International Journal of Internet, Broadcasting and Communication Vol.16 No.2 161-168 (2024)
http://dx.doi.org/10.7236/IJIBC.2024.16.2.161

IJIBC 24-2-19

Design of a ParamHub for Machine Learning in a Distributed Cloud Environment

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 big data models grows, distributed training is emerging as an essential element for large-scale machine learning tasks. In this paper, we propose ParamHub for distributed data training. During the training process, this agent utilizes the provided data to adjust various conditions of the model's parameters, such as the model structure, learning algorithm, hyperparameters, and bias, aiming to minimize the error between the model's predictions and the actual values. Furthermore, it operates autonomously, collecting and updating data in a distributed environment, thereby reducing the burden of load balancing that occurs in a centralized system. And Through communication between agents, resource management and learning processes can be coordinated, enabling efficient management of distributed data and resources. This approach enhances the scalability and stability of distributed machine learning systems while providing flexibility to be applied in various learning environments.

Keywords: *Agent System, Parameter, Distributed Training, Efficiency, Machine Learning*

1. INTRODUCTION

In recent years, machine learning has been applied to various fields such as business, science, and online services, with the size of data and training models continuing to increase [1]. Machine learning models typically have many parameters, and effectively training such models requires vast amounts of data and computation [2]. Examples include image recognition [3], face recognition [4], and self-driving cars [5]. However, efficient training has become challenging due to the additional time costs incurred by training models, mainly because of large datasets and complex model architectures [6]. To optimize the performance of your model or achieve the desired results, you need to adjust parameters such as model structure, performance tuning, hyperparameters, and biases. Therefore, as dataset size and model complexity continue to increase, training models in a distributed environment has become essential for large-scale machine learning tasks [7]. In this paper, we propose ParamHub for distributed data training to minimize and expedite the costs incurred in learning when models based on large-scale datasets are applied. The proposed system consists of three layers: DataSet, Worker, and ParamHub. The Worker layer consists of the Task Scheduler module, the Resource Monitoring module, and multiple nodes, responsible for training. The Task Scheduler module

Manuscript Received: April. 1, 2024 / Revised: April. 13, 2024 / Accepted: April. 20, 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

partitions the data among nodes, while the Resource Monitoring module allocates necessary resources through monitoring. The ParamHub layer is responsible for updating data such as weight condition and is comprised of the Server Manager, Controller, and Update Model modules. The Server Manager module receives and manages trained data information from each node. The Controller module repeatedly trains data information and sends the information to the Update Model module when parameter values are optimized. The structure of this paper is as follows: Section 2 describes the system outline and operational flow and describes the overall flow with a sequence diagram. Section 3 offers a comparative analysis of the performance between the existing system and the proposed system, while Section 4 concludes with a conclusion.

2. PROPOSED SYSTEM

2.1. System Component

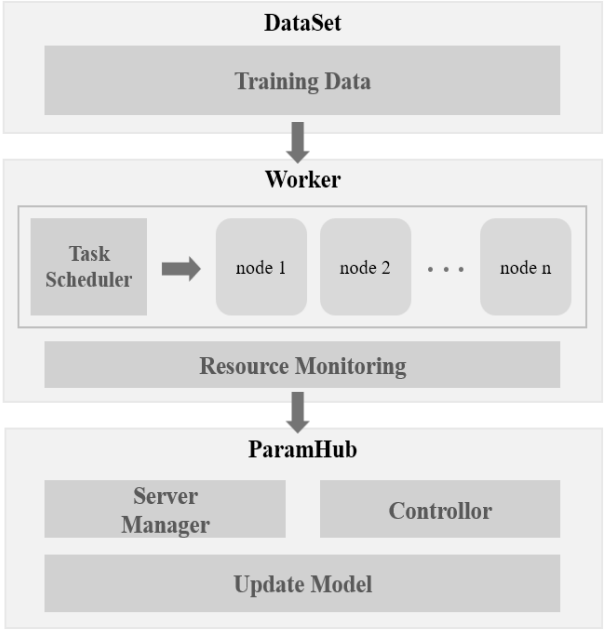


Figure 1. Overview of the proposed system

Figure 1 presents an overview of the system proposed in this paper. The proposed system consists of three layers: DataSet, Worker, and ParamHub. The DataSet consists of training data. The Worker layer consists of the Task Scheduler module and the Resource Monitoring module. ParamHub consists of the Server Manager module, Controller module, and Update Model module. The description of each module is as follows.

