A Synchronous Parallel Method with Parameters Communication Prediction for Distributed Machine Learning

Yanguo Zeng[1], Meiting Xue[5(✉)], Peiran Xu[1], Yukun Shi[5], Kaisheng Zeng[4,6], Jilin Zhang[1,2,3], and Lupeng Yue[1]

[1] School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
{jilin.zhang,lupengyue}@hdu.edu.cn
[2] Key Laboratory for Modeling and Simulation of Complex Systems, Ministry of Education, Hangzhou 310018, China
[3] Data Security Governance Zhejiang Engineering Research Center, Hangzhou 310018, China
[4] National University of Defense Technology, Changsha, China
zks@nudt.edu.cn
[5] School of Cyberspace, HangZhou Dianzi University, Hangzhou, China
{munuan,202243270074}@hdu.edu.cn
[6] Department of Computer Science and Technology, Tsinghua University, Beijing, China

Abstract. With the development of machine learning technology in various fields, such as medical care, smart manufacturing, etc., the data has exploded. It is a challenge to train a deep learning model for different application domains with large-scale data and limited resources of a single device. The distributed machine-learning technology, which uses a parameter server and multiple clients to train a model collaboratively, is an excellent method to solve this problem. However, it needs much communication between different devices with limited communication resources. The stale synchronous parallel method is a mainstream communication method to solve this problem, but it always leads to high synchronization delay and low computing efficiency as the inappropriate delay threshold value set by the user based on experience. This paper proposes a synchronous parallel method with parameters communication prediction for distributed machine learning. It predicts the optimal timing for synchronization, which can solve the problem of long synchronization waiting time caused by the inappropriate threshold settings in the stale synchronous parallel method. Moreover, it allows fast nodes to continue local training while performing global synchronization, which can improve the resource utilization of work nodes. Experimental results show that compared with the delayed synchronous parallel method, the training time and quality, and resource usage of our method are both significantly improved.

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 563, pp. 385–403, 2024.
https://doi.org/10.1007/978-3-031-54531-3_21

Keywords: Distributed Machine Learning · Synchronous Parallel · Communication Prediction · Collaborative Computing

1 Introduction

With the development of 5G, AI (artificial intelligence) technology, information technology, etc., intelligent collaborative computing is a good way to cope with the changing world. As AI technology is widely used in all walks of life, such as natural language processing [1], image classification [2], network traffic control [3], speech recognition [4], and other fields [5,6], the data has grown explosively, from the PB level to the EB level. It is a challenge to deal with such a huge amount of data with AI technology in a single device, as the limited resources of the single device. So, distributed machine learning technology, dealing with large-scale data with multiple devices, has become an inevitable trend and research hotspot.

The parameter server system is one popular distributed machine learning method to deal with large-scale data with AI technology. It trains a global model with the corporation of parameter server and worker nodes. Where the worker nodes use the subset of the data set to train the local models and update the local models to the parameter server, and the parameter server trains a global model by aggregating local models [7,8]. As deep learning models require multiple rounds of iterations to converge, they need to transport large-scale data, such as the parameters of the local model and global model, between servers and workers to complete the gradient descent method [9,10]. There is extensive communication between servers and workers. How to train a global model efficiently with low communication cost is an important problem for the parameter server system.

The bulk synchronous parallel method [11] is one of the mainstream parametric synchronization methods in the parameter server system. When the parameter server computes the global model by aggregating the local models, it needs to wait for all work nodes to upload the current version of the local models. The convergence time of model training depends on the slowest working node, which leads to low resource utilization and long training time [12]. In order to solve this problem, Dean et al. proposed the asynchronous parallel method [13], where each worker node is trained asynchronously and communicates with the parameter server to exchange models after completing a round of training without waiting for other worker nodes, which significantly utilizes the computation resources of worker nodes.

However, the uncontrollability of each node in the cluster in this method often leads to a significant difference in the number of iterations between fast and slow nodes, which finally makes the machine learning model converge poorly or even fail to converge. Combining the characteristics of the above two methods, Ho [14] et al. proposed the stale synchronous parallel method. It defines a delay parameter representing the maximum iteration difference between the working nodes to control the synchronization time of the work nodes. If the iteration

difference between the work nodes is less than the delay parameter, the work nodes will use the asynchronous communication method. Otherwise, the work nodes will use the synchronous communication method, waiting until all working nodes have completed the current round of training and performing a global synchronization. That is, the setting of the delay parameter affects the performance of this method. However, the value of the delay parameter is hard to set, as it relies on expert experience. The synchronization delay with unreasonable delay parameters still leads to low computing performance.

In summary, the existing parameter communication methods of distributed machine learning still have some shortcomings: (1) the bulk synchronous parallel method cannot fully utilize the computational performance; (2) The asynchronous parallel method over-exploits the fault tolerance of machine learning, which may eventually lead to the non-convergence of the model; (3) The stale synchronous parallel model, in which most delay thresholds are set based on expert experience, needs to be better adapted to the cluster environment and wastes computational resources.

In order to solve these problems, this paper proposes a synchronous parallel method with parameters communication prediction for distributed machine learning, and we call this method the Prediction Synchronous Parallel (PSP) method in this paper. This method controls the synchronous time of work nodes by analyzing the last iteration of cluster training to predict the future cluster performance and set the optimal synchronization timing to reduce the synchronization delay. Furthermore, to further improve the utilization of cluster computing resources, the fast node still keeps training if it enters the synchronization barrier, and when it receives the latest global model parameters, it aggregates the incremental local model training at the synchronization barrier and uses the global model parameters as the initial model for a new round of training. The experiment results show that our method can effectively improve the computation performance and convergence performance and also improve the resource utilization compared with the bulk synchronous parallel method and asynchronous parallel method.

