IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON NOVEL LEARNING APPLICATIONS AND SERVICES FOR SMART CAMPUS

Received February 5, 2018, accepted March 19, 2018, date of publication March 29, 2018, date of current version April 23, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2820899

An Adaptive Synchronous Parallel Strategy for Distributed Machine Learning

JILIN ZHANG[1,2,3,4,5], HANGDI TU[1,2], YONGJIAN REN[1,2], JIAN WAN[1,2,4,5], LI ZHOU[1,2], MINGWEI LI[1,2] AND JUE WANG[6]

[1]School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China
[2]Key Laboratory of Complex Systems Modeling and Simulation, Ministry of Education, Hangzhou 310018, China
[3]College of Electrical Engineering, Zhejiang University, Hangzhou 310058, China
[4]School of Information and Electronic engineering, Zhejiang University of Science and Technology, Hangzhou 310023, China
[5]Zhejiang Provincial Engineering Center on Media Data Cloud Processing and Analysis, Hangzhou 310018, China
[6]Supercomputing Center of Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China

Corresponding author: Yongjian Ren (yongjian.ren@hdu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672200 and Grant 61572163, in part by the National Key Technology Research and Development Program of China under Grant 2015BAH17F02, in part by the Zhejiang Natural Science Funds under Grant LY16F020018 and Grant LY17F020029, in part by the National High Technology Research and Development Program of China under Grant 2015AA01A303, in part by the Hangzhou Dianzi University Construction Project of Graduate Enterprise Innovation Practice Base under Grant SJJD2014005, in part by the Zhejiang Province Education Department Scientific Research Fund Project under Grant Y201016492, in part by the CERNET Innovation Project under Grant NGII20160212, in part by the Ministry of Housing and Urban Construction Science and Technology Project of China under Grant 2017-R2-018, in part by the Scientific Research Foundation of BUCEA under Grant 00331616023, in part by the Key Research and Development Project of Zhejiang Province under Grant 2017C03024, in part by the Opening Foundation of the Institute of Computing Technology, Chinese Academy of Sciences, under Grant CARCH201712, and in part by the Postgraduate Cultivation Project under Grant hxkc2016004.

ABSTRACT In recent years, distributed systems have mainly been used to train machine learning (ML) models. However, as a result of the different performances among computational nodes in a distributed cluster and delays in network transmission, the accuracies and convergence rates of ML models are relatively low. Therefore, it is necessary to design a reasonable strategy that provides dynamic communication optimization to improve the utilization of the cluster, accelerate the training times, and strengthen the accuracy of the training model. In this paper, we propose the adaptive synchronous parallel strategy for distributed ML. Through the performance monitoring model, the synchronization strategy of each computational node with the parameter server is adjusted adaptively by considering the full performance of each node, thereby ensuring higher accuracy. Furthermore, our strategy prevents the ML model from being affected by irrelevant tasks in the same cluster. Experiments show that our strategy fully improves clustering performance, and it ensures the accuracy and convergence speed of the model, increases the model training speed, and has good expansibility.

INDEX TERMS Distributed machine learning, adaptive synchronous parallel, communication strategy, parameter server.

I. INTRODUCTION

In the age of big data, distributed machine learning (ML) has become a hot research field due to its ability to adapt to the complexity of big data, obtain higher prediction accuracy and support more intelligent tasks [1].

Distributed optimization of ML is becoming a prerequisite for solving large-scale ML problems [2]–[4]. Due to the increase in data volume and the complexity of the ML model, especially the increase in the number of parameters, single machines have been unable to fully and quickly train the ML model. However, it is not easy to realize an efficient distributed algorithm [5], [6]. The system design needs to consider the large amounts of computation and data traffic. The actual size of the training data will be between 1TB and 1PB, which can be used to create a powerful and complex model. These models are usually shared globally by all computational nodes. A computational node computes local updates on its data subset in each iteration and then submits the local update to the parameter server, which updates the global model parameters; the parameter server distributes the new global model parameters to each computational node. There are three challenges to this sharing approach: (1) Access parameters require a great deal of network bandwidth [7]. (2) When the synchronization costs and machine

delays are high, the resulting hurdles can impair performance. (3) In large-scale ML, fault tolerance [8] must be considered. Model training tasks are usually performed in the cluster. However, computational nodes may not be reliable, and the job may be pre-empted.

