Executing and Testing Distributed Model Training on Cloud Platforms with Horovod

Aashish Ghimire
Department of Computer Science
University of South Dakota
Vermillion, USA
aashish.ghimire@coyotes.usd.edu

Sandeep Chataut
Department of Computer Science
University of South Dakota
Vermillion, USA
sandeep.chataut@coyotes.usd.edu

Sabin Adhikari
Department of Computer Science
University of South Dakota
Vermillion, USA
sabin.adhikari@coyotes.usd.edu

***Abstract*—Processing enormous volumes of data is typically required for a deep learning model. It is not feasible to store and handle the dataset on a single system if it is larger than several terabytes and is required for model training. Additionally, preparing those data takes a significant amount of time. Thus, we plan to investigate a distributed data management system in which preprocessing is carried out across clusters of distributed datasets. After that, batch datasets will be obtained from each of those distinct data clusters and used to train a model on a Master Node. We plan to look at the suggested system's viability for training models and compare its results to those of the conventional approach. We plan to communicate with the cluster using the Horovod platform. Tensorflow, a popular deep-learning framework, will be used for model training. We will properly investigate the system's scalability in terms of the number of clusters it can support and possible issues. This experiment aims to contribute valuable insights into the domain of distributed machine learning, highlighting the balance between computational resources and training efficacy in large-scale deep learning projects.**

***Index Terms*—Deep learning, Tensorflow, Horovod.**

I. Introduction

The current interest in deep learning was significantly spurred by breakthrough research results, which made it possible to build models to solve complex tasks, such as computer vision, speech recognition, natural language processing, etc. The massive parallel processing power of graphics processing units (GPUs) has been largely responsible for the recent successes in training deep learning models [1]. Along with the development of libraries, the decrease in the cost of hardware, and the wide range of cloud offerings, in the future, we can predict an increase in the development of distributed machine learning allowing students, researchers, and institutions to improve their models and their results. We plan to implement a straightforward methodology, briefly describing the most used frameworks that allow a comprehensive view of this subject at the same time allowing the reader to locate himself temporally and technologically in this frame. Our research not only presents a Distributed Machine Learning implementation but also explains it in a simple way, focusing on the fundamental concepts and guiding readers to use the tools they are most comfortable with in their own implementation.

Large data sets are ideally suited for deep learning algorithms, and training deep learning models also requires a lot of computing power. Training such models on large datasets to convergence can easily take weeks or even months on a single GPU [1]. GPUs and TPUs are readily available on a pay-per-use basis or for free (Google Colab for instance). However, it is a system with several GPU cores sufficient to train sizable models. Technically, sure, but training the model can take weeks. How can we therefore shorten the training period? Scale-up or scale-out methodologies can be used to resolve any scaling challenge. In this project, we will learn how to use scaled-out, or more precisely how to implement distributed machine learning techniques on the Cloud. We will explore how to use Cloud and Horovod to go from a Jupyter notebook phase—the fastest approach to developing machine learning models—to a production-ready training script that can run on a cluster of GPUs.

II. Related Work

The concept of distributed model training in machine learning, a cornerstone of our project, has evolved through significant research contributions. Early foundational work in distributed computing for machine learning has established the groundwork for parallelizing computational tasks in neural networks. Subsequent advancements, such as the development of Horovod by Uber [2], marked a pivotal step in simplifying and accelerating distributed deep learning by integrating efficiently with TensorFlow, Keras, and PyTorch. Studies like Krizhevsky's work on data parallelism [3] further expanded the understanding of handling large-scale model training, focusing on synchronization efficiency and optimal distribution strategies.

In addition to algorithmic advancements, the integration of cloud computing in machine learning is revolutionizing distributed computing by offering scalable and cost-effective solutions. This shift significantly influenced our project's reliance on cloud platforms. Recent works, including benchmarking studies for scalability and performance [4], have been instrumental in guiding the development of robust and efficient distributed learning systems. These collective insights from past studies have been critical in shaping the methodologies and implementation approaches of our project.

III. METHODS

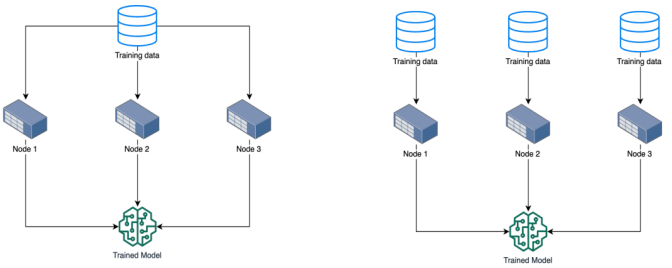
A. Data Preparation

We use The MNIST database, which contains 60,000 training images and 10K testing images. The dataset was evenly distributed among the available GPUs, with the training process spanning 24 epochs. The primary objective was to assess the system's performance across different hardware configurations, specifically using 1, 2, and 4 GPUs.