The rest of this paper is organized as follows. Firstly, related work is reviewed in Sect. 2. Then, this paper describes the prediction synchronous parallel method for distributed machine learning in Sect. 3. Experiments follow in Sect. 4. Finally, the conclusion is in Sect. 5.

2 Related Work

Many distributed machine learning systems have been proposed to deal with large-scale data with AI technology, such as Spark and Hadoop, which are implemented based on MapReduce schema. For these systems, the server must wait for all work nodes to update the local models before proceeding to global model aggregation in each iteration, which causes a significant delay. In order to solve these problems, the parameter server system has been proposed, such as Multiverso, Ray [15], etc., which can support bulk synchronous parallel method, asynchronous parallel method, and stale synchronous parallel method.

The bulk synchronous parallel method is one of the dominant communication methods for distributed machine learning, such as the spark-5 [16] and MLIB16 [17]. It requires that the performance of worker nodes is similar. Otherwise, the end-to-end training time of the deep learning model will be dragged down by the worst-performing worker node. Haozhao Wang [18,19] et al. have proved that the performance loss due to synchronous communication is vast, even in clusters with similar computational performance.

To solve the synchronization delay problem in the bulk synchronous parallel method, Dean [13] et al. proposed an asynchronous parallel method that can fully utilize the computational resources of the working nodes. For the asynchronous parallel method, the local model is sent to the server as long as the work node calculates it, and the server updates the global model according to the local model parameters rather than waiting for all other work nodes. Therefore, the cluster's performance can be fully utilized as the slow nodes do not slow down the fast nodes. Furthermore, the asynchronous parallel method has been widely used in Tensorflow [20]. The advantage of this method is that it can be much faster than the bulk synchronous parallel method in the clusters with heterogeneous computing performance of work nodes. However, as the server doesn't need to wait for the slowest work nodes if the parameters of the local model uploaded by the slowest work node lag far behind the local models of other work nodes, the accuracy of the global model will be reduced, or even not convergent [5].

In order to solve the above problems, the stale synchronous parallel method [14] has been proposed. It combines the advantages of the bulk synchronous parallel method and the asynchronous parallel method by introducing a delay threshold to limit the iteration difference between the fastest and slowest work node. Similar to the asynchronous parallel method, the delayed synchronous parallel method allows the work nodes not to be globally synchronized until the iteration interval between the fasted work node and the slowest work node reaches an obsolescence threshold. There are some distributed machine learning systems supporting the stale synchronous parallel method, such as Petuum [21] and Bosen [22].

The stale synchronous parallel method alleviates the delay problem of the bulk synchronous parallel method and the low accuracy caused by the slowest work node in the asynchronous parallel method. However, it still needs to solve the problem of low calculated performance or low convergence performance due to an unreasonable delay threshold. Where the value of the delay threshold is set by users, and it requires users to have knowledge of machine learning, distributed computing, architecture, etc. So it is hard to set a reasonable value for the delay threshold. In this paper, we propose a prediction synchronous parallel method for distributed machine learning to improve the calculated performance and convergence performance of model training.

3 Method

In this section, we first analyze the synchronization lag problem in detail in the stale synchronous parallel model and then propose the synchronous prediction by

leveraging the continuity of cluster performance and parallelizing computation and synchronization. Finally, we prove that our proposed method is feasible from algorithm and theory.

3.1 Features of Synchronization

The main advantage of the stale synchronous parallel model is that it combines the characteristics of the global synchronous parallel method and the asynchronous parallel method, accelerating the computation of distributed machine learning models while ensuring convergence. However, there are still some problems in the previous section, not just the lag problem. This section discusses the design philosophy and algorithm implementation of the stale synchronous parallel model and points out the issues it faces when running on a real distributed machine learning cluster.

The utilization of computing resources in the stale synchronous parallel method is:

$$resource\_usage = \frac{\sum_{i=1}^{p} t_i}{p * \max(t_i)}, \tag{1}$$

where p represents the number of worker nodes and t_i represents the time spent on local computation in this round of synchronization. In the case of the global synchronous parallel model, $\max(t_i)$ represents the time spent on local model training by the worst-performing node. If the performance of all worker nodes is similar, higher utilization of computing resources can be achieved. In the case of the stale synchronous parallel model, $\max(t_i)$ represents the time taken by the last worker node to enter global synchronization. If a suitable stale threshold is set to make all worker nodes enter global synchronization at the same time, the stale synchronous parallel method can achieve higher utilization of computing resources. However, in practice, users often do not have a complete understanding of the performance of the cluster, so the stale threshold they set may not be able to achieve optimal utilization of computing resources. Additionally, the performance of each worker node in the cluster may change in real-time, so a fixed stale threshold may not be suitable for a real-world cluster environment.

We assume the ratio of the time required for one round of model training on these three worker nodes is $1 : 2 : 3$, and then there are three worker nodes performing machine learning model training in a parameter server system.

As shown in Fig. 1, worker node 1 and worker node 2 have to wait for worker node 3 to complete local model training, as it has not yet finished, resulting in the stale synchronous parallel model degrading into the synchronous parallel model. In Fig. 2, worker node 1 enters the synchronization barrier after completing 6 local model training iterations when the stale threshold is reached. At this point, worker node 2 and 3 have also completed their local model training, and different from Fig. 1, worker nodes 1 and 2 do not have to stop local training and wait for worker node 3. Under ideal conditions, there is no synchronization stale, thus fully utilizing the cluster computing performance.