- Task Scheduler: This module checks node status and is responsible for transmitting data to each node. Subsequently, it receives and processes the data to be trained from the DataSet layer.
- Resource Monitoring: The Resource Monitoring module collects trained data from each node, manages resources through monitoring, and transmits it to the Server Manager module.
- Server Manager: This module aggregates the data from each node into a single result and then transmits it to the Controller module.
- Controller: The Controller module judges the result and transmits it to the DataSet when additional data training is needed, and then repeats the process.

• Update Model: This module repeats the above process several times, updating the values when parameters of the entire model have been optimized.

Figure 2 shows the operational flow of data distribution training for ParamHub proposed in this paper. The data to be trained from the DataSet is sent to the Task Scheduler module. After receiving the data, the Task Scheduler module divides it into N nodes and transmits it. The number of nodes is determined by considering factors such as system requirements, workload, data volume, and processing speed. Each node uses the provided data to adjust and learn model parameters. Afterward, the learned values are transmitted to the Resource Monitoring module. This module is responsible for collecting training data from each node and monitoring and managing it. And this information is transmitted to the Server Manager module. Then, it adds up all the values received from the nodes and sends the result to the Controller module. The Controller module trains a model by repeatedly using data based on the received results. As the data is trained repeatedly, predictions can be calculated, and patterns can be identified. The final results are sent to the Update Model module and are updated.

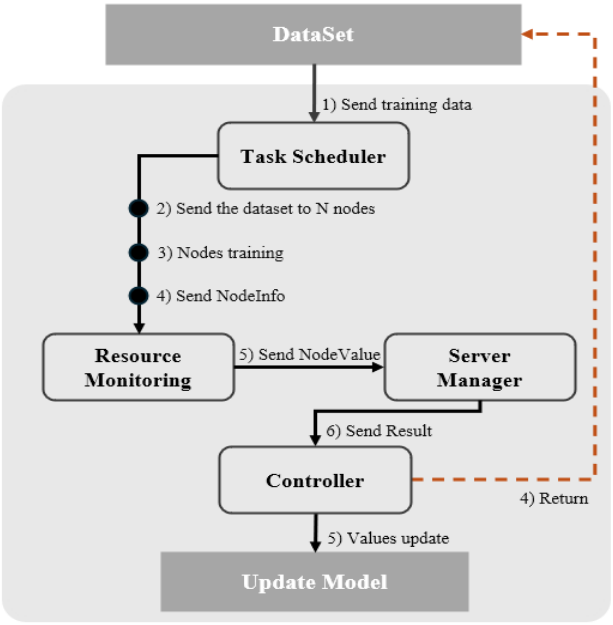


Figure 2. ParamHub workflow

2.2. Sequence Diagram

This section describes the overall flow of the system proposed in this paper and the data flow between each module. Figure 3 shows the flow of data and operations of the proposed system.

