Performance Evaluation of Distributed Deep Learning: A Networking Perspective

Peng Cheng Rui Liu Takuya Kurihana

Abstract

Distributed deep learning frameworks have been widely deployed for various applications in both academia and industry. Training deep neural networks (DNNs) requires computation of various types of kernels and algorithms, such as convolutional neural networks (CNNs) and stochastic gradient descent (SGD). Moreover, computing nodes have to mutually communicate to update the weight and gradient in distributed training, but the training performance varies in different network environments (e.g., bandwidth, latency, network protocol, etc.). In this project, we evaluate the performance of various deep learning models in a distributed environment based on three factors: network latency, network bandwidth, and packet loss. The results indicate that the performance of distributed deep learning is largely depends on the network condition, subtle changes may significantly affect the performance. We believe this performance evaluation not only provides a guideline to setup suitable network environment for various training models but also points out potential future directions for developers to optimize deep learning frameworks.

1 Introduction

The high demand for analyzing and processing big data has encouraged the use of data-hungry machine learning algorithms like deep learning. Deep learning is gaining much popularity due to its supremacy in terms of accuracy when trained with a huge amount of data. However, training deep learning models like convolution neural networks is a non-trivial job; thus, many open-source deep learning frameworks are developed to increase the efficiency of Deep learning methods (e.g., TensorFlow [1] and PyTorch [12]).

Nowadays, DNNs are becoming larger and larger so that they are often trained in a distributed way. This distributed training processing can be last for days or weeks due to the complication of the training process. During the training process, large-scale distributed training requires significant communication bandwidth and low latency for gradient update/exchange between each node. Thus, a bad networking environment may add additional communication overhead, increase the overall training time or even break the whole training process. Previous works develop techniques to save networking bandwidth [5] or propose an efficient host-based communication layer to maintain stable network [10]. However, few works take an effort to evaluate how the network condition affects the performance of distributed deep learning. Therefore, we are motivated to conduct a comprehensive evaluation from a networking perspective with respect to latency, bandwidth, packet loss to investigate this issue.

2 Related Work

In this section, we discuss some existing work about distributed training for deep learning deployment [2, 4, 7, 8, 11–13, 18, 20] and classify them into two categories: protocol-specific and network-specific.

The existing works about protocol-specific mainly focus on distributed training over different protocols like MPI (Message Passing Interface) and RPC. Tal and Torsten [4] survey 80 distributed training papers using multi-node parallelism and report that over 68% of papers after 2016 among them selected MPI (including CUDA-Aware MPI) for their experiments. For example, Horovod is a distributed training framework for TensorFlow, Keras, PyTorch, and MXNet using MPI [16]. Besides MPI, NCCL (NVIDIA Collective Communications Library) and gRPC (Google RPC) are two alternatives that can be used for distributed training. Amrita et al. [11] attempted to explore scaling of TensorFlow with gRPC primitives on 512 nodes of Cori supercomputer. Their results indicate that gRPC deteriorates the training throughput because of suboptimal for utilizing interconnect bandwidth and inefficiency of gRPC on the high speed interconnect. Ammar [2] reviewed these communications libraries used for distributed training, including MPI, NCCL, and gRPC.

The network-specific works usually deploy the distributed deep learning over new network environment on top of Ethernet [19]. For example, RDMA (Remote Direct Memory Access) is one of the most widely used network environ-

ment [7]. According to the recent evaluation, distribute training over RDMA can provide up to 169% improvement in TensorFlow [20].

