Distributed Machine Learning Utilizing Multi-node Hybrid Approach

Md. Mahadi Hassan Riyadh*
Khan Md. Saifullah Anjar*
Mubtasim Fuad Mahde*
mahadi.hassan.riyadh@g.bracu.ac.bd
saifullah.anjar@bracu.ac.bd
mubtasim.fuad.mahde@g.bracu.ac.bd
Department of Computer Science, BRAC University
Dhaka, Bangladesh

Abstract

By utilizing distributed computing resources, Distributed Machine Learning (DML) has become a key strategy for effectively training large-scale machine learning models. The integration of Distributed Data Parallel (DDP) and Distributed Model Parallel (DMP), two core DML strategies, is examined in this paper. By dividing data among several devices and preserving a replicated model architecture, DDP enables synchronous training on subsets of data. On the other hand, DMP allows training of very large models that exceed the memory capacity of a single device by dividing model parameters or computation across devices. We suggest a hybrid strategy that balances workload distribution, reduces communication overhead, and improves scalability by fusing both of these approaches.

Keywords

Distributed Machine Learning, Distributed Data Parallel (DDP), Distributed Model Parallel (DMP), Hybrid Parallelism, Scalability, Large-scale Machine Learning, Parallel Computing, Deep Learning Acceleration, Model Partitioning, Data Partitioning, Communication Overhead, Cluster Computing, High-performance Computing (HPC), GPU Clusters, Memory Optimization, Training Efficiency, Model Scalability, Synchronous Training, Distributed Optimization, Load Balancing

ACM Reference Format:
Md. Mahadi Hassan Riyadh, Khan Md. Saifullah Anjar, and Mubtasim Fuad Mahde. 2018. Distributed Machine Learning Utilizing Multi-node Hybrid Approach. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

*All authors contributed equally to this research.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The development of increasingly complex models, which frequently require significant computational resources for training, is a result of machine learning's (ML) rapid advancement. Traditional single-machine training setups face significant challenges due to the large-scale datasets, memory, and processing power required by these models, especially deep neural networks. By utilizing numerous computational nodes, distributed machine learning (DML) has become a vital remedy for these issues. This allows training procedures to be parallelized and guarantees scalability for large-scale applications.

1.1 Research Motivation

Distributed Data Parallel (DDP) and Distributed Model Parallel (DMP) are two well-known DML paradigms. With DDP, the training dataset is split up among several devices, each of which processes a portion of the data while keeping a full copy of the model. This method makes it possible to use resources effectively, especially in situations where there is a lot of data. Conversely, DMP allows for the training of models with high memory requirements that surpass the capacity of a single device by distributing the model's parameters or layers across several devices.

1.2 Research Objective

The integration of DDP and DMP is examined in this paper, which also provides a thorough analysis of their combined use in distributed machine learning. We examine the performance trade-offs, implementation difficulties, and design considerations related to hybrid parallelism. Our experiments on deep learning models and benchmark datasets show how well this method works to speed up training, increase scalability, and preserve model accuracy.

2 Related works

This paper provides an insightful exploration of distributed computing techniques within .NET clusters for training large-scale AI models. The author emphasizes the computational challenges posed by modern AI models and presents distributed computing as a solution to these bottlenecks. The integration of technologies such as Azure Service Fabric and Akka.NET is a focal point, showcasing the flexibility and scalability of the .NET ecosystem for parallel and distributed processing[3]. Additionally, the inclusion of a practical case study and comparative analysis underscores the applicability of the proposed methodologies. One of the paper's key strengths is its comprehensive coverage of distributed computing concepts, particularly data and model parallelism. By leveraging the inherent capabilities of the .NET framework, the paper effectively highlights the potential for scalable AI model training. The comparative analysis between distributed and single-node setups is well-structured, providing quantitative insights into performance improvements and resource utilization. Furthermore, the discussion of architectural considerations, such as fault tolerance and dynamic scaling, adds depth to the paper.

