Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230835

A hybrid parallelization approach based on workers grouping algorithm

Ruoyu Lu

SouthWest Jiaotong University, Chengdu, Sichuan, 611756, China

ruoyu.lu@outlook.com

Abstract. As the volume of model data increases, traditional machine learning is not able to train models efficiently, so distributed machine learning is gradually used in large-scale data training. Currently, commonly used distributed machine learning algorithms are based on data parallelism, and often use an overall synchronous parallel strategy when passing data, but using this strategy makes the overall training speed limited by the computation speed of the slower workers in the cluster. While the asynchronous parallel strategy maximizes the computational speed of the cluster, there is a delay in updating the parameters of the global model, which may lead to excessive computational errors or non-convergence of the model. In this paper, the author combines these two data delivery methods by grouping workers together and using synchronous parallelism for the workers in the group and asynchronous parallelism for the components for training. The experiment shows that the hybrid parallelism strategy can reduce the training time with guaranteed correctness.

Keywords: Deep learning, Data parallel, Distributed machine learning, Synchronous parallel, Asynchronous parallel.

1. Introduction

With the wide spread of smart devices in recent years, machine learning has become very popular. However, with the explosive growth of data volume, iterative computation of models requires multiple days of training on a single GPU, so using computing resources of clusters to speed up the training of deep networks becomes a currently more optimal way, i.e., distributed machine learning. However, the training speed of distributed does not increase linearly with the number of workers. For example, on MapReduce, a programming model for parallel computing with large-scale datasets, training a deep learning model with 10 GPUs is only 4.4 times faster than training with 1 GPU [1]. This is because distributed workers require frequent large-scale model information interactions with each other, making network communication one of the main factors affecting distributed efficiency. On the one hand, with the deployment of underlying hardware gas pedals (e.g., TPU, GPU, etc.), each worker generates more data per unit time, making the network pressure further aggravated. On the other hand, the current hardware computation speed is higher than the network communication speed between two machines, making the computation speed block the network communication. Microsoft research shows that in a 100 Gbps RDMA (Remote Direct Memory Access) network environment, even if there are only four GPU workers on a server, the training speed of each GPU is reduced by about 50% compared to one GPU [2][3]. In summary, it is of high practical value to improve the training efficiency of machine

DOI: 10.54254/2755-2721/6/20230835

learning algorithms in a distributed environment by optimizing and improving the communication strategy of distributed machine learning.

2. Related work

2.1. Overall synchronization parallel strategy

Currently, the most common training mode in machine learning is the iterative convergence training mode, the current mainstream distributed implementation process is to iterate gradient descent for each worker, submit the obtained local gradient to the parameter server, enter the synchronization barrier until all workers complete the iteration, and then release the synchronization barrier for the next iteration [4]. As shown in Figure 1, this parameter communication strategy that adds a synchronization barrier to ensure global consistency when updating parameters is called the overall synchronization parallel strategy.

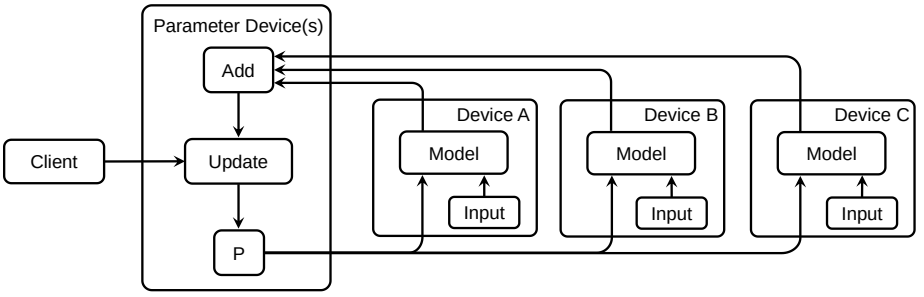

Figure 1. Synchronous Data Parallelism [5].

