On The Importance of Execution Ordering in Graph-Based Distributed Machine Learning Systems

Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, Roy Campbell
University of Illinois at Urbana-Champaign

ABSTRACT

Execution of operations in distributed machine learning systems has largely ignored dependencies between communication and computation ops. In this paper, we make the case that model-aware ordering of operations at individual machines can decrease the step time of training iteration in distributed machine learning systems while also improving network utilization. The contributions of this work are:

- We introduce a metric for quantitatively measuring the efficiency of ordering of ops (§1).
- We propose an ordering heuristic for Model-Replica with Parameter Server systems (§2.1).
- We chalk out a roadmap for developing fast heuristics for model-aware ordering of ops in Model-Replica systems with all-reduce synchronization (§2.2).
- We evaluate our ordering mechanism on Model-Replica with Parameter Server on TensorFlow and show that the training efficiency can be improved by up to 78% through better ordering of tasks with 46% reduction in step time (§3).

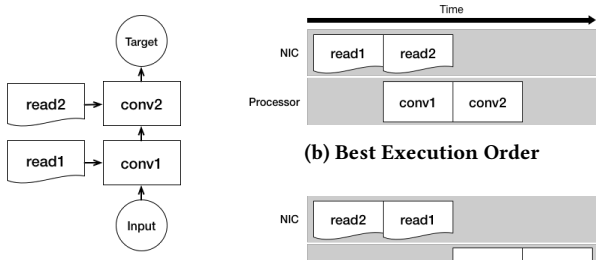
1 PROBLEM DEFINITION

Graph-Based Machine Learning Systems such as TensorFlow [1] and PyTorch [12] evolved as a response to the growing complexity of problems that artificial intelligence is trying to solve. In these systems, a machine learning model (model) is represented by a directed acyclic graph (DAG) with predefined operations (ops) as nodes and their data/control dependencies as edges.

When a distributed model is sent for execution, each op is assigned a tag [1]. This tag determines the device on which the op with will run. Using the type of op, we can also infer the associated resource on the device: computation unit, or communication channel from a specific source. Thus, the tagging of ops in a distributed model allows to precisely determine the number and type of ops assigned to a given device. In addition, the DAG gives us the relative ordering of ops on a device, with multiple possible combinations. We show that the performance of the model can vary widely across the multiple feasible combinations. Our goal is to select and enforce execution orders with minimal makespan.

First, we illustrate the significance of ordering with a simple example. In Figure 1a, there are two communication ops ($read_1$,$read_2$) and two computation ops ($conv_1$,$conv_2$) assigned to a device. While both $r_1 \rightarrow r_2 \rightarrow c_1 \rightarrow c_2$ and $r_2 \rightarrow r_1 \rightarrow c_1 \rightarrow c_2$ are valid orders topologically, as shown in figures 1b and 1c, the latter has the worst step time due to lack of overlap between transfer and computation.

Next, we define the problem formally. An execution order is defined as a function that maps an op to a real number. This mapping determines the relative ordering of ops on a given resource. Our goal is to determine an execution order that minimizes the makespan of the model. This problem of finding an order that minimizes the makespan can be mapped to the job shop problem[9] which is known to be NP-complete.



(a) DAG

(b) Best Execution Order

(c) Worst Execution Order

Figure 1: Example of impact of ordering on performance

Given a performance oracle, $Cost$, which predicts the actual cost of running an op on a resource, the upper and lower boundaries of the makespan respectively are:

$$\overline{Cost}(G) = \sum_{op \in G} Cost(op), \quad \underline{Cost}(G) = \max_{r \in R} \sum_{op \in G_r} Cost(op) \quad (1)$$

where G is the set of all ops, R refers to all resources, and G_r refers to all ops assigned to the resource r. For a given makespan, t, "Ordering Efficiency" (E) is defined as [1]:

$$E(G, Cost, t) = \frac{\overline{Cost}(G) - t}{\overline{Cost}(G) - \underline{Cost}(G)} \quad (2)$$

With Ordering Efficiency, we can compare two execution order quantitatively regardless of variations in execution scenarios. $E = 1$ means perfect ordering (which may not be possible at all), and $E = 0$ means serialized execution.