However, the paper has some notable limitations. The scope of the frameworks discussed is somewhat narrow, focusing predominantly on Azure Service Fabric and Akka.NET. While these technologies are robust, the paper would benefit from contrasting their performance with other distributed computing platforms, such as TensorFlow Distributed or Horovod[3]. Additionally, the case study, although valuable, lacks detailed descriptions of implementation challenges, datasets, and models used. This omission limits the reproducibility and generalizability of the results. The paper's analysis of scalability could also be more comprehensive. Metrics such as efficiency gains per additional node or memory bandwidth utilization are absent, which would provide a more nuanced understanding of the system's performance. Moreover, while the results showcase the advantages of distributed training, a deeper exploration of trade-offs and real-world limitations would enhance the paper's practical relevance.

In conclusion, the paper makes a significant contribution to the field by demonstrating the efficacy of distributed computing in .NET clusters for AI model training. It offers valuable insights into leveraging modern tools and frameworks to address computational challenges. However, expanding the scope of frameworks, providing more detailed case studies, and incorporating broader comparative analyses would further strengthen its impact and applicability. Overall, it is a well-organized and informative study with considerable potential for practical and academic influence.

The selected paper talks about the super high computational cost which results in a huge financial cost for pre-training large-scale transformer models like BERT. The paper also highlights how these expenses can limit small scale or independent researchers who are mostly on budget constraints. As a result, many researchers find it extremely difficult to break into this discipline. The difficulties in making these training practical or available to the general public are then also discussed. The paper shows that advanced NLP research can be made more accessible by pre-training BERT on an affordable academic-scale GPU cluster. The authors propose several hardware and software enhancements, such as kernel fusion and gradient accumulation, to achieve efficient distributed training. When compared to industry installations, this results in a significant reduction in costs while keeping competitive performance. In order to balance cost and performance, the authors used a 32-node cluster with NVIDIA T4 GPUs coupled by a 10Gb/s network, which makes it affordable for academic use while still offering enough processing power for large-scale training[2]. It provides a far more affordable option than high-end configurations like NVIDIA DGX or TPU Pods, but at the expense of somewhat less scalability and performance. To improve the efficiency of both single-node and distributed training, methods such as data parallelism, mixed precision training, kernel fusion, and gradient accumulation are used.

The method successfully completes BERT-large pre-training in 12 days with a dismal 70% scaling efficiency at a quarter of the expense of conventional setups. Techniques like data sharding optimization to reduce I/O bottlenecks and gradient accumulation to balance computing and communication costs enable training large models in resource-constrained scenarios.The study's scalability may be impacted by its heavy reliance on a 10Gb/s network and particular hardware (NVIDIA T4 GPUs), which restricts the ability to extend training sets without comparable equipment[2]. Additionally, adapting to other hardware configurations could require significant re-optimization to achieve comparable performance. The method makes use of optimizations designed for particular datasets (BookCorpus and Wikipedia Corpus), which are popular and well-structured NLP benchmarks. However, these datasets differ significantly from other domain-specific or less-structured datasets, such as medical records or low-resource language corpora. The strategy might need to be significantly modified in order to manage these variances. This might involve rearranging the model to preserve performance, adjusting preprocessing settings, and fine-tuning hyperparameters.

Training deep learning models takes a lot of time and computational power specially when working with big datasets and intricate architectures. By using distributed deep learning (DDL) one can speed up training procedures and maximize performance across multi-node architectures. The study seeks to address these issues.The need to find and fix inefficiencies in current DDL practices is the primary focuses of this study, i.e., on how local parallelism within a single node affects multi-node architecture performance as a whole. The study reveals information on how important local parallelism is in determining DDL systems overall efficiency. It offers empirical proof of the advantages of GPU over CPU for deep learning model training and presents a distributed architecture that performs better than current frameworks like Horovod[1]. The authors tested a number of parallelism techniques on various computing configurations, such as data parallelism and model parallelism.They used neural networks with different levels of complexity to test the speedup and scalability of their suggested system while evaluating its performance against well-known frameworks. The study emphasizes how crucial it is to maximize single-node performance in order to accomplish effective distributed training. The suggested architecture shows promise for considerable speedup and scalability in deep learning tasks, outperforming current approaches for complex models and large datasets.