We further study the works which center performance evaluation for distributed training, which are considered as the most relevant works. Soheil et al. presented a comparative study of five deep learning frameworks, including Caffe, Neon, TensorFlow, Theano, and Torch from three aspects: extensibility, hardware utilization, and speed [3]. Shayan et al. analyzed the performance of three different frameworks, Caffe, TensorFlow, and Apache SINGA. They focus on the performance characteristics of the state-of-the-art hardware technology for deep learning, including NVIDIA's NVLink and Intel's Knights Landing [17]. Xingzhou et al. analyzed the performance of machine learning packages on the edges instead of on the cloud. They investigated the latency, memory footprint, and energy of TensorFlow, Caffe2, MXNet, PyTorch, and TensorFlow Lite on different edge devices [21]. The aforementioned research summarized the existing distributed deep learning frameworks from the perspectives of extensibility, hardware utilization, memory footprint, and energy. Although some of them touch the network environment for distributed training, to the best of our knowledge, none of them conduct a comprehensive performance evaluation for distributed training from the viewpoint of network condition.

3 Methodology

In this section, we detail the performance evaluation. The whole performance evaluation is based on a cluster with four nodes and each nodes has one GPU for training models, as shown in Figure 1.

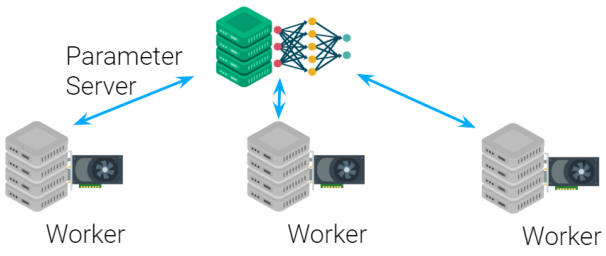

Figure 1: Diagram of our distributed training workload in Tensorflow framework. Parameter server(green) and worker servers(gray) with a GPU communicate each other every iteration to update training parameter information.

3.1 Evaluation Environment

We deploy evaluation performance on TensorFlow, the most widely used deep learning framework. We evaluate in various networking environments, including different latency, bandwidth, and packet loss. More specifically, we tested 1ms, 10ms and 100ms for latency, 250 Mbits, 500 Mbits, 1000 Mbits for bandwidth, and 0%, 1%, 10% for packet loss. We use `tc` command in Linux to realize these network configurations.

3.2 Evaluation Metric

We measure the average time of a single training step since we found some context initialization are triggered at the beginning of training process in TensorFlow. This initialization should be isolated in the evaluation since it is completely local. Also, since a single training epoch can be regarded as a series of repeating training steps and a complete training process is made with multiple epochs, measuring the single training step is sufficient to estimate the performance of the whole training process. To make the results unambiguous, we convert the time of single training step to throughput which is defined as *image/sec*.

3.3 Evaluation Workload and Dataset

We conduct a workload for image processing. The workload consists of three kinds of models, ResNet, MobileNet, and MLP (Multiple Layer Perceptron). We set the batch size to 32 as a starting point and then measured the performance when it increases to 128 and 1024, which implies the workload includes nine combinations. The models usually have an order of $O(10^9)$ parameters which have to be updated every batch. This leads a heavy communication and synchronization between nodes, and then we can evaluate the performance of different network environments accordingly.

ResNet (residual neural network) is a widely used convolutional neural network for image recognition [6], which has a substantial amount of layers by exploiting the residual block. ResNet is considerably large due to its deep architecture, namely ResNet has many parameters to update during the training process. Compared with ResNet, MobileNet is a relatively lightweight convolutional neural network for deployment in mobile devices/applications [15]. Due to their size difference, we choose ResNet and MobileNet as two representative models. Also, evaluating these two models can provide insights into network performance when training heavy and light models respectively. MLP is a classic and simple feedforward artificial neural network which only consists of an input layer, at least a hidden layer, and an output layer. This basic model can serve as a baseline in this performance evaluation.

We choose two popular datasets as training data for our workload models, MNIST [9] and ImageNet [14]. The former is a dataset of handwritten digits (0-9), and the latter is a large visual database that is widely used in the real world for image processing or visual object recognition.