In general, distributed ML uses the Bulk Synchronous Parallel (BSP) strategy [9], [10] to parallelize data. Under this strategy, the computational nodes do not start the next iteration after committing their updates to the parameter server until all computational nodes commit their updates and receive the new global model parameters from the parameter server. Due to the different performances among the computational nodes, BSP has a load imbalance problem in computing.

To solve this problem, Dean *et al.* [11] proposed an asynchronous iteration strategy [12] for distributed ML, which allows computational nodes to use local model parameters for the next iteration. This strategy improves fault tolerance without a limit; however, it causes the model to become trapped in a local optimum, so it cannot converge to the globally optimal solution and has no accuracy guarantee. Ho *et al.* [13] proposed the Stale Synchronous Parallel (SSP) strategy [14], [15]. This strategy allows computational nodes to use stale global model parameters to train the model, thereby reducing the cost of synchronizing parameters with the parameter server. However, it has the limitation of using stale global model parameters, so it cannot provide convergence guarantees. This strategy can improve training speed, but due to its lack of local updates, it will accumulate parallel faults, which will reduce the convergence speed. To solve the problems in the BSP and SSP schemes, this paper improves SSP and proposes the Adaptive Synchronous Parallel (ASP) strategy. The strategy dynamically adjusts the communication mechanism between computational nodes and the parameter server according to the performance of each computational node, which significantly reduces the impact of different performances among the computational nodes when training an ML model. The training speed is greatly accelerated, thereby ensuring better accuracy and higher convergence speed. This paper introduces the relevant scenario of distributed ML parameter communication in the next section. The design of the ASP strategy is described in the third section. In the fourth section, we test the strategy in distributed ML experiments, and analyse and verify the corresponding theory and function. The last section presents a summary and discussion of this paper.

II. RELATED WORK

This paper proposes ASP, which is related to parameter servers, BSP and SSP. This section will introduce the research in these areas.

A. PARAMETER SERVER FRAMEWORK

The use of the parameter server framework is expanding in both academia and industry. Relevant systems have been implemented on Tencent [16], [17], Baidu [18], Alibaba [19], [20], Google [11], [21], and Yahoo [5]. Open-source codes are available, such as YahooLDA [5], MXNet [22], and Petuum [23].

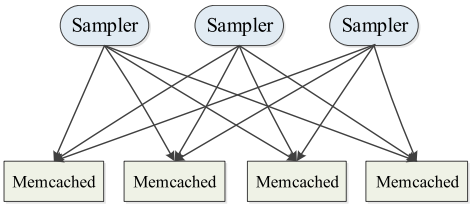


FIGURE 1. First generation of the parameter server.

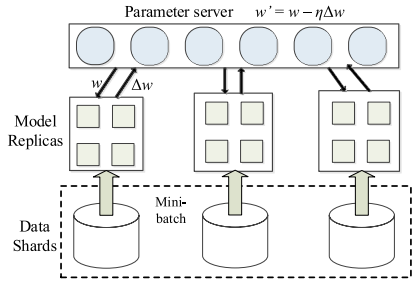


FIGURE 2. Second generation of the parameter server.

Alex Smola put forward a parallel topic model [24], [25] architecture in 2010, which we call the first generation of the parameter server. Essentially, it is a distributed shared storage framework, in which each sampler can share and access global model parameters through a key-value interface. The model uses memcached distributed storage for parameter storage (as shown in Figure 1). Each sampler only retains some of the parameters for necessary calculations and the global model parameters can be synchronized by each sampler. However, this parameter server framework is only a simple concept. It neither optimizes the communication cost nor is it designed specifically for distributed ML.