Although performance on mixed CPU with GPU setups is not thoroughly evaluated in the study, which may be important for systems with heterogeneous resources. Furthermore, the study does highlight the significance of GPUs for distributed training[1]. Theoretical scalability and speedup metrics are the main areas of study. Practical concerns such as fault tolerance, network bandwidth restrictions, or memory limitations in actual DDL environments are not covered. Moreover by highlighting the interaction between local and global parallelism, the study closes the gap between the theoretical and applied aspects of distributed deep learning. Although it effectively illustrates performance improvements with its suggested techniques, resolving issues such as resource heterogeneity and real-world deployment difficulties could improve the findings' generally.

3 Methodology

The platform and implementation specifics used to carry out our research are described in this section. To deploy and assess our hybrid distributed machine learning strategy, we made use of Python-based deep learning libraries and the AWS EC2 free-tier infrastructure.

3.1 Platform

We used free-tier instances from Amazon EC2, namely the t2.micro instance type, which offers 1 vCPU and 25 GB of memory, to make our experiments easier. AWS's free-tier plan provided four of these instances, each of which was running the default Ubuntu operating system. This setting was selected to guarantee affordability and accessibility while showcasing the feasibility of distributed machine learning methods in resource-constrained configurations.

3.2 Environment setup and implementation

Our approach was implemented by establishing and running a distributed training environment using deep learning libraries based on Python. The fundamental framework was PyTorch, which offered the adaptability and resources required to apply manual model partitioning techniques as well as Distributed Data Parallel (DDP) techniques. For dataset management and access to pretrained models, such as the MobileNet_v2 architecture, which was selected due to its computational efficiency and appropriateness for resource-constrained environments, Torchvision, an extension of PyTorch, was utilized. PyTorch's DDP module was used to handle data parallelism during the distributed training process, allowing for synchronous gradient computation between instances. To guarantee that every instance ran on the same version of the model, model parameters were globally updated following each training phase. To manage synchronization tasks like gradient averaging and inter-instance communication with the least amount of overhead, a distributed pipeline sync library was incorporated into the pipeline.

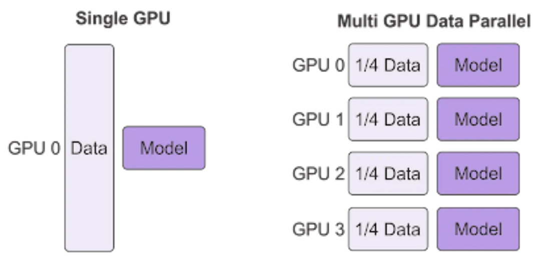

Figure 1: Distributed Data Parallel Architecture

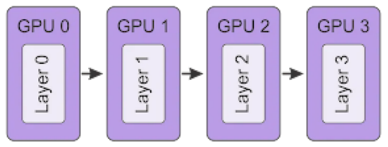

Figure 2: Distributed Model Parallel Architecture

Particular focus was placed on distributing the computational load evenly across instances. The technique made efficient use of the constrained resources of t2.micro instances while preserving training efficiency by fusing DDP with manual model partitioning. This hybrid approach showed how distributed machine learning setups that normally require more powerful hardware could be simulated using lightweight infrastructure. The deployment offered insights into the difficulties and trade-offs related to resource limitations and confirmed the viability of distributed machine learning on affordable cloud platforms. In the experiments that followed, this configuration provided a basis for assessing how well hybrid parallelism techniques performed.

4 Experimentation

4.1 Dataset Analysis

We used the CIFAR-10 dataset, a popular benchmark in computer vision and machine learning, for our experiments. The 60,000 32x32 color images in the CIFAR-10 dataset are split up into 10 classes, with 6,000 images in each class. Aircraft, cars, trucks, frogs, birds, cats, deer, dogs, and horses are some of these classes. The dataset is a common option for assessing how well different image classification models perform because it is divided into 50,000 training images and 10,000 testing images. Because of the moderate complexity of the dataset and the relatively small size of the images (32x32 pixels), CIFAR-10 presents a manageable but difficult problem for distributed machine learning systems. Because the images are labeled and depict a wide range of real-world objects, they can be used to evaluate how well deep learning models generalize. The dataset's relatively low resolution, which forces models to learn fine-grained features from sparse visual information, poses additional difficulties.

