Performance Comparison and Code Refactoring Analysis of Distributed Tensorflow and Horovod DNN Training with Custom Training Loop API

1st Matin Raayai Ardakani
Department of Electrical and Computer Engineering
Northeastern University
Boston, MA
raayaiardakani.m@northeastern.edu

***Abstract*—One of the main challenges of training Deep Neural Networks is the large amount of data, memory and computation required to achieve satisfactory inference results while meeting efficiency and timing requirements, which has led to the creation and adoption of distributed deep learning frameworks in both industry and academia; However, leveraging these tools still seems to be out of the reach of deep learning practitioners and researchers since they require a working knowledge of distributed systems and high-performance computing.**

Introduced by Uber Technologies in 2018 to address this need, Horovod is a distribution framework that aims to distribute the training/inference process of popular DNN frameworks including Google's Tensorflow across multiple workers in a cluster with minimal code refactoring [1]. At that time, Tensorflow supported distributed training implementations of DNNs [2]; However, Sergeev and Balso [1] demonstrated that Horovod outperforms Tensorflow in terms of training speed and scalability in Uber's production setting.

In this paper, we recreated the study done by Sergeev and Balso [1] to best of our ability by attempting to write 2 distributed versions of a single-GPU VGG-16 training script on the CIFAR-100 dataset using the Custom Loop API, one using only the Tensorflow Strategy API and the other only using Horovod. We have found that since Horovod's release, Tensorflow developers have made a significant effort to make the migration of single-GPU code to different distributed scenarios (called strategies) as seamless as possible for the Keras-API. However, the framework currently fails to provide any useful support for migration of a single-GPU Custom Loop API code to different strategies except single host MirroredStrategy [2], which renders these efforts almost useless. This is all while Horovod can use its minimally refactored code for single-host and multi-host scenarios with minimal changes to its launching command.

In terms of training speed, we have found that Horovod achieves equal performance to Distributed Tensorflow's Mirrored Strategy in a single-host, multi-GPU scenario on a node with 4 NVIDIA V100 SXM-2 GPUs on Northeastern University's Discovery Cluster. Horovod's training speed is analysed and its weak scaling is found to be close to linear across different hardware configurations in the Discovery cluster.

I. Introduction

One of the main challenges of training Deep Neural Networks is the large amount of data, memory and computation required to achieve satisfactory inference results while meeting efficiency and timing requirements. This has motivated the machine learning and distributed and high-performance computing communities to work together to accelerate the development and testing cycles of DNNs in both research and production. The endeavours made in this area has come a long way since the revival of DNNs in the beginning of the previous decade and has found its way to the popular deep learning frameworks widely available today; However, leveraging these features still seems to be out of the reach of deep learning practitioners and researchers since it requires them to put a significant amount of time and effort to learn about their inner-workings.

In 2018, Uber Technologies open-sourced a distributed DNN framework called Horovod currently hosted by the Linux Foundation, which was the result of the company's effort to address this problem, by building on previous efforts made by Facebook and Baidu [1]. Horovod can use both MPI and Facebook's Gloo to distribute the training process of several popular DNN frameworks, including Google's Tensorflow.

At the time, Tensorflow supported distributed training implementations of DNNs [2]; However, the short tutorials, incomplete API documentation and implementations of Tensorflow always proved to be a challenge in leveraging them. Horovod's significance was shown by Sergeev and Balso [1], that unlike the common parallelism granularity-level trade-offs seen in leveraging distributed systems, it is indeed possible to achieve scalability while keeping single-GPU code refactoring to a minimum in training DNNs.

Since the release of Horovod, Tensorflow has gone under drastic changes to improve performance and usability and now supports auto differentiation of custom-defined user computation in real-time, called "Eager Execution" [3], while still supporting the "Graph Execution", still used with the "Keras API". Writing custom loops in training DNNs (refereed to as Custom Loop Training API by Tensorflow) is widely used by the practitioners on the bleeding edge of machine learning since it allows them to define very complicated training scenarios in newer DNN architectures like Generative Adverserial Networks.

To investigate whether this rather rare scenario in high-performance computing still holds, we decided to recreate the

study on different hardware configurations with the Custom Training Loop API.

