On the Time Complexity of Common All-Reduce Methods

Kun Yuan

(Internal Reference Only)

September 7, 2019

Outline

1. Training framework for distributed deep learning
2. Time complexity for each all-reduce method
3. Decentralized algorithms

Outline

1. Training framework for distributed deep learning
2. Time complexity for each all-reduce method
3. Decentralized algorithms

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

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

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

Gradient Descent Method

- The gradient descent method to solve problem (1) is

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

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

  where α is the step-size.

- The computation of the true gradient $\nabla F_p(w^k)$ is time-consuming.

Stochastic Gradient Descent

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

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

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

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

- The batch is usually sampled without replacement.
- SGD is a standard training algorithm for deep learning.
- More advanced training methods: ADAM and its variant

Training framework for deep learning: Parameter Server

Algorithm 1 Distributed training framework: Parameter Server

Setting: P nodes; Partitioned data set $\{\{x_{1,n}\}_{n=1}^{N/K}, \cdots, \{x_{K,n}\}_{n=1}^{N/K}\}$

For $k = 1, 2, 3 \cdots$ until **meet a criterion**:

All $(P-1)$ **slave nodes do in parallel**:

Pull w^k from the master node P;

Sample a batch of local data $\{x_{p,\mathcal{B}}\}$;

Evaluate a local stochastic gradient $\widehat{\nabla F}_p(w^k; \{x_{p,\mathcal{B}}\})$;

Push $\widehat{\nabla F}_p(w^k; \{x_{p,\mathcal{B}}\})$ to the master node P;

The master node P **does**:

Reduce all received gradients: $\widehat{\nabla F}(w^k) = \frac{1}{P} \sum_{n=1}^{P} \widehat{\nabla F}_p(w^k; \{x_{p,\mathcal{B}}\})$

Update the variable: $w^{k+1} = w^k - \alpha \widehat{\nabla F}(w^k)$

Output: w^k

Training framework for deep learning: Parameter Server

- The all-reduce procedure in PS bases on "reduce $\rightarrow$ broadcast"
- Has to distinguish master node from slave nodes
- Four implementations on "reduce $\rightarrow$ broadcast"
  - Star implementation
  - Tree implementation
  - Cycle implementation
  - BytePS implementation
- Will discuss each implementation later

Training framework for deep learning: Ring-Allreduce

Algorithm 1 Distributed training framework: Ring-Allreduce

Setting: P nodes; Partitioned data set $\{\{x_{1,n}\}_{n=1}^{N/K}, \cdots, \{x_{K,n}\}_{n=1}^{N/K}\}$

For $k = 1, 2, 3 \cdots$ until **meet a criterion**:

Each node does in parallel:

Sample a batch of local data $\{x_{p,\mathcal{B}}\}$;

Evaluate a local stochastic gradient $\widehat{\nabla F}_p(w^k; \{x_{p,\mathcal{B}}\})$;

Run ring-allreduce and get $\widehat{\nabla F}(w^k) = \frac{1}{P}\sum_{n=1}^{P} \widehat{\nabla F}_p(w^k; \{x_{p,\mathcal{B}}\})$;

Update the variable: $w^{k+1} = w^k - \alpha \widehat{\nabla F}(w^k)$

Output: w^k

Important note: after ring-allreduce, every node will get $\widehat{\nabla F}(w^k)$

Training framework for deep learning: Ring-Allreduce

- Ring-allreduce framework is fundamentally different from PS
- In ring-allreduce:
  - Every node will get $\widehat{\nabla F}(w^k)$ after ring-allreduce
  - Every node will perform the SGD update
- In parameter server:
  - The master node will get $\widehat{\nabla F}(w^k)$ after reduce
  - The master node will perform the SGD update

Outline

1. Training framework for distributed deep learning
2. Time complexity for each all-reduce method
3. Decentralized algorithms

Star All-reduce

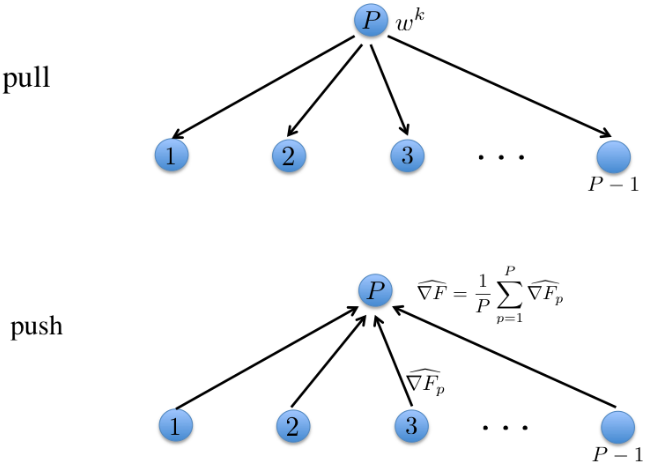

Figure: Illustration of the star all-reduce.

