Communication Patterns in Distributed Deep Learning

Amir Farhat Manya Ghobadi
Massachusetts Institute of Technology
amirf@mit.edu,ghobadi@csail.mit.edu

ABSTRACT

Machine learning has been increasingly deployed in the cloud to take advantage of massive scaling capability as a means of reducing the time-to-accuracy of training. To this end, different machine learning training distribution frameworks are put to use, with Horovod from Uber emerging as a popular choice. To squeeze as much performance as possible from the distribution framework, it is important to maximally overlap computation and communication while maintaining high GPU utilization as a way of reducing the duration of each iteration of training. As a first step in this direction, this project sets out to study the communication component of training. We train Deep Neural Network (DNN) models of various sizes on sixteen GPUs in Google Cloud Compute Engine platform and record information about the data the workers exchange as well as the timing of each iteration of training. Our two main observations are: (i) the amount of data exchanged between workers at each training iteration is proportional to the model size; and (ii) the duration of training is not fully determined by the model size, it depends also on the compute hardware, communication bandwidth, and batch sizes in addition. The significance of these findings does not offer a complete enough picture for improving the TTA for models, but can do that in combination with information about computation.

1 INTRODUCTION

Machine learning models are instrumental in solving complex non-traditional problems such as image processing, controlling autonomous vehicles, natural language processing, and more. The power of such techniques, specifically deep learning, has inspired the development of increasingly complex and large models. To ensure their effectiveness, machine learning engineers put these models through several rounds of training, fine-tuning, and testing before deployment. As models increase in size, engineers have begun distributing the training on multiple servers in order to leverage parallelism of training tasks. Specifically, engineers, through the training environment, coordinate each server to train on a subset of the data that is disjoint from other subsets of the data that other servers handle. In this fashion, an N-fold increase in the number of servers participating in the training process will ideally lead to an N-fold speedup in training performance. In particular, a crucial metric used for evaluating training is the time-to-accuracy (TTA), a measure of the time required to train a given model until it achieves a specific accuracy [1]. Thus, the machine learning community has adopted distributed machine learning frameworks for use in their training in an effort to improve the speed of their training [2]. Tensorflow, a widely used machine learning training framework, comes equipped with a method to distribute training across multiple machines, but it is hard to use in a distributed fashion and can be slow [2, 3]. Users were consequently motivated to develop different machine learning distribution frameworks like Horovod, a flexible open source platform for distributing training from Uber.

Horovod includes a profiler named Horovod Timeline, but its information is limited in that it only displays computation time [2, 3]. A full analysis of training must measure not only computation (on each server and on its GPUs), it must also measure communication (between servers and between GPUs). In the case that there is little or no overlap between the computation and communication stages during training, this method of profiling would identify opportunities to increase the overlap between communication and computation as a means of reducing the overall TTA of a model. In addition to overlap, this method, when combined with knowledge of the inner-workings of the distribution framework at hand, exposes potential bottlenecks in the computation and communication stages. These can be used to optimize the training process for even better TTA.

As a first step to achieving this large goal, this project focuses on the communication patterns during distributed training. Specifically, we make use of five different models of varying size (highlighted in Table 1) for training with the Horovod distributed training framework. We design a measurement tool to measure inter-server communication. In particular, we measure all-to-all server communication over the network using tcpdump while training is running [4]. Later, we post-process the raw communication data to extract information about the communication. Our observations are as follows. First, we confirm empirically that Horovod establishes a ring topology between its workers according to the order they appear in the execution command. Second, we observe that inter-server TCP flows come

Model Name	Number of Parameters	Model Size (MB)
MobileNet [5]	4, 253, 864	16
DenseNet121 [6]	8, 062, 504	33
InceptionV3 [7]	23, 851, 784	92
ResNet50 [8]	25, 636, 712	98
VGG19 [9]	143, 667, 240	549

Figure 1: *Model Metadata.* The table shows information about each of the three models we used to run distributed training. Information obtained from Keras applications [10]

in small, medium, and large sizes. Thirdly, we find a direct relationship between the model size and size of the flows transmitted during each iteration of training. Finally, we find that the model size does not fully determine the iteration duration on its own, though it is a factor.

2 EXPERIMENT METHODOLOGY

First we discuss the setup of servers and the workflows we run. Then, we discuss in detail how we measure communication between the servers as they engage in distributed machine learning training.

2.1 Infrastructure and Workflows

2.1.1 Servers. To study different aspects of servers running distributed training in the cloud, we acquire and configure servers to run training on. We use Google Cloud Compute Engine virtual machines due to the large capacity of compute, memory, and GPUs. Specifically, we set up four GPU-enabled VMs running Ubuntu 18.04. We use the words server and VM interchangeably, with both pointing to a GPU-enabled Google Cloud Compute Engine instance unit that we run training on. Each server has four NVIDIA Tesla T4 GPUs, sixteen virtual CPUs, and 104GB of RAM. Atop these servers we install Tensorflow 1.13 for executing training on each GPU, and install Horovod 0.18.1 for distributing training across servers [11].

