Distributed Deep Learning Framework Based on Cloud Oss for Efficient Model Synchronization

Zinan Weng

College of information engineering, Shanghai Maritime University, Shanghai, China

Abstract. With the rapid development of deep learning technology, training deep neural networks (DNN) under limited computational resources has become a hot topic, especially in fields like computer vision and natural language processing. Traditional distributed training often relies on centralized parameter servers, which can lead to communication bottlenecks, limiting system performance. To address the issue, this paper proposes a distributed training framework based on cloud Object Storage Service (OSS), utilizing OSS for parameter synchronization and avoiding the bottlenecks associated with traditional parameter server models. Experimental results show that the OSS-based framework achieves faster convergence speed, higher training efficiency, and shorter training time compared to traditional distributed training frameworks. Specifically, the OSS framework reduces training time by approximately 30% and demonstrates better flexibility and scalability across different worker configurations, effectively enhancing performance in large-scale distributed training tasks. The significance of this study lies in providing an efficient and scalable solution for training deep neural networks under limited computational resources.

1. Introduction

With the rapid development of deep learning technology, how to train DNN under limited computing resources has become a hot topic, especially in the wide application in computer vision, natural language processing and other fields [1]. As the data scale and model complexity keep increasing, when it comes to training complex models on large-scale datasets, single-machine training often encounters bottlenecks and is unable to meet the increasing computing demands. Therefore, distributed machine learning (DML) emerged as a result [2]. However, DML often requires an upfront investment of massive hardware resources, while in the meantime, complex tasks such as environment configuration, deployment and maintenance also need to be completed, which is both time-consuming and costly for researchers and enterprises.

Common methods for distributed training include data parallelism and model parallelism. These approaches address the computational bottlenecks inherent in distributed training to some extent. However, data parallelism incurs significant parameter synchronization overhead, particularly with large-scale datasets, where communication

202210310195@stu.shmtu.edu.cn

costs become a critical factor influencing training efficiency. On the other hand, model parallelism is often constrained by the model partitioning strategy, which can lead to instability during the training process. Additionally, most existing distributed training frameworks rely on centralized parameter server models, which can create a performance bottleneck. In this regard, DML based on cloud servers can effectively alleviate these problems [3]. Cloud platforms provide adjustable computing resources, and users do not need to purchase hardware themselves. Researchers and enterprises only need to adjust resources dynamically according to computing needs, thereby saving costs on hardware procurement and maintenance [4].

This study aims to address the limitations of traditional distributed training by developing a framework that optimizes both resource allocation and training efficiency. The proposed approach leverages cloud servers and OSS for parameter synchronization, reducing the burden on centralized parameter servers and mitigating bottlenecks.

In conclusion, the main contributions of the proposed method are as follows:

• Distributed Training Framework: Utilizes cloud servers and OSS for parameter storage and synchronization, reducing the burden on centralized servers and addressing performance bottlenecks.

• Efficient Update Mechanism: Implements periodic model updates and asynchronous strategies to minimize communication overhead, enhance training stability, and improve efficiency by reducing global synchronization.

2. Methodologies

Traditionally, distributed training relies on a centralized parameter server to store and synchronize model parameters across all workers. However, this approach can lead to significant communication bottlenecks, as the parameter server becomes a single point of contention, handling all model updates from the workers. As the number of workers increases, the parameter server may become overwhelmed, resulting in slower training speeds due to increased network traffic and synchronization delays.