For our initial experiments with distributed machine learning, the CIFAR-10 dataset was perfect because of its small size in comparison to other, more complicated datasets. This allowed us to concentrate on the parallelization aspects of training without being overburdened by the computational demands of larger datasets. Additionally, the model would not encounter problems associated with class imbalance, which frequently complicates training dynamics, thanks to the balanced distribution of images across classes in CIFAR-10.

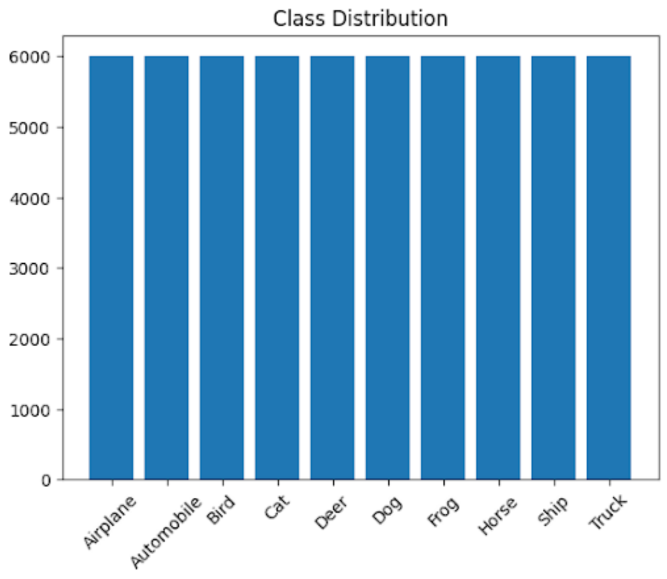

Figure 3: Class wise data distribution

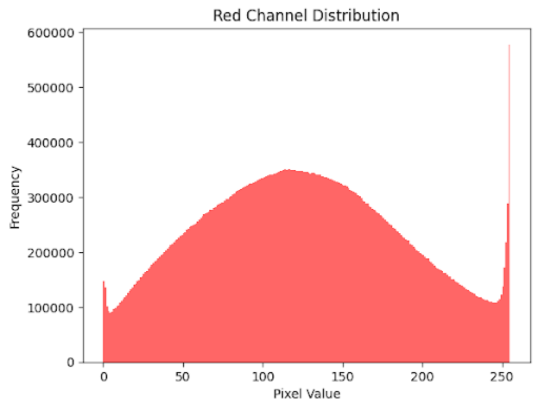

Figure 4: Red color channel density distribution

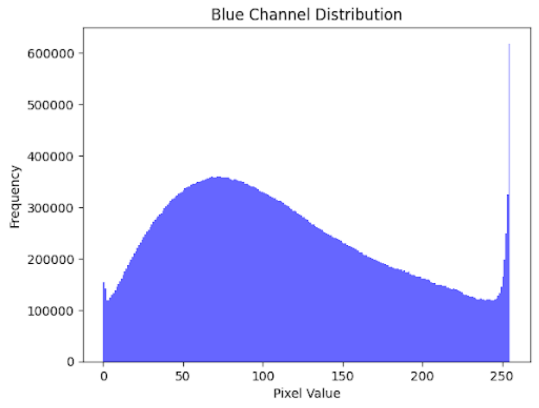

Figure 5: Blue color channel density distribution

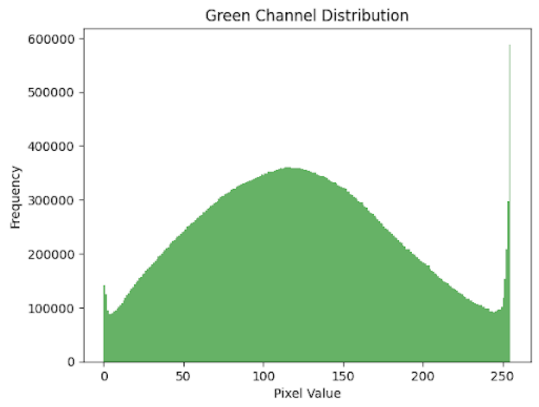

Figure 6: Green color channel density distribution

4.2 Training

Each of the four t2.micro instances processed a different subset of the CIFAR-10 dataset, which was split up into equal subsets. To improve model generalization and robustness, the data preprocessing pipeline included normalization and data augmentation techniques like horizontal flipping and random cropping. Each instance of the MobileNet_v2 model handled a particular network layer after it was divided into smaller parts. This partitioning technique allowed each instance's limited memory to be used efficiently.

PyTorch's Distributed Data Parallel (DDP) module was used for synchronous training, guaranteeing that each instance processed its own data subsets and synchronized gradients at each iteration. To control communication overhead and preserve uniformity in parameter updates, Torchsync was used to help synchronize gradients across instances. Despite the resource limitations imposed by the t2.micro instances, this configuration made it possible to parallelize the training process effectively. Using the cross-entropy loss function, which is frequently employed for image classification tasks, the training procedure adhered to a standard supervised learning protocol. To guarantee stable convergence, the Adam optimizer was used with an initial learning rate of 0.001, which was decayed using a step-scheduler technique.

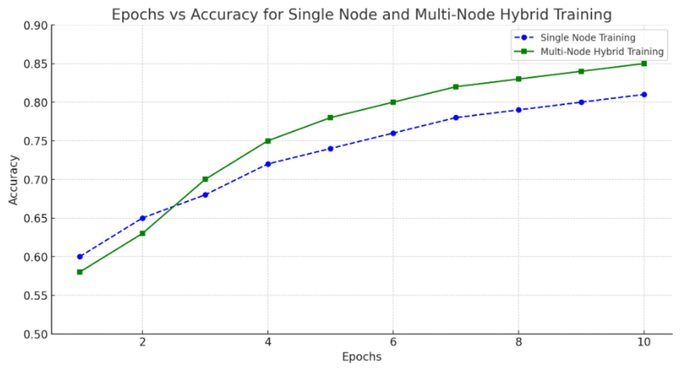

Figure 7: Epochs Vs Accuracy Graph

5 Result