II. Distributed Training Schemes Supported By Each Framework

A. Distributed Tensorflow

Tensorflow, regardless the API used, aims to support the following distributed training schemes according to [2]:

1) **MirroredStrategy** that creates one replica of the model per GPU device on a single host. Each variable in the model is mirrored across all the replicas. Together, these variables form a single conceptual variable called MirroredVariable. These variables are kept in sync with each other by applying identical updates.
2) **CentralStorageStrategy** does synchronous training as well, with the difference that variables are not mirrored, instead they are placed on the CPU and operations are replicated across all local GPUs. If there is only one GPU, all variables and operations will be placed on that GPU.
3) **MultiWorkerMirroredStrategy** is also very similar to MirroredStrategy. It implements synchronous distributed training across multiple workers, each with potentially multiple GPUs. Similar to MirroredStrategy, it creates copies of all variables in the model on each device across all workers.
4) **ParameterServerStrategy** where some machines are designated as workers and some as parameter servers. Each variable of the model is placed on one parameter server and computation is replicated across all GPUs of all the workers.
5) **TPUStrategy** supports deploying the single-GPU code on Google TPUs.
6) **OneDeviceStrategy** is a debugging strategy that mimics the single-GPU code.

B. Horovod

Horovod's documentation does not provide a clear naming convention for different distribution schemes that can be used with Horovod; However, from the code examples available, it seems that Horovod is meant to support the MultiWorkerMirroredStrategy and MirroredStrategy of Tensorflow all in a single piece of code. Hence, to do a fair performance comparison, these two Distributed Tensorflow Strategies along with Horovod's default scheme received our undivided attention.

It is worth noting that since Horovod is written on different distribution back-ends such as OpenMPI or GLOO, the training code can be written such that it supports more complicated schemes as well.

III. Refactoring Experiment and "Rewriting the Benchmark Code"

A. Starter Horovod Code

Investigation into the Horovod repository showed that to produce the numbers in [1], the authors leveraged the official High-Performance CNN benchmarks provided by Tensorflow [4]. However, there were two main issues with this repository which would have not been a problem if we were benchmarking the Keras API:

1) The code was no longer actively maintained.
2) The code was written for Tensorflow 1 and in order to be executable using Tensorflow 2, it's new behavior was to be disabled.

Another noteworthy high-performance benchmark is [5], which was written for Tensorpack, an interface for Tensorflow, which was not the focus of this study.

Thankfully, the Horovod repository included new synthetic benchmark for Tensorflow 2's Custom Training Loop API that mimicked what was done in [1]. We used that piece of code as the started code.

The starter code had the following issues:

- It generated a single synthetic batch that would be used in every single pass of the program, which did not simulate the tf.Dataloader class in real-life training.
- The benchmark assumed that each GPU will show roughly the same performance; Therefore, instead of measuring the execution time across all GPUs, the root rank of the training's statistics were used instead of reducing the statistics across all the devices.

These problems were fixed before migrating the code to its single-GPU version by taking the following actions:

- Instead of synthetic data, the CIFAR-100 dataset was used as training data and pre-processed in real-time by the Tensorflow Dataset class before being fed to the model.
- Using Horovod's allreduce API, the total throughput of the program was reduced across all nodes before being printed for the user.

Additionally, the code was modified to use only the standard Keras VGG-16 architecture with 100 labels and max-pooling. VGG-16 was selected for this study due to its poor scaling performance in [1]. Some minor modifications were made to the optimizer used (Adam instead of SGD) and the benchmark command-line arguments to give more control over the benchmarking process, which was measuring a single forward and backward propagation of training.

B. Single-GPU Tensorflow

Migration of the Horovod code to single-GPU TF proved to be very easy, with the only relevant modification being the replacement of the Horovod distributed gradient tape with the regular TF tape. This showed that Horovod indeed achieves its goal of providing parallelism with minor modifications in the single-GPU code.

C. MirroredStrategy in Distributed Tensorflow

To write the equivalent MirroredStrategy benchmark, we went on the same adventure as Sergeev and Balso [1] in Tensorflow documentations. Compared to what they experienced, creating the Custom Loop API MirroredStrategy benchmark proved to be quiet easy. However, the Single-GPU code had to undergo these major modifications:

1) Instead of using the original tf.Dataset, an "Experimental Distributed Dataset" from the strategy object was used to distribute the dataset among the GPUs, with each GPU getting the fraction of the global batch size. The API for the distributed dataset is somewhat different than a regular TF Dataset object and one has to call specialized functions to access their per-device values/properties.
2) Any function or object related to distributed training (model, optimizer, etc) that was not created using the strategy object had to be declared in the strategy object's scope.
3) A per-replica loss function was written that calculated the loss on a single replica by dividing the intermediate result by the global batch size.
4) A distributed benchmark step function was written that would first have the strategy object run the original benchmark step from the single-GPU code (with per-replica loss) and then reduce the per-replica loss results to sync it across all devices.