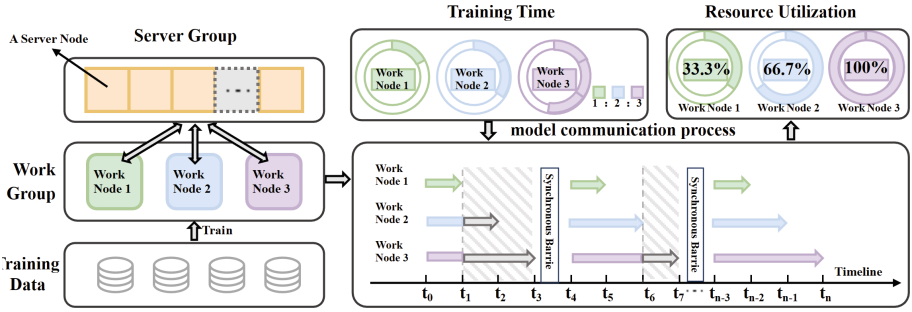


Fig. 1. The delayed synchronous parallel method communication process with a delay threshold of 1.

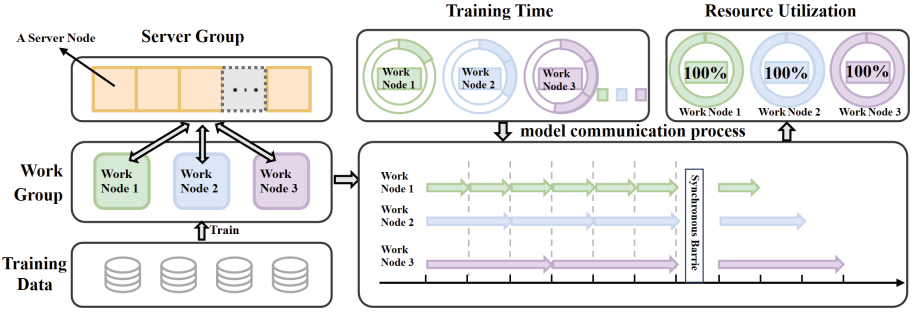


Fig. 2. The delayed synchronous parallel method communication process with a delay threshold of 5.

As shown in Figs. 1 and 2, the first problem is that the setting of the stale threshold in the stale synchronous parallel model will directly affect the efficiency of distributed machine learning model training. However, users often cannot set the appropriate stale threshold based on the performance of each worker node because they do not understand the cluster's performance.

The second problem is that external factors that may interfere with distributed machine learning model training were not taken into consideration. In real cluster environments, worker nodes typically perform tasks other than distributed machine learning model training. Therefore, the performance of each worker node is constantly changing, and a fixed stale threshold cannot adapt to the real distributed cluster environment.

As shown in Fig. 3, the initial time ratio required for one round of model iteration among worker node 1, worker node 2, and worker node 3 is 1:1:2. At this time, setting the stale threshold to 2 results in the minimum synchronization

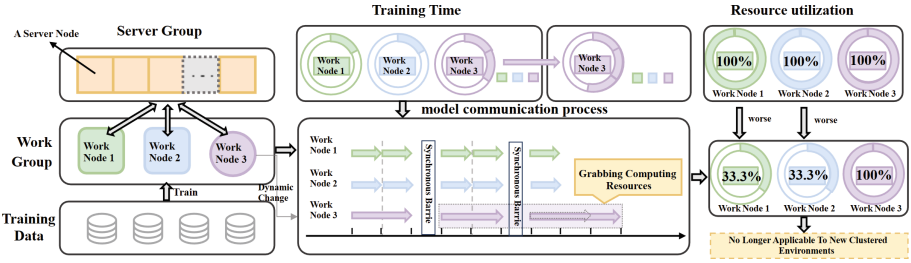


Fig. 3. The delayed synchronous parallel method under dynamic changes in the performance of the worker nodes.

delay. However, after two global synchronizations, other tasks on worker node 3 preempt computing resources, resulting in a performance decline. The originally designed stale threshold is no longer suitable for the new cluster environment.

In addition, when using the stale synchronization parallel model for distributed machine learning training in a real cluster environment, global synchronization of the worker nodes is unavoidable. When a fast node enters global synchronization, it will stop local training until every worker node in the cluster completes local model training before proceeding to the next round of training. Therefore, nodes that complete model training earlier are still held back by slower nodes, which affects the cluster's computational performance. To address this issue, this paper proposes a synchronous parallel method with parameters communication prediction, which uses the optimal synchronization time instead of a fixed stale threshold to solve the design problem of the stale synchronous parallel model. This method allows for simultaneous global synchronization and local computation, further improving the cluster's computing efficiency.

3.2 Synchronization Prediction

In the stale synchronous parallel method, setting the stale threshold to 0 transforms the stale synchronous parallel method into a synchronous parallel method, and setting the stale threshold to infinity transforms the stale synchronous parallel method into an asynchronous parallel method. The design of the stale threshold in the stale synchronous parallel model directly determines the global synchronization timing of nodes and directly determines the efficiency of the computing cluster. The following will use a typical scenario of distributed machine learning model training as an example to illustrate.