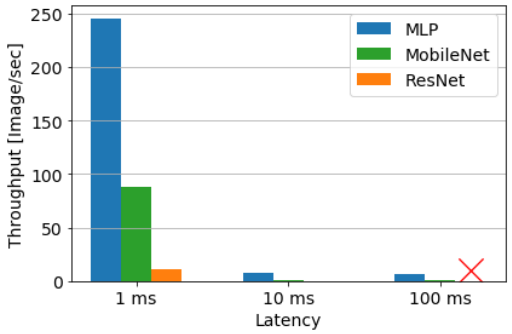

Figure 2: Results in MLP, MobileNet and ResNet models at 1, 10, 100 ms latency training with MNIST under one parameter-worker server environment. Red cross depicts a shut down session for too long training time to finialize first training step.

4 Performance Evaluation

4.1 Latency

We first explore the impact of latency against the average single step for distributing training. Figure 2 shows the results in a two-nodes distributed environment where one is the parameter server and the other is worker. We focus on three models under the configuration of 32 batch size, 1000 Mbit bandwidth and 0% packet loss. We find adding 10ms latency in the training network deteriorates the throughput performance by 1 order of magnitude. The distributed training of ResNet cannot finalize single step with 30 minutes in 100 ms case so we forcibly shut down this case, denoting red cross in the Figure 2. Here, we observe that the performance of distributed training is sensitive to network latency regardless of the model architecture.

We then evaluate the impact of batch size and vary the batch size between 32, 128, and 1024. Figure 3 compares difference of training throughput in MNIST and ImageNet. We find throughput almost increases linearly with the batch size. Also, throughput of batch size 1024 is approximately 30 times larger than that of 32 in 1ms latency case. Same as in 100 ms case, we observe the same tendency as Figure 2 that the performance of throughput larger than 1ms downgrades exponentially. As the model is light and has fewer parameters than common convolutional neural networks, users can obtain the same output in 1ms latency with 32 batch size when the network latency increases from 1ms to 100ms. In contrast to the consistency in MLP, adding $>$ 1ms latency deteriorates the throughput of ResNet and we only observe the ResNet at 1ms latency experiment scales well. In this configuration, batch size 1024 is able to process 324 image per second, whereas that of 32 is 10.9 image per second. From this experiment, we summarize that when users have to train MLP in higher network latency environment, it is better to set the batch size to 1024 or other uncommon large batch sizes. The results also indicate training efficiency cannot be improved by changing batch size at poor network bandwidth in ResNet.

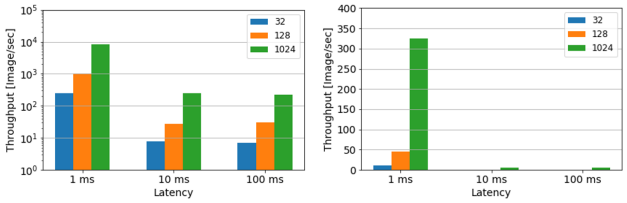

Figure 3: Results in batch size 32, 128, and 1024 at same range of latency under same configuration of Figure 2. Left: Results for MLP. Right: Results for ResNet.

We investigate the influence of latency against the distributed training with more practical situation. Here, we set 3 computing server to worker servers and one for the parameter sever. The result is shown in Figure 4, revealing that communication-heavy distributed training is more vulnerable to the network latency. We find that both MLP with 100 ms latency and ResNet with 10ms latency cannot terminate the training process after 40 minutes. From Figure 4, we can see adding one order of latency makes the entire training far slower. Moreover, throughput of both models declines as we add more worker servers for training. In the case of one parameter-worker sever with ImageNet, the average single step takes around 6s, which indicates the throughput is 5.3 image per second; while in the case of 1 parameter sever and 3 worker sever, the throughput is 1.17 image per second. This difference shows that communication and synchronization between multiple servers cause additional overhead in the training process.

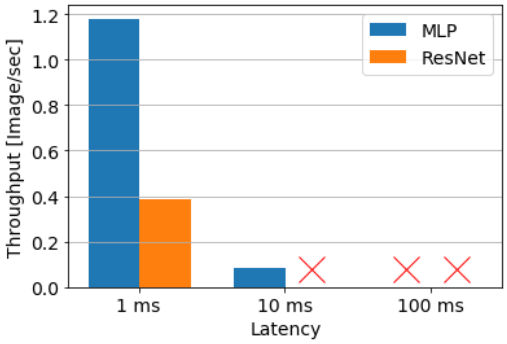

Figure 4: Results in MLP and ResNet models in 1, 10, 100 ms latency under 1 parameter with 3 worker servers.