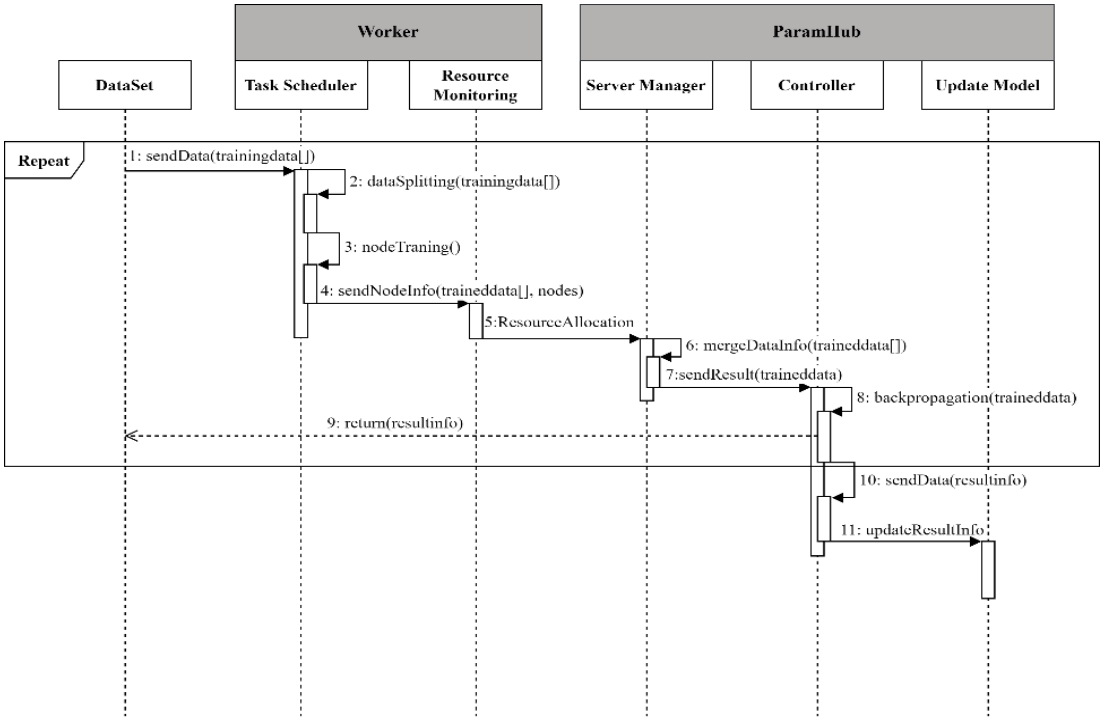


Figure 3. Sequence diagram of proposed system

1. sendData(trainingdata[]): The training data is sent from the DataSet to the Task Scheduler module.

2. dataSplitting(trainingdata[]): Split the data to be trained according to the number of N nodes.

3. nodeTraning(): Train the split data at each node.

4. sendNodeInfo(trainingdata[], nodes): Information on the trained data from each node is transmitted to the Resource Monitoring module.

5. resourceAllocation(): It receives resource allocation and transfers the information to the Server Manager module.

6. mergeDataInfo(traineddata[]): Add up all the training data values received from the N nodes.

7. SendResult(traineddata): Send the trained data to the controller module. This is used as input data for model updating.

8. backpropagation(traineddata): Adjust errors in the trained data. This process aims to find optimal model parameters.

9. return(resultinfo): Return the resulting data and train again. The model's parameters are repeatedly trained to minimize the error between the model's predictions and actual values.

10. sendData(resultinfo): The results obtained through multiple repeated training are updated in the Update Model module.

11. updateResultInfo(): The resulting information is sent to the Update Model module.

Algorithm 1 implements the ParmHub process for distributed machine learning, encompassing the adjustment of models through a central server using distributed data and updating each node's model.