As shown in Fig. 1, if the time ratios required for one local model training for three worker nodes are 1:2:3, then setting the stale threshold to 1, in an ideal situation, worker node 1 has a computational resource utilization rate of only 33.3%, and worker node 2 has a utilization rate of only 66.7%. The overall computational resource utilization rate of the computing cluster is 66.7%. As shown in Fig. 2, if the stale threshold is set to 5, that is, worker node 1 enters the synchronization barrier after completing 6 local model training, then in an ideal situation, the

resource utilization rate of the computing cluster is 100%. Similarly, if the time ratio for one model iteration training for three worker nodes is 1:1:3, setting the stale threshold to 1 results in a computational resource utilization rate of 55.5% for the computing cluster, while if the global synchronization is performed after worker node 1 completes 3 local model iteration training, the ideal computational resource utilization rate of the cluster can reach 100%. Therefore, selecting different synchronization times based on different cluster performances directly affects the computational efficiency of the computing cluster. In this paper, the definition of the optimal synchronization time is when all worker nodes perform global synchronization at that moment, achieving the highest computational resource utilization rate.

However, in a real computing cluster environment, users often do not have knowledge about the performance of each working node in the cluster, making it difficult to design a stale threshold that can achieve the optimal utilization of computing resources. Moreover, the computing performance of each working node may change unpredictably, and a fixed stale threshold may no longer be suitable for the real-time changes in the performance of each working node.

Most distributed computing clusters have performance that varies in real-time but also has continuity, meaning that the computing performance of various working nodes in the computing cluster will not change significantly in a short period of time. To address the above issues, the synchronous parallel method with parameters communication prediction replaces the stale threshold with the optimal synchronization time. Since the cluster's computing performance has continuity, the computing performance of each working node in the next synchronization can be predicted using the performance of each working node in the previous round of synchronization. If the time ratio of model training for the three working nodes in the previous round of synchronization was 1:2:3, the time ratio of model training for the three working nodes in the next round of synchronization is also approximately 1:2:3.

According to above analysis, assuming there are P working nodes in a cluster, and the parameter server obtains the training time $\{t_1, t_2, t_3, \ldots, t_P\}$ of each working node from the previous iteration, then the optimal synchronization time can be represented as the least common multiple of $\{t_1, t_2, t_3, \ldots, t_P\}$, show in Eq. (2), i.e., working node i enters global synchronization after T/t_i iterations,

$$gbc(t_1, t_2, t_3, \ldots, t_P). \tag{2}$$

As shown in Fig. 4, during the first global synchronization, the parameter server obtains the local model training time ratios of the three worker nodes, which are 1:2:3. Then, the parameter server calculates the optimal time for worker node 1 to perform global synchronization is after completing six iterations.

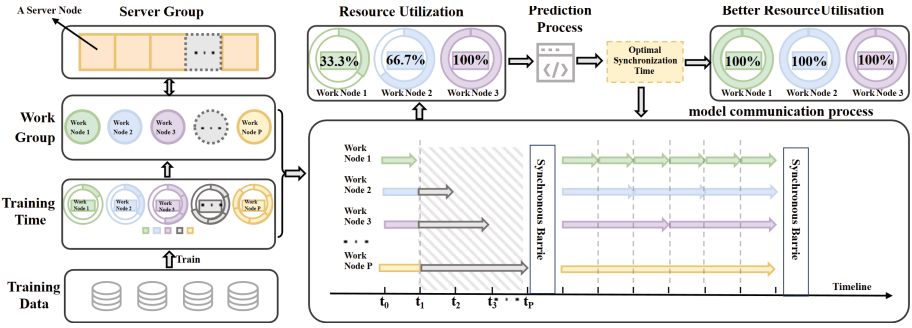


Fig. 4. The predictive synchronous parallel model communication process.

3.3 Implement

This section presents the implementation of the synchronous parallel method with parameters communication prediction. It is implemented under the parameter server system, where nodes are divided into working nodes and parameter servers. The algorithmic details of the working nodes and parameter servers are described in Algorithm 1 and Algorithm 2.

The specific execution process of Algorithm 1 is as follows:

(1) Load the sub-dataset on this worker node in line 1; (2) Line 2 - Line 9, the work node receives the iteration number of the current round of training from the parameter server and checks if the current iteration number is -1. If it is V1, end the model training. Otherwise, go to (3); (3) Receive the optimal synchronization time of this worker node. If the current iteration number has not reached the optimal synchronization time, continue to use the local dataset to compute the local model in line 10 - line 15. Otherwise, go to (4); (4) Calculate the average time consumption of one local model training on this worker node, and send the local model parameters and the average time consumption to the parameter server in line 16- line 19; (5) Use the local dataset to train the local model until receiving the new global model pushed by the parameter server in line 19 -line 26.

The specific execution process of Algorithm 2 is as follows:

(1) Initialize the iteration number and optimal synchronization time for the corresponding computing process in line 1 - line 2; (2) Receives the local model of the corresponding computing process through MPI communication and increment the iteration number of the corresponding computing process in line 3 - line 4; (3) If the optimal synchronization time is reached, enter global synchronization, wait for all work nodes to upload local models to the parameter server, calculate the optimal synchronization time based on the time required for each work node to perform one local model training, and wait for the parameter thread to aggregate the new global model parameters. Lastly, sends the new global model parameters and the optimal synchronization time to the corresponding computing process through MPI communication.

