Performance Evaluation of Distributed Deep Learning Training on Low-Bandwidth Networks

Rayan Raddatz de Matos[1], **Marcelo Cardoso Oliveira Gulart**[1], **Kenichi Brumato**[1], **Lucas Mello Schnorr**[1]

[1] Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

`rayan.raddatz, kbrumati, mcogulart, schnorr@inf.ufrgs.br`

Abstract. *The growing complexity of Deep Learning models makes training on single devices difficult, requiring distributed strategies like Data Parallelism. However, this approach introduces critical synchronization challenges between processing units. This work evaluates TensorFlow parallel strategies on an HPC cluster with low-bandwidth network to analyze how application factors contribute to time reduction. The study compares the standard synchronous method against a proposed Custom Training Loop with gradient accumulation designed to reduce the bottleneck by synchronizing gradients less frequently. Our results indicate that optimizing synchronization intervals and batch sizes allows for scalable performance gains even on low-bandwidth infrastructure.*

1. Introduction

In the last decade, deep learning has established itself as state-of-the-art for solving complex problems such as computer vision and natural language processing. Given the progress in algorithmic precision, there has been an exponential growth in the complexity of neural network architectures and the volume of datasets. This scenario imposes a massive computational load, making the training of robust models on a single processing unit a difficult task in terms of time and resources.

To mitigate these challenges, distributed training has become a standard practice, with emphasis on the Data Parallelism strategy. In this approach, a complete replica of the model is instantiated on each processing unit, and the global dataset is partitioned into smaller, locally-focused batches. Although effective, this technique introduces critical synchronization challenges, as locally calculated gradients must be aggregated and communicated among all devices at every training step via All-Reduce operations. The network communication channel then becomes the primary bottleneck, especially in environments where high-performance networking hardware is not available.

Therefore, this work analyzes data parallelism for deep learning model training using TensorFlow in a cluster with a low-bandwidth network to investigate how the factors related to the application contribute to time reduction in a parallel scenario.

Software and Data Availability. We endeavor to make our analysis reproducible for a better science. We made available a companion material hosted in a public GitHub repository at `https://github.com/kenichi220/comp-sys-perf-analysis-final-work/tree/main`. Our companion contains the source code of this article and the software necessary to handle the created datasets. We also include instructions to run the experiment and figures.

2. Related Work

There are several studies regarding the training of deep learning models using distributed strategies. A comprehensive taxonomy of Distributed deep learning Systems (DDLS) is provided in [Langer et al. 2020], categorizing strategies into Model vs. Data Parallelism and Centralized vs. Decentralized optimization. They identify that in Data Parallelism, the network communication channel is typically the primary bottleneck. This topic of distributed training is also studied in [Shen et al. 2024], where, among various strategies, it is highlighted that large batch sizes are preferred in distributed scenarios. In this work we evaluate some of the most used strategies for parallel training, and also introduce the use of a Custom Training Loop with gradient accumulation that enable us to reduce the frequency of synchronization, allowing for lower training times for low-bandwidth networks.

3. Methods and Materials

3.1. Hardware and Software Configuration

Experiments were run using Python 3.11.2 and TensorFlow 2.15 as newer versions showed errors when using more than one node with TensorFlow `MultiWorkerMirroredStrategy` methods. We used `pip` to manage the Python libraries. The NVIDIA driver version was 550.54.15 along with CUDA 12.4. The platform used for the experiments has five nodes connected with a network of 1 Gbps, but only three were available for use. Each node on the platform contains an Intel Core i7-14700KF processor at 3.40 GHz with 28 threads and 20 cores, 96 GB DDR5 RAM and NVIDIA GeForce RTX 4070 with 8GB.

3.2. Custom Training Loop

The expected behavior of training with the `MultiWorkerMirroredStrategy` would be to have a lower training time as the number of nodes is increased. However, one common problem that surfaces in low-bandwidth networks is the synchronization overhead, resulting in a higher training time with the increase of nodes. This behavior happens because the gradients and weights of the model are synchronized between all nodes after every batch is computed when using the `model.fit()` method. To mitigate this problem, we proposed a Custom Training Loop that implements gradient accumulation, meaning that it synchronizes the gradients at an interval of batches, resulting in lower synchronization overhead. We empirically defined this interval of batches as four batches.

3.3. Design of Experiments (DoE)

We selected two different deep learning models to train, one big and one small: **ResNet50** [He et al. 2015] as our big model, with ~25.6M parameters, and **EfficientNetB0**[Tan and Le 2019] as our small model, with ~5.3M parameters. The two models were trained for five epochs using the TinyImageNet-200, a subset of the ImageNet[Deng et al. 2009] containing 100000 colored images downsized to 64×64 pixels, divided among 200 classes. We created a full factorial project for each model, varying the local batch size (32, 64 or 100), the number of nodes used (one, two or three) and the type of the training (ctl for Custom Training Loop and sync for training with the `model.fit` method). This is a total of 24 (2x3x3x2) different configurations, each execution was replicated five times in a random order to tackle variability.