Compared to the Horovod version, more modifications were made to the single-GPU code and some of them required the knowledge of how the underlying reduction operations and data distribution work. Compared to TF1, however, these modifications seem very minor.

D. MultiworkerMirroredStrategy in Distributed Tensorflow

Transforming the MirroredStrategy code to support MultiworkerMirroredStrategy was as simple as reassigning the strategy object. However, a closer look at [2] showed that Custom Training Loop API is currently not supported for this strategy and is scheduled to be added in subsequent versions of TF. Therefore, in the multi-host scenario, Horovod is declared winner by default.

IV. Modifications to the Original Proposal

The Horovod benchmarks were done on Uber computing resources with 512 NVIDIA Tesla GPUs, with each node having 4 GPUs, TCP (25 GB/s) and RDMA interconnects available [1]; Originally, this study was to be recreated on the NVIDIA Pascal P100 or NVIDIA Kepler K80 nodes on the multigpu partition of Northeastern University's Discovery cluster to be as close as possible to [1]. However, due to those nodes being occupied, the study was instead recreated on the NVIDIA V100-SXM2 nodes on the same partition, which proved to be a better choice. Furthermore, due to the partition having a 12 GPU per user limit, the performance analysis was done in a smaller scale than originally proposed.

After experimenting with creating training scripts for all scenarios and finding Horovod to be winner by default in the multi-host realm, instead of comparing the validation accuracy of distributed TF and Horovod (which were identical in terms of logic in the first place), we opted in for measuring the training speed of Horovod across different combinations of number of hosts and number of GPUs to demonstrate the flexibility of Horovod while maintaining simplicity. In future studies, however, it is worth investigating how these two training schemes compare to each other in terms of final testing outcome and can be done with very minor modifications to the training scripts provided.

TABLE I
Performance Comparision Results Between Distributed Tensorflow and Horovod on a Single Node.

# GPUs	Horovod (imgs/sec)	Distributed Tensorflow (imgs/sec)
1	272.445 $\pm$25.582	256.676 $\pm$33.711
2	472.425 $\pm$26.247	458.816 $\pm$2.550
3	702.127 $\pm$26.435	694.008 $\pm$5.072
4	924.093 $\pm$66.829	929.747 $\pm$6.866

V. Hardware Configuration

As mentioned earlier, the experiments were done on the multigpu partition of the discovery cluster, with each node of the experiment having 4 NVIDIA V100-SXM2 attached. The interconnect between each node in that partition is Intel Ethernet 10G 2P X520 with a throughput of 10 GB/s. Each node had an Intel(R) Xeon(R) Gold 6132 CPU running at 2.60GHz.

VI. Software Configuration

The CUDA version used in this experiment was 10.0 with CUDNN 7, latest available on the Discovery Cluster. To enhance GPU reduction operations, NVIDIA NCCL library local installation version 2.5.6-1 was installed locally.

The Tensorflow version used was 2.0.0 and the Horovod version 0.19.0, with Horovod being compiled for Tensorflow, OpenMPI, GLOO, and NCCL support.

The operating system running on each node was CentOS7 and each node was configured to have no-authentication SSH to each other for TCP connection during run-time.

VII. Experiments and Results

First, to compare the performance of Distributed Tensorflow and Horovod, we ran the benchmark script on a single node on 1 to 4 GPUs with 2 epochs, 2000 iterations through the dataset for each replica and a batch size of 64. Each image of the dataset was a Float Tensor of shape (224, 224, 3). Table I shows the results of this benchmark.

As it can be seen from the table, the average training step performance of Distributed Tensorflow and Horovod are almost identical; Therefore leveraging Horovod on a single host seems to be the better option when adding in the code refactoring effort to the equation.

As for Horovod itself, we ran the benchmark script with the same parameters as before with different combinations of number of nodes and number of GPUs as was permitted on the partition. Table II shows the raw results and figures 2 and 1 show the weak scaling of Horovod in this benchmark.

