Efficient Partial Reduce Across Clouds

Renyi Wang Shouxi Luo Ke Li Huanlai Xing
Southwest Jiaotong University
Chengdu, China

CCS CONCEPTS

• **Networks → Cloud computing**.

KEYWORDS

Partial reduce, flow scheduling, cloud computing

ACM Reference Format:
Renyi Wang Shouxi Luo Ke Li Huanlai Xing. 2022. Efficient Partial Reduce Across Clouds. In *6th Asia-Pacific Workshop on Networking (APNet 2022), July 1–2, 2022, Fuzhou, China.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3542637.3543707

1 INTRODUCTION

Geo-distributed machine learning (Geo-DML) systems, which provide the ability of learning models from massive data across the globe, have become an essential infrastructure for the design, development, and deployment of large-scale artificial intelligence (AI) services [1, 4]. In these systems, recent studies show that performing model synchronizations across wide-area networks (WANs) could dominate the time cost of the entire training; thus, optimizing the involved communication becomes the key to improve the efficiency of Geo-DML training [4].

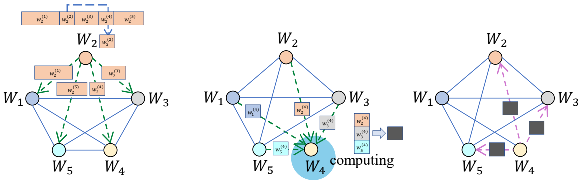
Basically, the synchronization of models can be carried out using the collective operation of All- or Partial- Reduce, depending on the training algorithm designs. Partial Reduce is a recently proposed variant of All Reduce [3]. By allowing p out of n workers to conduct All Reduce operations for a round of synchronization, it is able to ease the impacts of stragglers with a slowed-yet-controllable convergence speed, thus promising for data-parallel distributed training in heterological environments [3]. Then, an interesting question is: *how could we achieve efficient partial reduce for data-parallel distributed training in the context of inter-cloud Geo-DML, where workers are hosted in different clouds networked with heterogeneous WAN connections?*

In this work, we propose FMReduce, a flexible and efficient partial reduce implementation to pursue the goal. Compared with existing solutions, the novelties of FMReduce are two-fold, making it distinguished for inter-cloud data-parallel training.

Corresponding author: Huanlai Xing (hxx@swjtu.edu.cn). This work was partially supported by NSFC Project 62002300, NSFSC Project 2022NSFSC0944, and Project of Network and Data Security Key Laboratory in Sichuan Province NDS2022-1.

APNet 2022, July 1–2, 2022, Fuzhou, China

ACM ISBN 978-1-4503-9748-3/22/07…$15.00
https://doi.org/10.1145/3542637.3543707



(a) weighted scatter (b) controlled reduce computation (c) selected broadcast

Figure 1: The 3-phase workflow of FMReduce.

- Firstly, as the involved computation of reduce is generally not computationally intensive, instead of employing only the p selected workers, FMReduce leverages all the n available workers to conduct the operations needed by a partial reduce (via "scatter⇒reduce⇒broadcast" as Figure 1 shows and see §2 for details), thus can make efficient usage of all the abundant inter-cloud WAN connections.
- Secondly, rather than starting the involved transmissions only after all the p workers are already established, if desired, FMReduce allows a worker to scatter the updated model to others immediately once it is ready, making the spare yet scarce and expensive WAN link capacities get better usages.

2 FMREDUCE

Overview. As Figure 1 shows, FMReduce (Full-Mesh Reduce) abstracts the inter-cloud network involving n workers as a fully connected graph, where the directed link from worker i to worker j is with the available bandwidth of $b_{i,j}$ for partial reduce.[1] And because of the presence of coexisting traffic, $b_{i,j}$ is probably unequal to $b_{j,i}$. To make full use of the processing capacity of all workers and all the available bandwidth, when worker i completes its local training, it splits its model parameters W_i into n non-overlapping blocks: $W_i^1, \cdots, W_i^j, \cdots, W_i^n$ for synchronization, respecting the available bandwidth around the network. Then, the j-th block W_i^j is scattered to worker j. Note that, the scatter of W_i^i will not trigger network transmissions as the destination and the source are the same. On getting these blocks, the worker would cache them to conduct all or partial reduce (Figure 1.b), and broadcast/multicast the result back once the computation is done (Figure 1.c).

To release the full power of the above design, however, three problems must be addressed carefully. Next, we briefly introduce what the problems are and sketch how FMReduce solves them.

1) How to split the model into blocks to make the most usage of the available bandwidth? Intuitively, each worker should split its model parameters in the proportion of its up- and down- link bandwidths to others for the best usage of link capacities. However, if each worker only considers its own bandwidths, inconsistent

[1] For each i, we assume that $b_{i,i}$ is set to a large value and would not be the bottleneck.

scatters of the model would occur. To address this, FMReduce computes the splitting scheme upon a global view of the involved links' available bandwidth.

Let v be the volume of model W and x_j be the size of the j-th partition $W_i^j (1 \le j \le n)$ under consistent splitting. Then, obviously, the time costs of the scatter and broadcast phases specified in Figure 1, can be formulated by Eq. (1a) and (1b), respectively.

$$T_{scatter} \coloneqq \max_i T^i_{scatter} \coloneqq \max_i \max_j \frac{x_j}{b_{i,j}} = \max_j \frac{x_j}{\min_i b_{i,j}} \quad (1a)$$

$$T_{broadcast} \coloneqq \max_i T^i_{broadcast} \coloneqq \max_i \frac{x_i}{\min_j b_{i,j}} \quad (1b)$$

Following this, the upper bound of the communication time involved in a round of partial reduce can be formulated as $T_{scatter} + T_{broadcast}$. To make partial reduce more efficient, FMReduce tries to minimize this upper bound by splitting the model appropriately, according to the results of *linear programming* (LP) (2). As the implementation, FMReduce lets the worker with the lowest ID compute the splitting scheme based on the updated view of the inter-cloud network and send the results to involved workers on demand.

$$\textbf{Minimize } \{T_{scatter} + T_{broadcast} : \sum_{i=1}^{n} x_i = v, \text{where } x_i \ge 0\} \quad (2)$$

2) How to guarantee consistent partial reduce operation? FMReduce supports both blocking and non-blocking partial reduce. In the blocking mode, workers could not begin the scatter until FMReduce has figured out which p workers are involved and has decided how to split the model based on that; while in the non-blocking mode, workers could immediately start the scatter following predefined splitting plans. When performing non-blocking partial reduce for $p < n$, despite the launch of scatters being non-blocking, FMReduce still needs to decide which p workers could participate in a round of synchronization and ensures that all workers do agree with this for consistency. To this end, FMReduce configures the worker with the smallest ID to act as the leader. Once getting p scattered model blocks from p workers (including that of itself if it is ready), it broadcasts which p are involved in this round to all workers. On getting this information, each worker conducts partial reduce on the selected p blocks, then broadcasts the result back to these corresponding p workers immediately, or later in case that some blocks are not obtained yet, as Figure 1.b and Figure 1.c show.

3) How to schedule concurrent transfers for the optimization of training QoS? Sometimes, flows triggered by the new scatter might compete with flows belonging to a previous ongoing broadcast on some links. It is obvious that, in such a scenario, the default *max-min* fairness (i.e., FS) is sub-optimal for the allocation of bandwidth; and instead, the principle of *First-Come-First-Service* (FCFS) is a simple-yet-efficient design.

3 PRELIMINARY EVALUATION

Mainly, FMReduce embodies four types of alternative designs for the implementation of partial reduce:

- p *vs.* n workers used for each partial reduce;
- **Blocking** *vs.* **non-blocking** scatters for partial reduce;
- **Evenly-divided** *vs.* **network-aware** model splits/scatters;
- **FS** *vs.* **FCFS** flow scheduling for concurrent scatters/broadcasts.

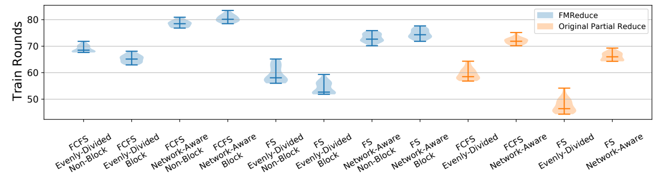


Figure 2: FMReduce vs Origial Partial Reduce when $p = 50$. FMReduce uses n workers for each partial reduce while Original Partial Reduce uses p workers only supporting *blocking*.

To verify their benefits, we develop a flow-level simulator with Python 3, which precisely simulates the behaviors of partial reduce operations with and without the enhancement of FMReduce. We conduct 12 comparative simulation experiments and assess them by the number of training rounds in a fixed period of time.

Network and workloads. We consider a geo-DML environment involving 60 workers, in which, the available bandwidths of inter-cloud connections (i.e., $b_{i,j}$) are calculated by $k * u$, where k is an integer in the range of $[1, 10]$ sampled from $N(5, 1)$ and u is the minimum bandwidth unit with the value of $25Mbps$. We assume that the model is with the size of $200MB$; the time of workers' each round of local training follows the distribution reported by [2].

Case Study. As shown in Figure 2, when other conditions keep the same, **network-aware** and **FCFS** scheme can always efficiently increase the train rounds compared to **evenly-divided** and **FS**. Regarding **blocking** and **non-blocking**, **non-blocking** performs better than **blocking** unless the model splitting scheme is **network-aware**. This is because the composition of p workers can be known under **blocking**, which can help to calculate a more precise model splitting results according to the bandwidths of the p workers when $p < n$. However, if $p = n$ exactly, the composition of p workers is known in advance, then **non-blocking** can reduce the waiting time without cost, outperforming **blocking**. Most importantly, as Figure 2 shows, by employing all n workers to conduct each partial reduce, FMReduce achieves a remarkable speedup for model synchronization compared to the original partial reduce.

4 CONCLUSION

We have presented FMReduce, an efficient partial reduce implementation for inter-cloud Geo-DML training. By balancing the involved communication and reduce operations among all available workers respecting the state of inter-cloud connections, and conducting FCFS flow scheduling, FMReduce could make effective usage of the available inter-cloud network to achieve efficient partial reduce.

REFERENCES

[1] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds. In *14th NSDI*. USENIX Association, 629–647.
[2] Shigang Li, Tal Ben-Nun, Salvatore Di Girolamo, Dan Alistarh, and Torsten Hoefler. 2020. Taming Unbalanced Training Workloads in Deep Learning with Partial Collective Operations. In *25th PPoPP*. ACM, 45–61.
[3] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. 2021. Heterogeneity-Aware Distributed Machine Learning Training via Partial Reduce. In *SIGMOD*. ACM, 2262–2270.
[4] Pan Zhou, Qian Lin, Dumitrel Loghin, Beng Chin Ooi, Yuncheng Wu, and Hongfang Yu. 2021. Communication-efficient Decentralized Machine Learning over Heterogeneous Networks. In *37th ICDE*. IEEE, 384–395.