2 MODEL-REPLICA DISTRIBUTION

Many large-scale machine learning systems in practice are distributed using a paradigm called Model-Replica [5–7]. In MR, a base model is replicated on several devices (also known as workers) and the input data is partitioned among these replicas. The goal of replicas is to collectively update a persistent collection of parameters. Therefore, at each step of training, MR requires synchronization among workers.

There are two major patterns for synchronization: **(a) Parameter server:** where one or more special devices called Parameter Servers (PS) store the master copy of the parameters and aggregate

[1] This metric is an update to "Scheduling Efficiency" metric in the original job shop problem

updates to them([1, 4, 6, 11]). A worker loads a fresh copy of parameters at the beginning of each step and sends updates back to PS during the step. **(b) All Reduce:** where each worker keeps a copy of all parameters([2, 5, 7]). Changes are propagated directly to other workers with collective algorithms such as bucket algorithm [3] or recursive halving-doubling [15].

2.1 Ordering in Parameter Server

Search space for ordering in PS pattern can be reduced to the ordering of read ops on a worker. We observe that (i) Computation load on PS is insignificant compared to communication, which makes the ordering less influential on the overall performance, and (ii) read ops on workers are leaves in the model, that are immediately executable at the start of the step. Hence, the order of the computation ops does not impact the makespan.

We calculate the order of a *read* op as[2]:

$$Order(read) = \min\{|Dep(op)| \; for\; op \; in\; G \; if\; read \in Dep(op), |Dep(op)| > 1\} \quad (3)$$

where G is the set of all the ops in the DAG, and $Dep(op)$ is the set of read ops that op depends on. Intuitively this heuristic prioritizes read ops that are connected to computation ops with less dependencies. Evaluation is presented in §3.

2.2 Ordering in All Reduce

All Reduce requires all nodes to actively synchronize at the same time. As a result, in MR+AR, a perfect execution order should ensures that all workers reach the sync ops for each parameter simultaneously. Hence, we need to order both computation and communication ops.

Additionally, there are two windows for synchronizing each parameter: (a) at the beginning of a step before the parameter is modified or (b) at the end of a step after the parameter is updated. This adds further complexity to the ordering problem[3]. Ordering in MR+AR is ongoing work. .

3 EVALUATION

We extend TensorFlow [4] tracing capability to measure the cost of both computational and communication ops[5] to calculate the "Ordering Efficiency". We implement the ordering algorithm as a static analyzer and make changes in TensorFlow core to enforce a given order on the sender side just before the response is sent to the destination. [6]. The experiments were run on a commodity cluster [7] with one PS and one worker on separate nodes. We choose a batch size for each model in a way that balances communication and computation loads ($\frac{\overline{Cost}(G)-\underline{Cost}(G)}{\overline{Cost}(G)} > 90\%$). We tested our method on well-known models, Alexnet [10], ResNetV2-152 [8], InceptionV2 [14], VGG16 [13] as well as two extreme toy models:

- Par-32: A flat model with 32 concurrent layers. All the topological orders are the best order in this model .
- Seq-32: A sequential model with 32 layers similar to recurrent models. Only one topological order out of 32! is the best order.

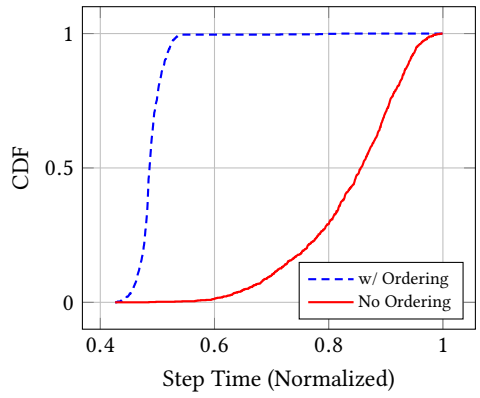


Figure 2: Distribution of step time for different ordering algorithms on InceprionV2 model.

