Pipe-SGD: A Decentralized Pipelined SGD Framework for Distributed Deep Net Training

Youjie Li[1], Mingchao Yu[2], Songze Li[2], Salman Avestimehr[2], Nam Sung Kim[1], Alexander Schwing[1] [1]UIUC, [2]USC

Introduction

Motivation:

- Distributed training of deep nets is essential to address ever-increasing computational demands and memory consumption
- Main objective is to minimize the end-to-end training time

Prior Work:

- Centralized parameter-server framework, especially with asynchronous training (Fig. 1)
  - Pros: low synchronization overhead + interleaved training iterations
  - Cons: uncontrolled staleness of gradient + congested communication
- Decentralized framework with synchronous training (Fig. 2)
  - Pros: no stale gradient + balanced communication via AllReduce
  - Cons: high synchronization overhead + sequential execution of local iterations

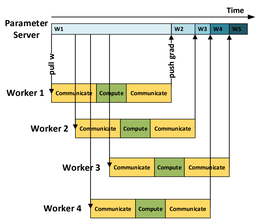


Fig. 1. Parameter server with asynchronous training.

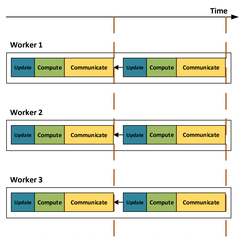


Fig. 2. Decentralized synchronous training.

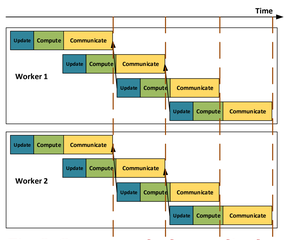


Fig. 3. Our approach: decentralized pipelined training.

Our Approach:

- To combine the best of both, our approach **Pipe-SGD**: decentralized pipelined training (Fig. 3)
  - Balances communication via AllReduce → low communication time
  - Pipelines local iterations to hide compute/communicate time → low execution time
  - Controls staleness of gradients → good convergence

Approach

Three Major Questions (Fig. 4):

- Staleness: can we restrict staleness of gradient? what is the best restricted staleness?
- Level-of-pipeline: besides pipelining iterations (level-1), can we further pipeline communication in each iteration (level-2)?
- Communication ratio: what ratio of communication overhead to per-iteration time is desirable?

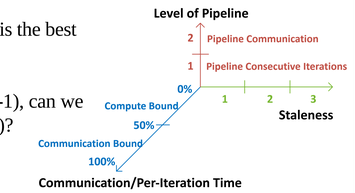


Fig. 4. Design space of **Pipe-SGD**.

Timing Model and Notation:

To achieve the optimal setting for **Pipe-SGD**, we propose timing models to analyze the end-to-end training time. We use the following notation:

- l_*: the time taken by any component, e.g., update, computation, and communication
- k: staleness
- T: total number of training iterations
- p: the number of workers
- α: the network latency
- n: the model size in bytes
- β: the byte transfer time
- γ: the byte sum reduction time
- S: the global synchronization time
- L: the number of gradient segments
- T_{single}: the number of iterations for single node training
- l_{single}: the per-iteration time for single node training

Approach (Continued)

Finding the Optimal Staleness under Resource Constraints (Fig. 5~6):

- Assume ideal conditions where resources are unlimited and constant T, then total train time is:

$$l_{total\_pipe} = T/(k+1) \cdot (l_{up} + l_{comp} + l_{comm})$$

- However, in reality, both computation and communication resources are strictly limited, thus:

$$l_{total\_pipe} = T \cdot \max(l_{up} + l_{comp}, l_{comm})$$

where the total time is solely determined by compute or communication resources, regardless of k.

- Also, since gradient staleness is always k iterations, increasing k only harms. Thus **the optimal value: k^* = 1, i.e., each iteration depends on the 2nd last iteration** (as in Fig. 6).

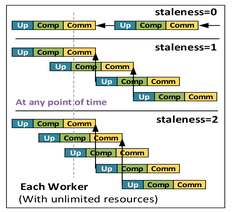


Fig. 5. Under ideal conditions.

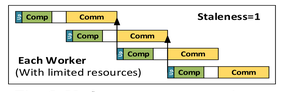


Fig. 6. Under resource constraints.

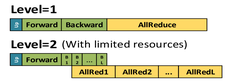


Fig. 7. Further pipelined communication.

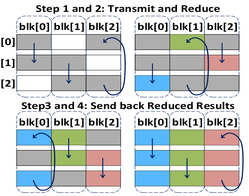


Fig. 8. Gradient communication using Ring-AllReduce.