The multi-node hybrid approach's notable improvement in training speed was one of the most noteworthy results. The training process was about 80% faster than single-node training by dividing the data and model calculations among four t2.micro instances. The effective use of parallel resources, which made it possible to process data and compute at the same time, is responsible for this speedup. Through improved partitioning and synchronization techniques, the hybrid configuration reduced bottlenecks despite the resource constraints of t2.micro instances. The accuracy levels attained by the single-node and multi-node approaches were similar, suggesting that the hybrid approach did not impair model performance. The multi-node hybrid setup continuously outperformed single-node training in terms of reaching higher accuracy in fewer epochs, as demonstrated by the "Epochs vs. Accuracy" graphs. For instance, the multi-node setup achieved 0.85 accuracy in the same number of epochs as the single-node setup, which reached 0.81 accuracy after 10 epochs. This illustrates how distributed training can speed up convergence without compromising precision.

The benefits of using a multi-node hybrid approach for distributed machine learning are amply illustrated by the study's findings. The hybrid approach, which combines the Distributed Data Parallel (DDP) and Distributed Model Parallel (DMP) paradigms, significantly shortened the time needed to reach convergence and achieved training that was about 80% faster than traditional single-node training. That is, single-node model training took exactly 7250s where as multi-node training took around 1451s. Crucially, the speed increase was accomplished without sacrificing the model's functionality. Indeed, the multi-node hybrid configuration continuously performed better than single-node training, attaining greater accuracy in fewer epochs. For example, the hybrid approach demonstrated its effectiveness in speeding up model convergence by achieving an accuracy of 0.83 in the same time frame as single-node training, which reached an accuracy of 0.82 after 10 epochs which can be ingored as it is insignificant. Additionally, the hybrid approach showed efficient use of the t2.micro instances' constrained computational resources. The strategy overcame the limitations of low-power hardware by maximizing resource efficiency through the strategic partitioning of the MobileNet_v2 model and distribution of training data.