However, the overall synchronous parallelism of the model has a high peer overhead, which also makes the problem of distributed deployment of large machine learning models difficult to achieve. The results of the literature show that when 32 machines are in overall synchronous parallelism, the workers spend six times more time in communication than the time spent in the computational iteration process [6]. Also during the iterative process there will be some workers slower than others, which makes the overall speed receive impact and waste the computational power of the rest of the workers. This problem will appear more serious when the number of workers is larger.

In the literature [7], the training duration of 50 workers in the Amazon EC2 cluster was counted, and the measurement results are shown in Figure 2. A quarter of these workers have latency higher than 0.1 s. And among these slow workers, about one-third of them have about 5 times higher latency compared to the average worker.

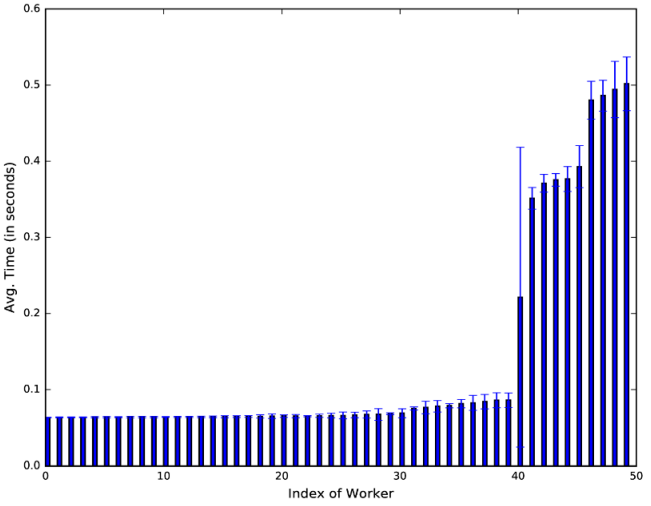

Figure 2. Diagram of the time required for communication between 50 workers in an Amazon EC2 cluster [7].

DOI: 10.54254/2755-2721/6/20230835

2.2. Asynchronous parallelism strategy

To solve the problems of synchronous parallelism strategy, A. Ahmed et al. [8][9] proposed the concept of asynchronous parallelism as shown in Fig. 3. The asynchronous update strategy means that each worker does not need to wait for other workers. After completing a computation, the result is sent directly to the parameter server and then the global update parameters are obtained for the next iteration. This approach results in the highest utilization of the computational power of each worker.

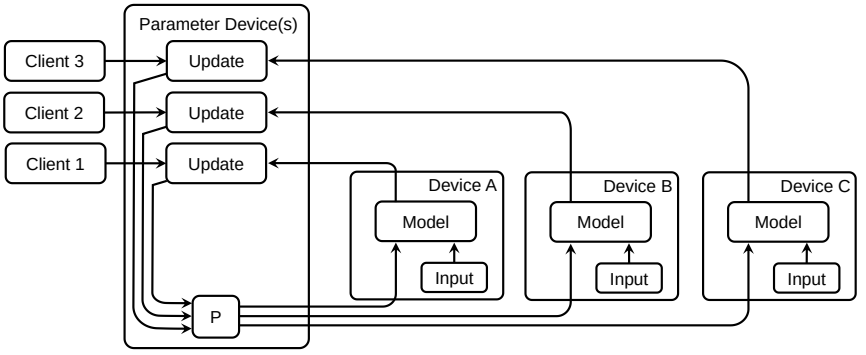

Figure 3. Asynchronous Data Parallelism [5].

Although the overall iteration speed problem is solved, the asynchronous parallel strategy suffers from the problem of gradient obsolescence [10]. The computationally faster workers precede the update of the global parameters, while the computation of some of the slower workers results in a relatively outdated gradient. This slows down the convergence of the model when the overall parameters are updated, and it is even difficult to reach convergence when the arithmetic power gap between two workers is too large [11-13].