Algorithm 1: Prediction synchronous parallel method on worker node

Input : dataset subdata,iterations *iteration*,steps η,communication threshold τ,synchronization iterations i
output:

load *subdata*// load local dataset
while *ture* **do**
  /* Receiving the iteration number and the optimal synchronization opportunity iteration of the worker node from the parameter server. */
  $iternumber \leftarrow MPI\_RECV(ITER)$
  $i \leftarrow MPI\_RECV(i)$
  /* If the iteration number of this iteration is -1, the model training is finished. */
  if $iternumber == -1$ **then**
    break
  end
  $start\_time = now$
  $end\_iternumber = iternumber + i$
  /* The local model is used for model training before reaching the optimal synchronization opportunity. */
  while $iternumber < end\_iternumber$ **do**
    /* Calculating the parameter gradient of local model by back propagation method */
    $gradient \leftarrow ForwardBackward(parameters, subdata)$
    /* Updating local model parameters with gradient */
    $parameters \leftarrow Update(gradient, \eta, parameters)$
    $iternumber++$
  end
  $end\_time = now$
  /* Calculating the average time consumption of each round of model training. */
  $average\_time = (end\_time - start\_time)/i$
  $MPI\_ISED(parameters)$ // Send local model
  $MPI\_ISED(average\_time)$ // Send average time
  $next\_flag = false$
  /* Continue to train the local model while waiting synchronously. */
  while !$next\_flag$ **do**
    $next\_flag \leftarrow MPI\_RECV(next\_flag)$
    $gradient \leftarrow ForwardBackward(parameters, subdata)$
    $parameters \leftarrow Update(gradient, \eta, parameters)$
  end
  /* Receiving a new global model from the parameter server */
  $parameters \leftarrow MPI\_RECV(global\_parameters)$
end

Algorithm 2: Prediction synchronous parallel method on the server

Input : Calculation process number pid stale threshold s worker nodes p
output:
/* Initialize the iteration number and optimal synchronization opportunity of the corresponding calculation process. */
1 $iters\_pid \leftarrow 0$
2 $iters\_end\_pid \leftarrow iters\_pid + i\_pid$
/* Receiving a local model sent by a corresponding computing process */
3 $parameters\_pid \leftarrow MPI\_RECV(pid, parameters)$
4 $iters\_pid = isters\_pid + 1$
/* Judge whether the optimal synchronization opportunity is reached, and if so, perform global synchronization. */
5 **if** $iters\_pid = iters\_end\_pid$ **then**
  /* Global synchronization waiting for all working nodes to upload local models and average time consumption */
6 **for** $i = 1$ **to** p **do**
7 $wait(\&send)$
8 **end**
9 $next\_flag\_pid = true$
  /* Calculate that corresponding optimal synchronization opportunity */
10 $i\_pid \leftarrow Compute(time\_pid)$
  /* Sending the global synchronization end signal and the optimal synchronization opportunity to the corresponding computing thread. */
11 $MPI\_ISEND(pid, nex\_flag\_pid)$
12 $MPI\_ISEND(pid, i\_pid)$
13 **end**

3.4 Theoretical Analysis

To ensure the correctness of using the implementation of the synchronous parallel method with parameters communication prediction for distributed machine learning, the following will theoretically prove that this method has the same correctness as the stale synchronous parallel model. We adopt the convergence of the method as the criterion for judging the correctness of distributed machine learning.

For the convenience of this chapter's proof, the following assumptions are made:

Assumption 1. The objective function F is continuously differentiable, and the gradient of the objective function is Lipschitz continuous [23], with a Lipschitz constant $L > 0$ as given in Eq. (3):

$$\|\nabla F(\omega) - \nabla F(\widetilde{\omega})\|_2 \leq L\|\omega - \widetilde{\omega}\|_2, \tag{3}$$

where ω represents the model parameters.

Assumption 2. The loss function has an upper bound, which means:

$$\|\nabla F(\omega)\| \leq K, \tag{4}$$

where K is a constant.

Assumption 3. The gradient in stochastic gradient descent is bounded, that is:

$$D(\omega\|\omega') = \frac{1}{2}\|\omega - \omega'\|^2 \leq F^2. \tag{5}$$

Most machine learning algorithms follow an iterative training pattern that involves an optimization process. The optimization function is represented by Eq. (6):

$$L = f(\omega) = f(I_{i=1}^{N}\{x_i, y_i\}, \omega). \tag{6}$$

Here, f is the loss function, x_i, y_i is a sample in the dataset, ω is the machine learning model parameter, and y_i is the expected output of the input data x_i. The loss function represents the difference between the actual output x_i and the expected output of the input. The machine learning program iterates using the dataset to minimize the loss function. We will now demonstrate that the efficient synchronous parallel model can ensure the final convergence of the model, using an optimization function shown in Eq. (7):

$$L = f(\omega) = \sum_{t=1}^{T} f_t(\omega_i). \tag{7}$$

where f_t is the loss function at the $t - th$ iteration and f is a convex function. The goal is to find the optimal solution ω^* of the machine learning model to minimize the loss function.

