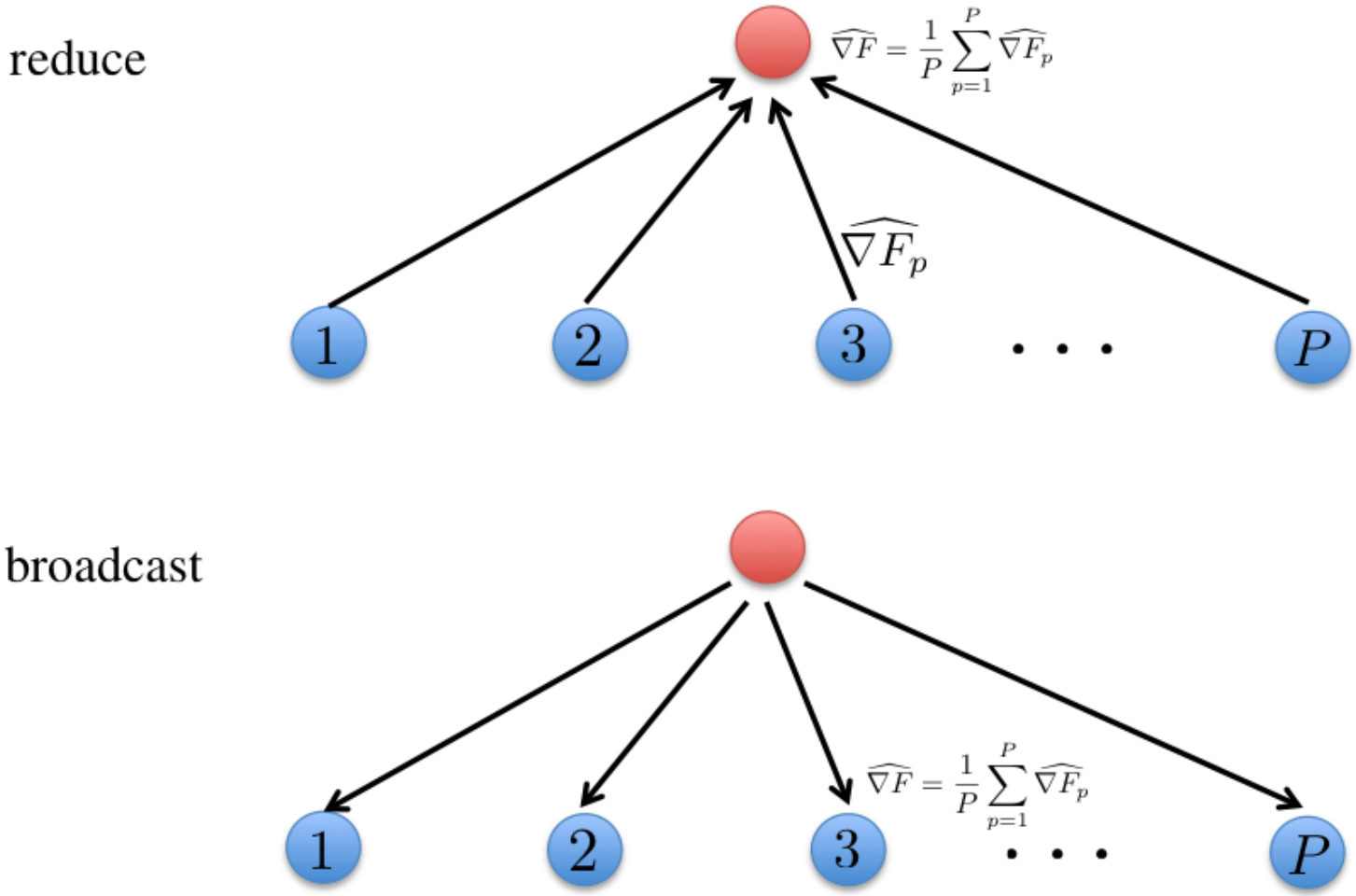

Figure: Illustration of the star allreduce.

Star allreduce

- The communication from workers to the master is not parallel
  - The NIC throughput is the bottleneck
  - The gradients have to be taken in one by one

- The time complexity for one star-allreduce step:
  - Time spent in comm. from worker p to master: $MBt_m + t_c$
  - There are P such communications in "reduce": $P(MBt_m + t_c)$
  - There are "reduce" and "broadcast": $2P(MBt_m + t_c)$