Figure 4 shows the update process of the global parameters by four workers in an asynchronous parallel strategy, where W_n is the global parameter, n is the number of parameter updates, ΔW_n is the local gradient, and λ is the learning rate. As shown in the figure, Worker 1 submits the local gradient ΔW_{97} to the parameter server after completing the iteration with the global parameter W_{97}. Then, the parameter server updates the global parameter and sends the updated global parameter W_{101} to Worker 1 for the next iteration. Worker 2 finishes the iteration with global parameter W_{98}, sends the local gradient ΔW_{98} to the parameter server, and gets the updated global parameter W_{102} for the next iteration. Then, the global parameter ΔW_{100} is sent to the parameter server to get the updated global parameter W_{103} for the next iteration. Finally, Worker 4 completes the iteration with the global parameter W_{99} and updates the global parameter for the 104th time. It can be seen that due to the slow computation speed of worker 4, the local gradient calculated from the global parameters obtained from the 99th update is used for the 104th update of the global parameters, thus feeding the model with an outdated gradient. This problem is aggravated when the computational delay of the lagging workers is too large or when the number of workers in the cluster is large [14].

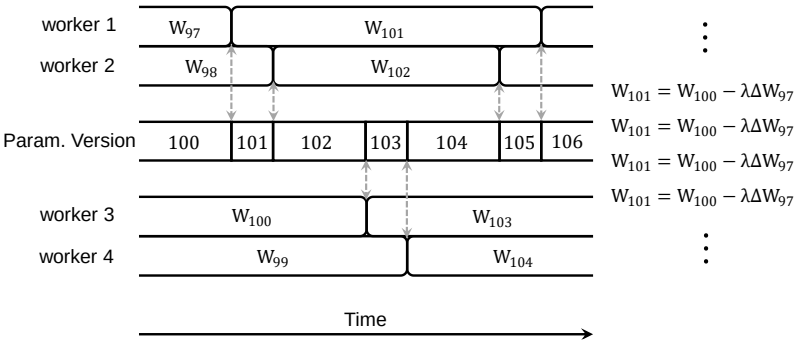

Figure 4. Asynchronous parallel strategy update schematic.

DOI: 10.54254/2755-2721/6/20230835

2.3. Hybrid parallelism

In order to solve the problems of synchronous parallel strategy and asynchronous parallel strategy, this paper proposes a new type of parallelism, namely the synchronous-asynchronous hybrid parallel strategy. By combining synchronous parallelism and asynchronous parallelism, a balance between convergence speed and convergence progress can be achieved.

The main idea of synchronous-asynchronous hybrid parallelism is to group different workers according to their computational speed, so that workers with a similar speed are divided into groups. The communication between the parameter server and the worker group is carried out through an overall synchronous parallel strategy, while within the group, an asynchronous parallel strategy is used between the workers. In this way, the synchronization overhead of the overall synchronous parallel policy can be effectively reduced due to the small speed difference of the workers within the group. Moreover, after grouping, the gradient results submitted by each group to the parameter server contain the computation results of multiple nodes, thus reducing the impact of the asynchronous parallel strategy on the convergence of the model.

The overall design of the synchronous-asynchronous hybrid model is shown in Figure 5, in which there are nine computational workers, which are divided into three groups according to the computational speed, and each computational worker corresponds to the gradient of one data block.

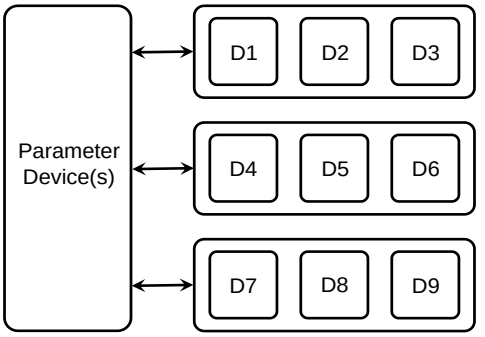

Figure 5. Synchronous and asynchronous mixed parallel strategy graph..

3. The proposed approach

There are many different grouping schemes for multiple workers, for example, a total of $C_9^3 \times C_6^3 \times C_3^3 = 1680$ results under 9 compute workers. If one grouping scheme is randomly selected, the computing power of each worker may not be fully exploited. Therefore, in this paper, the workers are grouped by the training speed of each worker.