We introduce a regret value to represent the deviation between the currently trained model and the optimal solution, and its mathematical expression is shown in Eq. (8).

$$R[\omega] = \frac{1}{T}\sum_{t=1}^{T} f_t(\omega) - f(\omega^*). \tag{8}$$

If T tends to infinity, $R[\omega]$ tends to 0, it can be proven that the efficient synchronous parallel model can ultimately make the machine learning model converge. Based on Eq. (8), we can obtain Eq. (9):

$$R[\omega] = \frac{1}{T}\sum_{t=1}^{T} f_t(\omega) - f(\omega^*) \quad \leq \frac{1}{T}\sum_{t=1}^{T}(\nabla f_t(\widetilde{\omega}), \widetilde{\omega} - \omega^*). \tag{9}$$

According to Ho [14], we can get the Eq. (10):

$$\frac{1}{T}\sum_{t=1}^{T}(\nabla f_t(\widetilde{\omega_t})\widetilde{\omega_t} - \omega^*) \leq \sigma K^2 \frac{1}{\sqrt{T}} + \frac{F^2}{\sigma}\frac{1}{\sqrt{T}} + \frac{1}{T}(\omega_t - \widetilde{\omega_t}, \widetilde{g_t}), \tag{10}$$

where $\sigma = \frac{F}{K\sqrt{2(\tau s+1)P}}$. When T tends to positive infinity, both $\sigma K^2 \frac{1}{\sqrt{T}}$ and $\frac{F^2}{\sigma}\frac{1}{\sqrt{T}}$ tend to 0. Therefore, if we want to prove that the regret value $R[\omega]$ tends to 0 when T tends to positive infinity, we only need to prove that $\frac{1}{T}(\omega_t - \widetilde{\omega_t}, \widetilde{g_t})$ tends to 0 when T tends to positive infinity.

$$\begin{aligned}\frac{1}{T}(\omega_t - \widetilde{\omega_t}, \widetilde{g_t}) &= \frac{1}{T}\sum_{t=1}^{T}([\sum_{i\subseteq A_t} u_i - \sum_{i\subseteq B_t} u_i], \widetilde{g_t}) \\ &\leq \frac{1}{T}\sum_{t=1}^{T}[\sum_{i\subseteq A_t}\eta_i(\widetilde{g_i}, \widetilde{g_t}) - \sum_{i\subseteq B_t}\eta_i(\widetilde{g_i}, \widetilde{g_t})] \\ &\leq \frac{1}{T}\sum_{t=1}^{T}[|A_t| + |B_t|]\eta_t K^2\end{aligned} \tag{11}$$

Let the iteration difference between the fast node and the slow node be s, from which we can get $|A_t| + |B_t| \leq s$. Let the learning rate η_t of the t iteration be $\frac{\sigma}{\sqrt{T}}$. Therefore, Eq. (12) can be obtained based on Eq. (11):

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}[|A_t| + |B_t|]\eta_t K^2 &\leq \frac{1}{T}\sum_{t=1}^{T} s\eta_t K^2 \\ &\leq \frac{1}{T} sK^2\sqrt{T} \\ &= sK^2\frac{1}{\sqrt{T}}\end{aligned} \tag{12}$$

Therefore, we can get the Eq. (13):

$$R[\omega] \leq \frac{sK^2 + \sigma K^2 + \frac{K^2}{\sigma}}{\sqrt{T}} \tag{13}$$

where, as $sK^2 + \sigma K^2 + \frac{K^2}{\sigma}$ is a fixed value, when T tends to infinity, $\sqrt{T}$ tends to infinity and $\frac{sK^2+\sigma K^2+\frac{K^2}{\sigma}}{\sqrt{T}}$ tends to infinitesimal, that is, the regret value $R[\omega]$ tends to infinitesimal.

The Eq. (13) shows that when the number of training iterations of the distributed machine learning model tends to infinity, the gap between the model obtained by training calculation and the optimal model tends to zero. Overall,

we prove that the efficient synchronous parallel model can ensure the final convergence of the model; that is, it proves the correctness of using this model for distributed machine learning training.

4 Experiments

4.1 Experimental Setup

Dataset. We use two widely used and publicly available datasets to validate the effectiveness of our approach: CIFAR-10 [24] and MNIST [25]. **1)** The CIFAR-10 consists of 60,000 samples, each of which is a 32×32 pixel colour image divided into three channels: R, G, and B. Among these 60,000 samples, five sets are reserved for training and one for testing. It is used for supervised learning. Each sample contains a label indicating the category of the object, with ten categories of images, including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. **2)** The MNIST dataset is specifically designed for handwritten digit recognition. It consists of a training set with 60,000 samples and a separate test set containing 10,000 samples. Each sample in the MNIST dataset is a grayscale image of a handwritten digit, with a size of 28×28 pixels. The dataset is widely used as a benchmark for various machine learning algorithms and models.

Baselines. The baselines used in this study are the parameter communication methods currently mainstream in distributed machine learning, including the bulk synchronous parallel method [11], the asynchronous parallel method [13], and the stale synchronous parallel method [14]. The bulk synchronous parallel method, in which the server needs to wait for all work nodes to upload the current local models, performs global synchronization after each local model training iteration. For the asynchronous parallel method, each worker node is trained asynchronously and communicates with the parameter server. The stale synchronous parallel method only performs global synchronization when the iteration difference between fast and slow nodes reaches a delay threshold.

Metrics. We use three metrics to evaluate the models over the above-mentioned benchmarks: training time, training quality, and resource usage. The training time will be measured by the time spent per epoch, the training quality will be measured by the decrease of the loss function over time, and the computational resource utilization will be defined as the time spent on computation as a percentage of the total execution time, as shown in Eq. (14),

$$resource_{usage} = \frac{\sum_{i \subset workers} compute_{time_i}}{\sum_{i \subset workers} totall_{time_i}}. \tag{14}$$

Environments. The hardware environment used in this paper is a cluster of four servers, with one server as the parameter server and the other three servers as worker nodes. It has the following key components: 1) Parameter server: the CPU is an Intel Xeon processor (E7-4807) with 1.87 GHz, containing 12 physical cores in total; 2) Hyper-Threading: it has 24 logical cores, 32 GB of memory, and 256 GB of hard disk capacity; 3) Worker node: the CPU is AMD processor (Processor 6136) with 2.4 GHz. The nodes in the cluster are connected to each other via Gigabit Ethernet. In addition, a scenario with 3 and 6 clients is developed by using threads to simulate clients.