Many related frameworks have been implemented in industry. Dean and Ghemawat [26] introduced the second generation of the parameter server, and it can be found in a deep learning system, namely DistBelief [11]. DistBelief is mainly used for the training of super-large-scale learning models in Google Brain. Xing *et al.* [23] developed it as a general ML platform, which uses the second generation of the parameter server, and the ML model is stored distributive in the computational nodes, as shown in Figure 2. In this framework, the parameter server is used to communicate with all computational nodes and transmit all model parameters to them, and the computational nodes no longer communicate with one another. The second generation of the parameter server makes possible the use of distributed algorithms in ML, such as the distributed stochastic gradient descent (SGD) [27], [28]. However, it lacks adequate consideration of the performances of different computational nodes, so the utilization rate of individual computational nodes in the distributed system is relatively low.

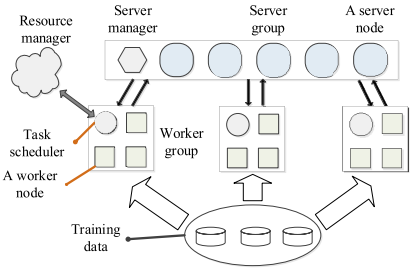


FIGURE 3. Third generation of the parameter server.

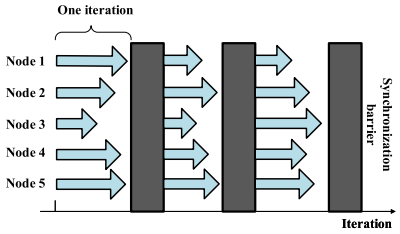


FIGURE 4. BSP sets a synchronization barrier to ensure the consistency of the global parameters.

Li *et al.* [29]–[31] described the third generation of the application-specific parameter server framework. This framework creates a more general and flexible design, and contains a flexible group of parameter servers and multifunctional groups of computational nodes (as shown in Figure 3). In this framework, each parameter server maintains a partition of global model parameters and the parameter servers communicate with one another to maintain reliability and consistency. A server manager maintains the entire group of parameter servers. The multifunctional groups of computational nodes can run many different ML applications. As with parameter servers of the previous generation, all parameters are transmitted by the parameter server. Each group of computational nodes has a task scheduler, which is set up for assigning tasks to computational nodes and monitoring their progress. If a computational node gives no response or a new computational node has been added, the task scheduler will redistribute the remaining tasks without having to restart the model training. This framework overcomes the limitations of previous generations and increases flexibility by designing a scheduler. However, this single strategy can only handle problems by increasing or reducing the number of computational nodes and does not consider countermeasures if external disturbances are encountered during the training process.

B. BULK SYNCHRONOUS PARALLEL

In an actual cluster, it is very possible for the performances of different computational nodes to be different. For the most common iteration-convergence algorithm in ML, the main procedure of distributed ML implementation is as follows: each computational node will enter the synchronization barrier after committing the parameters to the parameter server and wait for other computational nodes to commit their local model parameters. From then on, every computational node will obtain the latest global model parameters from the parameter server for the next iteration (as shown in Figure 4). The method for setting a synchronization barrier to ensure the consistency of the global parameters is called BSP. There are two obvious disadvantages of BSP: First, it requires large communication overhead for each iteration. Qirong Ho's research [13] shows that the parameter communication time is more than six times the iterative calculation cost. Second, the next iteration does not begin until all computational nodes have completed the calculations in the first iteration, which requires the load on the cluster to be very balanced. However, Chilimbi's research [32] shows that even in a load-balanced cluster, a portion of the computational nodes can be randomly and unpredictably slower than the others.

C. STALE SYNCHRONOUS PARALLEL

To overcome problems of BSP, Ho et al. proposed SSP [13]. Due to the iterative-convergence nature of most ML algorithms, SSP does not use a synchronization barrier immediately after an iteration. Therefore, the computational nodes can perform the next iteration directly instead of waiting for other computational nodes to finish. It only synchronizes all computational nodes when the fastest computational node exceeds the slowest computational node by s iterations; s is called the stale threshold. For distributed ML algorithms, SSP yields faster training efficiency and convergence. Wei et al. uses SSP to propose a parameter server framework for distributed ML [33], [34]. Cui *et al.* [35] improves this framework so that it can be used in distributed GPUs.