4.2 Bandwidth

We continue to examine the impact of bandwidth to the distributing training in the same experiment set-up. Figure 5

shows the result of one parameter-worker server experiment using MNIST with batch size 32, 1ms latency and 0% packet loss. We observe that there is no severe performance degradation (4.3 percent) when we decrease the bandwidth from 1000Mbit (247 image per second) to 500 Mbit (231 image per second). However, the throughput for MobileNet decreases from 87.6 image per second to only 2.5 image per second. This difference shows that models with more parameters is more sensitive to the change of bandwidth than the simple one dimensional fully connected network. Thus, users have to allocate enough bandwidth for these convolutional neural models.

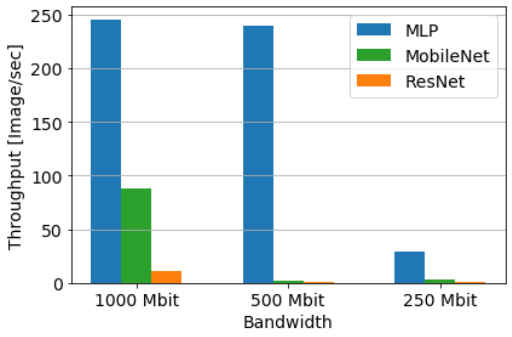

Figure 5: Results in MLP, MobileNet and ResNet models at 250, 500, 1000 Mbits training with MNIST under one parameter-worker server environment.

We also evaluate the impact of batch sizes to aforementioned models. As shown in Figure 6, the scalability of throughput is preserved when the batch size changes; thus, user can select batch size according to their available capacity of memory. Next, we extend the simulation to four-servers situation. Figure 7 reveals that the throughput is proportional to network bandwidth; more specifically, ResNet achieves 0.4 image per second, 0.2 image per sec, and 0.12 image per sec in 1000Mbit, 500Mbit, and 250Mbit, respectively. MLP is more stable when the available bandwidth is becoming less until the available bandwidth is narrow down to the 250Mbit. The reason behind this is that communication time is far greater than the actual computation time on GPU so that the difference tendency we see in the one parameter-worker result with MNIST becomes negligible.

4.3 Packet Loss

We then analyze the influence of packet loss against the average single step for TensorFlow distributed training. We first conduct an experiment using MNIST on one parameter sever and one worker, and set the latency to 1ms and bandwidth to 1000Mbits, as shown in Figure 8. In this experimental set-up, MLP, MobileNet and ResNet have similar degradation behaviors, which degrades approximately 10% as packet loss increases 1%.

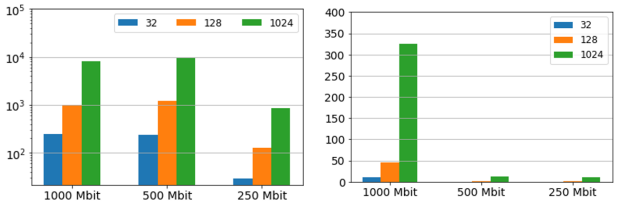

Figure 6: Results in batch size 32, 128, and 1024 at same range of bandwidth under same configuration of Figure 5. Left: Results for MLP. Right: Results for ResNet.

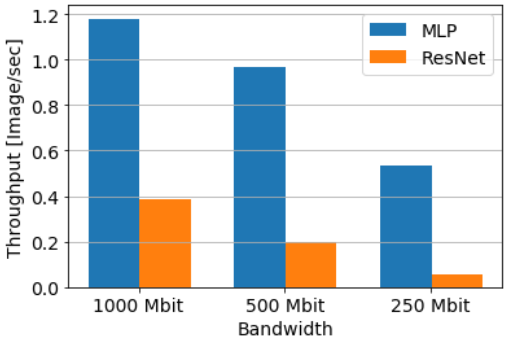

Figure 7: Results in MLP and ResNet models in 250, 500, 1000 Mbits bandwidth under 1 parameter with 3 worker servers.