4.2 Experimental Results

Training Time

Tables 1 and 2 show the time required for different methods to achieve a given accuracy. We report the Prediction Synchronous Parallel method as the distributed machine learning communication method, the training time of the VGG-11 model on the CIFAR-10 dataset is reduced by up to 19.9%, and up to 55.8% reduction is achieved when training logistic regression on the MNIST dataset. This is because the Prediction Synchronous Method can predict the optimal synchronization time according to the history information, which can reduce the synchronization relay. So that, the method proposed in this paper is better suited for natural cluster environments where the performance varies in real-time.

Table 1. Training Time for VGG-11 model on CIFAR-10 using different methods with 75% accuracy

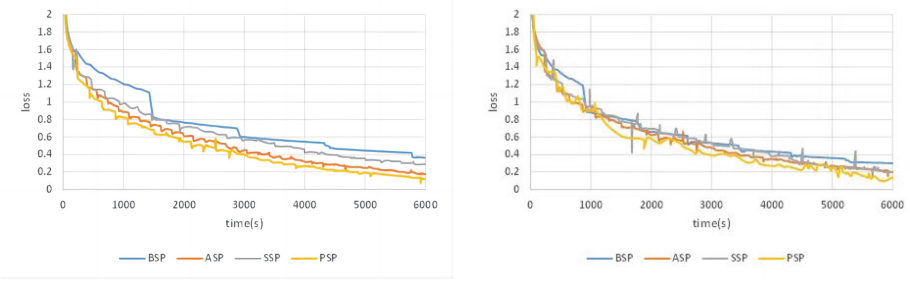
	Bulk synchronous parallel	Asynchronous parallel	Stale Synchronous Parallel	Predictive synchronous parallel
3 clients	4511 s	4230 s	4133 s	3756 s
6 clients	4225 s	3846 s	4033 s	3419 s

Table 2. Training Time for logistic regression on MNIST using different methods with 92% accuracy

	Bulk synchronous parallel	Asynchronous parallel	Stale Synchronous Parallel	Predictive synchronous parallel
3 clients	221 s	258	353 s	204 s
6 clients	208 s	231 s	445 s	197 s

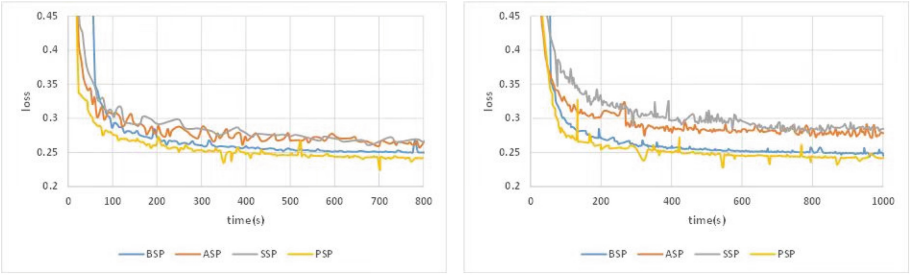
Training Quality

According to Figs. 5 and 6, the method proposed in this paper has better convergence performance than other baselines. Although the asynchronous parallel method has a faster speedup, it excessively utilizes the limited fault tolerance of machine learning, resulting in inferior convergence performance compared to our method in this paper. Moreover, while ensuring model convergence, the Predictive Synchronous Parallel method proposed in this paper has better training speed than the bulk and stale synchronous parallel methods. Our method with the optimal synchronization time can reduce the synchronization delay in the bulk synchronous parallel method, solve the parameter stale problem in the asynchronous parallel method, and the unreasonable delay parameters in the stale synchronous parallel method. Therefore, regarding convergence performance, our method is superior to the other two methods, the final model converges, and the loss function of the Predicted Synchronous Parallel method is reduced by about 15% compared to the mainstream parametric communication method.



(a) (b)

Fig. 5. Convergence performance of different methods trained with VGG-11 on CIFAR-10



(a) (b)

Fig. 6. Convergence performance of different methods trained with logistic regression on MNIST

Resource Usage

In Figs. 7 and 8, the utilization of network bandwidth is used as a metric to measure the utilization of computational resources. This is because computational tasks require data transmission and communication between different nodes. Our method exhibits a computational resource utilization rate second only to asynchronous parallel methods. This is because our method has lower synchronization latency compared to batch synchronous parallel methods and outdated synchronous parallel methods. In contrast, asynchronous parallel methods do not require global synchronization, resulting in higher computational resource utilization. Compared to mainstream parameter communication methods, our method improves computational resource utilization by 31%. As the number of worker nodes increases, the computational resource utilization rates of batch synchronous parallel methods, asynchronous parallel methods, sluggish synchronous parallel methods, and our method all exhibit varying degrees of decline. This is due to the increased number of worker nodes communicating with the parameter server in the cluster network, leading to network congestion. More resources are allocated to model transmission between worker nodes and the parameter server, ultimately resulting in a decrease in cluster computational resource utilization.

