A Comparative Analysis of Distributed Training Strategies

Appari Lalith - alalith@asu.edu
Navyasri Reddy - nmallu@asu.edu
Anoop Vallabhajosyula - avallab5@asu.edu

Problem Statement:

The rising need for advanced machine learning models in a variety of real-world settings presents significant challenges to current computing frameworks, especially as these models become more complex and larger in size. This complexity is fueled by the increasingly sophisticated tasks these models must handle, such as making real-time decisions in autonomous vehicles or managing interactions between humans and machines in customer service roles. As a result, there's a growing need for training methods that are not only efficient but also capable of scaling up; this is where distributed training strategies become essential.

Distributed training uses multiple computing units—like GPUs or whole servers—to train neural networks across several processors at once. This approach is crucial because it significantly cuts down the time needed to train models and helps manage the large amounts of data these models work with. Traditional methods, where a single machine does all the work, often fall short because they don't have enough memory, processing power, or speed. Distributed training steps in as a vital solution, enabling the use of bigger and more complex models that can learn from bigger datasets more effectively.

This study looks into three main types of distributed training strategies: Data Parallelism, Pipeline Parallelism, and Model Parallelism. Each strategy has its own unique way of handling training tasks. The effectiveness of these strategies is assessed through rigorous evaluation of training speed, resource utilization, and scalability across various tasks in computer vision and natural language processing. The choice of these tasks is strategic, reflecting common yet diverse applications of deep learning that benefit from distributed training approaches. Computer vision tasks, such as image recognition and segmentation, and natural language processing tasks, such as sentiment analysis and language translation, typically require extensive computational resources and can greatly benefit from optimized training strategies.

Literature Survey:

For distributed training strategies, a comprehensive examination of existing literature is crucial to provide context and background. The following papers covers foundational concepts, advances in distributed computing, and specific applications in machine learning. Here’s a detailed breakdown of the types of literature and specific works that we have reviewed:

1. Foundational Theories and Algorithms in Machine Learning:

- “Stochastic Gradient Descent” - A fundamental algorithm for training neural networks. This algorithm updates model parameters iteratively based on the gradient of the loss function with respect to the parameters. Understanding SGD is crucial as many distributed training methods build upon or modify this basic approach to optimize for parallel processing.
- “Neural Networks and Deep Learning" by Michael Nielsen - This book provides a solid foundation in the mechanics of neural networks, which is essential for understanding how they can be adapted to distributed training frameworks.

2. Specific Distributed Training Strategies:

- "Large Scale Distributed Deep Networks" by Jeffrey Dean et al. - This paper from Google discusses the challenges and methodologies for training deep neural networks across a large number of computational resources. It introduces the concept of Downpour SGD, an asynchronous SGD variant, which is pivotal for discussions on model scalability.
- "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism" by Yanping Huang et al. - Offers insights into pipeline parallelism, explaining how splitting a model into different segments for concurrent processing can enhance training efficiency.

3. Advancements and Comparative Studies:

- "Distributed Deep Learning: A survey on recent trends and future directions" by researchers in the field - This survey provides an overview of the latest trends in distributed deep learning, including new frameworks and architectures developed to facilitate more efficient distributed training.
- "Efficient Large-Scale Distributed Training of Convolutional Networks" by Adam Coates et al. - Explores various strategies for effectively scaling up the training of convolutional networks, a common model type in computer vision applications.

4. Practical Implementations and Framework Reviews:

- "PyTorch Distributed: Experiences on Accelerating Data Parallel Training" - A detailed look into PyTorch's distributed training capabilities, including its DistributedDataParallel module. Understanding this will be crucial for implementing and benchmarking data parallelism strategies.
- "TensorFlow Distributed Deep Learning" - A guide and review of TensorFlow's capabilities for distributed learning, providing insights into different configuration and optimization options available within the framework.

Proposed Solution:

To perform the comparative analysis, we propose to evaluate a wide ranging models over different sets of tasks while varying the underlying distributed training setup. We plan to perform the following experiments to understand the distributed training approaches we have presented in the literature survey (overlaps with the discussions from class).

- Three deep learning tasks are chosen: Image Classification, Image Segmentation and Sentiment Analysis.
- Reasoning for choosing these set of tasks:
    - Popular use cases for deep learning models encompassing both visual and textual domains.
    - Include both multi-class and single-class classification.
    - Well studied in normal training setups.

- For Image Classification:
    - Models: VGG, ResNet
    - Datasets: MNIST

- For Image Segmentation:
    - Models: UNet, SegNet
    - Datasets: COCO

- We propose to setup experimentation across five different training environments for better understanding of the distributed training approaches:
    - Training on a single GPU without parallelism using various batch sizes.
    - Training using Data Parallelism on multiple GPUs.
    - Training using Model Parallelism on multiple GPUs, employing frameworks such as DistBelief, Hogwild, or SHAT.
    - Training using Pipeline Parallelism on multiple GPUs, utilizing frameworks like PipeDream and Gpipe.
    - The evaluation plan includes metrics such as GPU utilization, training time, convergence rate, and cost analysis.
    - The comparison of results focuses on identifying the most suitable distributed training approach for different tasks based on the experimental setups and evaluation metrics.