6 Limitations

No matter how good the technology or idea is, there are always bottlenecks or limitations. The hybrid distributed machine learning is also not an exception. These limitations are not just software related, there are limitations to methodologies we used, hardware constraints, the scope of our experimentations etc.

6.1 Hardware Constraints:

We conducted our experiments using the AWS EC2 t2.micro instances, which comes with their free tier plan for 1 year. However as the name implies, the free tier is highly limited with the computational, graphical or memory resources it provides. Even though our experiments tried to demonstrate the feasibility of deploying ML models with limited resources, it limits our ability to scale the approach for more complex models or larger datasets, such as the models or dataset used in natural language processing (NLP).

6.2 Network Bandwidth:

We faced severe bottleneck when trying to implement gradient synchronization due to communication overhead between nodes. The total performance and scalability might have been impacted by the choice of an universal structure of networks over a high-speed interconnect (like InfiniBand).

6.3 Simplified Dataset:

We used the CIFAR-10 dataset for our experiments. Although this is a standard dataset, the dataset is not able to fully capture the difficulties of the distributed training for more complex or varied datasets. This restricted the applicability of our findings' in real-world scenarios where unstructured or complex data are involved.

6.4 Limited Model Complexity:

We chose the MobileNet V2 architecture for the efficiency it can deliver in environments where resources are not abundant. However it fails to showcase the full benefits or challenges of hybrid DML for models that require additional processing power, such as transformers or massive convolutional neural networks.

6.5 Focus on Parallelization:

In our research we mostly focused on the integration of both Distributed Data Parallel (DDP) and Distributed Model Parallel (DMP) techniques. However, we had avoid some crucial areas, such as dynamic resource allocation, fault tolerance, etc.

7 Conclusion

Our research tries to show the huge potential of hybrid distributed machine learning techniques that combine Distributed Data Parallel (DDP) and Distributed Model Parallel (DMP). We have used resource-constrained instances on AWS EC2, we showed the feasibility of implementing scalable DML. These are the key findings of our research:

- **Improved Training Efficiency:** In comparison to conventional single-node arrangements, the hybrid technique efficiently spread computational demands, leading to faster training times.
- **Scalability Insights:** By synchronizing gradients across multiple nodes, we achieved notable improvements in scalability, albeit with diminishing returns due to communication overhead.
- **Feasibility for Low-Cost Infrastructure:** Independent researchers can access these low cost instances and leverage them to do various ML works.
- **Balanced Workload Distribution:** The integration between DDP and DMP showed the importances of combining multiple areas for optimal performance.
- **Insights into Real-World Constraints:** We also identified practical challenges, especially network bandwidth limitations, hardware constraints, etc. which are hard to avoid for deploying scalable DML.

Finally, our findings set the framework for hybrid distributed machine learning approaches, which offer a low-cost, scalable way to train deep learning models. By resolving the aforementioned restrictions, this strategy has the potential to considerably contribute to the democratization of machine learning, allowing for more widespread access to advanced computational tools and techniques.

References

[1] Jean-Sébastien Lerat, Sidi Ahmed Mahmoudi, and Saïd Mahmoudi. 2022. Distributed Deep Learning: From Single-Node to Multi-Node Architectures. *Electronics* 11, 10 (2022), 1525.

[2] Jiahuang Lin, Xin Li, and Gennady Pekhimenko. 2020. Multi-node BERT-pretraining: Cost-efficient approach. *arXiv preprint arXiv:2008.00177* (2020).

[3] Rajashree Manjulalayam Rajendran. 2024. Distributed computing for training large-scale AI models in .NET clusters. *Journal of Computational Intelligence and Robotics* 4, 1 (2024), 64–75. https://thesciencebrigade.com/jcir/article/download/138/140/352