However, according to our previous research [36], there are two obvious problems with SSP. We considered the typical distributed ML model training scenario and examined the problems with the strategy. The first problem is that SSP cannot optimize a distributed ML algorithm that is running on the cluster, which consists of similar-performance computational nodes. We set five computational nodes to perform the training of the distributed ML model, as illustrated in Figure 5. The axis shows the number of iterations for which the computational nodes train after a synchronization barrier; we found that each computational node has similar performance. Since SSP only sets a global stale threshold, if we set the value of stale threshold to greater than or equal to 3, each computational node will train for multiple iterations using stale global model parameters. Thus, the convergence and accuracy of the distributed ML model training cannot be guaranteed.

Without considering the extraneous factors that can disturb the model training process, the second problem of SSP arises. In Figure 6, the axis shows the number of iterations for which the computational nodes train after each synchronization barrier. If the difference in the number of iterations between computational node 1 and 3 is more than s, computational node 1 will enter the synchronization barrier to wait for the other computational nodes to finish their iterations.

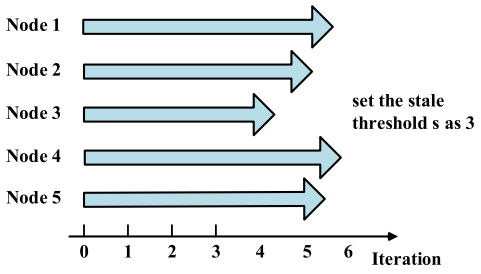


FIGURE 5. Cluster of nodes with similar performance running a distributed ML algorithm.

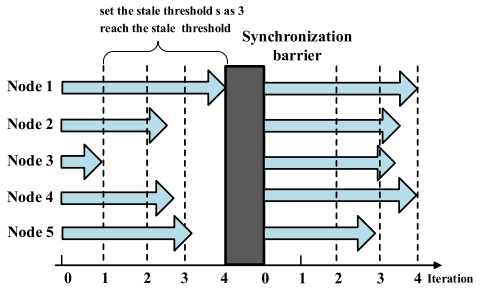


FIGURE 6. The training model may encounter external influences.

After that, all computational nodes perform global model parameter updating so that each computational node can use the new global model parameters for the next iteration. However, in the next round of iteration, if the computing performance of computational node 3 increases for some reason (such as completing other unrelated computational tasks). At the same time, the number of iterations that have been finished by each computational node is similar, and because the computational nodes cannot reach the stale threshold, they cannot perform enough iterations to guarantee the convergence and accuracy of the distributed ML model training.

III. DYNAMIC COMMUNICATION STRATEGY

In this section, we first introduce the theoretical analysis of ASP and then propose the performance monitoring model for implementing the dynamic parameter adjustment mode. Finally, we introduce the improvements for SSP in detail and present ASP.

A. THEORETICAL ANALYSIS OF ASP

Most of the ML programmes have an iterative-convergent nature and can be expressed as Equation 1:

$$L = f(X, M) = f(I_{i=1}^{N}\{x_i, y_i\}, M) \tag{1}$$

In Equation 1, N is the total number of samples in the data set, $\{x_i, y_i\}$ is one of the samples in the data set, y_i only appears in the marked data set, and M is the ML model. The programme uses the data samples to fit the ML model through constant iteration.

Because of the massive sizes of the data set and the ML model, parallelism and distributed training are required. Here, we introduce the principle of SSP. Assuming that the stale threshold is S, the model $\tilde{M}_{p,t}$ that it can access is made up of the initial model M_0 and the update $u_{p,t}$ for a computational node P with an iteration time of t. The model that adopts the SSP is as follows:

$$\tilde{M}_{p,t} = M_0 + \left[\sum_{i=1}^{t-s-1}\sum_{j=1}^{P} u_{j,i}\right] + \left[\sum_{(i,j)\in U_{p,t}} u_{j,i}\right] \tag{2}$$

In Equation 2, $U_{p,t}$ represents an update subset of all computational nodes P in the 2s iterations from t-s to t + s-1. $\left[\sum_{i=1}^{t-s-1}\sum_{j=1}^{P} u_{j,i}\right]$ is the update that takes full advantage of the performance during this iteration period.