2.1.2 Workflows. We chose to run training for five different models that, together, represented a relatively wide span in the model size. The model size is determined by the number of parameters in the model. At 4, 253, 864 parameters, the smallest model, MobileNet has a model size of 16MB. The medium model, InceptionV3, has 23, 851, 784 parameters which produce a model size of 92MB. The largest, VGG19, comes with 143, 667, 240 parameters and has a model size of 549MB. This information is detailed in Table 1. We coordinated training of these models to be distributed on the four servers, each of which further divided its training responsibilities to its four GPUs. The servers ran 200 iterations of training while the current time before and after each iteration of training was recorded. We made use of one batch of synthetic data per iteration, where each batch contained 64 synthetic images, in addition to five warm up batches before actual training began. The virtual topology of communication during training forms a ring (as per ring-allreduce), where each server sends data to the next server and receives data from the previous server [2]. Open MPI is used as the underlying communication mechanism between nodes through Horovod. The training data used is synthetic, since real data and synthetic data don't affect the computation and communication during training. We ran communication measurements during the training interval, and the methods are highlighted in more detail in section 2.2.

2.2 Measuring Communication

We limit our focus to inter-server communication. Inter-server communication consists of the packets sent over the network between servers as they undergo training. We observe that the servers communicate using TCP. Therefore, we capture incoming and outgoing packets on each of the servers using the tcpdump [1] command by running it before training begins and terminating it when training ends. Each packet in the tcpdump output contains a superset of the crucial fields we need to construct a high-level view of communication. In particular, the packet's Ethernet and TCP/IP headers contain this information. Consequently, we capture only the first 150 bytes of the packet so save space.

The source/destination and address/port combinations in addition to TCP flags enable us to distinguish between different TCP connections between different servers. For example, recognizing the first and last packets of a TCP connection via the SYN/FIN TCP flag proved to be useful for flow duration delimiting. The `len` field tells us the size of the packet that is sent, and the `timestamp` field informs us of the capture time of this packet. Putting these together and cross-referencing with the time before and after training iterations, we can determine all the TCP connections between our servers, and can tell exactly when each packet was sent and the size of the data that it contained. Filtering by source/destination and address/port combination is done during post-processing, after the tcpdump capture terminates. After that, a script we wrote determines the time boundaries for each iteration of training and bins all the captured packets from each worker machine into their appropriate iteration.

[1] `sudo /usr/sbin/tcpdump -s 150`

Model Name	Packets Before	Packets During
MobileNet	10.02%	89.96%
DenseNet121	9.68%	90.29%
InceptionV3	6.44%	93.55%
ResNet50	4.81%	95.18%
VGG19	3.05%	96.95%

Figure 2: *Packet Count Binning*. The Table shows the percentage of packets which fall in different time positions of training

Model Name	Data Before	Data During
MobileNet	2.68%	97.32%
DenseNet121	2.89%	97.11%
InceptionV3	2.66%	97.34%
ResNet50	2.65%	97.35%
VGG19	2.63%	97.37%

Figure 3: *Transmitted Data Binning*. The Table shows the percentage of transmitted data via TCP packets which fall in different time positions of training

3 RESULTS

We present the findings of the experiment in three forms for each model. First, we plot markers representing the start and end of training and overlay the timestamps of packets sent by data flows during training in Figures 2 and 3. Second, we plot CDFs of iteration durations in Figure 4. Third, we plot CDFs of flow sizes in Figure 5. To generate these results, we ran distributed learning of the different models mentioned above for 200 iterations and measure the communication between the four different servers during training via system command tcpdump, which captures packets sent and received by the NIC. We additionally recorded timing iteration about training: start and end timestamp of each iteration. This helps to differentiate between communication happening in different iterations. Then, and most importantly, we used the timestamp information from each packet to determine which iteration that packet belongs to, if any.

The results show that, as expected, the flow size per iteration increases linearly with the model size. In particular, data from all models showed flow sizes around twice the size of the model (see Figures 1 and 5). Timestamp data from Figures 2 and 3 shows packets which lie outside of the training interval in all cases, even though a majority of packets are found inside the training interval. Interestingly, the median iteration duration does not directly correlate with the model size, as can be seen in Figure 4. For example, DenseNet121 and ResNet50 had nearly identical median iteration durations, at around 0.7 seconds, but the ResNet50 is larger than DenseNet121 by a factor of 3. Similarly, even though InceptionV3 and ResNet50 have approximately the same model size, 92MB and 98MB respectively, their median iteration durations differ by 200 milliseconds, with InceptionV3 taking 0.5 seconds and ResNet taking 0.7 seconds per iteration. We suspect that this disparity is due to the way GPU performance changes with the architecture of the model at hand, where GPUs can more efficiently do calculations for some models because of a simpler architecture.