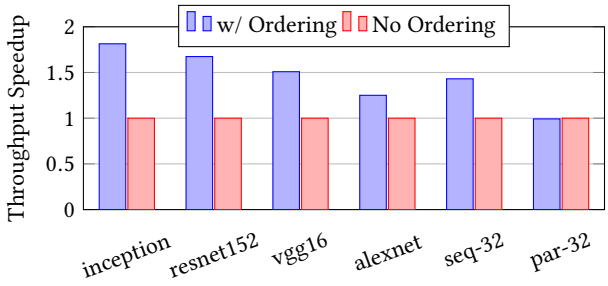


Figure 3: Throughput (Image/second) speedup of ordering algorithms for different models. Higher the better. Seq-32 is a 32-layer sequential model (similar to Recurrent models.) Par-32 is a 32-layer parallel model.

3.1 Results

Figure 2 shows the forward-pass step time cdf of 1-Worker 1-PS training on InceptionV2. We repeat 1000 runs each with and without the ordering heuristic. Enforcing ordering reduces the step time by 46% on average. Moreover, the standard deviation in step time is reduced by nearly 4× (48.3ms vs 187.1ms). We have observed similar pattern with other models except Par-32.

Figure 3 shows the average speedup of different models in forward-pass. Notably, Resnet-152 has 78% higher ordering efficiency (which results in 65% speed up in step time). Par-32, with all topological orderings already optimum, is the only model whose performance is hit marginally due to the ordering overhead.

[2]We use a heuristic-based approach to order read ops. We have the analysis with the complete heuristic that uses "Cost Oracle" which is beyond the scope of this abstract. The performance with the simple heuristic is within 5% of that obtained with the complete cost function. Hence simple one has been chosen as the representative in this abstract for ease of explanation.

[3]Most systems in practice perform the synchronization at the end of the step. A simulation of running synchronization in the beginning can be found in systems which removes the step barrier such as [7]

[4]top of the tree as of December 1, 2017: https://github.com/tensorflow/tensorflow/commit/efbdc15b280374607895ab0ada467de4a0512e0c

[5]github.com/tensorflow/tensorflow/pull/14604

[6]The code is accessible from github.com/xldrx/orderedtf

[7]32-core Xeon processor and 1Gbps ethernet network

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. 2015. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. *CoRR* abs/1512.02595 (2015). http://arxiv.org/abs/1512.02595

[3] Mike Barnett, Lance Shuler, Robert van De Geijn, Satya Gupta, David G Payne, and Jerrell Watts. 1994. Interprocessor collective communication library (InterCom). In *Scalable High-Performance Computing Conference, 1994., Proceedings of the*. IEEE, 357–364.

[4] Trishul M Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. 2014. Project Adam: Building an Efficient and Scalable Deep Learning Training System.. In *OSDI*, Vol. 14. 571–582.

[5] Minsik Cho, Ulrich Finkler, Sameer Kumar, David S. Kung, Vaibhav Saxena, and Dheeraj Sreedhar. 2017. PowerAI DDL. *CoRR* abs/1708.02188 (2017). arXiv:1708.02188 http://arxiv.org/abs/1708.02188

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[7] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European Conference on Computer Vision*. Springer, 630–645.

[9] Anant Singh Jain and Sheik Meeran. 1999. Deterministic job-shop scheduling: Past, present and future. *European journal of operational research* 113, 2 (1999), 390–434.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[11] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server.. In *OSDI*, Vol. 1. 3.

[12] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration. (2017).

[13] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[14] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2818–2826.

[15] Rajeev Thakur and William D Gropp. 2003. Improving the performance of collective operations in MPICH. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 257–267.