Finding the Optimal Level of Pipelining (Fig. 7~8):

- The bandwidth-optimal Ring-AllReduce is adopted for gradient communication (Fig. 8)
- When level of pipeline is 1 (i.e. staying sequential within each iteration) (Fig. 7), total time is:

$$l_{total\_pipe\_1} = T \cdot \max\left(l_{up} + l_{for} + l_{back},\ 2(p-1)\cdot\alpha + 2(\frac{p-1}{p})\cdot n\cdot\beta + (\frac{p-1}{p})\cdot n\cdot\gamma + S\right)$$

- If we increase the level to 2 (i.e. further pipelining gradient communication) (Fig. 7), then:

$$l_{total\_pipe\_2} = T \cdot \max\left(l_{up} + l_{for} + l_{b1},\ 2(p-1)L\cdot\alpha + 2(\frac{p-1}{p})\cdot n\cdot\beta + (\frac{p-1}{p})\cdot n\cdot\gamma + L\cdot S\right)$$

- From $l_{total\_pipe\_1}$ and $l_{total\_pipe\_2}$, we note that increasing the level results in longer communication time, which worsens the already communication-dominant system in practice. Thus **the optimal value: level-of-pipeline = 1**.

Finding the Optimal Communication Ratio:

- Assume that: a) given a cluster we use the same batch size on each worker as in the single-node training; b) the single node and **Pipe-SGD** train the same epochs on the dataset. Thus, the "scaling efficiency" SE of **Pipe-SGD** is:

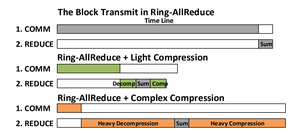
$$\text{SE} = \frac{\text{Actual Speedup}}{\text{Ideal Speedup}} = \frac{\frac{l_{single}\cdot T_{single}}{l_{total\_pipe}}}{p} = \frac{\frac{l_{single}\cdot T_{single}}{\max(l_{up}+l_{comp},\ l_{comm})\cdot\frac{T_{single}}{p}}}{p} = \frac{l_{up}+l_{comp}}{\max(l_{up}+l_{comp},\ l_{comm})}.$$

- Hence, once system is compute bound, linear speedup can be achieved as the cluster size scales, i.e. SE=1, which means **the optimal value: communication ratio < 50%**.

To sum up, Pipe-SGD is optimal for: k = 1, level-of-pipeline = 1, system is compute bound (after compression).

Compression

- For the desired communication ratio, **simple and light-weight** compression schemes are used in **Pipe-SGD**, such as floating-point truncation on 16-bit LSBs, and scalar 8-bit quantization.
- We find that complex compression algorithms aren't suitable for the bandwidth-optimal Ring-Allreduce:
  - Compression itself is compute-heavy
  - Compression is invoked repeatedly, complexity: *O(p)*



Pipe-SGD Algorithm and Convergence Rate

Algorithm 1: Decentralized Pipe-SGD training algorithm for each worker.

On the computation thread of each worker:

1: Initialize by the same model $w[0]$, learning rate γ, staleness k, and number of iterations T.
2: **for** $t = 1, \ldots, T$ **do**
3: Wait until aggregated gradient g^c_{sum} in compressed format at iteration $[t-(k+1)]$ is ready
4: Decompress $g_{sum}[t-(k+1)] \leftarrow \mathscr{D}(g^c_{sum}[t-(k+1)])$
5: Update $w[t] \leftarrow w[t-1] - \gamma \cdot g_{sum}[t-(k+1)]$
6: Load a batch $\mathscr{B}$ of training data
7: Forward pass to compute current loss $f_{\mathscr{B}}$
8: Backward pass to compute gradient $g_{local}[t] \leftarrow \frac{\partial f_{\mathscr{B}}}{\partial w[t]}$
9: Compress $g^c_{local}[t] \leftarrow \mathscr{C}(g_{local}[t])$
10: Denote local gradient $g^c_{local}[t]$ as ready
11: **end for**

On the communication thread of each worker:

1: Initialize aggregated gradients g^c_{sum} of iteration $[-k, 1-k, \ldots, 0]$ as zero and mark them as ready
2: **for** $t = 1, \ldots, (T-(k+1))$ **do**
3: Wait until local gradient $g^c_{local}[t]$ is ready
4: AllReduce $g^c_{sum}[t] \leftarrow \sum g^c_{local}[t]$
5: Denote aggregated gradient $g^c_{sum}[t]$ as ready
6: **end for**

Convergence: For convex objectives, the convergence rate of **Pipe-SGD** is $8FL\sqrt{\frac{k+1}{T}}$, where F and L are constants for gradient descent and Lipschitz continuity, respectively. For strongly convex objectives, the convergence rate of **Pipe-SGD** is $O(\frac{\log T}{T})$ for gradient descent.

Experiments

- Wall-clock measurement on total train time; the same number of iterations per benchmark
- Four-node GPU cluster (TitanXP, Xeon E5), 10Gb Ethernet, an extra node for parameter server
- **Pipe-SGD: 3.2~5.4x speedups compared to conventional approaches** without accuracy loss