$$T_{\text{star}} = 2P(MBt_m + t_c)$$

- Since M is very large, it holds that $MBt_m \gg t_c$ which implies

$$\boxed{T_{\text{star}} = 2PMBt_m = O(P)}$$

Tree allreduce

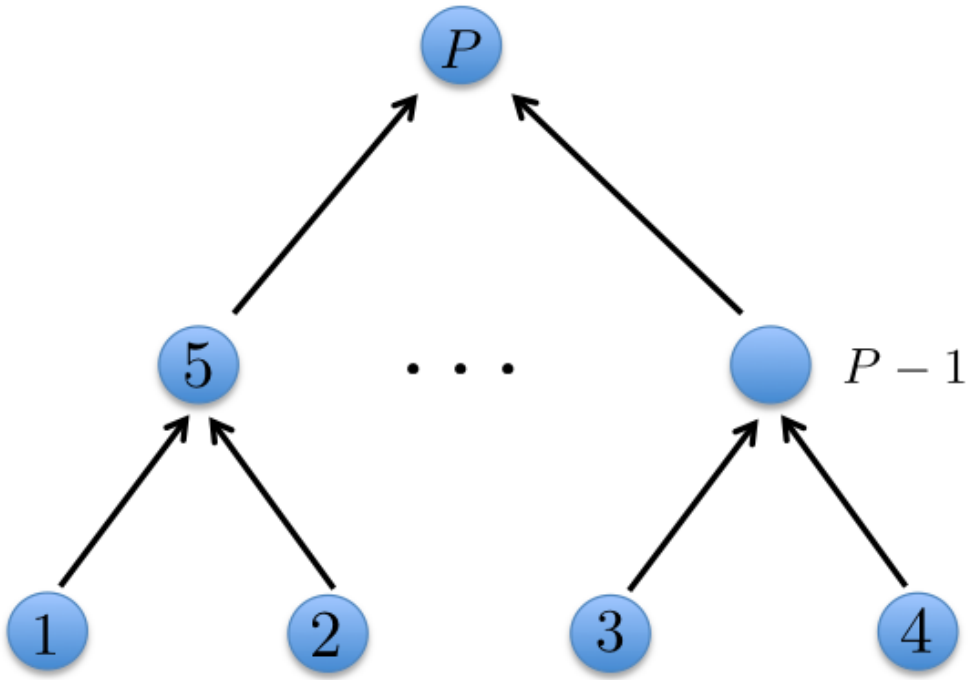

Figure: Illustration of the tree all-reduce.

Tree allreduce

- The time complexity for one tree-allreduce step:
  - Time spent in each level of the tree: $2MBt_m$ (we omit t_c)
  - There are $\log(P)$ levels in total
  - There is a "reduce" step and a "broadcast" step

$$T_{\text{tree}} = 4\log(P)MBt_m$$

- A smart implementation can further reduce the complexity as

$$\boxed{T_{\text{tree}} = 2\log(P)MBt_m = O\Big(\log(P)\Big)}$$

- Star or Tree allreduce is widely employed in Parameter Server

A smart tree allreduce

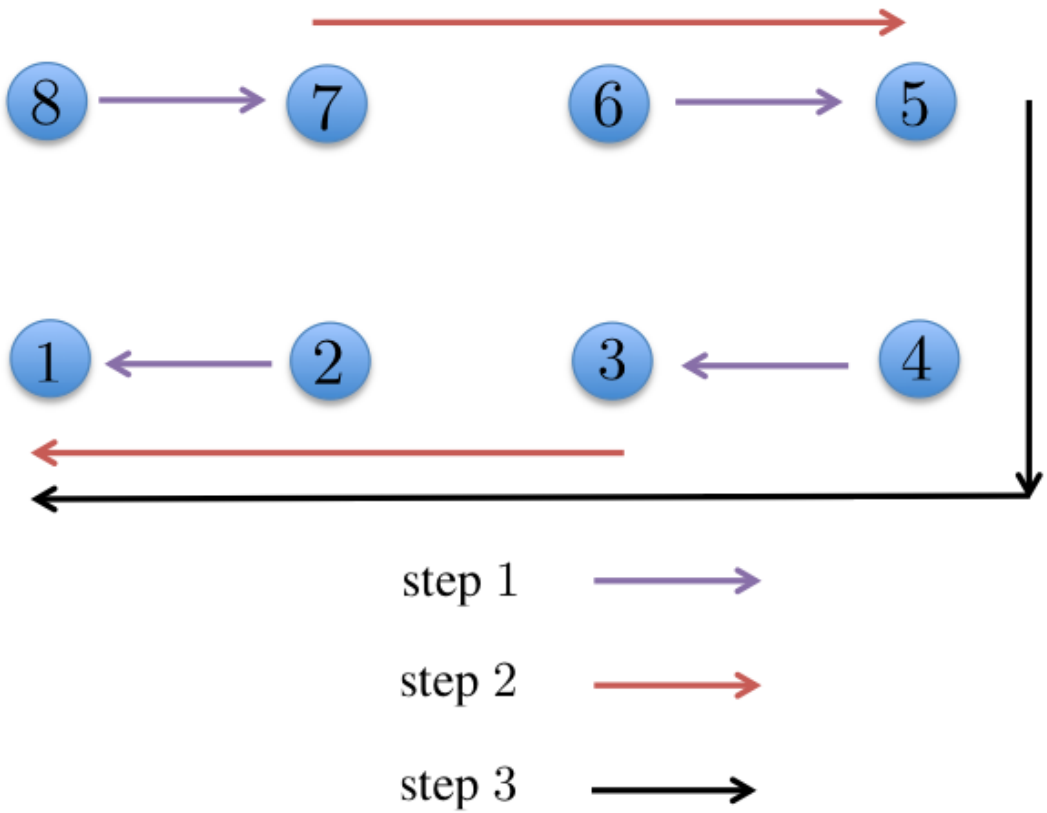

Figure: Illustration of a smart tree all-reduce. In the figure, all information are merged to node 1.