Our evaluation will primarily focus on maximizing hardware resource utilization, measured through metrics like GPU usage (% memory access). Time efficiency will be assessed in two dimensions: (i) the duration for a model using a specific training approach to achieve a target training accuracy and (ii) the average time taken to process an epoch. Additionally, we will compare the number of epochs required for each approach to reach certain threshold accuracies. Furthermore, we will analyze the costs associated with running these training tasks on cloud platforms such as AWS and Google Cloud Platform. This comprehensive evaluation

will provide insights into the suitability of different distributed approaches for specific tasks based on factors including resource utilization, time efficiency, and cost-effectiveness.

Experimental Setup and Data:

Below are the data used for training the models in different GPU environments and training setup for each model

	Samples	Features	Classes
CIFAR-10	60k	32 x 32	10
CIFAR-100	60k	32 x 32	100
MNIST	70k	28 x 28	10

Model	# GPUs	Batch Size	Other Params
naive-128	1	128	-
dataparallel-256	2	256	-
gpipe-256	2	256	chunk size: 4

Training setup:

- 2 x Nvidia RTX 3070
- GPU RAM - 8 GB
- CUDA version - 12.2
- Driver version - 535.129.03

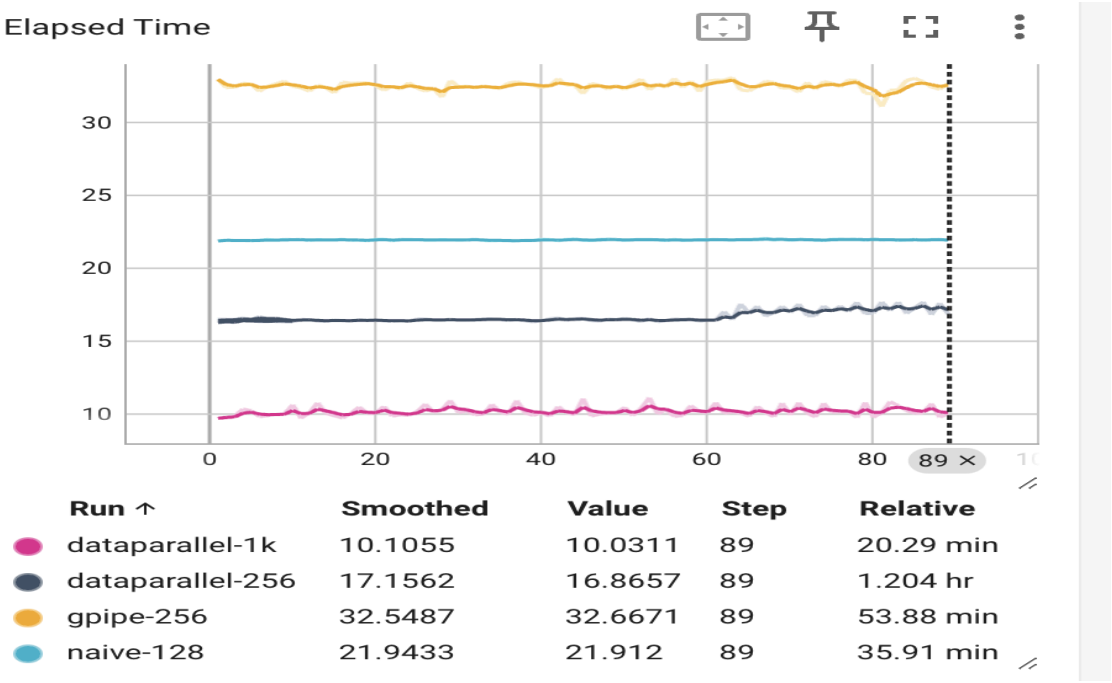
Results and Conclusion:

Time per Epoch Analysis:

The analysis of training time per epoch reveals significant insights into the efficiency of different

distributed training strategies. The TensorBoard chart, which visually represents the duration each model took per epoch, indicates that Data Parallelism effectively reduced the training time per epoch. This outcome was anticipated, given the inherent advantages of Data Parallelism in distributing data across multiple GPUs, thereby enhancing computational speed by parallel processing.

Unexpectedly, the results for GPipe presented a different scenario. Contrary to expectations that Pipeline Parallelism would streamline processing and reduce training times, GPipe extended the duration of each epoch considerably. This increase in time could be attributed to the overhead associated with managing pipeline stages and synchronizing data across different segments of the model during training.



Run	Smoothed	Value	Step	Relative
dataparallel-1k	10.1055	10.0311	89	20.29 min
dataparallel-256	17.1562	16.8657	89	1.204 hr
gpipe-256	32.5487	32.6671	89	53.88 min
naive-128	21.9433	21.912	89	35.91 min

Throughput Analysis:

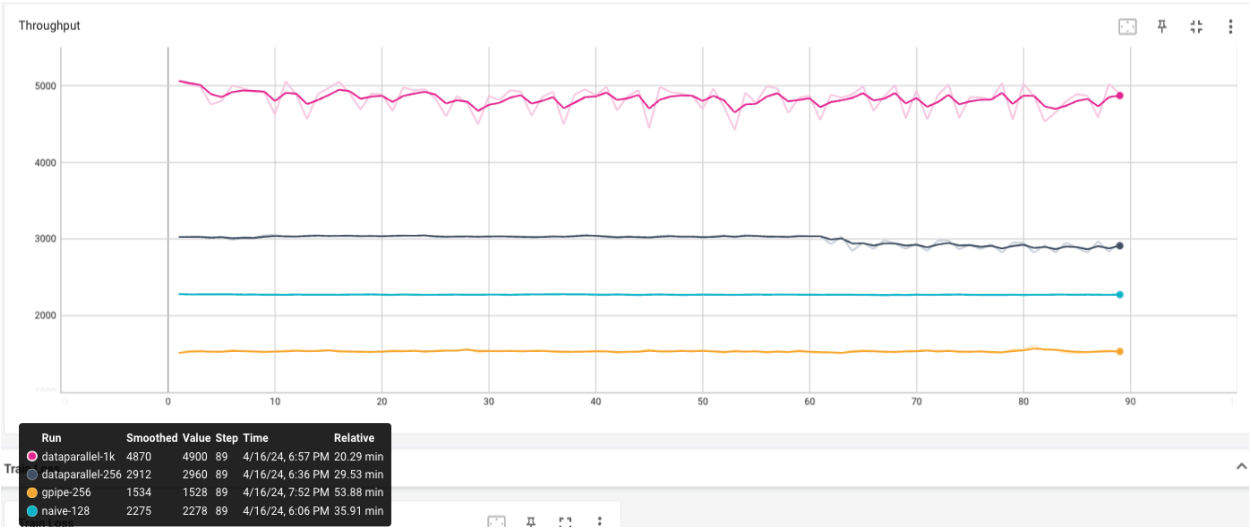
Further insights were gained from analyzing the throughput, measured in samples processed per second, across different models. The TensorBoard chart provided a clear quantitative measure of throughput for each training configuration:

Baseline (Single GPU): Achieved a throughput of 2200 samples per second, serving as our control metric.

DataParallel-256 (Using Data Parallelism on multiple GPUs): Showed a significant improvement, reaching a throughput of 3000 samples per second. This increase underscores the effectiveness of Data Parallelism in handling larger batches of data efficiently across multiple GPUs.

GPipe: Contrary to its expected performance, GPipe managed a throughput of only 1500

samples per second. This result is notably lower than both the baseline and Data Parallelism configurations, suggesting that while Pipeline Parallelism might offer benefits in certain scenarios, it may also introduce inefficiencies, possibly due to the complexity of managing micro-batches and coordinating between different pipeline stages.



Model	Throughput (total)	GPU Utilization (Avg)	Memory Usage
naive-128	2275 /s	92%	1.1 GB
dataparallel-256	2968 /s	88%	1.2+1.1 GB
GPipe	1528 /s	68%	0.9+0.5 GB

Future work:

It would be valuable to explore the implications of using both homogeneous and heterogeneous GPU configurations for distributed training. Testing with homogeneous systems, where all GPUs are identical, could provide insights into optimizing training protocols under ideal conditions with uniform computational resources. Conversely, heterogeneous environments that mix different types of GPUs, or even GPUs and CPUs together, could reflect more realistic scenarios in various industry settings. This approach would help identify adaptive strategies that optimize

performance despite hardware disparities, which is critical for deploying machine learning solutions in diverse technological landscapes.

Additionally, expanding the research to include distributed training strategies for stateful models like recurrent neural networks (RNNs) would address a significant gap in current methodologies. RNNs, crucial for tasks involving sequential data such as language processing or time-series analysis, present unique challenges in distributed settings due to their dependencies across time steps. Investigating how different distributed training strategies can maintain the state across multiple GPUs without compromising the temporal integrity of the model could lead to breakthroughs in efficiently training more complex neural networks. These efforts combined would not only enhance the understanding of distributed training dynamics but also push the boundaries of what is achievable in advanced machine learning applications.

References:

1. Zhou, Xinyu, et al. "East: an efficient and accurate scene text detector." Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. 2017.
2. Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, 2014.
3. Recht, Benjamin et al. "Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent." NIPS (2011).
4. Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems 25 (2012).
5. Ko, Yunyong, and Sang-Wook Kim. "SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning." Applied Sciences 12.1 (2022).
6. Narayanan, Deepak, et al. "PipeDream: generalized pipeline parallelism for DNN training." In Proceedings of the 27th ACM Symposium on Operating Systems Principles, pp. 1-15. 2019.
7. Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." Advances in neural information processing systems 32 (2019).
8. Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.
9. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition.
10. LeCun, Y., Cortes, C., & Burges, C. J. (2010). MNIST handwritten digit database.
11. Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context.
12. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.
13. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). RoBERTa: A robustly optimized BERT pretraining approach.