As it can be seen in the figures, Horovod's weak scaling in small scale applications is almost linear, which is a great news for Machine Learning researchers that want to quickly prototype their models but don't want to write complicated

TABLE II
HOROVOD TRAINING BENCHMARK RESULTS ON A COMBINATION OF NUMBER OF NODES AND GPUS)

	1 Node	2 Nodes	3 Nodes	4 Nodes
1 GPU	272.445 ±25.582	472.425 ±26.247	702.127 ±26.435	924.093 ±66.829
2 GPUs	451.698 ±17.742	910.767 ±42.640	1321.872 ±86.017	1768.105 ±171.072
3 GPUs	677.059 ±26.387	1300.992 ±89.301	1956.980 ±130.057	2436.152 ±187.640
4 GPUs	894.088 ±41.936	1734.713 ±107.277	2428.542 ±189.716	
5 GPUs	1114.151 ±52.754	1855.495 ±136.407		
6 GPUs	1332.648 ±68.411	2385.434 ±197.324		
7 GPUs	1487.610 ±58.900			

Fig. 1. Weak Scaling of Horovod With Respect to Number of Hosts

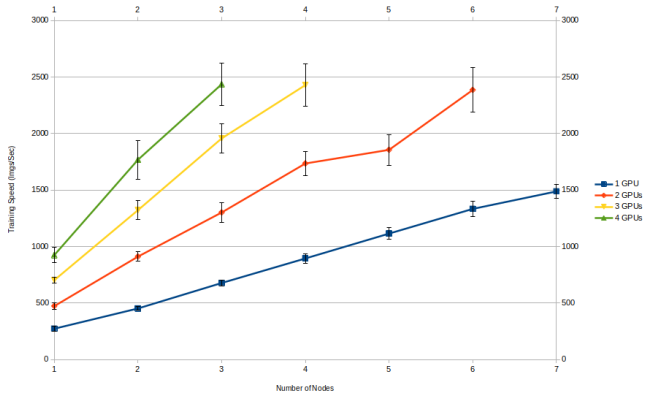
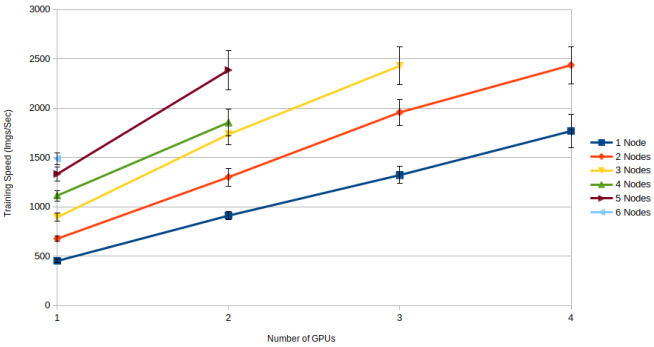


Fig. 2. Weak Scaling of Horovod With Respect to Number of GPUs



distributed versions of their code. The figures also shows the communication penalty of distributing resources across more hosts in contrast to gathering them all in a single host, something we witnessed in class very often. In other words, having more GPUs attached to a single host results in a significantly better performance than having them spread across multiple nodes.

VIII. CONCLUSION

We witnessed that Horovod, in the scale of our study does live up to its name and provides significant performance benefits over single-GPU Tensorflow and performs as well as Mirrored Strategy distributed Tensorflow while not requiring a fine granularity of parallelism. However, this study should not end here as there are many avenues of future work. Some of them are the following:

1) How Horovod compares to higher-level interfaces of Tensorflow such as Tensorpack and Tensorflow Mesh.
2) How Horovod compares to Distributed Tensorflow when using advanced optimization techniques during training such as adaptive learning rate.
3) How Horovod compares to other frameworks in other classes of DNNs such as Recurrent Neural Networks or Generative Adverserial Networks with more complicated back-propagation schemes.

REFERENCES

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

[2] Distributed training with tensorflow: Tensorflow core, . URL https://www.tensorflow.org/guide/distributed_training.

[3] Xuechen Li. Code with eager execution, run with graphs: Optimizing your code with revnet as an example. URL https://blog.tensorflow.org/2018/08/code-with-eager-execution-run-with-graphs.html.

[4] Tensorflow benchmark repository. URL https://github.com/tensorflow/benchmarks.

[5] Tensorpack benchmark repository, . URL https://github.com/tensorpack/benchmarks.