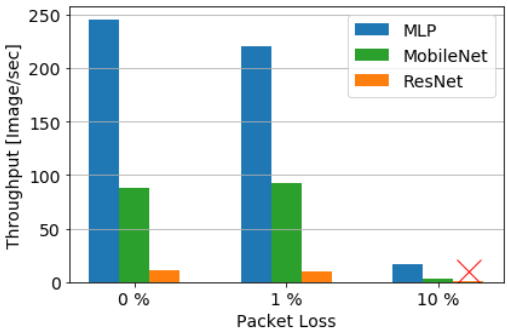

Figure 8: Results in MLP, MobileNet and ResNet models at 0%, 1%, 10% packet loss training with MNIST under one parameter-worker server environment.

To further evaluate the relationship between packet loss and throughput in distributed environment, we measure the throughput with packet loss of 0%, 1% and 10%, varying the batch size between 32, 128, and 1024. The results are shown in Figure 9 with MNIST in the left and ImageNet in the right. We can see the change of throughput is approximately proportional to the change of batch size. Also, the throughput in the ImageNet is around 30 times less than the throughput in MNIST since ImageNet has more parameters to be synchronized in each step.

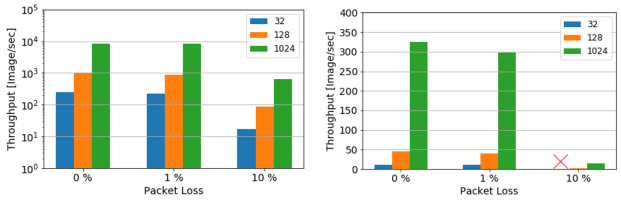

Figure 9: Results in batch size 32, 128, and 1024 at same range of packet loss under same configuration of Figure 8. Left: Results for MLP. Right: Results for ResNet.

We then investigate the impact of packet loss using a more practical scenario using ImageNet and with one parameter server and three workers. The result is shown in Figure 10, indicating the different behaviors between MLP and ResNet; For MLP, each percent increase leads to a 3% throughput decrease while in ResNet, throughput degrades around 30%. Therefore, we conclude ResNet is more sensitive to the change of packet loss. Moreover, we observed when packet loss is greater than 10%, the performance of both models degrades drastically; the throughput for MLP decreases by more than 95%. For ResNet, the training process is still in the very first step after waiting 20 minutes, which is much longer than the cases having lower packer loss. Therefore, we believe the total training process takes a long time and is unrealistic to use. Those results is similar as we expect since ResNet has substantial amount of layers and has more parameters to be synchronized.

4.4 Fault Tolerance

Training a model may take several hours or days, even using a large number of machines together. Even worse, in some cases, we may not have the complete and same training data for the whole training process. Thus, fault tolerance is needed to deal with failure or preemption. Unlike frameworks like Spark, instead of on an individual worker node failing, the recovery is at the checkpoint, which is a stateful computation that makes it resilient to hardware failures by periodically saving its program state to persistent storage. One of the workers, named the chief worker, initializes the process, coordinates other works, counts the number of training steps completed,

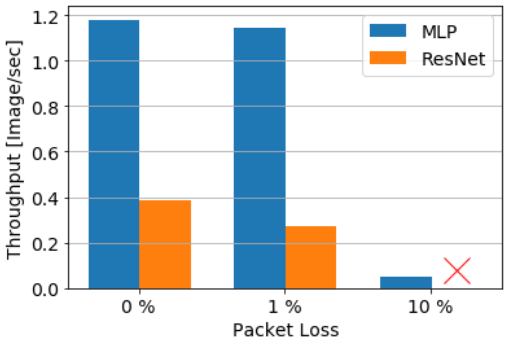

Figure 10: Results in MLP and ResNet models at 0%, 1%, 10% packet loss under 1 parameter with 3 worker servers.

and saves and restores model checkpoints to recover from failures. Since all the workers are kept in sync in terms of training epochs and steps, other workers would need to wait for the failed or preempted worker to restart to continue. If the chief worker itself shuts down, training will need to be restarted from the most recent checkpoint. Therefore, TensorFlow has weak support for classic fault-tolerance.