In order to measure the training speed of the workers conveniently, a pre-training process is set up in this paper. The pre-training orders each worker to train asynchronously and in parallel, and sets a small global training step number. First, the local training step is initialized and then the training loop is entered. At each training step, the batch data is read from the dataset and the feed_dict is generated, then the train_step is called to execute the training once, and the training is stopped when the global training step reaches the preset value. By pulling the number of iterations of each worker in this process, the computational speed of each worker can be known, and then the grouping is completed.

The grouping algorithm of the overall workers is shown in the following.

Algorithm: workers grouping

input： dataset, number of workers N, number of groups n, number of global training steps i

output： Grouping Results

```
load dataset
k = N/n
K=[]
    g←gradienti// Asynchronous parallel iteration with workers uploading the gradient
    iternumber++
```

DOI: 10.54254/2755-2721/6/20230835

```
6 /* Iterations*/
7 for t=1,2.....N do
8     K←{"Wt":iternumber(t)}// Pull the number of iterations for each worker
9 End for
10 sorted K(value)
11 for t=1,2.....n do
12     for a=1,2.....k do
13         Print(K.key)
14         Ki.append(K.key)// Derive the results for each group
15     End for 16 Print(Kn)
17     Print("one group")
18 End for
```

The experiments are performed on the TensorFlow platform using an asynchronous parallel strategy and deployed on the parameter server side to achieve hybrid parallelism. First, pre-training of workers is required, and then the workers pass the computed gradient and index values to the parameter server in the iterative computation phase. Finally, the model is updated for the next iteration after the response from the parameter server. For the parameter server, after receiving the pre-training results, the grouping of workers is carried out first, and after the worker pushes the gradient value, the grouping information is found according to the index value and the gradient value is saved.

4. Experiment

The hybrid parallel strategy proposed in this paper is implemented on the TensorFlow distributed platform, and the parameters of the experimental hardware platform are shown in Table 1.

Table 1. Hardware platform parameters.

Hardware Configuration	Parameters
CPU	Intel i7-7700
GPU	GTX 1060
RAM	16G
Disk	512G

The parameter servers and workers used in this paper all work under the windows operating system, and their software versions are shown in Table 2.

Table 2. Software Version.

Software Name	Versions
Python	3.9
TensorFlow	2.6.0
cuda	10.0
cuDNN	7.4

In this paper, experiments are conducted using the cifar-100 dataset. A 9-layer convolutional network neural model is used for the experiments. The convolutional kernel size is set to 3*3, and 64, 64, 128, and 256 convolutional kernels exist in the input layer, respectively, and the RELU function is used for the convolutional layer activation function. After that, the feature map is input to a 2*2 pooling layer for maximum pooling. After the pooling process, the feature map is fed into the Flatten layer for one-dimensionalization. The final result is input to the fully connected layer, where the RELU function is used for the activation function of the fully connected layer. The dropout operation is used in the fully connected layer to prevent overfitting. The results are then fed to the fully connected layer with SoftMax activation function for classification. The loss function is calculated based on the output values and the data label values, and then the gradient values are calculated for the model update.

DOI: 10.54254/2755-2721/6/20230835

The experimental parameters of this paper are set as shown in Table 3.

Table 3. Neural network parameter setting.

Parameter Name	Numerical value
Batch-size	128
Dropout	0.5
Epochs	70
Learning Rate	0.01

5. Experimental results and discussion

In Fig. 6, it can be seen that the final accuracy values of all three parallel strategies decrease with the increase in the number of workers. This is because the experiments use a small batch gradient descent algorithm for model update training. It has a certain error that is amplified when the number of workers increases. In the case of 9 workers, the error of this algorithm has a low impact on the accuracy of all three communication strategies. From the experimental results, it can be seen that the overall synchronous parallel strategy is less affected and the asynchronous parallel strategy is more affected by this error. The reason for this is that as the number of workers increases, the gradient obsolescence problem becomes more severe, and therefore the accuracy of the final model of the asynchronous parallel strategy is more severely affected. Compared to the asynchronous-only strategy, the mixed synchronous-asynchronous parallel strategy is less affected by errors. This is because by grouping, the synchronous-asynchronous hybrid parallelism aggregates the computational results of multiple workers to update the global parameters in each group. This can reduce the impact of the possible gradient obsolescence problem in the asynchronous parallel strategy on the slow convergence of the model to a certain extent. After comprehensive consideration, a cluster of 9 workers are used in this chapter for further experiments.