The following analysis demonstrates the feasibility of SGD in the ASP scenario. For a convex function $L = f(M) = \sum_{c=1}^{C} f_c(M)$, to obtain the minimum value of the model, we utilize SGD. The gradient of each computational node is denoted as ∇f_c, s denotes the stale threshold, the differences between the all computational nodesą́ŕ performance factors are denoted as α and the total number of computational nodes is denoted as P. Denote by c the update of the iterative period $u_c := -\eta_c \nabla_c f_c(\tilde{M}_c)$. The step size is $\eta_c = \frac{\sigma}{\sqrt{c}}$, and F and L are constants in $\sigma = \frac{F}{L\sqrt{2(\frac{s}{\alpha}+1)P}}$. Adapting from [1], [13], and [33], we can obtain the ASP formula as follows:

$$\begin{aligned} R[M] :&= \left[\frac{1}{C}\sum_{c=1}^{C} f_c(\tilde{M}_c)\right] - f(M^*) \\ &\le \sum_{c=1}^{C}\left\langle \nabla f_c(\tilde{M}_c), \tilde{M}_c - M^*\right\rangle \\ &= \sigma L^2\sqrt{C} + F^2\frac{\sqrt{C}}{\sigma} + 2\sigma L^2\left[\left(\frac{s}{\alpha}+1\right)P\right]^2 \\ &\quad + 4\sigma L^2\left(\frac{s}{\alpha}+1\right)P\sqrt{C} \\ &\le 4FL\sqrt{2\left(\frac{s}{\alpha}+1\right)PC} \end{aligned} \tag{3}$$

The formula describes the ML model that is obtained by using ASP. We can establish that ASP converges to $O(\sqrt{C})$ and provides correctness guarantees.

The formula that is obtained from [1], [13], and [33] takes the number of error updates into account with the upper limit 2(s+1)P, so convergence rates may be poor and the training efficiency may temporarily decrease. Our formula (Equation 3) solves problems in SSP and generalizes this formula. We propose the weak threshold w in this paper to represent the number of iterations that are completed by the worst-performing node of all the computational nodes, and the difference factor α represents the difference in performance of each computational node. α close to 1 means that each computational node in the cluster has similar performance, which causes the stale threshold s to fail; in this case, we change the restriction condition of ASP to the synchronization barrier from the stale threshold s to the weak threshold w. If the performance difference among the computational nodes is

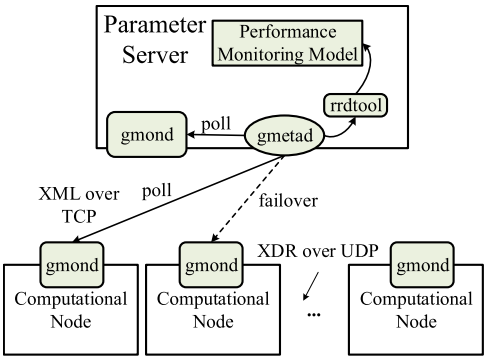


FIGURE 7. Performance monitoring model is used to realize the dynamic adjustment of parameter communication.

larger than the average, α is larger. In this case, we can increase the stale threshold s by referring to Equation 3 to guarantee the convergence rate. We modify Equation 4 as follows:

$$R[M] \leq \begin{cases} 4FL\sqrt{2\left(\frac{s}{\alpha}+1\right)PC}, & \alpha \gg 1 \\ 4FL\sqrt{2\left(w+1\right)PC}, & \alpha \approx 1 \end{cases} \quad (4)$$

B. PERFORMANCE MONITORING MODEL

To realize the dynamic adjustment of parameter communication, this paper implements a performance monitoring model. The model takes advantage of the open-source distributed monitoring system Ganglia [37] to obtain the performance information of each computational node, including the CPU utilization, memory utilization, hard disk utilization, I/O load, and network traffic.

Each node is configured with a Ganglia monitoring daemon (gmond), which can collect the monitoring data of the node and send it to the other computational nodes via the UDP protocol. In this way, each computational node can obtain the entire cluster of monitoring data.