B. Parallelism Strategies

Modern distributed VLS deep learning is divided mainly into two categories: data parallelization and model parallelization. As implied by their names, data parallelization is used to distribute data on different devices for parallelization, while model parallelization is to distribute parts of the model on different devices [5]. When we have a huge model that cannot fit into memory, we use distributed ML for "Model Parallelism". This only works for specific models that can be distributed onto multiple machines. So you have each node working on part of the model and then you combine the results from all nodes to end one iteration. Each model instance operates on a subset of the data. The Model parallel method deploys a layer (or group of layers) of the model on one node of a cluster, and the entire dataset is copied to the node; each node trains on the entire dataset [6].

In the other case where we have lots of training data, we use a concept called "Data Parallelism" where you distribute chunks of training data to various nodes. Each node has the complete model but only works with part of the training data. You then need a way to combine the parameters from each node to go to the next iteration and eventually converge to a solution [7]. In Our implementation, we use this method, data parallelism.



A. Model Parallelism B. Data Parallelism

Fig. 1. Model and data parallelism [7]

C. Centralized and decentralized training

The communication between the nodes is critical in this type of architecture; it's critical to define how the parameters are initialized and how the weights/biases are updated. There are two methods of communication. A parameter server is a node or group of nodes in a centralized communication pattern that is responsible for synchronizing the model parameters. The approach has the advantage of being simple to synchronize the model parameters; however, the parameter server can become a bottleneck for a large cluster. It also serves as a single point of failure. Of course, by introducing multiple parallel servers and ensuring proper storage redundancy, the bottleneck problem can be mitigated to some extent [1]. To update the parameters, each node in the decentralized communication pattern communicates with every other node. The benefit of this approach is that peer-to-peer updates are faster, sparse updates can be made by exchanging only what has changed, and no single point of failure exists.

D. Tools

To run a Distributed training we will use Google Cloud using frameworks like PyTorch [8], and TensorFlow. Google Cloud Platform provides infrastructure as a service, platform as a service, and serverless computing environments. Tensorflow's distributed training support both centralized and decentralized training methods, if you already have a notebook using distributed TF you can easily import it into Google Cloud. In our implementation, we will use Horovod(To ensure the efficient distribution of the training independently of the deep learning framework used [6]). The computational experiments were conducted on a Google Cloud virtual machine configured with 8 vCPUs, 30 GB RAM, and 4 NVIDIA T4 GPUs, as detailed in the system specifications (see Fig. 2).

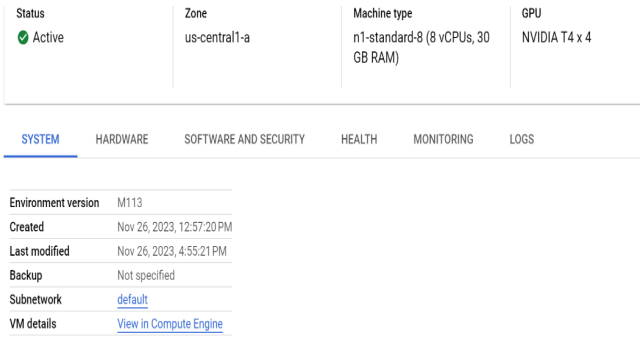


Fig. 2. Google Cloud configurations

The architecture of our implementation is shown in Fig. 3.

IV. RESULTS AND DISCUSSION

A. Performance Metrics

We use The MNIST database, which contains 60,000 training images and 10K testing images. The dataset was equally split among the number of GPUs. Tests were done using 1, 2, and 4 GPU. Execution times and accuracy were recorded for setups using 1, 2, and 4 GPUs. After running 24 epochs on the MNIST dataset, the execution time on each GPU was as shown in Table I.