4. Results

Figure 1 shows a full view of our results. The graph presents the mean training time in minutes as a function of the number of nodes, faceted by model and batch size and with the color representing the training type. Training time is defined as the time between the start of the first batch and the end of the last. Each bar represents the mean for the five executions for each configuration. The black transparent dots represent our observations. Error bars represent the standard error with a confidence interval of 99%. Two main results can be observed. First, the inclusion of more than one node introduces the need for communication, also introducing synchronization overhead. This overhead is better seen with smaller batches as the computation is faster and more communication is needed. The synchronization overhead is also more present when using the ResNet50 model as it is almost five times bigger than the EfficientNetB0. Secondly, the ctl showed a considerably smaller communication overhead even for small batch sizes. The execution using ctl was approximately 2.24x faster than the sync for the ResNet50 and 1.33x for the EfficientNetB0 when using batch size 32. The ctl showed Out-Of-Memory (OOM) errors when running with the batch size 100 for training the EfficientNetB0 model. We notice that after the introduction of overhead by the inclusion of a second node, adding a third node decreased the training time. We suppose that even with a low-bandwidth network, with enough nodes the time gained by the distributed computing can overcome the overhead and show better results than running it in a single machine.

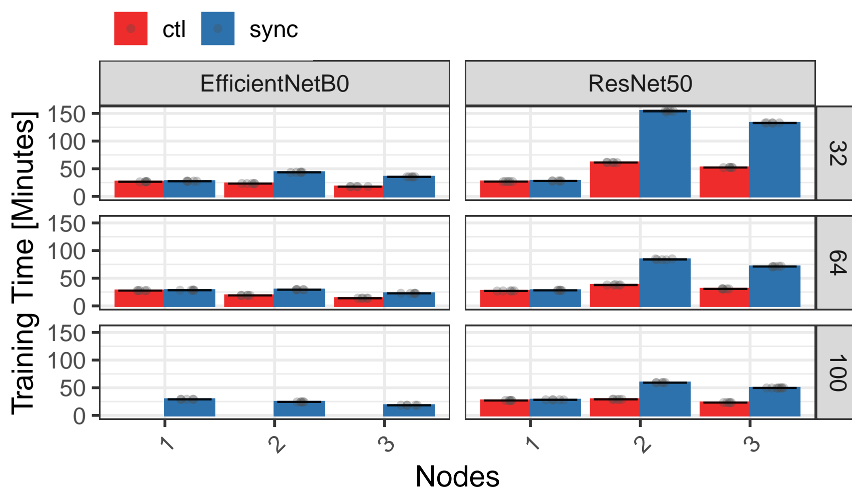

Figure 1. Mean of experimental results.

4.1. The impact of the nodes number and batch size

Figure 2 shows the mean training time aggregated by node count. This enables us to better observe the performance gain with more than two nodes, even though this gain is small. Figure 3 demonstrates that increasing batch size consistently shows performance gains for the tested sizes, while also decreasing the variance. This means that as the batch size grows the training time is almost the same despite the node count.

5. Conclusion

We could conclude that the network and the model size have a significant impact on distributed training, but that even with a low-bandwidth network it is possible to gain per-

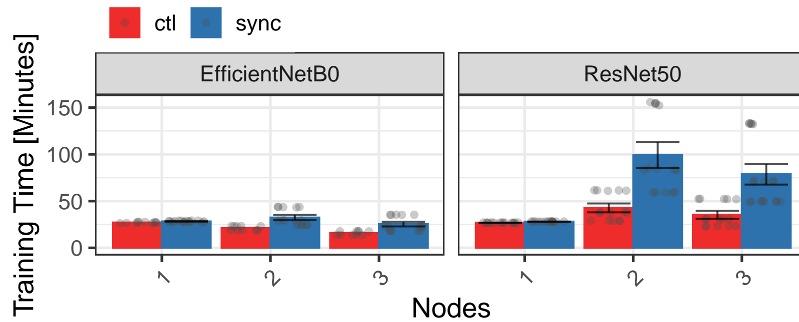

Figure 2. Mean of experimental results grouped by node count.

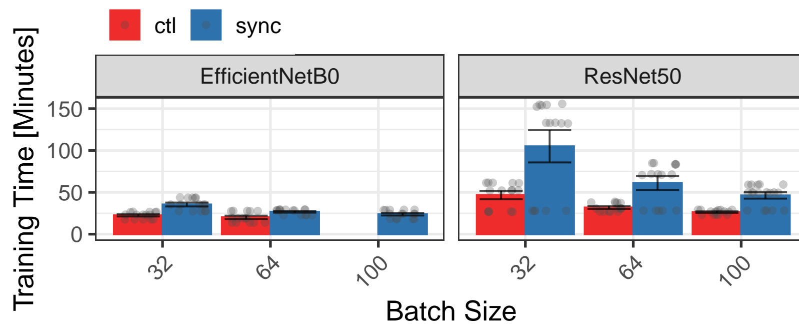

Figure 3. Mean of experimental results grouped by batch size.

formance when training by correctly configuring the environment and model parameters. Future work intends to further investigate the correlation between model size and network for effective parallel training, including studies about the impacts on model accuracy.

Acknowledgements

We would like to thank the PCAD at INF/UFRGS for making infrastructure and hardware used in the experiments available. We also acknowledge the Brazilian National Council for Scientific Technological Development (CNPq) for their financial support through the PIBIC-UFRGS scholarship.

References

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE CVPR*, pages 248–255.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*.

Langer, M., He, Z., Rahayu, W., and Xue, Y. (2020). Distributed training of deep learning models: A taxonomic perspective. *IEEE TPDS*.

Shen, L., Sun, Y., Yu, Z., Ding, L., Tian, X., and Tao, D. (2024). On efficient training of large-scale deep learning models. *ACM Computing Surveys*, 57:1 – 36.

Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946.