If the Ganglia Meta daemon (gmetad) is configured on the parameter server, the gmetad will collect the monitoring data by the gmond through the periodic polls of the TCP protocol and then store it in the Round Robin Database. When the gmetad fails to extract the monitoring data from a gmond, since each gmond has the complete cluster monitoring data, the gmetad can extract the monitoring data from others to ensure the robustness of the monitoring system.

In this paper, the performance monitoring model (shown in Figure 7) is used to directly obtain the parameters that are the most relevant to the distributed ML model training from the circular database. The CPU occupancy rate is used to determine whether the stale threshold s should be changed according to the CPU occupancy rate of each computational node. When the CPU usages of the computational nodes are very different, the current performance differences among the computational nodes are large. Therefore, the stale threshold s needs to be improved. On the other hand, if the CPU occupancy rates of the computational nodes are not greatly different, the current performance differences among the nodes are small and it is necessary to adopt a smaller stale threshold to avoid the performance of many iterations of computational nodes without updating the global model parameters, which will seriously affect the convergence rate and accuracy of the ML model.

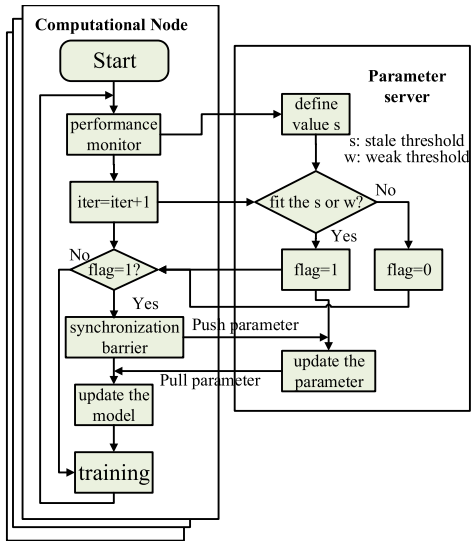


FIGURE 8. Adaptive Synchronous Parallel strategy generalizes and improves upon SSP.

C. ADAPTIVE SYNCHRONOUS PARALLEL STRATEGY

We present a dynamic communication strategy called ASP, which generalizes and improves upon SSP. The flow chart of the strategy is shown in Figure 8.

The flow of each computational node is similar, so we take a computational node and a parameter server as an example. When the computational node starts iterating, the gmond will monitor the performance of the computational node and the parameter server will extract the performance data of each computational node and compare their CPU occupancy rates. Then, the parameter server will adjust the stale threshold s dynamically according to the performance differences among the nodes. At this point, the computational node will upload the number of iterations to the parameter server. The parameter server will judge whether the number of iterations of the computational nodes satisfies the stale threshold s and the weak threshold w.

If the number of iterations reaches the condition of the synchronization barrier, the flag will be set to 1 and sent to all computational nodes. The computational node enters the synchronization barrier and then sends its local parameter to the parameter server so that the parameter server can update the global model parameters and return them to the computational nodes. The computational nodes use the new global model parameters to update the model and train it. However, if the condition of the synchronization barrier is not reached, the computational node does not need to enter the synchronization barrier. Therefore, in this case, the training

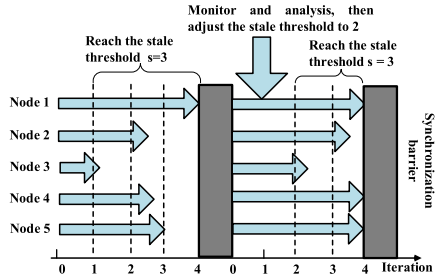


FIGURE 9. We implement a performance monitoring model to dynamically adjust the delay threshold s.

is carried out directly with the stale model to improve the training efficiency of the model.

ASP solves the two main problems of SSP, which are mentioned in section 2.3, and has a better effect on the actual operation of the cluster than the approach that was used in our previous research [36]. We add another synchronization barrier condition: the minimum number of training iterations of the computational nodes for completing the iteration (we set w as the weak threshold), or we set s as the stale threshold. If the computational nodes satisfy one of these conditions, then they will enter the synchronization barrier and update the global model parameters after finishing their iterations. We implement a performance monitoring model to dynamically adjust the delay threshold s. As shown in Figure 9, the performance monitoring model monitors the change in the threshold of computational node 3 after a new round of iteration through the synchronization barrier, and obtains that the performance differences between computational node 3 and the other computational nodes have been reduced through increasing the computational performance of computational node 3. Thus, by decreasing the stale threshold s to 2, we can avoid unnecessary iterations of each computational node, thereby solving problem regarding the failure of the stale threshold s.