Fig. 7. Computational resource utilization of different models trained with CIFAR-10 using VGG-11 under three and six threads

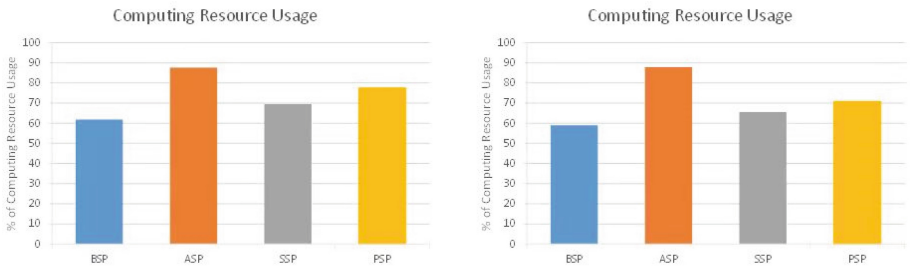


Fig. 8. Computational resource utilization of different models trained with MNIST

5 Summary

In this paper, we propose a synchronous parallel method with parameters communication prediction for distributed machine learning to solve the degraded resource utilization problem caused by the constant delay thresholds in the stale synchronous parallel method. It predicts the next round of training based on the previous round of training of each node in the cluster to select the optimal synchronization timing, thus reducing the synchronization waiting time. Moreover, the fast nodes in this method continue to train locally with local datasets instead of stopping training while waiting synchronously, which further increases the computational resource utilization of the cluster. The experiments show that the method proposed in this paper has good improvements in training time, training quality, and resource usage.

Acknowledgment. I would like to express my gratitude to all those who helped me during the writing of this work. This work is supported by the Key Technology Research and Development Program of China under Grant No. 2022YFB2901200.

References

1. Ahmad, F., et al.: A deep learning architecture for psychometric natural language processing. ACM Trans. Inf. Syst. (TOIS) **38**(1), 1–29 (2020)
2. Dabare, R., Wong, K.W., Shiratuddin, M.F., Koutsakis, P.: A fuzzy data augmentation technique to improve regularisation. Int. J. Intell. Syst. **37**(8), 4561–4585 (2022)
3. Liu, W.-X., Jinjie, L., Cai, J., Zhu, Y., Ling, S., Chen, Q.: DRL-PLink: deep reinforcement learning with private link approach for mix-flow scheduling in software-defined data-center networks. IEEE Trans. Netw. Serv. Manage. **19**(2), 1049–1064 (2021)
4. Kriman, S., et al.: QuartzNet: deep automatic speech recognition with 1d time-channel separable convolutions. In: ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6124–6128. IEEE (2020)
5. Wang, Y., Wang, K., Huang, H., Miyazaki, T., Guo, S.: Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. IEEE Trans. Industr. Inf. **15**(2), 976–986 (2018)
6. Xu, C., Wang, K., Sun, Y., Guo, S., Zomaya, A.Y.: Redundancy avoidance for big data in data centers: a conventional neural network approach. IEEE Trans. Netw. Sci. Eng. **7**(1), 104–114 (2018)
7. Xu, C., Wang, K., Li, P., Xia, R., Guo, S., Guo, M.: Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning. IEEE Trans. Netw. Sci. Eng. **7**(1), 205–215 (2018)
8. Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., Guo, C.: A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2020), pp. 463–479 (2020)
9. Liang, X., et al.: Accelerating local SGD for non-IID data using variance reduction. Front. Comp. Sci. **17**(2), 172311 (2023)

10. Lin, G., et al.: Understanding adaptive gradient clipping in DP-SGD, empirically. Int. J. Intell. Syst. **37**(11), 9674–9700 (2022)
11. Gerbessiotis, A.V., Valiant, L.G.: Direct bulk-synchronous parallel algorithms. J. Parallel Distrib. Comput. **22**(2), 251–267 (1994)
12. Wang, Z., et al.: FSP: towards flexible synchronous parallel frameworks for distributed machine learning. IEEE Trans. Parallel Distrib. Syst. **34**(2), 687–703 (2022)
13. Dean, J., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems, vol. 25 (2012)
14. Ho, Q., et al.: More effective distributed ml via a stale synchronous parallel parameter server. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
15. Moritz, P., et al.: Ray: a distributed framework for emerging AI applications. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018), pp. 561–577 (2018)
16. Zaharia, M., et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2012), pp. 15–28 (2012)
17. Spark MLlib (2020). http://spark.apache.org/mllib/. Accessed Apr 2020
18. Wang, H., Guo, S., Li, R.: OSP: overlapping computation and communication in parameter server for fast machine learning. In: Proceedings of the 48th International Conference on Parallel Processing, pp. 1–10 (2019)
19. Wang, H., Zhihao, Q., Guo, S., Wang, N., Li, R., Zhuang, W.: LOSP: overlap synchronization parallel with local compensation for fast distributed training. IEEE J. Sel. Areas Commun. **39**(8), 2541–2557 (2021)
20. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2016), pp. 265–283 (2016)
21. Xing, E.P., et al.: Petuum: a new platform for distributed machine learning on big data. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1335–1344 (2015)
22. Wei, J., et al.: Managed communication and consistency for fast data-parallel iterative analytics. In: Proceedings of the Sixth ACM Symposium on Cloud Computing, pp. 381–394 (2015)
23. Khalil, H.: Nonlinear Systems, 3rd edn. Pearson, Upper Saddle River (2001)
24. Abouelnaga, Y., Ali, O.S., Rady, H., Moustafa, M.: CIFAR-10: KNN-based ensemble of classifiers. In: 2016 International Conference on Computational Science and Computational Intelligence (CSCI), pp. 1192–1195. IEEE (2016)
25. MNIST (2020). http://yann.lecun.com/exdb/mnist. Accessed June 2020