One conclusion from these results show that training iterations are relatively fast when there is GPU-backed distributed training. But perhaps the most important results are as follows. First, there is a linear increase in the flow size per iteration across all models as the model size grows, which empirically confirms that the model is being sent around between servers in a ring all-reduce fashion, consistently with Horovod's claim [2]. Second, the order and nodes of the ring are fully determined by the initial run command supplied to Open MPI. Specifically, server hostnames supplied to the `mpirun` command form a list; the elements of this list form the nodes of the ring while the order of the elements in the list determine the edges of the ring. Third, and perhaps more interesting is the inconsistency between training duration and model size. We expect that this is because training efficiency is highly model-dependent.

The nature of these measurements makes it difficult to define the notion of a baseline because these numbers are likely to change from setup to setup, with swings in network throughput and latency, and GPU utilization from other running jobs. Since we do not own the Google Cloud Compute Engine hardware running this training, we had to make some assumptions about the utilization of our setup. In future iterations, we wish to replicate our results on a physical testbed where we have more control over the hardware and network setup. Since all this dedicated hardware is in the cloud, we do not know with certainty whether the GPUs are being shared by other tenants. We are also unsure if the GPUs requested for a single VM are all found on the underlying hardware of the machine supporting the VM or if they are on a separate machine. Since we deployed these VMs in the same region and zone, we assume that they are utilizing the same network. However, we cannot say with certainty whether these VMs are on the same rack or in the same cluster.

4 CONCLUSION

Through measuring information about packets sent over the network during machine learning training distributed via Horovod on servers in the cloud, we learn more about the relationship between the model, the iteration durations,

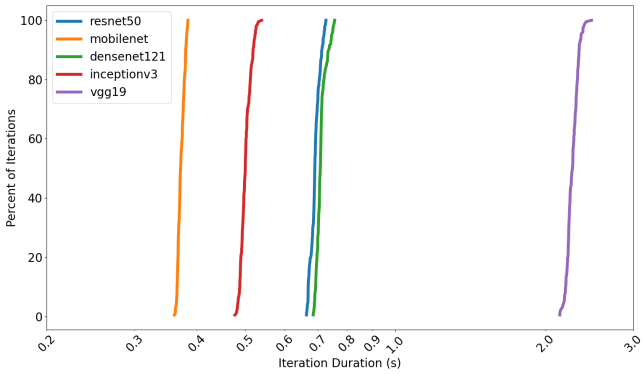

Figure 4: *CDF Iterations*. Time taken, for each model, to complete iterations of training during a distributed training session

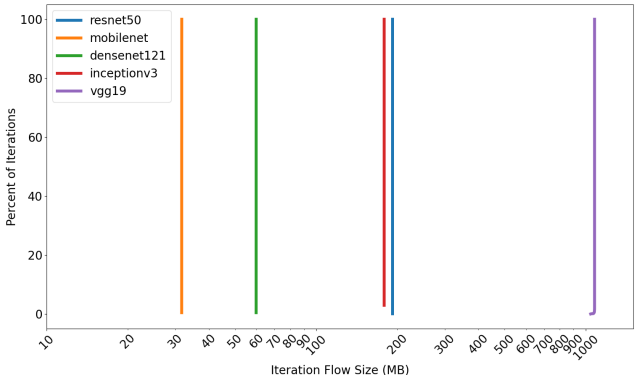

Figure 5: *CDF Flow Sizes*. Size of flows in megabytes that each model's data flows sent during each iteration of training

and the flow sizes resulting from training. We empirically confirm that Horovod runs ring-allreduce and learn that the ring is fully determined by the order of the servers provided to the training script. We observed that inter-server TCP flows come in small, medium, and large sizes where the medium flows are for control and the large flows primarily send the model. We found a direct relationship between the model size and size of the flows transmitted during each iteration of training. We found that the model size does not fully determine the iteration duration on its own, though it is a factor. Finally, not all packets that made up the large flows fell in the training interval. 90% of the packets were within the interval 10% were outside. Further, 97% of the bytes transmitted were in the interval, with 3% of bytes lying outside.

In the journey to improve TTA, this study offers a glimpse into the communication component necessary to understand the much needed overlap between communication and computation. The next steps for this research involve measuring inter-GPU communication and GPU utilization. These pieces, when put together with the communication profile highlighted in this paper's methods (see section 2.2), would then provide a fuller picture regarding communication and computation for a given training run. This study approximated the TTA by running a fixed number of training iterations, but a more realistic scenario would involve training the model to a high, predefined, specific accuracy, ergo TTA. Doing this will enable the user to observe the amount of overlap between communication and computation and find opportunities for additional overlap as a means of reducing the training time.

REFERENCES

[1] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Re, and Matei Zaharia. Analysis of dawnbench, a time-to-accuracy machine learning performance benchmark. 2018.

[2] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *CoRR*, abs/1802.05799, 2018.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[4] The Tcpdump Group. Tcpdump/Libpcap public repository. Library Catalog: www.tcpdump.org Publisher: The Tcpdump Group.

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017. arXiv: 1704.04861.

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, January 2018. arXiv: 1608.06993.

[7] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [cs]*, December 2015. arXiv: 1512.00567.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.
[9] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, April 2015. arXiv: 1409.1556.
[10] Applications - Keras Documentation.
[11] horovod/horovod. Library Catalog: github.com.