IV. EXPERIMENTS

A. ENVIRONMENT

In this paper, the distributed deep learning framework that is based on Caffe is implemented by ASP and the parameter server. The structure is shown in Figure 10.

The parameter server is made up of the global parameter storage function, the communication strategy control, the parameter update thread, the gmetad, the resource allocation scheduler, and the POSIX threads. The global parameter storage function is a vector-valued function that ensures that the global model parameters are up to date. The communication strategy control dynamically adjusts ASP. We set up a thread to calculate and update the global model parameters through the local model parameters, which are transmitted by the computational nodes. The gmetad collects the monitoring data and sends them to the resource allocation scheduler. The resource allocation scheduler carefully analyses the monitoring data and then decides whether to modify s and w.

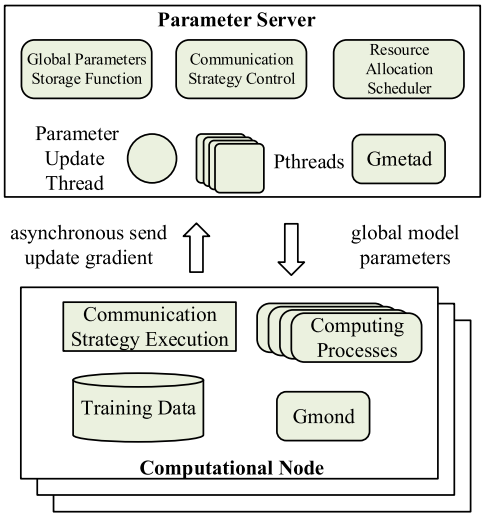


FIGURE 10. The structure of distributed ML framework based on Caffe.

Each computing process in the computational nodes has a corresponding POSIX thread in the parameter server, which is responsible for communicating information to the computational nodes, such as the number of iterations for each computing process and the local model parameters.

Each computational node is made up of the computing process, the training data, the gmod, and the communication strategy execution. The computing process is designed to train the ML model using data parallelism. The training data on computational nodes are averaged from the data set according to the number of computational nodes. The gmod collects the monitoring data of each computational node and sends it to the other computational nodes. The communication strategy execution judges whether computational nodes need to enter the synchronization barrier according to s and w, both of which are calculated by the parameter server. If neither threshold is reached, the communication strategy execution allows the computational node to use the stale global model parameters to go on the next iteration. If one of the thresholds is reached, all computing processes enter the wait-state as soon as their current iterations are completed and until they receive the latest global model parameters from the parameter server; then, they exit the wait-state.

The communication between the parameter server and computational nodes is executed by the MPICH. The asynchronous uploading of the gradient in the processes of the computational nodes can reduce the risk of communication congestion. The global model parameters are sent to the computational nodes in proper sequence on the basis of the parameter update module queue.

The experiments in this paper use the distributed ML framework that is described above. We implement this framework by using 4 nodes and connect them with gigabit Ethernet. Each node is configured with 16 cores of a 2.4 GHz AMD Opteron processor (Processor 6136) and 32 GB of RAM, running on top of CentOS7.0.

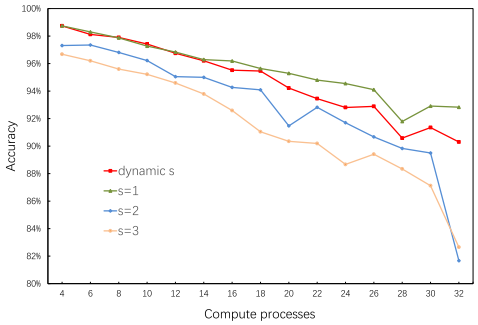


FIGURE 11. Comparing the accuracy of the ML model.