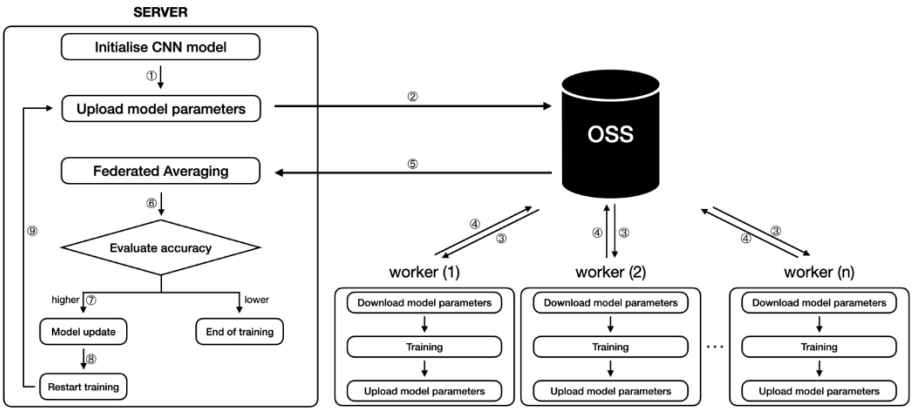

Figure 1. Distributed Training Process Flowchart (Picture credit : Original).

In contrast, this framework avoids the traditional parameter server model by utilizing OSS for asynchronous updates, which reduces communication bottlenecks and improves scalability. The training process begins with the server initializing a CNN model and uploading its parameters to OSS, making them accessible to all workers. Each worker independently downloads the latest global model, trains it on a local dataset using

Stochastic Gradient Descent (SGD), and periodically uploads the updated parameters back to OSS. Figure 1 illustrates the process of the proposed distributed training framework. The iterative training process continues until convergence, with asynchronous updates ensuring efficient resource utilization.

The server retrieves these updates from OSS and applies Federated Averaging (FedAvg) to aggregate them, ensuring that workers contributing more data have a greater influence on the global model [5]. After aggregation, the updated global model is re-uploaded to OSS for the next training round. To ensure training efficiency, the server periodically evaluates the model's accuracy. If the accuracy improves, training continues with updated parameters; otherwise, training is terminated. This iterative process enables asynchronous updates, allowing workers to train at their own pace without slowing down the entire system. By offloading synchronization tasks to OSS, the framework minimizes network congestion and enhances the overall training performance.

2.1 Overall Architecture

The distributed training framework is designed to leverage cloud-based storage for efficient model synchronization. It consists of a server and multiple workers, with OSS acting as an intermediary for model updates. The training tasks are executed in parallel by multiple worker nodes, with each worker responsible for a subset of the dataset. During the training process, model parameters are periodically uploaded to OSS, facilitating parameter storage and synchronization across the system. By utilizing OSS instead of a centralized parameter server, the framework reduces communication bottlenecks, as the server is no longer responsible for handling all parameter synchronization requests.

To ensure efficient model updates, the server aggregates all updates from the workers using a parameter aggregation strategy. The server then uploads the updated model parameters back to OSS for the next training round. This process supports asynchronous updates, allowing the workers to independently update and upload their model parameters without the need for global synchronization at each step. This minimizes communication overhead and enhances training efficiency. The periodic model updates, combined with the asynchronous strategy, help reduce global synchronization costs and improve the stability and scalability of the training process.

2.2 Role of OSS in the Framework

During the training process, OSS was utilized as an alternative to the traditional parameter server for parameter synchronization, aiming to reduce communication bottlenecks and enhance training efficiency. Conventional distributed training relies on parameter servers for synchronization, which often leads to excessive server load and degraded performance. Instead, this framework employs OSS as the synchronization medium, enabling workers to asynchronously upload parameters, while the server periodically aggregates them, thereby mitigating the burden of handling a large number of communication requests. Furthermore, asynchronous updates were adopted to address the issue of "slow workers" delaying overall training progress in synchronous updates. Under a synchronous update scheme, all workers must complete training and synchronize parameters simultaneously, whereas asynchronous updates allow each worker to train independently and upload parameters at its own pace. The server retrieves updates at fixed intervals, reducing computational resource wastage.

2.3 Server-Side Implementation

The server is responsible for initializing the global model, aggregating worker updates, and periodically synchronizing the model through OSS. At the start of training, the server initializes the model with random or pre-trained weights and uploads it to OSS, making it accessible to all workers. Throughout training, the server continuously fetches updated model parameters uploaded by workers. To aggregate model updates from multiple workers efficiently, the FedAvg algorithm is employed. Instead of averaging model parameters directly, FedAvg performs a weighted aggregation based on the number of training samples each worker processes. The global model update at round $t+1$ is computed as follows:

$$\theta_{t+1} = \sum_{i=1}^{N} \frac{n_i}{n} \theta_i^t \quad (1)$$

where N is the total number of active workers, n_i is the number of training samples used by worker i, $n = \sum_{i=1}^{N} n_i$ is the total number of training samples across all active workers, and θ_i^t represents the locally trained model from worker i at round t. This weighted averaging ensures that workers contributing more data have a larger influence on the global model, leading to a more balanced and representative update.

This aggregation process ensures that the global model reflects contributions from all workers while mitigating the impact of outliers or poorly trained local models. By performing updates at fixed intervals rather than synchronously collecting all worker updates, the server avoids delays caused by slow nodes and maintains efficient training progress (Table 1).

Table 1 Server-Side Training Process

Algorithm 1: Server-Side Training Process
Input: Number of workers *N*, training rounds *T*, initial model θ_0
Output: Final trained model θ_T
1: Initialize global model θ_0 and upload to OSS
2: for t = 1, 2, …, T do
3: Wait for worker updates in OSS
4: Download worker models $\{ \theta_i^t \}$ from OSS
5: Compute new global model using FedAvg
6: Upload updated global model θ_{t+1} to OSS
7: end for
8: return trained model θ_T

2.4 Worker-Side Implementation

Each worker operates independently, retrieving the latest global model from OSS before performing local training on a subset of the dataset. After training for a predefined number of epochs, the worker uploads the updated model parameters back to OSS. This asynchronous update mechanism prevents slower workers from delaying the overall training process.

During local training, each worker employs Stochastic Gradient Descent (SGD) to update model parameters. The training process iterates through mini-batches, computing gradients and adjusting the model accordingly [6]. By periodically synchronizing with OSS, workers contribute to the global model while maintaining flexibility in update timing.

Algorithm 2 outlines the worker-side training process, detailing how workers download the global model, perform training, and upload updates. This decentralized approach

enhances scalability by allowing workers to operate independently without requiring strict synchronization (Table 2).

Table 2 Worker-Side Training Process

Algorithm 2: Worker-Side Training Process
Input: Local dataset D_i, number of epochs E, batch size B, learning rate η, global model θ Output: Updated worker model θ_i 1: Download global model θ from OSS 2: Select a random subset D_i from the dataset 3: for epoch e = 1, 2, …, E do 4: for each batch (x_i, y_i) in D_i do 5: Compute gradient $g_j = \nabla_\theta L(f(x_j; \theta), y_j)$ 6: Update model parameters 7: end for 8: end for 9: Upload updated model θ_i to OSS

3. Experiment

3.1 Datasets and Model Setting

Datasets. For experimental evaluation, this framework utilizes CIFAR-100 as the training datasets. CIFAR-100 consists of 60,000 images of 32×32 color, categorized into 100 classes [7]. For training, 50,000 images are used, while 10,000 images are reserved for testing.

For experimental evaluation, a CNN model was used for classification tasks on the CIFAR-100 datasets [8]. The architecture consists of two convolutional layers followed by pooling layers to reduce spatial dimensions, and two fully connected layers to capture high-level features. ReLU activation is applied in the hidden layers for non-linearity, while the final Softmax output layer provides probabilities for multi-class classification.

3.2 Evaluation Metrics

The performance of the proposed distributed training framework was evaluated using four key metrics: model accuracy, which measures classification performance on the test dataset; model convergence speed, indicating how quickly the model reaches an optimal solution; training time, reflecting the efficiency of the distributed training process; and communication time, which tracks the time spent on communication between workers and OSS, impacting scalability and performance [9,10].

3.3 Ablation Experiment

In the ablation experiments, the performance and convergence speed of different training frameworks are evaluated. Specifically, the OSS framework based on cloud and the traditional distributed training framework are employed. The OSS framework updates the model through cloud storage as the medium, while the traditional distributed training framework updates it through the parameter server.

3.3.1 Experiment Setup

Both the traditional and proposed frameworks use the same CNN model for consistency and employ the SGD optimizer with a learning rate of 0.002. Both frameworks run 5 epochs per worker per round. Workers in both frameworks randomly select 10,000 samples from the CIFAR-100 dataset for local training, and the FedAvg strategy is used for model aggregation. The primary difference lies in the communication protocol, with the traditional architecture using TCP socket-based connections for synchronous communication, while the proposed framework utilizes Alibaba Cloud OSS for communication.

3.3.2 Experimental Results

In this ablation experiment, the comparison between the traditional socket-based framework and the OSS framework reveals several key differences in terms of model accuracy, convergence speed, training time, and communication time.

1) Model Accuracy. The model accuracy in the traditional framework steadily improves, reaching 56.14% by round 60. This suggests that while the model shows continuous improvement, the convergence process is relatively slow. On the other hand, the OSS framework reaches 56.7% after just 40 rounds, indicating a significantly faster convergence. The OSS framework's faster accuracy gain in the early rounds suggests that its asynchronous communication and efficient model update process contribute to quicker training.

2) Model Convergence Speed. The traditional framework takes longer to converge, with 60 rounds required to stabilize at an accuracy of 56.14%. In contrast, the OSS framework converges faster, reaching a steady state in 40 rounds. This improvement in convergence speed highlights the efficiency of asynchronous communication and the optimized handling of model updates in the OSS framework (Figure 2).

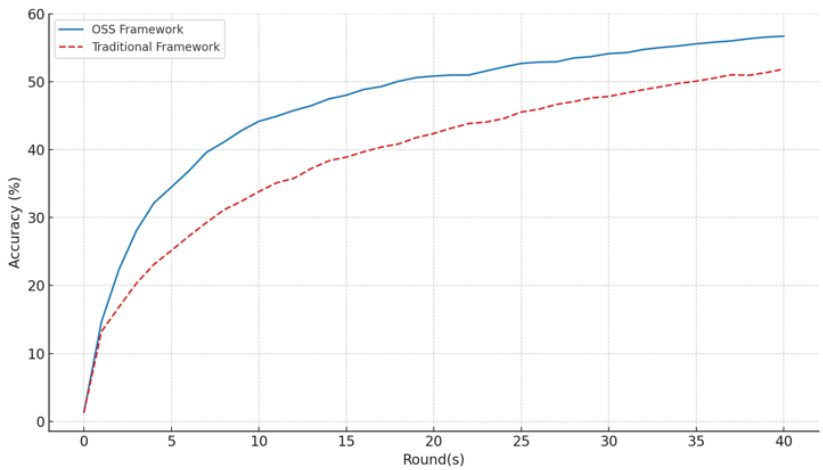

Figure 2. Accuracy comparison between OSS Framework and Traditional Framework(Picture credit : Original)

3) **Training Time**. The training time in the traditional framework is notably higher, with a range of 81.63 seconds (minimum) to 92.69 seconds (maximum), and an average of 84.30 seconds. This is mainly due to the synchronous nature of the communication, which introduces delays in model aggregation and synchronization. In contrast, the OSS framework reduces training time, with times ranging from 54.95 seconds (minimum) to

59.48 seconds (maximum), and an average of 57.10 seconds. The reduction in training time indicates that the asynchronous communication, coupled with OSS's scalable model upload and download process, leads to more efficient training.

4) Communication Time. The communication time between the two frameworks is similar, with the traditional framework having a range of 0.2014 seconds to 0.4052 seconds, averaging 0.2556 seconds. The OSS framework has slightly higher communication times, ranging from 0.1906 seconds to 0.4179 seconds, with an average of 0.2413 seconds. Despite the slight increase in communication time in the OSS framework, the trade-off is minimal when considering the advantages of asynchronous operations and scalable data management provided by object storage (Table 3).

Table 3. Comparison of Training and Communication Time Between OSS and Traditional Frameworks

Consumption	Training (s)		Communication (s)	
	OSS	Traditional	OSS	Traditional
Max	59.48	92.69	0.4179	0.4052
Min	54.95	81.63	0.1906	0.2014
Average	57.10	84.30	0.2413	0.2556

In conclusion, the OSS framework offers faster convergence, reduced training time, and comparable communication efficiency, making it more suitable for large-scale distributed federated learning tasks. The traditional framework, while effective for smaller, synchronous experiments, is slower in terms of both convergence and overall efficiency.

3.4 Impact of Varying Worker Counts

3.4.1 Specific Process

*T*he training process involves model initialization, distributed training, parameter synchronization, and updates. The specific steps are as follows:

The server initializes the CNN model and uploads its weight parameters to OSS for all workers. Each worker downloads the latest model, trains it locally on a subset of the CIFAR-100 dataset using the SGD optimizer, and uploads the updated parameters to OSS. The server periodically checks OSS for updates from all workers, aggregates the parameters using FedAvg, and generates a new global model. The server then evaluates the new model's accuracy against the previous one; if the new model performs better, its parameters are retained; otherwise, the training process stops. This cycle is repeated until the training is complete.

3.4.2 Experimental Results

Based on the analysis of the data, the impact of varying worker counts on the cloud server's OSS framework can be discussed across four key factors: model accuracy, convergence speed, training time, and communication time.

1) **Model Accuracy**. Increasing the number of workers leads to a steady improvement in model accuracy, but the rate of improvement diminishes as more workers are added. Initially, the increase from 5 to 10 workers shows a significant boost in accuracy, with a jump from 1.33% to 14.16%. However, by 20 workers, the accuracy stabilizes around 58%, suggesting that additional workers beyond this point yield diminishing returns in terms of accuracy gains. Interestingly, the performance seen at 15 workers provides a good balance,

offering substantial improvements without the diminishing returns seen at higher worker counts (Figure 3).

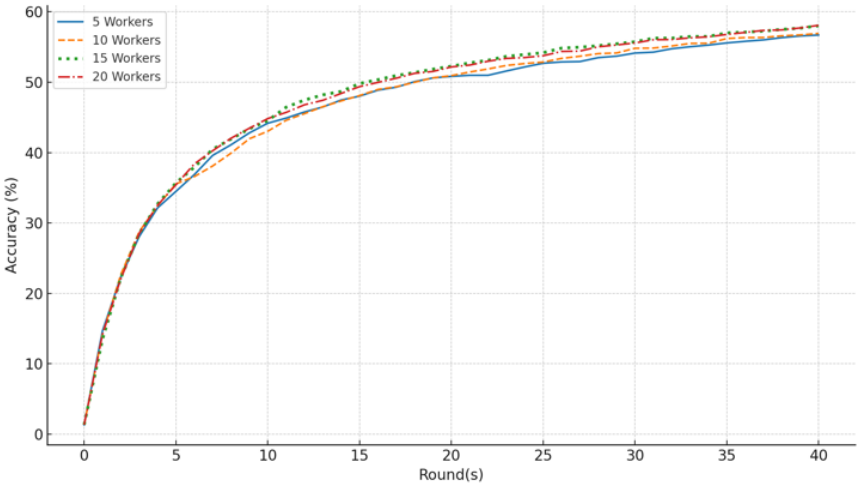

Figure 3. Impact of Worker Count on Accuracy Progression Across 40 Training Rounds (Picture credit : Original)

2) Convergence Speed. The number of workers plays a crucial role in accelerating the convergence speed of the model, particularly during the early stages of training. As the number of workers increases, the model's accuracy improves at a faster rate, reflecting quicker convergence. However, once the worker count reaches 20, the speed of convergence begins to slow down significantly, indicating that the system is approaching a convergence point. Beyond this point, further increases in worker numbers offer limited benefits. In this context, 15 workers seem to offer the best trade-off, providing a substantial improvement in convergence speed without the diminishing returns observed at higher worker counts.

3) Training Time. The addition of workers does not result in a substantial decrease in training time. In fact, both the maximum and average training times tend to slightly increase with more workers. For instance, the maximum training time for 20 workers is 59.48 seconds, which is marginally higher than the 57.57 seconds observed with 5 workers. This suggests that the parallel computing overhead and communication costs associated with increasing the number of workers may offset any potential time savings in training. Thus, despite the increase in workers, the training time does not experience a significant reduction, especially as the number of workers exceeds 15.

4) Communication Time. The impact of worker count on communication time is relatively minor. The average communication time remains stable across different worker configurations, with only small fluctuations. For instance, the average communication time for 5 workers is 0.23 seconds, which remains the same for 20 workers. This suggests that the cloud server's architecture is well-optimized for communication, with the bandwidth and latency not significantly affected by an increase in the number of workers (Table 4).

Table 4. Training Time and Communication Time Comparison for Different Worker Counts

Time	Training (s)			Communication (s)		
	Max	Min	Avg	Max	Min	Avg
5 workers	57.57	54.95	56.13	0.55	0.15	0.23
10 workers	58.22	55.34	56.63	0.32	0.16	0.22

15 workers	57.66	56.49	57.11	0.36	0.18	0.22
20 workers	59.48	56.69	58.02	0.35	0.16	0.23

While increasing the number of workers enhances both model accuracy and convergence speed, the improvement becomes less pronounced beyond a certain threshold, particularly around 20 workers. Additionally, increasing the number of workers does not lead to a significant reduction in training time, and may even slightly increase it due to additional computational and communication overhead. Communication time remains relatively unaffected by the increase in workers. Overall, 15 workers appear to provide the best balance, offering substantial improvements in performance without the diminishing returns and resource overhead seen at higher worker counts. Thus, in a cloud server environment, choosing an optimal number of workers requires balancing performance gains with resource overhead.

4. Conclusion

This paper proposes a novel distributed training framework based on cloud OSS, aiming to solve the communication bottlenecks and efficiency issues inherent in traditional distributed training frameworks. By adopting an asynchronous update mechanism and FedAvg algorithm, the framework effectively reduces the communication overhead of parameter synchronization while improving training stability and efficiency. Experimental results show that the OSS-based framework outperforms traditional parameter server-based frameworks in terms of faster convergence and reduced training time, especially in terms of training efficiency. Furthermore, as the number of worker nodes increases, the OSS framework maintains good performance and demonstrates excellent scalability. However, the experiments also indicate that too many worker nodes can lead to increased computational and communication overhead. Therefore, selecting an optimal number of worker nodes is crucial in practical applications. Future research can focus on further optimizing resource allocation and communication strategies to enhance the framework's performance in larger-scale distributed learning tasks.

References

1. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M. A., Senior, A. W., Tucker, P. A., Yang, K., and Ng, A. Y.: 'Large scale distributed deep networks', Proc. NIPS, 2012, pp. 1232–1240.
2. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y.: 'Scaling distributed machine learning with the parameter server', Proc. 11th USENIX Symp. Operating Syst. Design Implement. Broomfield, CO, 2014, pp. 583–598.
3. Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y.: 'Scaling distributed machine learning with the parameter server', Proc. 11th USENIX Symp. Operating Syst. Design Implement., Broomfield, CO, 2014, pp. 583–598.
4. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M.: 'A View of Cloud Computing', Commun. ACM, 2010, 53, (4), pp. 50–58.

5. McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. Y.: 'Communication-efficient learning of deep networks from decentralized data', Proc. 20th Int. Conf. Artif. Intell. Stat., 2017, pp. 1273–1282.
6. Bottou, L.: 'Stochastic gradient descent tricks', Neural Networks: Tricks of the Trade, Springer, 2012, pp. 421–436.
7. Krizhevsky, A., Nair, V., and Hinton, G. E.: 'Learning multiple layers of features from tiny images', Technical report, University of Toronto, 2009.
8. LeCun, Y., et al.: 'Gradient-based learning applied to document recognition', Proceedings of the IEEE, 1998, 86, (11), pp. 2278–2324.
9. Abdullah, R. M., Abdulrahman, L. M., Abdulkareem, N. M., et al.: 'Modular Platforms based on Clouded Web Technology and Distributed Deep Learning Systems', Journal of Smart Internet of Things (JSIoT), 2023,(02), pp. 162–173.
10. Zhang, H., Li, Y., Deng, Z., et al.: 'Autosync: Learning to synchronize for data-parallel distributed deep learning', Advances in Neural Information Processing Systems, 2020, 33, pp. 906–917.