International Journal of Advanced Smart Convergence Vol.13 No.3 143-149 (2024)
http://dx.doi.org/10.7236/IJASC.2024.13.3.143

IJASC 24-3-16

Study on Accelerating Distributed ML Training in Orchestration

Su-Yeon Kim*, Seok-Jae Moon**

** The master's course, Graduate School of Smart Convergence, Kwangwoon University, Seoul, Korea*
*** Professor, Graduate School of Smart Convergence, KwangWoon University, Seoul, Korea*
E-mail : {rlatndus0304, msj8086}@kw.ac.kr

Abstract

As the size of data and models in machine learning training continues to grow, training on a single server is becoming increasingly challenging. Consequently, the importance of distributed machine learning, which distributes computational loads across multiple machines, is becoming more prominent. However, several unresolved issues remain regarding the performance enhancement of distributed machine learning, including communication overhead, inter-node synchronization challenges, data imbalance and bias, as well as resource management and scheduling. In this paper, we propose ParamHub, which utilizes orchestration to accelerate training speed. This system monitors the performance of each node after the first iteration and reallocates resources to slow nodes, thereby speeding up the training process. This approach ensures that resources are appropriately allocated to nodes in need, maximizing the overall efficiency of resource utilization and enabling all nodes to perform tasks uniformly, resulting in a faster training speed overall. Furthermore, this method enhances the system's scalability and flexibility, allowing for effective application in clusters of various sizes.

Keywords: *Distributed Machine Learning, ML Model Training, Orchestration, Parameter Server, Resource Allocation*

1. INTRODUCTION

As the size and complexity of models rapidly increase, training times on large datasets are significantly rising [1, 2]. Training these large deep learning models, which include billions of parameters, on a system with a single GPU or CPU can take a considerable amount of time [3, 4]. However, by utilizing distributed training to split data across multiple nodes and process it in parallel, training times can be reduced [5]. In a distributed environment, if the resource utilization among nodes is uneven, some nodes may end up waiting for others to catch up. This resource allocation issue can lead to bottlenecks, ultimately slowing down the overall training speed [6]. Without effective resource management and optimized distributed strategies, it can be challenging to achieve performance improvements due to communication overhead and inefficient resource usage. Therefore, optimizing resource management and distributed strategies is crucial for the distributed training of large models. This paper proposes a method for accelerating distributed machine learning training using

Manuscript Received: July. 18. 2024 / Revised: July. 24. 2024 / Accepted: July. 29. 2024
Corresponding Author: msj8086@kw.ac.kr
Tel: 02-940-8283, Fax: 02-940-5443
Author's affiliation: Professor, Graduate School of Smart Convergence, Kwangwoon University, Seoul, Korea

orchestration. The proposed system consists of two layers: Worker and ParamHub. The Worker layer includes the Orchestration and NodeCore components, which are responsible for resource allocation and training across each node. Within the Orchestration component, there are two key modules: the Scheduler and the Resource Manager. The Scheduler module efficiently distributes resources among multiple running nodes, while the Resource Manager module plays a critical role in monitoring and managing these nodes. The NodeCore component consists of the Node Manager, Container, and Data Node modules. The Node Manager is responsible for managing the containers and monitoring resource usage, reporting this information to the Resource Manager. The Container provides the necessary resources for training, while the Data Node is responsible for training on the data assigned to it. The ParamHub layer includes an Aggregator module, which collects the results trained from each node and updates them by summing their values. The structure of this paper is as follows. Chapter 2 describes the overview and components of the proposed system and explains the overall operational flow of the system. Chapter 3 conducts comparative analysis for performance analysis, and Chapter 4 concludes the paper with a discussion of the conclusion and future research directions.