BytePS allreduce

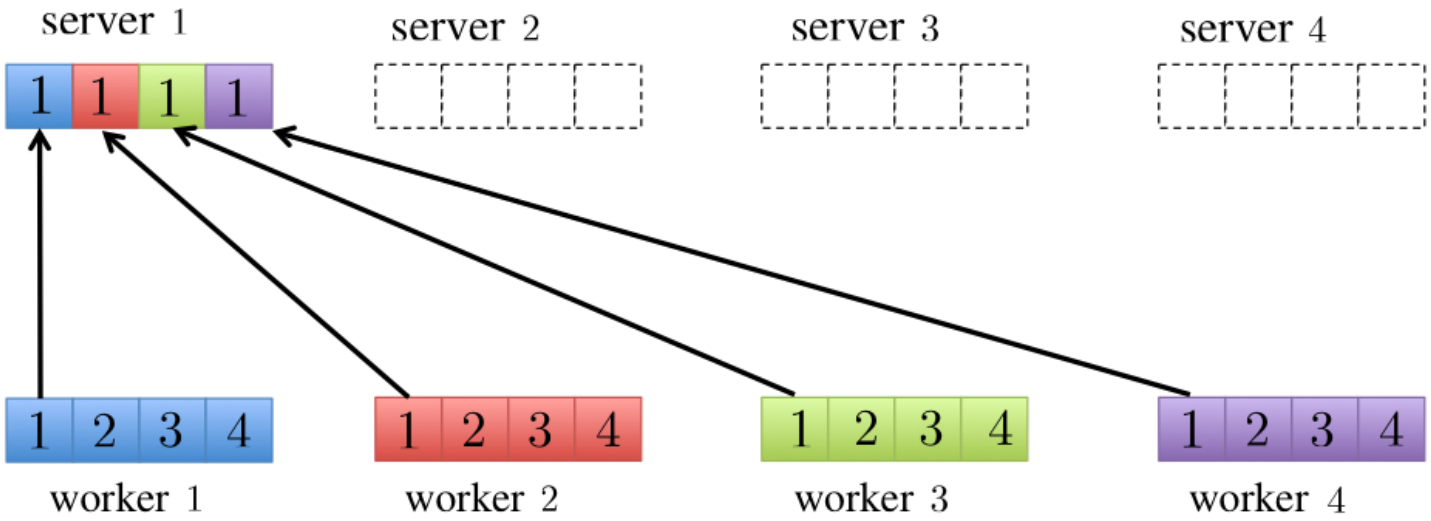

Figure: Each worker partitions its own data into P blocks and send its p-th block to server p. Note that P additional servers are required for this method. Since servers only perform the average operation, they can be cheap CPU servers. This figure illustrates that each worker is sending its 1st block to server 1.

BytePS allreduce

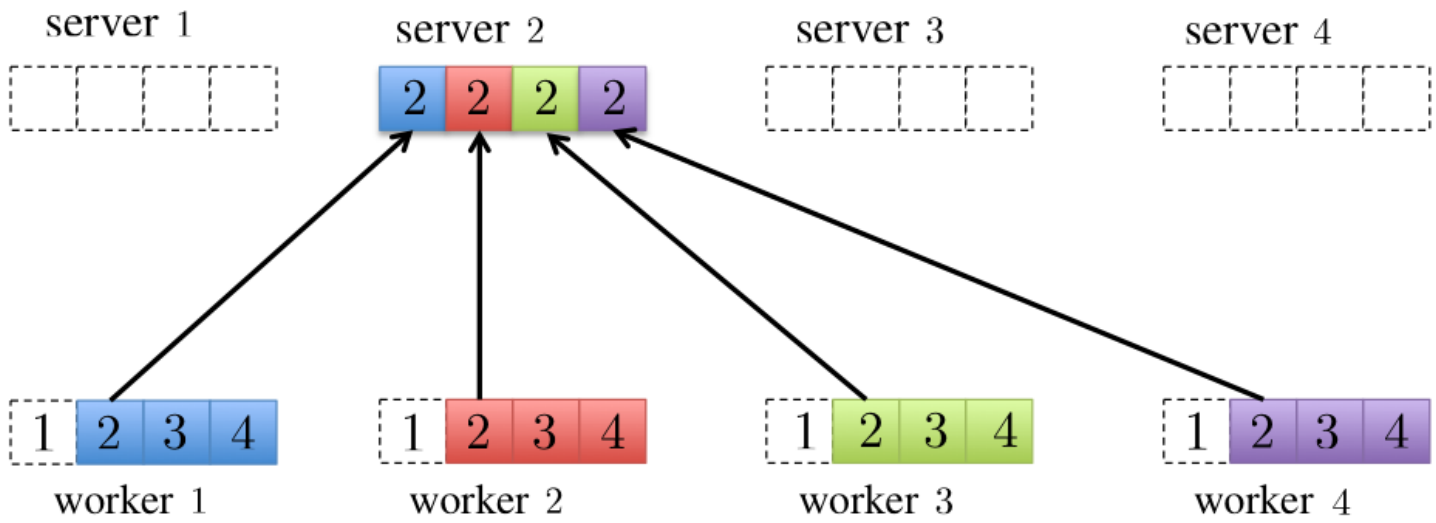

Figure: This figure illustrates that each worker is sending its 2ed block to server 2.

BytePS allreduce

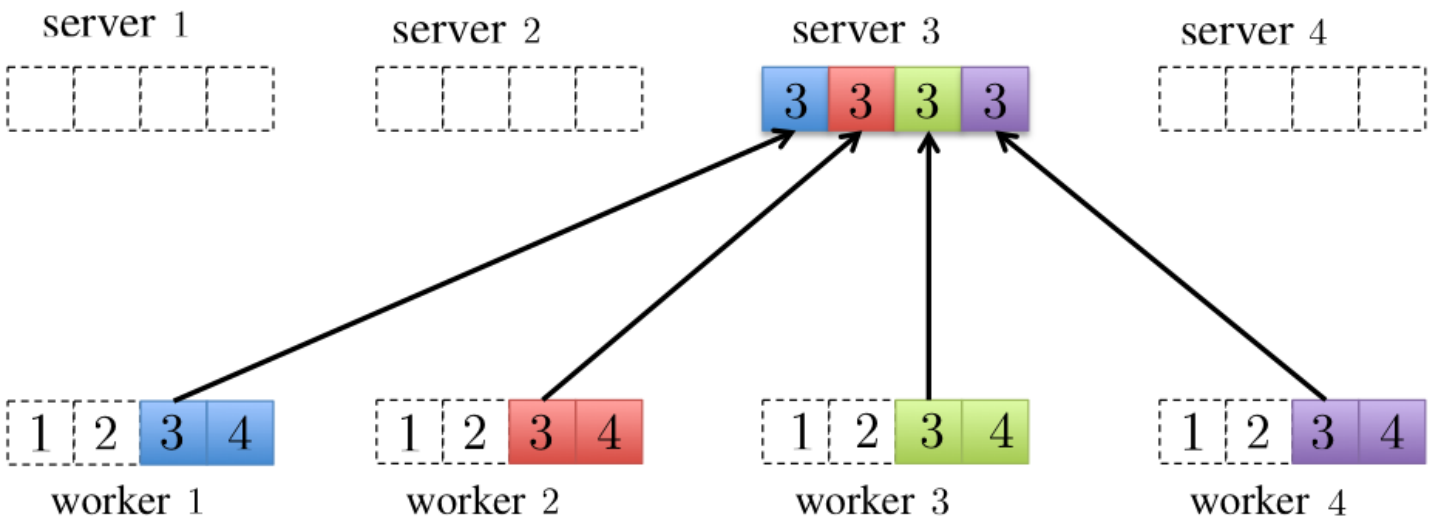

Figure: This figure illustrates that each worker is sending its 3rd block to server 3.