B. Scalability and Efficiency

The results clearly demonstrate the scalability and efficiency benefits of using multiple GPUs. As the number of GPUs increased, the execution time for training the model significantly

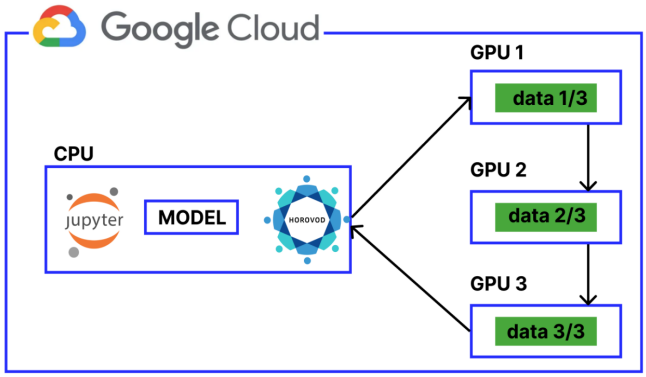


Fig. 3. Project Architecture

TABLE I
GPU NUMBER AND THEIR CORRESPONDING EXECUTION TIME AND ACCURACY

No. of GPU	Time(Average)	Accuracy
1 GPU	4s 9ms	0.9965
2 GPU	2s 11ms	0.9956
4 GPU	1s 13ms	0.9953

decreased. This indicates that the distributed system effectively utilizes the additional computational resources.

C. Accuracy Trade-off

The slight decrease in accuracy with more GPUs indicates a trade-off between speed and model precision. This could be due to factors such as the efficiency of data synchronization across GPUs or the nuances of parallel processing affecting model convergence.

D. Optimization Potential

The results suggest room for optimization in the distributed training process. Strategies such as fine-tuning data parallelism techniques or improving synchronization mechanisms might help in maintaining or even enhancing accuracy while reducing training time.

E. Resource Utilization

The reduced execution time with more GPUs underscores the potential for substantial gains in resource utilization efficiency. This is particularly relevant for large-scale deep learning tasks where time and resource efficiency are critical.

V. Conclusion

The exploration and experimentation of distributed model training using the Horovod platform and TensorFlow, as applied to the MNIST dataset, have yielded significant insights into the dynamics and potential of distributed systems in deep learning environments. The project successfully demonstrates the effectiveness of leveraging multiple GPUs in a distributed setup, highlighting key aspects such as scalability, execution efficiency, and resource utilization. A critical observation from the study is the inverse relationship between execution time and the number of GPUs employed. The substantial reduction in training time with the increase in GPU count underscores the scalability and efficiency benefits of distributed computing in handling large-scale deep learning tasks. This finding is particularly relevant in the context of ever-growing data sizes and the computational demands of modern deep learning models.

However, the experiment also revealed a nuanced trade-off between execution speed and model accuracy. The slight decline in accuracy with an increased number of GPUs points to the complexities inherent in parallel processing and data synchronization in distributed environments. This aspect suggests an area for further research and development, particularly in optimizing data handling and processing techniques to enhance both the speed and accuracy of distributed training systems. Overall, the project underscores the transformative potential of distributed machine learning systems, especially in cloud-based environments. It provides a valuable foundation for future research in this area, emphasizing the importance of developing more sophisticated methods for managing the challenges of scalability, efficiency, and model fidelity in distributed settings. As machine learning continues to evolve, the insights gleaned from this project will undoubtedly contribute to the advancement of distributed training methodologies, paving the way for more efficient and effective machine learning solutions.

References

[1] M. Langer, Z. He, W. Rahayu, and Y. Xue. "Distributed Training of Deep Learning Models: A Taxonomic Perspective". IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 12, pp. 2802–2818, Dec. 2020. DOI: 10.1109/TPDS.2020.3003307. IEEE.

[2] A. Sergeev and M. Del Balso. "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow". 2018. ArXiv preprint abs/1802.05799.

[3] A. Krizhevsky. "One Weird Trick for Parallelizing Convolutional Neural Networks". 2014. ArXiv preprint abs/1404.5997.

[4] S. Shi, Q. Wang, P. Xu, and X. Chu. "Benchmarking State-of-the-Art Deep Learning Software Tools". 2017. ArXiv preprint abs/1608.07249.

[5] H. Bai. "Modern Distributed Data-Parallel Large-Scale Pre-training Strategies For NLP models". 2022. ArXiv preprint abs/2206.06356.

[6] Author Unknown. "How to Train Your Deep Learning Models in a Distributed Fashion". Towards Data Science, 2023. Available at: https://towardsdatascience.com/how-to-train-your-deep-learning-models-in-a-distributed-fashion-43a6f53f0484.

[7] Author Unknown. "Distributed Machine Learning – Part 2 Architecture". Studytrails, 2021. Available at: https://www.studytrails.com/2021/02/10/distributed-machine-learning-2-architecture.

[8] Author Unknown. "Distributed Training: Guide for Data Scientists". Neptune.ai, 2023. Available at: https://neptune.ai/blog/distributed-training.