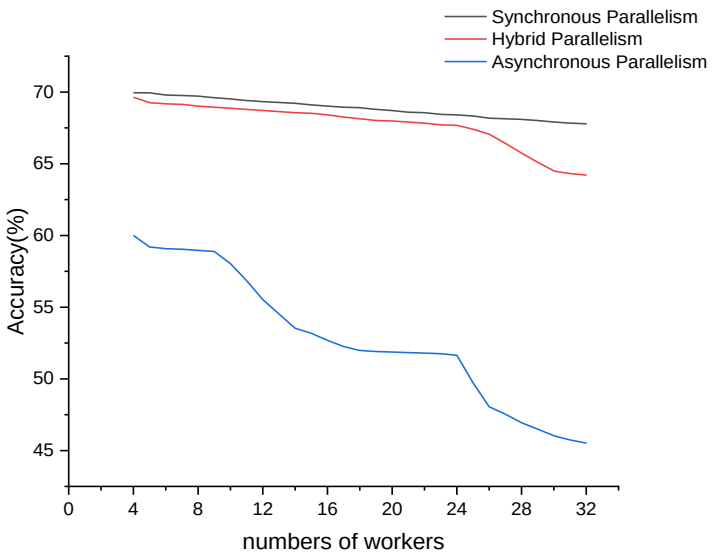

Figure 6. Accuracy comparison of parametric parallel strategies.

In the experiment, 9 workers and 1 parameter server are used, 2 workers are randomly selected for delay processing, and time.sleep(0.2) is used to set the delay value to 0.2s. The artificial delay is used to simulate the situation of slow workers with high load and slow computing speed in the distributed cluster, so as to simulate the cluster environment with uneven computing speed. By comparing the performance of the overall synchronous parallel strategy, asynchronous parallel strategy, and mixed synchronous-asynchronous parallel strategy, the model accuracy over time is shown in Figure 6.

DOI: 10.54254/2755-2721/6/20230835

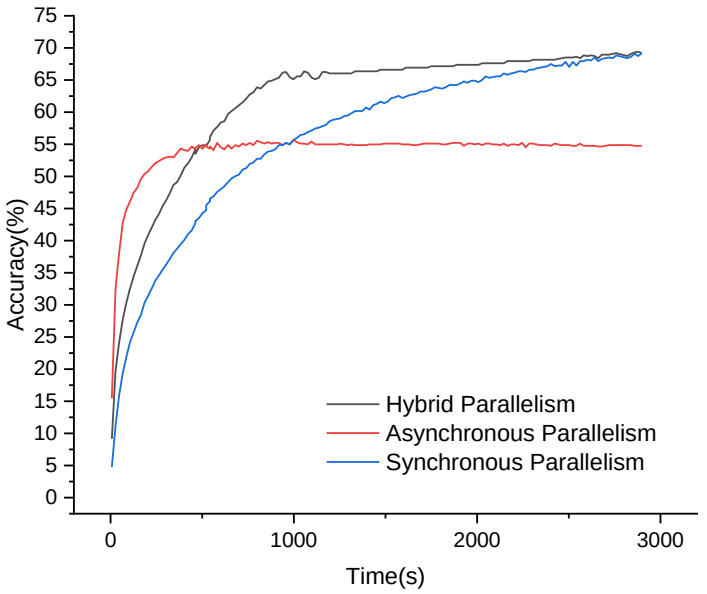

Figure 7. The accuracy of the three strategy models varies over time.