BytePS allreduce

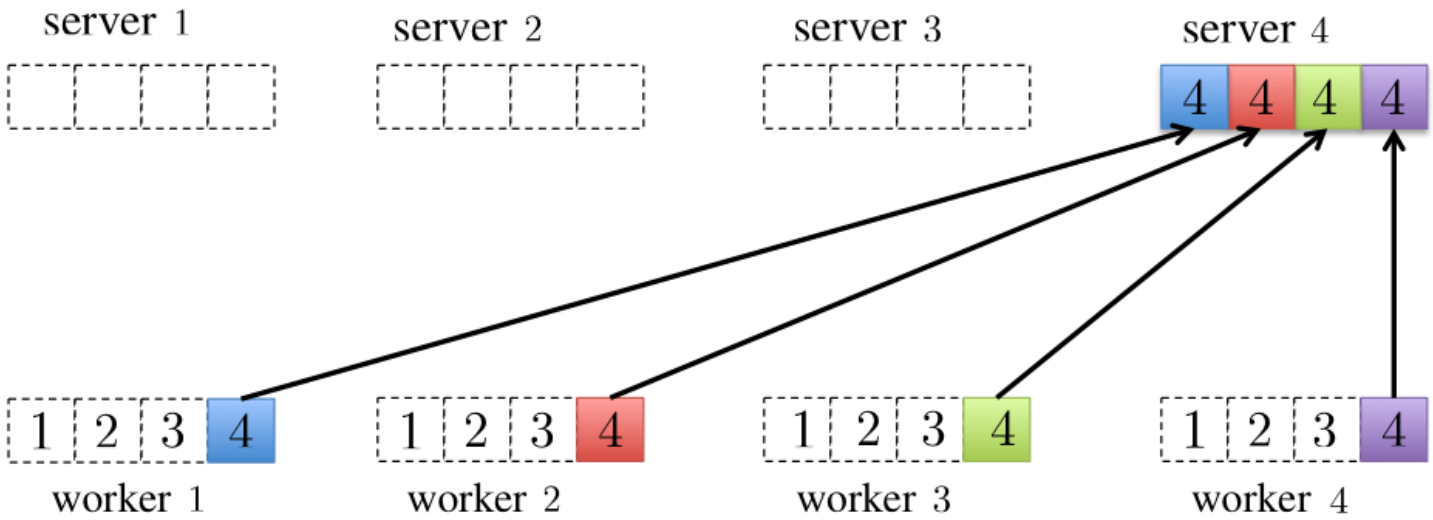

Figure: This figure illustrates that each worker is sending its 4th block to server 4. After this step, each server p collects the p-th block from all workers.

BytePS allreduce

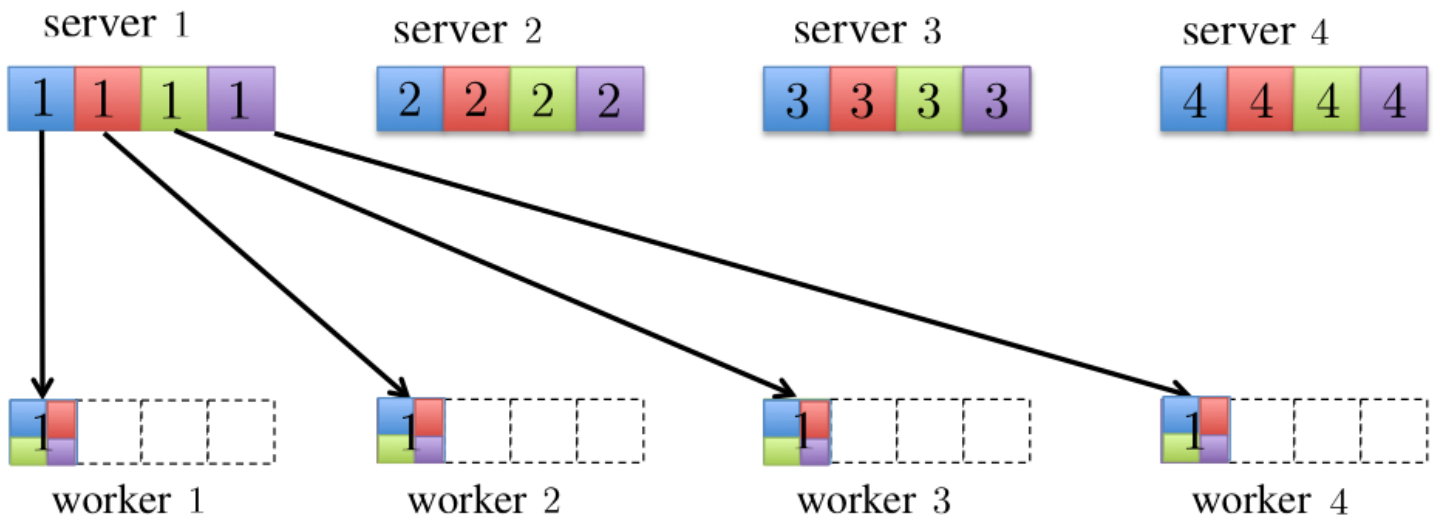

Figure: After the reduce step, server p reaches the average of the p-th block and sends it back to all workers. This figure illustrates that server 1 is sending the average of the 1st block to each worker.

BytePS allreduce

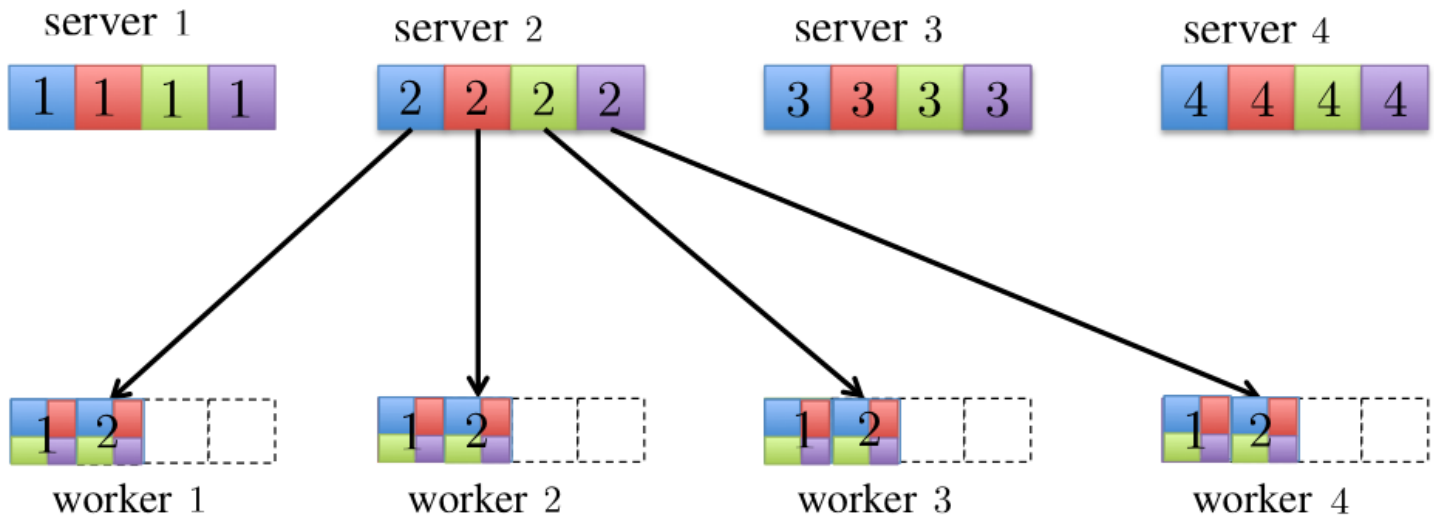

Figure: This figure illustrates that server 2 is sending the average of the 2ed block to each worker.

BytePS allreduce

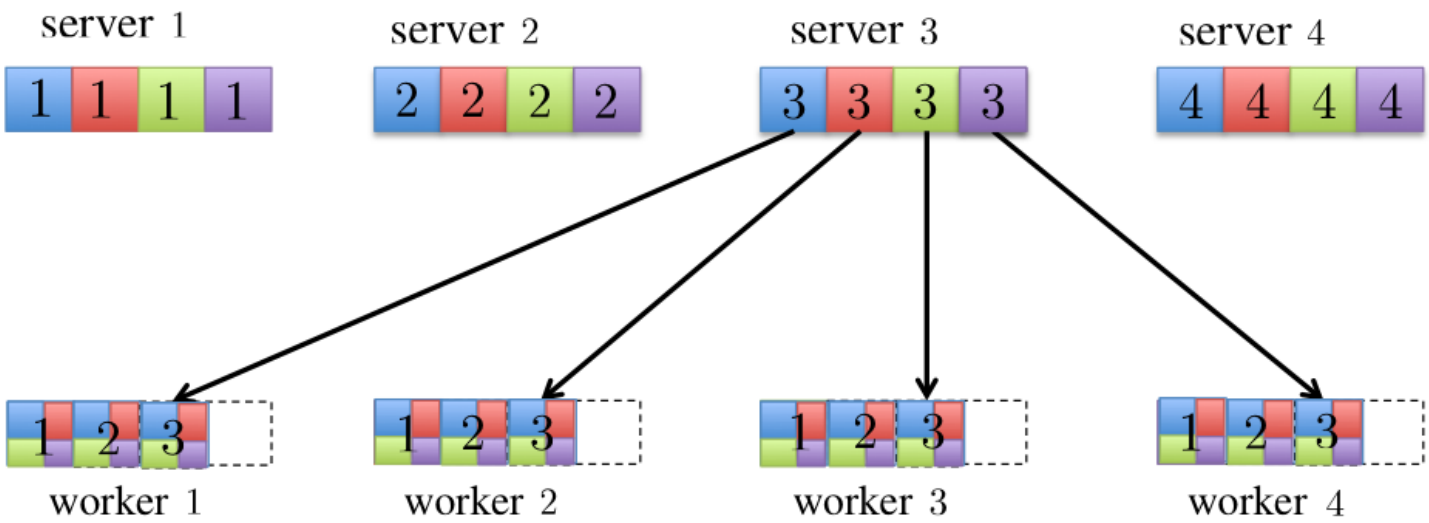

Figure: This figure illustrates that server 3 is sending the average of the 3rd block to each worker.

BytePS allreduce

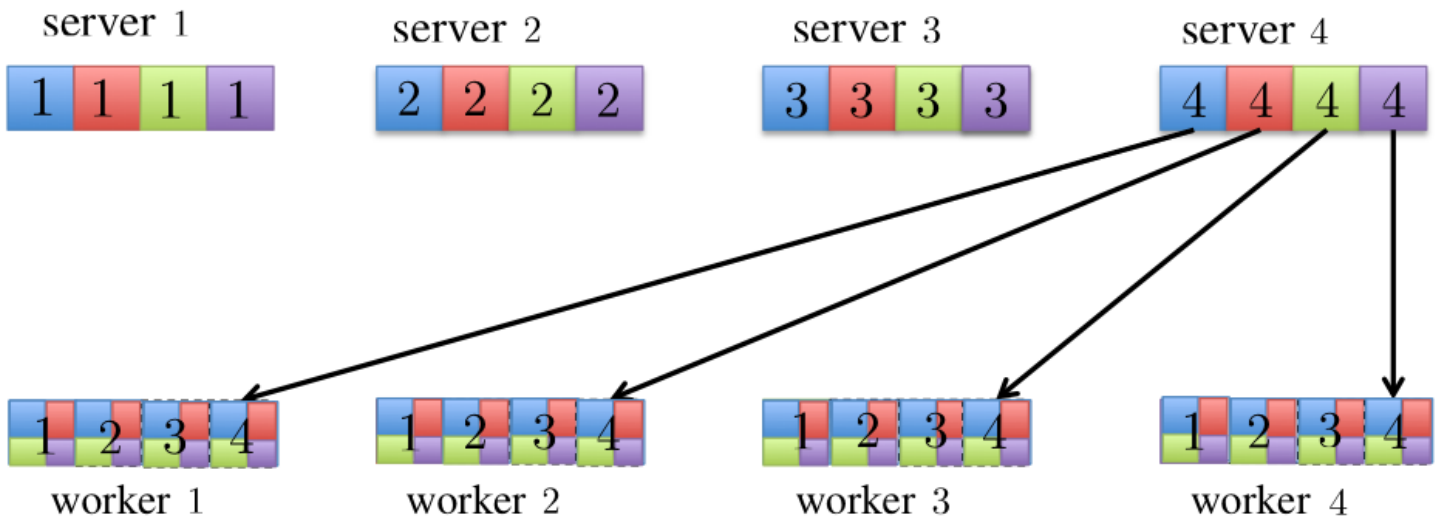

Figure: This figure illustrates that server 4 is sending the average of the 4th block to each worker. After this step, all workers get the copy of the global average of the gradients. The BytePS allreduce is finished

BytePS allreduce

- Step 1 – Step 4 can be conducted in parallel at the same time

$$\text{Time spent} = \frac{MB}{P} t_m \times P = MBt_m$$

- Step 5 – Step 8 can be conducted in parallel at the same time

$$\text{Time spent} = \frac{MB}{P} t_m \times P = MBt_m$$

- The total time complexity is

$$\boxed{T_{\text{BytePS}} = 2MBt_m = O(1)}$$

- Require P more (but cheap) servers
- BytePS is employed by Bytedance BytePS

Ring-allreduce

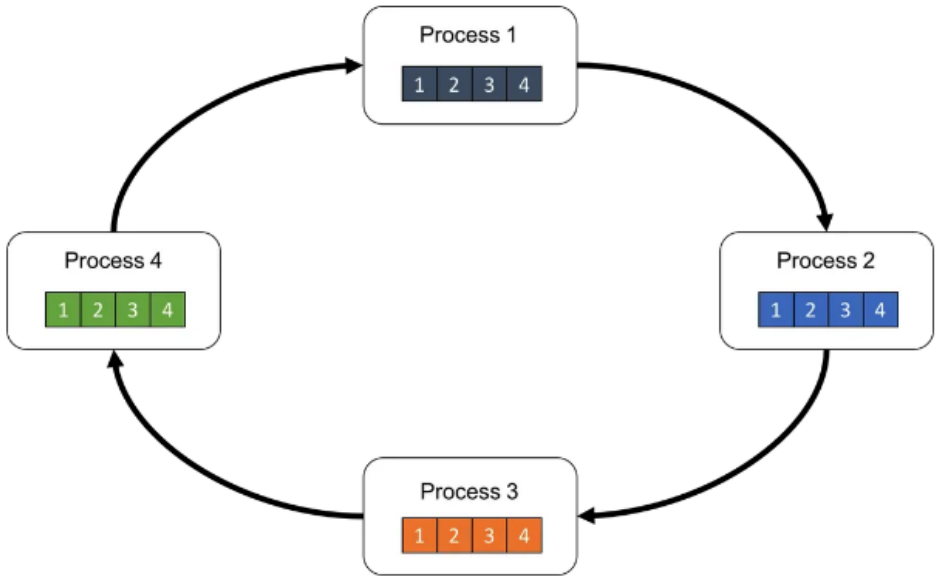

Fig.2 Example of a process ring

Figure: Illustration of the ring-allreduce. Each node partitions its data into P blocks[1].

[1] This figure is from the blog "Technologies behind Distributed Deep Learning: AllReduce"

Ring-allreduce

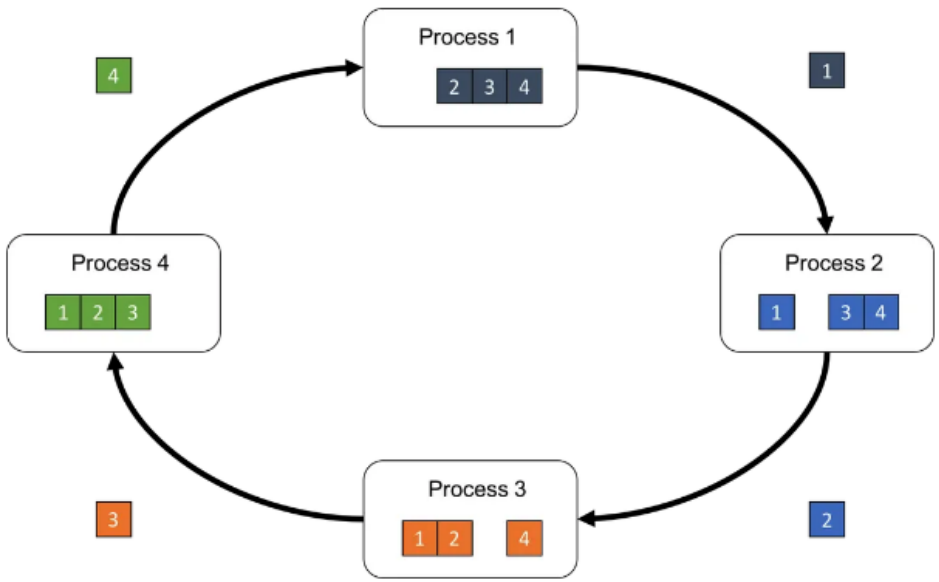

Figure: The p-th node sends its p-th block to the $(p+1)$-th node.

Ring-allreduce

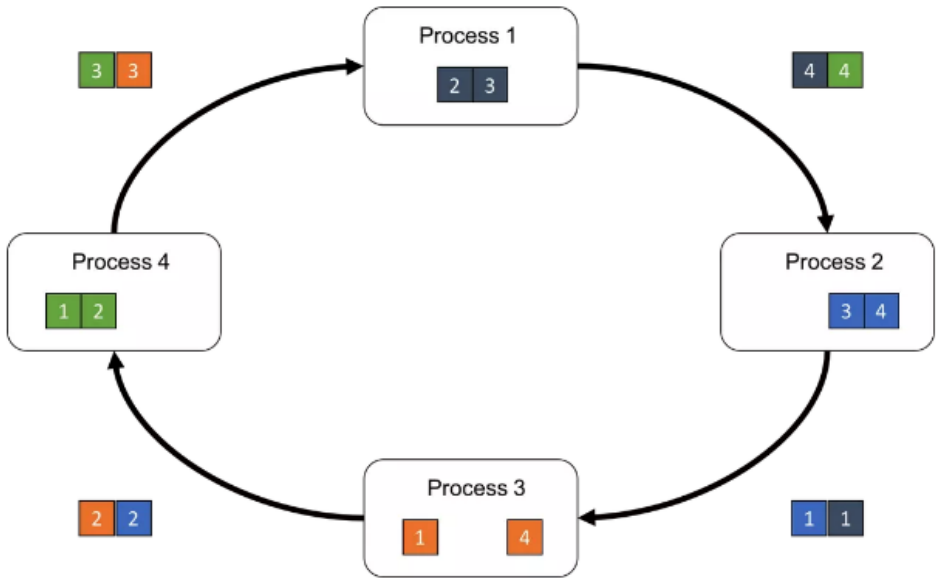

Figure: Each node sends a reduced block to the next node.

Ring-allreduce

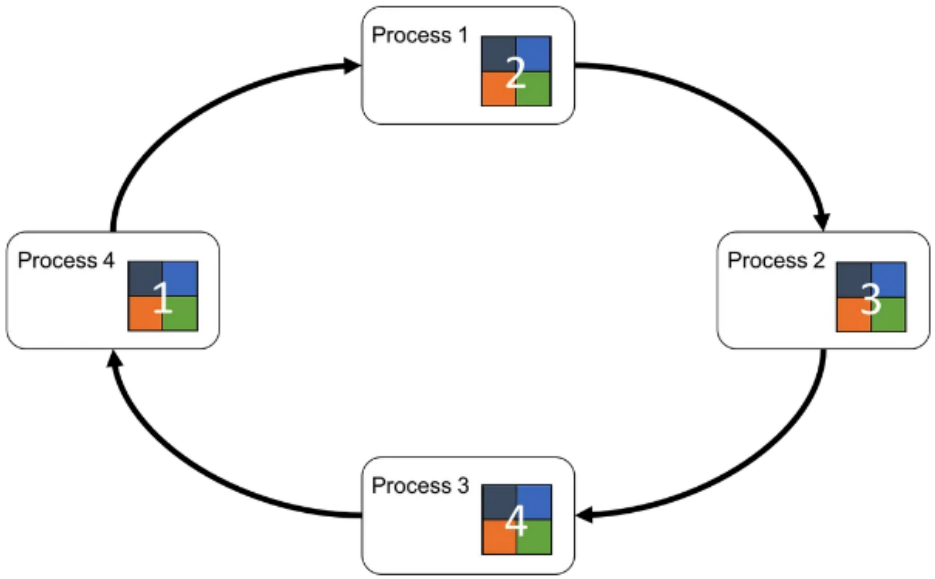

Fig.5 After P-1 steps, each process has a reduced subarray.

Figure: After $P-1$ steps, each node has a reduced sub-vector

Ring-allreduce

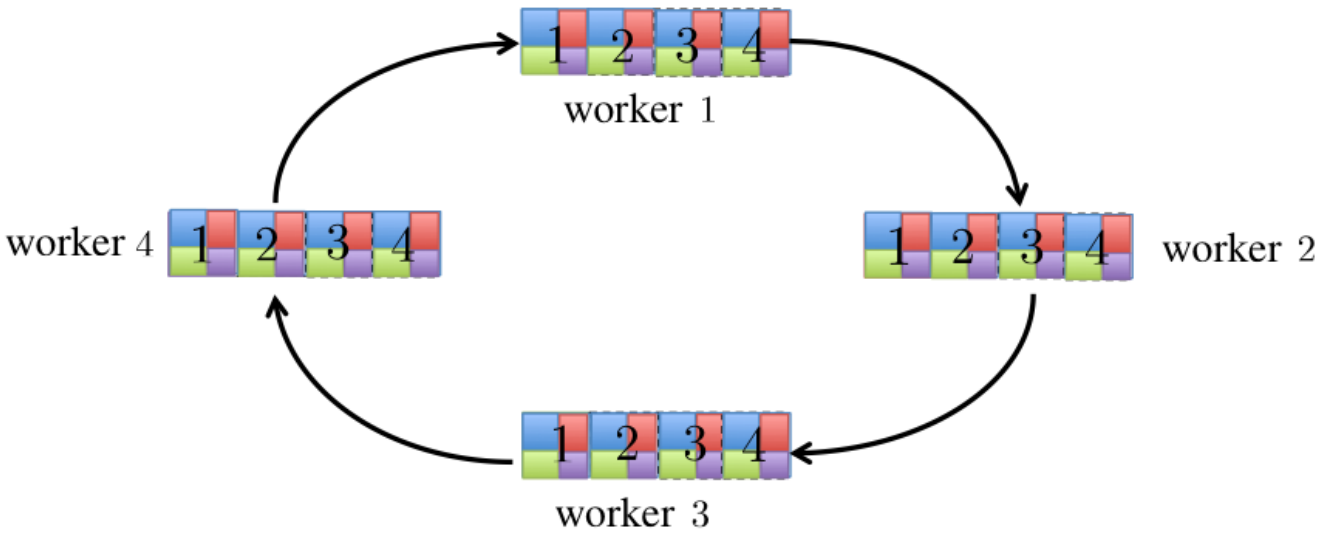

Figure: After another $P-1$ steps, all nodes get the averaged gradient

Ring-allreduce

- The time complexity for one ring-allreduce step:
  - Time spent at each step: $\frac{MB}{P} t_m$
  - There are $2(P-1)$ steps in total

$$T_{\text{ring}} = 2(P-1)\frac{MB}{P} t_m \longrightarrow 2MBt_m \quad \text{as} \quad P \to \infty$$

- Ring-allreduce has more elegant operations compared to BytePS
- No need for any additional server
- Ring-allreduce is employed in PeddlePeddle by Baidu and Horovod by Uber

Time complexities for various allreduce methods

allreduce methods	Time Complexity	Additional Requirements
star	$2PMBt_m$	N/A
tree	$2\log(P)MBt_m$	N/A
BytePS	$2MBt_m$	need P more servers
ring	$2MBt_m$	N/A

Table: Allreduce methods summary

- Ring-allreduce is preferable in both time complexity and implementation

Revisit distributed optimization

- Now we revisit the distributed optimization problem

$$\min_{w} \quad \frac{1}{P} \sum_{p=1}^{P} F_p(w)$$

  where each node has a local cost function $F_p(w)$.

- The standard distributed algorithm is

$$w^{k+1} = w^k - \frac{\alpha}{P} \sum_{p=1}^{P} \nabla F_p(w^k)$$

  where the average is computed by ring-allreduce.

Revisit distributed optimization

- Note that the time spent per iteration is $O(T_{\text{grad}} + MBt_m)$ where T_{grad} is the time to compute each $\nabla F_p(w^k)$ in parallel

- This time is independent of the number of nodes P; it is almost as good as the single-agent gradient descent

$$w^{k+1} = w^k - \alpha \nabla F(w^k) \qquad \text{(Time Cplex: } O(T_{\text{grad}})\text{)}$$

  with additional overhead $O(MBt_m)$. This conclusion is incredible!!

- If we overlap the computation and communication, the time complexity per iteration can be even better as

$$O\Big(\max\{T_{\text{grad}}, MBt_m\} \Big)$$

Revisit distributed optimization

- Suppose the algorithm (e.g., GD, SGD, or ADAM) requires "#iter" steps to converge to a ϵ-error

- The total time complexity is

$$O\Big(\#\text{iter} \times (T_{\text{grad}} + MBt_m)\Big)$$

  which is independent of the nodes' number P. Very nice scalability!

- I personally think this is the best time complexity in literature
  - best "$\#\text{iter}$" because it is the centralized bound
  - best $T_{\text{grad}} + MBt_m$ because of its scalability

- To further reduce the time complexity:
  - accelerate the algorithms convergence rate; improve "$\#\text{iter}$"
  - compress MB without affecting $\#\text{iter}$ too much

Revisit distributed optimization

- Is this conclusion correct? If it is true, I don't see too much improvement space. Many current research might not be that useful. For example:
- Decentralized algorithm: $w_i^{k+1} = \sum_{j \in \mathcal{N}_i} a_{ij} \left(w_j^k - \alpha \nabla F_j(w_j^k) \right)$
- Time complexity: $O\big(\#\text{d-iter} \times (T_{\text{grad}} + d_{\max} M B t_m)\big)$
  - $d_{\max}$ is the maximum degree in the network topology
  - $\#$d-iter is the iteration number of decentralized algo.; $\geq \#$iter
  - This time complexity is worse than GD with ring-allreduce

The future direction?

- Recall that ring-allreduce works in a **stable**, **homogeneous**, and **controllable** environment
- Ring-allreduce does not support asynchronous update. It can be severely downgraded by the slowest link or node.
- The system guys focus on building large system that provides ideal cond.
- We optimization guys may need to design new algorithms that work well under the **unstable**, **heterogeneous**, and **uncontrollable** environment.

After discussion with Hao

- The communication overhead t_c cannot be ignored
- t_c includes connection establishment overhead, synchronization overhead, and ACK for each package
- Each transmission may be partitioned into multiple smaller packages, and the communication of each package will incur t_c
- Time complexity involving t_c is summarized in the following table

Time complexities for various allreduce methods

allreduce methods	Time Complexity	Additional Requirements
star	$2P(MBt_m + t_c)$	N/A
tree	$2\log(P)(MBt_m + t_c)$	N/A
BytePS	$2MBt_m + Pt_c$	need P more servers
ring	$2MBt_m + Pt_c$	N/A

Table: Allreduce methods summary

- Ring-allreduce does not scale well with t_c
- The optimal lower bound for the time complexity is $O(MBt_m + \log(P)t_c)$, and there exist advanced but complicated allreduce methods to reach it.

References: [To be added]