Star All-reduce

- We assume some parameters
  - M: the dimension of the vector
  - B: the bytes used to store each element in the vector
  - t_m: the time used to transmit a unit byte
  - t_c: the time used to establish one connection
- The time complexity for one star-allreduce step:

$$T_{\text{star}} = 2(P-1)(MBt_m + t_c)$$

- Can be implemented in an asynchronous manner; has theoretical guarantee

Tree All-reduce

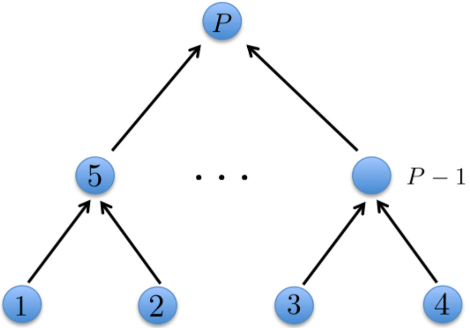

Figure: Illustration of the tree all-reduce.

Tree All-reduce

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

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

- Hard to be extended to the asynchronous version

Cycle All-reduce

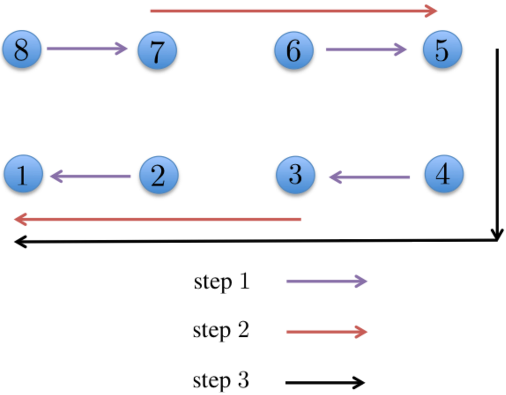

Figure: Illustration of the cycle all-reduce. In the figure, all information are merged to node 1.

Cycle All-reduce

- The time complexity for one cycle-allreduce step:
  - Time spent in each step: $MBt_m + t_c$
  - There are $\log(P)$ steps in total
  - There are a “reduce” step and a “broadcast” step

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

- Hard to be extended to the asynchronous version

BytePS All-reduce

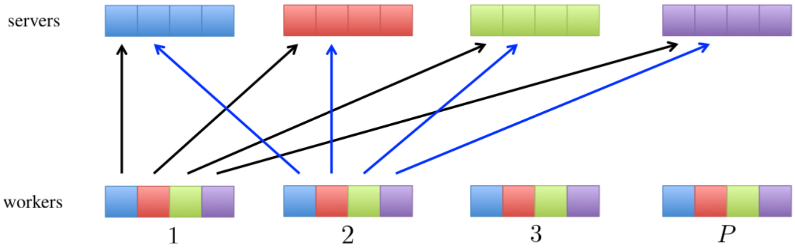

Figure: Illustration of the BytePS all-reduce.

BytePS All-reduce

- The time complexity for one BytePS-allreduce step:

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

- Can be improved to better complexity (but with more complicated ops.)

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

- Can be extended to the asynchronous version (not quite sure)

- Require more (but cheap) servers

Ring-allreduce

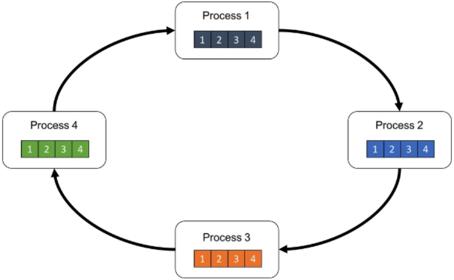

Fig.2 Example of a process ring

Figure: Illustration of the ring-allreduce.

Ring-allreduce

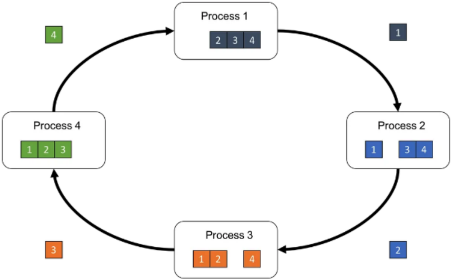

Figure: Illustration of the ring-allreduce.

Ring-allreduce

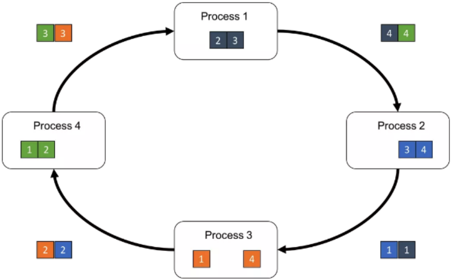

Figure: Illustration of the ring-allreduce.