Algorithm 1: ParmHub distributed machine learning algorithm

```
PamHub Procedure start:
        traingdataset[] = DataSet of n:

       tdata[] = Worker::TaskScheduler(traingdataset[])
distributed_data = distribute_data_round_robin(tdata[], num_nodes)
       tres = Worker::ResourceMoitoring(distributed_data):

merge_models = ParmHub::ServerManager(tres):
       ParmHub::Controller(merge_models):

procedure end;

func_Worker::TaskScheduler(traingdataset[], num_nodes):    // Distribute the data to each node
in a round-robin fashion.

        nodes_data = [[] for _ in range(num_nodes)]
for i, item in enumerate(traingdataset[]):
            node_index = i % num_nodes
            nodes_data[node_index].append(item)

return nodes_data

func_Worker::ResourceMoitoring(distributed_data[]):
X = np.array([x for x, _ in data]).reshape(-1, 1)
y = np.array([y for _, y in data])

model = LinearRegression()
model.fit(X, y) return nodes_data
distributed_data[] = train_linear_regression_model(node_data)

return distributed_data

func_ParmHub::ServerManager(tres):
coefs = [coef for coef, _ in models]
      intercepts = [intercept for _, intercept in models]

mean_coef = sum(coefs) / len(coefs)
      mean_intercept = sum(intercepts) / len(intercepts)

      return merged_coef, merged_intercept

func_ParmHub::Controller(tres):
    def update_weights_central_server(node_weights):     // Update the weights of the nodes by
averaging them on the central server.
          total_weight = sum(weight for weight, bias in node_weights)
          total_bias = sum(bias for weight, bias in node_weights)
          num_nodes = len(node_weights)

      updated_weight = total_weight / num_nodes
          updated_bias = total_bias / num_nodes

    return updated_weight, updated_bias
```

```
func_ParmHub::UpdateModel(merge_models):    // Propagate the updated weights and biases to all nodes. Each node applies these values to its own model.

def propagate_updated_weights_to_nodes(updated_weight, updated_bias, num_nodes):
        updated_node_weights = [(updated_weight, updated_bias) for _ in range(num_nodes)]
return updated_node_weights
```

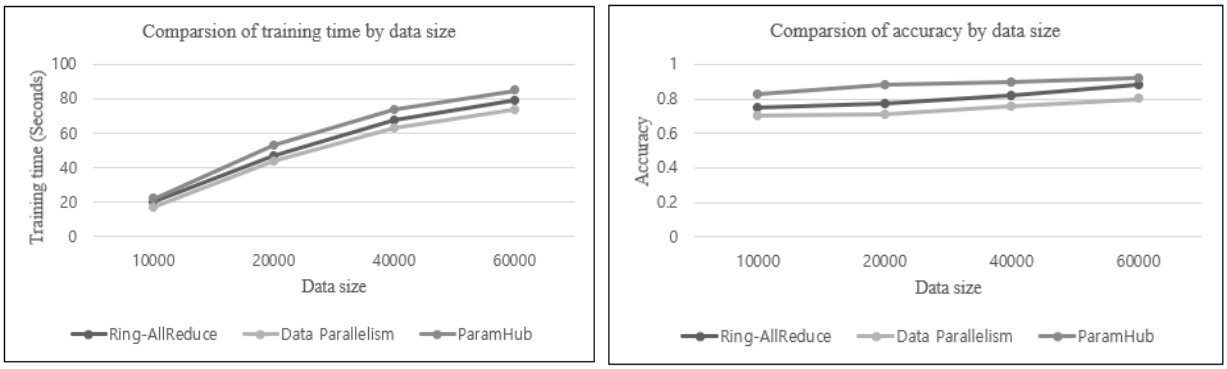
3. COMPERATIVE ANALYSIS

Table 2 summarizes the comparison results of major architectures used in distributed machine learning. The comparison targets are Ring-AllReduce [8] and data parallelism [9]. In this paper, we compared the communication method, consistency, and scalability of each architecture.

Table 2. Comparison of systems

	Ring-AllReduce[8]	Data Parallelism[9]	Proposed System
Communication	Efficient communication is achieved through the utilization of the Ring structure employing the AllReduce algorithm.	Data is processed in parallel, and direct communication is established between each device for updates.	Each agent communicates with the parameter server to perform updates.
Consistency	Consistency is maintained across all nodes with identical data	Due to multiple processing units concurrently handling the same data, consistency issues may arise.	The lack of guaranteed arrival order at the server can lead to consistency issues.
Scalability	It is suitable for scaling up and operates efficiently even as the number of nodes increases.	There are limitations regarding the amount of data and model size.	It is scalable by adjusting the number of agents.