2. PROPOSED SYSTEM

2.1. System Overview

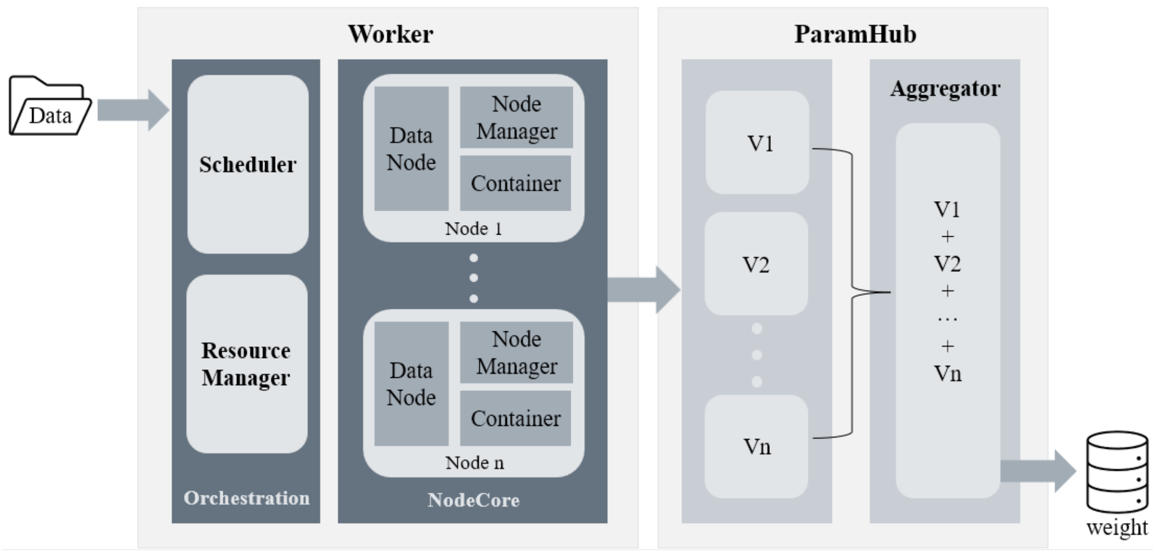


Figure 1. Overview of the Proposed System

Figure 1 shows the overview of the proposed system in this paper. The system consists of two layers: Worker and ParamHub. The Worker layer is responsible for resource allocation and training on each node and is composed of Orchestration and NodeCore. Orchestration includes the Scheduler and Resource Manager modules, while NodeCore contains the DataNode, NodeManager, and Container modules for each node. ParamHub consists of an Aggregator that aggregates and updates the data trained on each node. The descriptions of each module are as follows:

Orchestration allocates resources to nodes with decreased training speed, efficiently distributing resources among multiple running nodes and managing the nodes.

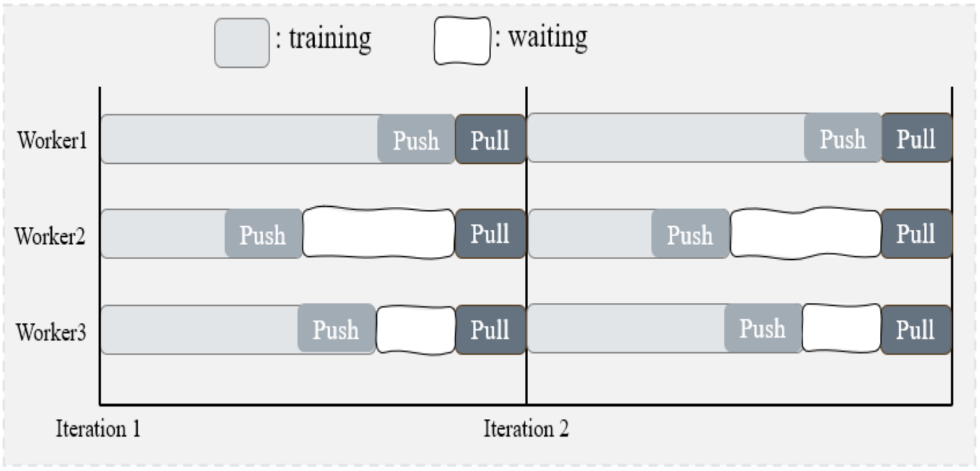
- Resource Manager: Performs initial configuration and resource requests for node execution, monitors the status of running nodes, reallocates resources when necessary, and recovers resources once node training is completed.

- Scheduler: Identifies available resources and allocates them to requesting nodes.

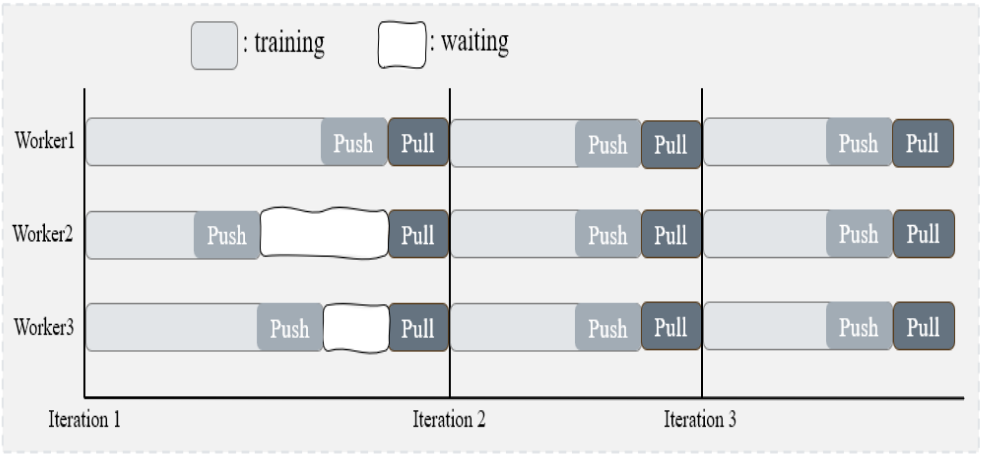
NodeCore consists of N nodes, each of which trains data.

- Node Manager: Manages containers and monitors their resource usage, reporting this information to the Resource Manager.
- Container: Each container is allocated a set of resources necessary for training and provides the required resources for execution.
- Data Node: Stores the training data assigned to it and sends the trained data to ParamHub after training.

ParamHub is responsible for receiving, aggregating, and updating the data trained by each node.



(a) Iterations Train without Orchestration



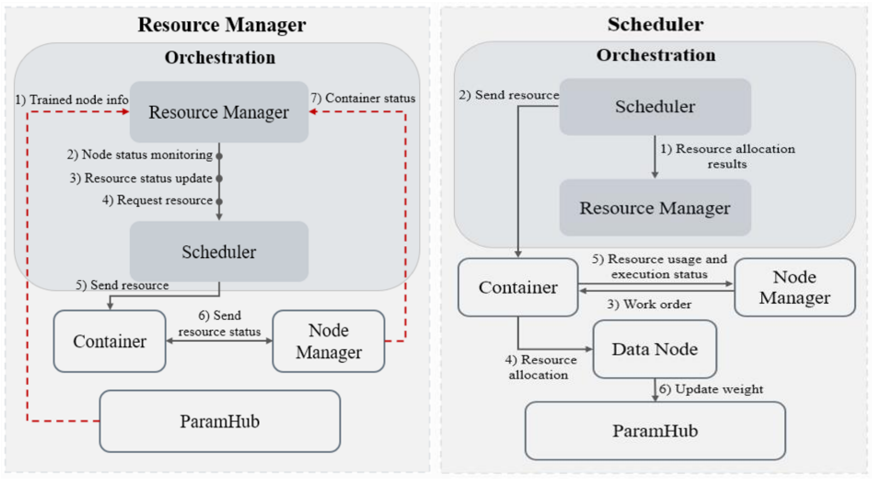
(b) Iterations Train with Orchestration

Figure 2. Comparative Analysis of Node Tasks with and without Orchestration

Figure 2 shows the task speed with and without Orchestration allocating resources to slow nodes. In a distributed environment, all nodes must wait for the slowest node to finish before starting the next iteration.

This results in idle nodes during each iteration, wasting time and increasing the overall training time. Orchestration allocates resources to delayed nodes, distributing the workload more evenly. This minimizes the occurrence of idle nodes during each iteration and reduces the overall training time. Figure (a) shows a situation where tasks are completed sequentially, but waiting time occurs because some nodes finish tasks faster than other nodes. This indicates an uneven workload distribution among nodes and results in inefficient time usage during task completion. Figure (b) shows a situation where Orchestration helps a node whose work is delayed. This reduces waiting times, allowing all nodes to complete their tasks at a consistent speed. By allocating resources to nodes experiencing delays after the first iteration, the overall system efficiency is maximized. Consequently, all nodes handle the workload more evenly, and the total training time is shortened.

2.2. System Component



(a) Resource Manager Flow **(b) Scheduler Flow**

Figure 3. Resource Management and Orchestration Structures in Distributed Systems

Figure 3 shows how the Resource Manager and Scheduler interact within the Orchestration. Figure (a) shows the flow of the Resource Manager. It begins with nodes sending their training information to the Resource Manager after the first iteration. The Resource Manager monitors the node status and updates the resource status to the Scheduler, requesting the necessary resources. The Scheduler allocates the requested resources to the Container and communicates this information to the Node Manager. Finally, the Node Manager reports the Container status back to the Resource Manager. Figure (b) depicts the flow of the Scheduler. It starts with the resource allocation results being sent to the Resource Manager. The Scheduler allocates resources to the Container, and the Node Manager issues work commands to the Container. The Container provides the allocated resources to the Data Node and reports resource usage and task status to the Node Manager. The Data Node uses the allocated resources to proceed with training and updates the results to ParamHub.

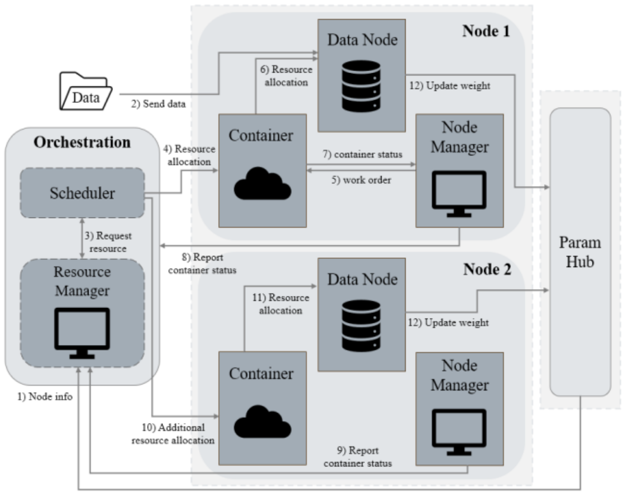


Figure 4. Resource allocation in distributed ML training

Figure 4 shows the overall system flow and the process of allocating resources to a slow node. The Resource Manager receives information about each node after the first iteration. This monitors and updates the status of each node, the number of tasks currently waiting, and the resource requirements of each task. The Resource Manager then requests the necessary resources from the Scheduler. The Scheduler accepts these requests and allocates resources, such as memory or CPU, to the Containers as needed by each node. The Node Manager manages the status of the Containers, monitors resource usage, and performs tasks such as starting or stopping Containers based on the node's execution state. Additionally, the Node Manager issues commands for the tasks that the Containers must perform and communicates any necessary status changes or updates. The Container allocates resources to the Data Node based on the received commands, and the Node Manager monitors the Container's resource status, reporting this to Orchestration. If Node 2 is a slow node, the Node Manager reports the insufficient resource allocation to the Resource Manager. Consequently, the Resource Manager requests additional resources from the Scheduler. The Scheduler then allocates the necessary additional resources to the Container, which in turn provides these resources to the Data Node. Finally, the Data Node updates the trained weight values to the ParamHub.

3. COMPERATIVE ANALYSIS

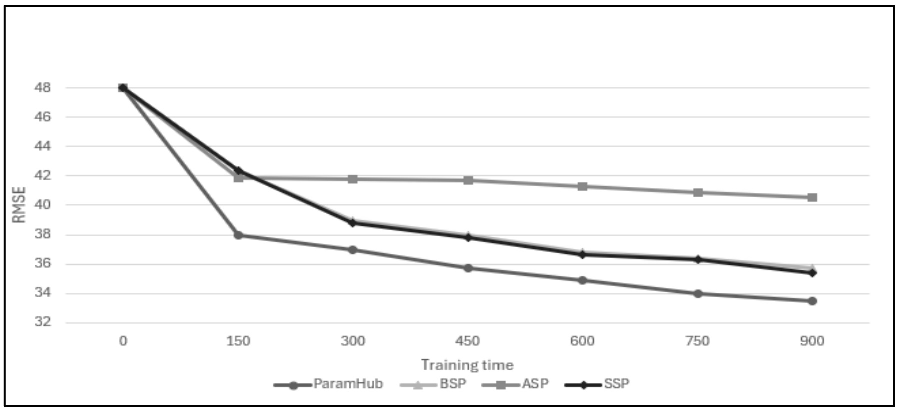


Figure 5. Comparsion of RMSE by training time

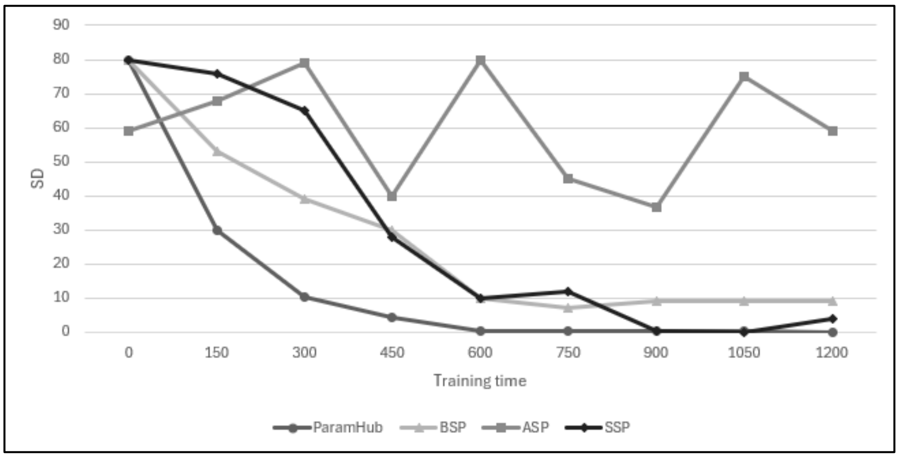


Figure 6. Comparsion of SD by training time

This section compares the performance of ParamHub with existing distributed machine learning algorithms such as BSP, SSP, and ASP. Figure 5 measures the RMSE over training time. When the threshold is set to 38, the proposed system reduces RMSE most quickly and maintains low RMSE values throughout the training period. This indicates that the proposed system accelerates the training speed most efficiently. BSP faces the straggler problem, where the slowest worker delays the entire training time as all workers wait for the slowest one to complete its task. Initially, BSP reduces RMSE faster than ASP and SSP, but over time, the gap with the proposed system widens. ASP attempts to mitigate the impact of slow workers by having workers perform tasks asynchronously, but this can lead to the stale parameter problem. While ASP does reduce RMSE, it does so at a relatively slower rate compared to other methods. SSP shows a similar trend to BSP but allows for some delay, enabling workers to perform updates asynchronously. Although SSP performs better than BSP, it still records higher RMSE values compared to the proposed system. Figure 6 measures the SD over training time. When the threshold is set to 30, the proposed system maintains a low SD value, indicating consistency in the training process. This means there is less fluctuation in performance during each iteration, leading to stable training. BSP can have high SD values due to significant waiting times during the training process, resulting in larger performance fluctuations and lower consistency. ASP can also exhibit high SD values because the parameters provided by slow workers may be outdated, reducing training consistency and causing larger performance fluctuations. SSP, while not waiting for slow workers and using the latest parameters for training, can still see an increase in SD values due to the influence of slow workers.

4. CONCLISION

This paper proposes an orchestration-based system to enhance training speed in large-scale distributed machine learning training by optimizing resource allocation and distribution strategies. The proposed system monitors resource allocation after the first iteration and redistributes resources to slower nodes, ensuring each node utilizes resources evenly. This approach balances resource usage and minimizes bottlenecks, thus reducing overall training time. Additionally, the proposed system demonstrated superior efficiency compared to traditional methods in comparative experiments. These results highlight the importance of resource management and distribution strategies in distributed machine learning environments and show that the proposed orchestration system offers substantial performance improvements. Future research is needed to improve the scalability of the system so that it can handle very large datasets and models.

ACKNOWLEDGMENT

※ This paper was supported by the KwangWoon University Research Grant of 2024.

REFERENCES

[1] A. Tariq, L. Cao, F. Ahmed, E. Rozner, and P. Sharma, “Accelerating Containerized Machine Learning Workloads,” NOMS 2024-2024 IEEE Network Operations and Management Symposium. IEEE, May 06, 2024.
DOI: https://doi.org/10.1109/NOMS59830.2024.10575188

[2] Y. Chen, Y. Peng, Y. Bao, C. Wu, Y. Zhu, and C. Guo, “Elastic parameter server load distribution in deep learning clusters,” Proceedings of the 11th ACM Symposium on Cloud Computing. ACM, Oct. 12, 2020.
DOI: https://doi.org/10.1145/3419111.3421307

[3] I. Thangakrishnan, D. Cavdar, C. Karakus, P. Ghai, Y. Selivonchyk, and C. Pruce, “Herring: Rethinking the Parameter Server at Scale for the Cloud,” SC20: International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, Nov. 2020.
DOI: https://doi.org/10.1109/sc41405.2020.00048

[4] A.-L. Jin, W. Xu, S. Guo, B. Hu, and K. Yeung, “PS+: A Simple yet Effective Framework for Fast Training on Parameter Server,” IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 12. Institute of Electrical and Electronics Engineers (IEEE), pp. 4625–4637, Dec. 01, 2022.
DOI: https://doi.org/10.1109/tpds.2022.3200518

[5] A. Renz-Wieland, R. Gemulla, S. Zeuch, and V. Markl, “Dynamic parameter allocation in parameter servers,” Proceedings of the VLDB Endowment, vol. 13, no. 12. Association for Computing Machinery (ACM), pp. 1877–1890, Aug. 2020.
DOI: https://doi.org/10.14778/3407790.3407796

[6] S. Wang, A. Pi, X. Zhou, J. Wang, and C.-Z. Xu, “Overlapping Communication With Computation in Parameter Server for Scalable DL Training,” IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 9. Institute of Electrical and Electronics Engineers (IEEE), pp. 2144–2159, Sep. 01, 2021.
DOI: https://doi.org/10.1109/tpds.2021.3062721