Ring-allreduce

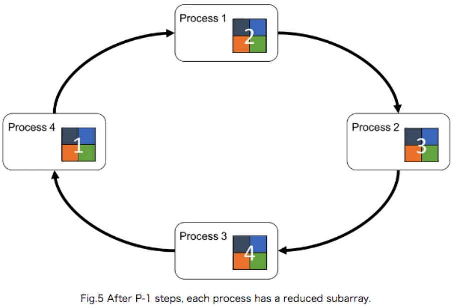

Figure: Illustration of the ring-allreduce.

Ring-allreduce

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

$$T_{\text{ring}} = 2(P-1)(\frac{MB}{P} t_m + t_c) \approx 2(MBt_m + Pt_c)$$

- Ring-allreduce has more elegant operations compared to BytePS
- Hard to be extended to the asynchronous version

Outline

1. Training framework for distributed deep learning

2. Time complexity for each all-reduce method

3. Decentralized algorithms

Core Idea

- All the above all-reduce methods traverse through all nodes
- A brand new idea:

  $\boxed{\text{Use the local average to approximate global average asympototically}}$

- Further lower down the above-mentioned time complexity

Average Consensus

- Calculate the average $\bar{x} = \frac{1}{P}\sum_{p=1}^{N} x_p$ over an arbitrary graph
- Core idea: each agent does local average

$$x_p^{k+1} = \sum_{q\in\mathcal{N}_p} a_{qp} x_q^k, \quad \text{where} \quad \sum_{q\in\mathcal{N}_p} a_{qp} = 1$$

- Convergence guarantee: $\|x_p^k - \bar{x}\| = \rho^k$ where $\rho < 1$ (linear convergence)
- This implies $x_p^k \to \bar{x}$ asymptotically for any $p = 1, 2, \cdots, P$.
- Proof:

$$\begin{aligned}\|\mathbf{x}^k - \bar{\mathbf{x}}^k\| = \|A\mathbf{x}^{k-1} - \frac{1}{n}\mathbb{1}\mathbb{1}^T\bar{\mathbf{x}}^{k-1}\| &\leq \|A - \frac{1}{n}\mathbb{1}\mathbb{1}^T\|\|\mathbf{x}^{k-1} - \bar{\mathbf{x}}^{k-1}\| \\ &\leq \rho\|\mathbf{x}^{k-1} - \bar{\mathbf{x}}^{k-1}\|\end{aligned}$$

where $\bar{\mathbf{x}}^k = \bar{\mathbf{x}}^{k-1} = \cdots = \bar{\mathbf{x}}^0$.

Average Consensus

$$x_P^{k+1} = \sum_{q \in \mathcal{N}_P} a_{qP} x_P^k$$

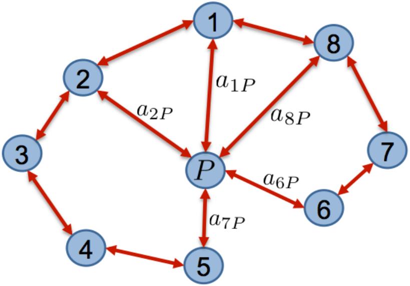

Figure: Illustration of an arbitrary graph and the consensus update.

Decentralized Stochastic Gradient Descent (Diffusion)

- Recall the standard SGD:

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

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

- Diffusion:

$$w_p^{k+1} = \sum_{q\in\mathcal{N}_p} a_{qp}\Big(w_p^k - \alpha\widehat{\nabla F_p}(w^k)\Big)\quad \text{(Local average)}$$

Theoretical Guarantee

- When each local cost function $F_p(w)$ is convex

$$\limsup_{k\to\infty} \frac{1}{P}\sum_{p=1}^{P} \mathbb{E}\|w_p^k - w^\star\|^2 = O\Big(\frac{\alpha\sigma^2}{P\nu}\Big), \quad \text{(SGD)}$$

$$\limsup_{k\to\infty} \frac{1}{P}\sum_{p=1}^{P} \mathbb{E}\|w_p^k - w^\star\|^2 = O\Big(\frac{\alpha\sigma^2}{P\nu} + \frac{\lambda^2\alpha^2\sigma^2}{1-\lambda}\Big), \quad \text{(Diffusion)}$$

  where σ^2 is the gradient noise, λ indicates the connectivity of the network

- When α is sufficiently small or the network is well-connected ($\lambda \to 0$),

$$\text{SGD} \approx \text{Diffusion}$$

- We can use local average per iteration without effecting the convergence!!

Local Averaging Time Complexity

- We use the Power-2 graph.
- Star local-averaging: $T = (MBt_m + t_s) \log P$; support asynchrony.
- Ring local-averaging: $T = 2(MBt_m + \sqrt{P}\, t_s)$; not support asynchrony.
- Better than their counterparts.

References: