Optimal Message Scheduling for Aggregation

Leyuan Wang
UC Davis, Computer Science
leywang@ucdavis.edu

Mu Li, Edo Liberty, Alex J. Smola
AWS Machine Learning Palo Alto
[mli,libertye,smola]@amazon.com

ABSTRACT

We derive algorithms for producing optimal aggregation schedules for automatically aggregating gradients across different compute units, both CPUs and GPUs, with arbitrary topologies. We show that this can be accomplished by solving a linear program on the spanning tree polytope. We give analytic bounds for the value of the optimal solution for arbitrary graphs. We also propose simple schedules that meet those bounds for some specific graphs.

ACM Reference Format:
Leyuan Wang and Mu Li, Edo Liberty, Alex J. Smola. 2018. Optimal Message Scheduling for Aggregation. In *Proceedings of ACM Conference on Systems and Machine Learning (SysML'18).* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Deep learning requires large amounts of computation for training. In recent years a number of frameworks for parallel training such as TensorFlow, CNTK, and MXNet have emerged to take advantage of multiple GPUs and entire clusters of machines. Distributed training is nontrivial since the target machines may have varying and large numbers of GPUs and CPUs connected nonuniformly, with significant variance in network topology and the bandwidth of different network links.

A key primitive in all deep learning frameworks is the gather-scatter operation [1, 5]. Computing parameter w requires a set of updates δ_s to be aggregated from all servers $s \in S$ via $\delta = \bigoplus_s \delta_s$. These updates are then applied to $w \leftarrow f(w, \delta)$. Subsequently, the new value of w is broadcast to all servers s.

Numerous variants of this communication protocol exist. Examples include bounded delay and asynchronous updates. For simplicity and conciseness, we focus on the delayless synchronous variant. In this paper, we further focus on the communication within a single server, though our algorithms hold for machines with multiple servers as well. Consider two popular high-performance GPU servers—Amazon's P3.16xlarge and NVIDIA's DGX-1 (Figure 1) and Amazon's P2.16xlarge instances (Figure 2). Even among just these two servers we see a multitude of (different) processors, network interconnection types, and bandwidths.

In the computations we consider here, a full set of gradients are broadcast between compute units, which perform the aggregations. The communication bandwidth between processors, not computation on the processors, is the bottleneck for the overall computation.

SysML'18, April 2018, Stanford, CA USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

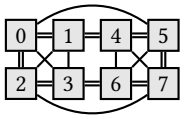


Figure 1: Communication design for AWS P3.16xlarge instances. Each edge amounts to an NVLink connection (we skipped PCI-E and QPI links for simplicity).

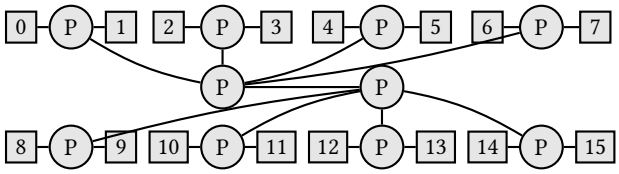


Figure 2: The Amazon AWS P2.16xlarge instance. Solid lines are PCI-E bus lanes (16 lanes wide).

We assume that the entire model fits in the memory of the compute units, justified by the realization that GPU memory size has outpaced model complexity—in fact, all recent image recognition models are *smaller* than Alexnet [4]. Hence, the main challenge in optimizing the overall computation is scheduling gather-scatter algorithms efficiently across compute units.

Related Work. The ParameterServer architecture offers a possible strategy for communication [1] via a bipartite graph of workers and servers. An alternative is to perform ring synchronization, such as Baidu-Allreduce and Horovod. We show that these algorithms arise as special cases of our optimization approach.

2 NETWORKS OF COMPUTE UNITS

Assume for now that the computer network consists only of nodes that are able to perform aggregation, e.g., a network of CPUs or of GPUs only. When we want to aggregate $\delta = \bigoplus_s \delta_s$ between servers, all changes ultimately need to reach one final server, say s^*. This is only possible if there is a spanning tree T connecting s^* to all other nodes. Since each node has computation capability, we note that we can always reduce network traffic if we aggregate incoming messages before transmitting them to the next destination. For efficiency we can aggregate by *streaming* data across the edges simultaneously. This minimizes the overhead due to tree depth.

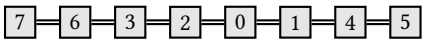
Denote by $G(V, E)$ an undirected graph with vertices V corresponding to compute units and edges E corresponding to communication links. Let $\mathcal{T}$ be the set of spanning trees on G. Let $c_e \geq 0$ be the total time to transmit all gradients through edge e. The time to synchronize all gradients along a single spanning T tree is dominated by the link with the lowest bandwidth used by the tree.

$$C(T) = \max_e c_e T_e \text{ where } T_e = 1 \text{ if } e \in T \tag{1}$$

In particular, Boruvka's minimum weight spanning-tree algorithm returns the optimal single spanning tree T_{MWST}, i.e.,

$$C(T) \geq C(T_{\text{MWST}}) \text{ for } T \in \mathcal{T}.$$

In the case of the P3.16xlarge architecture (Figure 1), an optimal single tree is given by the chain connecting all nodes with dual NVLink edges with the exception of the edge (5, 7).



This is not optimal, since all the single NVLink connections, e.g., between (0, 3) or the dual connection (5, 7), are not used. If we used multiple independent spanning trees, we could form 8 chains and balance the network traffic equally over them. This reduces the communication time by $^7/_8$, matching the bandwidth offered by ring synchronization. We now generalize this.

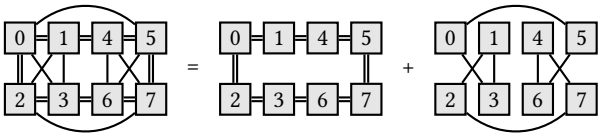
The Spanning Tree Polytope. When balancing synchronization over multiple spanning trees, each of which carries a portion of the data, the solution can be found as an element of the spanning tree polytope $\mathcal{P}(G)$, i.e., the convex combination of spanning trees on G. In other words, the schedule sends α_T traffic using spanning tree T and $\sum \alpha_T = 1$. It is convenient to define $P_e = \sum_{T \in \mathcal{T}} \alpha_T T_e$ which is the total amount of traffic across edge e. This leads to the following convex optimization problem:

$$\boxed{C(G) := \underset{P \in \mathcal{P}(G)}{\text{minimize}} \max_e P_e c_e} \tag{2}$$

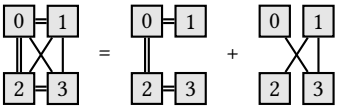
Lemma 2.1. *For any connected graph the value $C(G)$ of the optimization problem (2) satisfies*

$$\max_e c_e \geq C(T_{\text{MWST}}) \geq C(G) \geq \frac{n-1}{\sum_e c_e^{-1}} \tag{3}$$

This bound states that the highest bandwidth achievable by any graph is bounded from above by the sum of overall edge bandwidths uniformly distributed over a single spanning tree ($n-1$ edges). In the case of the P3.16xlarge network design, we obtain the optimal schedule, achieving $C(G) = 7/24$, as follows:



For P3.8xlarge servers which consist of half the GPUs, we obtain the following optimal scheduling with $C(G) = 1/3$.



3 NETWORKS WITH SWITCHES

When we add switches, the problem is more complicated. We cannot consider only spanning trees as before but rather schedules. A schedule is a fully specified communication pattern that can achieve the aggregation along the edges of the graph. Schedules are more complex than trees because data needs to flow through the switches that *cannot be aggregated*. An example of such a network design is that of P2.16xlarge AWS-EC2 servers. Any data crossing the central PCI express bridge also needs to flow through other bridges, thus consuming parts of their capacity.

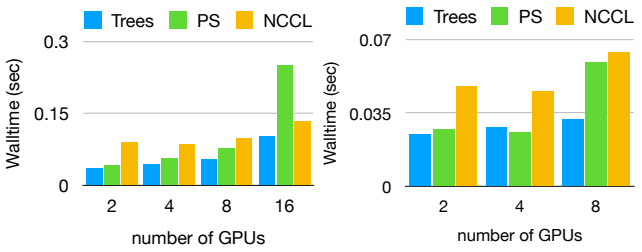


Figure 3: Compare the communication overhead of training ResNet-152 with the proposed scheme, parameter server (PS), and NCCL. Left: P2.16xlarge. Right: P3.16xlarge.

To solve the problem, the term P_e should be redefined as $P_e = \sum_{S \in \mathcal{S}} \alpha_T S_e$, where $\mathcal{S}$ is the set of all efficient schedules and $S_e \in \mathbb{Z}^+$ is the number of times edge e participates in S. Other than that, the problem remains unchanged. Given the set of schedules $\mathcal{S}$, the optimization is solvable using linear programming. Alas, $\mathcal{S}$ could be very large, which would make this computation very heavy. It is possible, however, to leverage characteristics of $\mathcal{S}$ and symmetries in the network to reduce the solution space and make the computation more manageable.

We can use this to solve the synchronization problem for any network of compute devices. In the specific case of the P2.16xlarge servers, we obtain the schedule shown in Figure 4. Quite surprisingly, the single 16-lane PLX between both halves is *not* the bottleneck for an efficient schedule. The utilization of the PLX within the cards, the PLX interconnecting the cards, and the connection between the two central PLX chips has the ratio 15:14:8. This can be seen, e.g., by edge-counting over the schedule in Figure 4 and by averaging over all schedules and lags. Furthermore, for P2.8xlarge, the optimal schedule is given by the first three stages of this diagram (we don't need to synchronize the last step). There, the utilization is 7:6 between intra-card PLX and inter-card PLX chips respectively. In both cases the bottleneck is the intra-card connectivity, a rather surprising result.

4 EXPERIMENTS

We implemented the proposed synchronization scheme in MXNet [2]. Each schedule is constructed as a computation graph. We leveraged MXNet's multi-threaded engine to execute these graphs in parallel. We used the data communication workload in the multi-GPU training of ResNet-152 [3] with data parallelism as our benchmark. We measured the walltime for each batch, specifically the data synchronization cost of a batch. For comparison, we also tested communication with a parameter server (PS) [5] and using NVIDIA's NCCL multi-GPU communication library. Figure 3 shows preliminary results: with 4 GPUs on P3 and 8 GPUs on P2, the proposed scheme performs as well as a PS that uses all-to-all communication. Both of them outperform the ring-based NCCL. When adding more GPUs, however, PS soon saturates the bottleneck connection, while the proposed scheme only drops by 13% on P3 and 20% on P2.

REFERENCES

[1] A. Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and A. J. Smola. Scalable inference in latent variable models. In *Proceedings of The 5th ACM International Conference on Web Search and Data Mining (WSDM)*, 2012.

[2] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[5] M. Li, D. G. Andersen, J. Park, A. J. Smola, A. Amhed, V. Josifovski, J. Long, E. Shekita, and B. Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

A VISUAL ILLUSTRATION OF P2.16XLARGE COMMUNICATION SCHEDULE

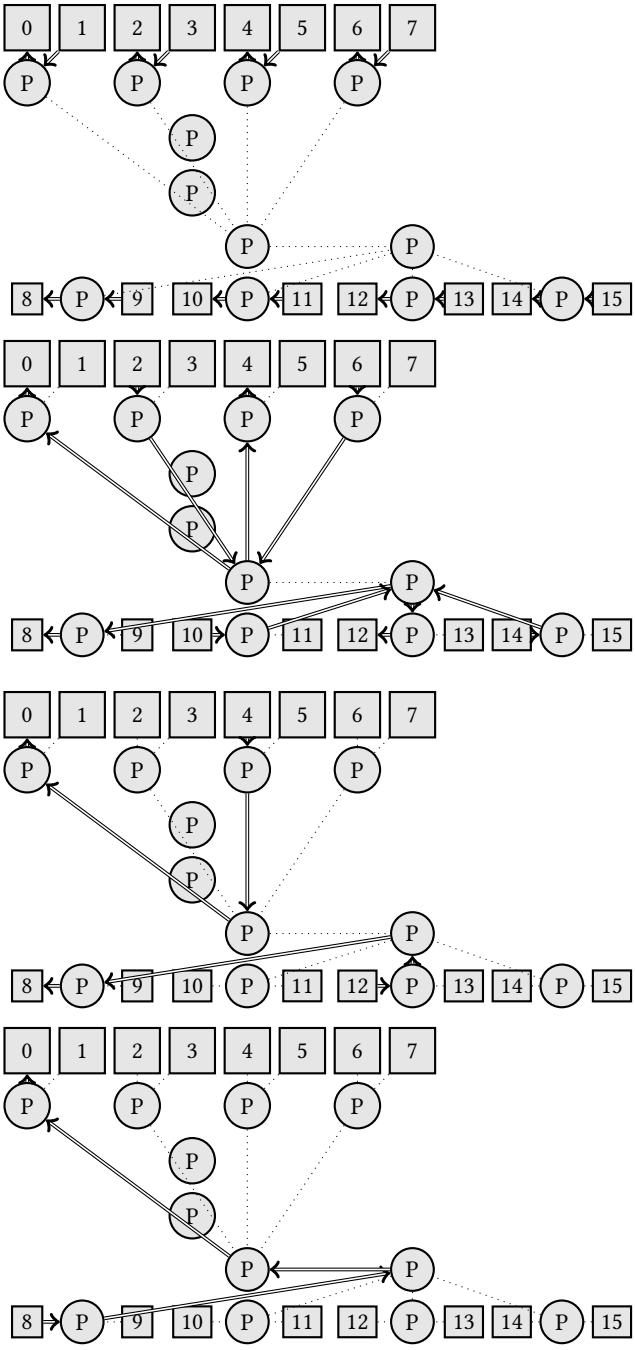


Figure 4: One of the 16 components of the optimal communication schedule for P2.16xlarge (the others use one of the other 15 nodes as the root node for aggregation). The synchronization proceeds in 4 steps, which are interleaved in 4 stages.