5 Conclusion

In this paper, we have conducted a comprehensive performance evaluation on distributed deep learning frameworks from a network perspective. We laid out three underpinning concepts behind the evaluation of network performance, i.e. latency, bandwidth and packet loss. To measure the performance, we implement MLP, MobileNet and ResNet, and train them with different batch sizes in a distributed environment which is cluster with 4 machines. The results showed that (1) distributed deep learning is not always beneficial due to overheads caused by communication between nodes; (2) the computation-intensive model like convolutional neural network are more sensitive to the change of network condition; (3) fault tolerance is lack in the mainstream deep learning frameworks, and users need to store the intermediate training status using checkpoint mechanism.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Ammar Ahmad Awan, Ching-Hsiang Chu, Hari Subramoni, Dhabaleswar K. Panda, and Jeroen Bédorf. Scalable distributed dnn training using tensorflow and cuda-

aware mpi: Characterization, designs, and performance evaluation. *IEEE/ACM Internal Symposium on Cluster, Cloud and Grid Computing*, pages 498–507, 2018.

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of deep learning software frameworks. *arXiv preprint arXiv:1511.06435*, 2015.

[4] Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 52(65):1–43, 2019.

[5] Song Han and William J. Dally. Invited: Bandwidth-efficient deep learning. *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, 2018.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, October 11-14, 2016*, pages 630–645.

[7] Chengfan Jia, Junnan Liu, Xu Jin, Han Lin, Hong An, Wenting Han, Zheng Wu, and Mengxian Ch. Improving the performance of distributed tensorflow with rdma. *International Journal of Parallel Programming*, 46:674–685, 2018.

[8] Thorsten Kurth, Mikhail Smorkalov, Peter Mendygral, Srinivas Sridharan, and Amrita Mathuriya. Tensorflow at scale - mpi, rdma and all that. 2018.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Luo Mai, Chuntao Hong, and Paolo Costa. Optimizing network performance in distributed machine learning. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 15)*, Santa Clara, CA, 2015.

[11] Amrita Mathuriya, Thorsten Kurth, Vivek Rane, Mustafa Mustaf, Lei Shao, Debbie Bard, Prabhat, and Victor W Lee. Scaling grpc tensorflow on up to 512 nodes of cori supercomputer. *Neural Information Processing Systems 2017 Workshop: Deep Learning At Supercomputer Scale*, 2017.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[13] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. 2019.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[15] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520.

[16] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. 2018.

[17] Shayan Shams, Richard Platania, Kisung Lee, and Seung-Jong Park. Evaluation of deep learning frameworks over different HPC architectures. In *IEEE International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, June 5-8*, pages 1389–1396, 2017.

[18] Shaohuai Shi and Xiaowen Chu. Performance modeling and evaluation of distributed deep learning frameworks on gpu. 2017.

[19] Shaohuai Shi, Qiang Wang, Kaiyong Zhao, Zhenheng Tang, Yuxin Wang, Xiang Huang, and Xiaowen Chu. A distributed synchronous SGD algorithm with global top-k sparsification for low bandwidth networks. *CoRR*, abs/1901.04359, 2019.

[20] Jilong Xue, Youshan Miao, Cheng Chen, Ming Wu, Lintao Zhang, and Lidong Zhou. Fast distributed deep learning over RDMA. In *European Conference on Computer Systems (EuroSys), Dresden, Germany, March 25-28, 2019*, pages 44:1–44:14, 2019.

[21] Xingzhou Zhang, Yifan Wang, and Weisong Shi. pcamp: Performance comparison of machine learning packages on the edges. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, 2018.