First, in the communication method, Ring-AllReduce uses the All-Reduce algorithm to effectively communicate through a ring structure. Data Parallelism processes data in parallel and communicates directly between each device to synchronize updates. In the proposed system, the agent performs updates by communicating with a centralized parameter server. In terms of consistency, Ring-AllReduce communicates directly with neighboring nodes, ensuring that the same data is maintained across all nodes. Data Parallelism entails each processing unit performing tasks independently, thus potentially causing consistency issues with changes to the same data. The proposed system is asynchronous, and the order of arrival at the server is not guaranteed, which may lead to consistency problems. In terms of scalability, Ring-AllReduce is suitable for scaling up and operates efficiently even as the number of nodes increases. On the other hand, Data Parallelism has limitations in terms of the amount of data and the size of the model. The proposed system is scalable by adjusting the number of agents.



(A) Training time **(B) Accuracy**

Figure 4. Comparing speed and accuracy

Figure 4 is a chart comparing training times by data size using the 60,000 Fashion MNIST dataset. (A) measures the learning speed according to each data size, and it can be observed that the speed of the proposed system is somewhat low at 85 seconds. The time taken to update parameters increases mainly due to communication overhead and communication with a centralized server. This results in network communication and data transfer taking time, leading to more overhead, especially as data size increases. (B) is a chart measuring accuracy. You can see that the accuracy of the proposed system is 0.92 when there are 60,000 pieces of data, which is higher than that of other systems. Because the same parameters are shared and updated, the model remains consistent, which helps achieve higher accuracy during the training process. Therefore, the ParamHub proposed in this paper may have somewhat lower speed but can provide higher accuracy compared to other systems such as data parallelism or ring-allreduce.

4. CONCLISION

In this paper, we proposed ParamHub for distributed data training to minimize and accelerate the cost added to learning in situations where large-scale dataset-based models are applied. The proposed system can operate independently in a distributed environment, with each agent designed to perform tasks independently, aggregating data and updating the model when necessary. This reduces bottlenecks that occur in centralized control systems and enables efficient resource management in a distributed environment. In addition, each agent can manage and update necessary data in its own environment and efficiently operate complex data and model learning. However, because each agent learns and acts independently, the complexity and consistency of the learning process must be considered. Therefore, future research is needed to improve the agent's learning process.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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

[2] N. Provatas, I. Konstantinou, and N. Koziris, “Is Systematic Data Sharding able to Stabilize Asynchronous Parameter Server Training?,” 2021 IEEE International Conference on Big Data (Big Data). IEEE, Dec. 15, 2021. DOI: https://doi.org/10.1109/bigdata52589.2021.9672001.

[3] A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” arXiv, 2020.
DOI: https://doi.org/10.48550/ARXIV.2010.11929.

[4] M. Wang and W. Deng, “Deep face recognition: A survey,” Neurocomputing, vol. 429. Elsevier BV, pp. 215–244, Mar. 2021.
DOI: https://doi.org/10.1016/j.neucom.2020.10.081.

[5] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” Journal of Field Robotics, vol. 37, no. 3. Wiley, pp. 362–386, Nov. 14, 2019.
DOI: https://doi.org/10.1002/rob.21918.

[6] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, “A Survey on Large-Scale Machine Learning,” IEEE Transactions on Knowledge and Data Engineering. Institute of Electrical and Electronics Engineers (IEEE), pp. 1–1, 2020.
DOI: https://doi.org/10.1109/tkde.2020.3015777.

[7] A. Renz-Wieland, R. Gemulla, S. Zeuch, and V. Markl, “Dynamic Parameter Allocation in Parameter Servers,” arXiv, 2020.DOI: https://doi.org/10.48550/ARXIV.2002.00655.

[8] Y. Chao, M. Liao, and J. Gao, “Task allocation for decentralized training in heterogeneous environment.” arXiv, 2021.
DOI: https://doi.org/10.48550/ARXIV.2111.08272.

[9] Y. M. Park, S. Y. Ahn, E. J. Lim, Y. S. Choi, Y. C. Woo, and W. Choi, “Deep Learning Model Parallelism,” Electronics and Telecommunications Trends, vol. 33, no. 4, pp. 1–13, Aug. 2018.
DOI: https://doi.org/10.22648/ETRI.2018.J.330401