It can be seen that when there are slow workers in the distributed cluster, the synchronous-asynchronous hybrid parallel strategy outperforms the overall synchronous-parallel strategy in terms of model convergence speed. It also outperforms the asynchronous-parallel strategy in terms of final model accuracy while being not worse than the overall synchronous-parallel strategy. Therefore, the hybrid synchronous-asynchronous parallel strategy can perform better than the overall synchronous-parallel strategy and the asynchronous-parallel strategy in the case of uneven computation speed of cluster workers.

6. Conclusion

At this stage, distributed machine learning mainly uses the overall synchronous parallel strategy and asynchronous parallel strategy. However, the above two approaches may slow down the training speed when there are slow workers, and even the computation results converge incorrectly.

To solve the problems of these two strategies, this paper combines the synchronous parallel strategy and the asynchronous parallel strategy. Workers in the distributed cluster are grouped according to the computational speed so that workers with similar speeds are in the same group. And the global parameters are updated by the overall synchronous parallel strategy within the group and the asynchronous parallel strategy is used between the groups to communicate with the parameter server to update the global parameters. After grouping, each group sends multiple results of computation to the parameter server uniformly for updating, which effectively eliminates the effect of gradient obsolescence on the overall model convergence. The experimental results show that the hybrid parallel strategy achieves the best balance between convergence speed and computational accuracy with good results in the case of an uneven computational speed of workers in distributed clusters.

However, this paper only investigates the case of several workers, and the extension of this strategy to the case of more workers is yet to be experimented with and studied, and it is hoped that its effectiveness in the case of more workers can be verified in the future.

References

[1] P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, "SparkNet: Training Deep Networks in Spark." arXiv, Feb. 28, 2016. Accessed: Jul. 13, 2022. [Online]. Available: http://arxiv.org/abs/1511.06051.

[2] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Y. M. Research, "Multi-

DOI: 10.54254/2755-2721/6/20230835

tenant GPU Clusters for Deep Learning Workloads: Analysis and Implications," p. 14.

[3] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads," p. 15.

[4] M. Li et al., "Improving the Performance of Distributed MXNet with RDMA," Int J Parallel Prog, vol. 47, no. 3, pp. 467–480, Jun. 2019, doi: 10.1007/s10766-018-00623-w.

[5] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." arXiv, Mar. 16, 2016. Accessed: Jul. 24, 2022. [Online]. Available: http://arxiv.org/abs/1603.04467.

[6] L. G. Valiant, "A bridging model for parallel computation," Commun. ACM, vol. 33, no. 8, pp. 103–111, Aug. 1990, doi: 10.1145/79173.79181.

[7] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient Coding: Avoiding Stragglers in Distributed Learning," p. 9.

[8] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in Proceedings of the fifth ACM international conference on Web search and data mining - WSDM '12, Seattle, Washington, USA, 2012, p. 123. doi: 10.1145/2124295.2124312.

[9] A. Ahmed, Q. Ho, J. Eisenstein, E. Xing, A. J. Smola, and C. H. Teo, "Unified analysis of streaming news," in Proceedings of the 20th international conference on World wide web - WWW '11, Hyderabad, India, 2011, p. 267. doi: 10.1145/1963405.1963445.

[10] L. Cannelli, F. Facchinei, V. Kungurtsev, and G. Scutari, "Asynchronous parallel nonconvex large-scale optimization," in 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, Mar. 2017, pp. 4706 – 4710. doi: 10.1109/ICASSP.2017.7953049.

[11] J. Langford, A. Smola, and M. Zinkevich, "Slow Learners are Fast." arXiv, Nov. 03, 2009. Accessed: Jul. 19, 2022. [Online]. Available: http://arxiv.org/abs/0911.0491.

[12] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware Distributed Parameter Servers," in Proceedings of the 2017 ACM International Conference on Management of Data, Chicago Illinois USA, May 2017, pp. 463–478. doi: 10.1145/3035918.3035933.

[13] W. Fan et al., "Adaptive Asynchronous Parallelization of Graph Algorithms," ACM Trans. Database Syst., vol. 45, no. 2, pp. 1–45, Jun. 2020, doi: 10.1145/3397491.

[14] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System," p. 13.