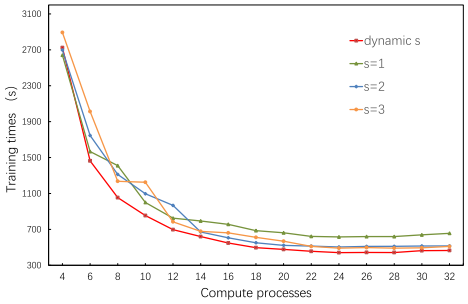


FIGURE 12. Comparing the training time of the ML model.

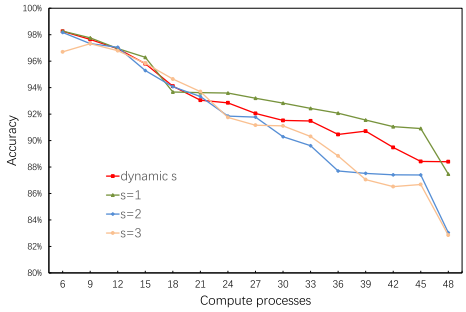


FIGURE 13. Verification of the extensibility of ASP based on the accuracy of the ML model.

Datasets: The MNIST handwritten numeric font data set [38] consists of a 60000-image training set and a 10000-image test set.

ML models: LeNet-5 [38] designs one input layer, one output layer, three convolution layers, two pooled layers and one fully connected layer. The batch size is 64 and the maximum number of iterations is 10000.

B. EFFECTIVENESS AND PERFORMANCE OF ASP

This section verifies the effectiveness of ASP and tests its performance by experiments. In this experiment, we apply three nodes of the distributed ML framework above to train the distributed ML model. One node in the framework acts as the parameter server; the others are set as computational nodes. When the number of ML model training iterations reaches 5000, this experiment adds an interference programme to the computational nodes to simulate the dynamic change in the performances of the computational nodes. The experiment compares SSP using different values of s and the influence of ASP on the accuracy and the training time of the distributed ML model. Figures 11 and 12 depict the results, in which we set the value of s in SSP to 1, 2, and 3. ASP dynamically adjusts the value of s. The weak threshold w is set to 1 in both strategies.

As shown in Figure 11, using either SSP or ASP to train the ML model, the accuracy is reduced as the number of processes increases. This is because the stochastic gradient descent algorithm is adopted for model training, which has some error, and when the number of processes increases, this error is amplified, which results in a decrease of accuracy. The stale synchronization strategy with a stale threshold s of 1 (in this situation, SSP is the same as BSP) has the highest accuracy. In addition, the ML model with ASP guarantees high accuracy due to the dynamic adjustment of stale threshold s. With the increase of the stale threshold, the accuracy of the model decreases greatly. As shown in Figure 12, due to communication between computational nodes and the parameter server, the training time does not decrease linearly as the number of computing processes increases; beyond a certain number of computing processes, the training time increases. The adaptive stale synchronous strategy requires the shortest time for model training, and the SSP with a stale threshold s of 1 requires the longest time.

It can be concluded from the above results that the distributed ML model with ASP reduces the training time while ensuring a certain accuracy. This verifies the validity of ASP and shows that the strategy achieves good performance.

C. EXTENSIBILITY OF THE ASP STRATEGY

To verify the extensibility of ASP, we apply four nodes of the distributed ML framework above to train the distributed ML model, where a node acts as a parameter server and the others act as computational nodes. To balance the number of iteration tasks among the computational nodes, the training processes are evenly distributed among three computational nodes. When the ML model training reaches 5000 iterations, an interference programme is added to one of the computational nodes to simulate the dynamic performance of the cluster. Other conditions are the same as in section 4.2. The specific results can be seen in Figures 13 and 14. We set the value of s in SSP to 1, 2, and 3, and ASP dynamically adjusts the value of s. In both strategies, we set the value of w to 1.

As shown in Figures 13 and 14, in terms of the accuracy, the effect of model training after adding nodes is roughly the same as that reported in section 4.2, and the accuracy decreases as the number of processes increases. Model training with ASP still achieves good accuracy, which is only lower than that of the DSP strategy with a stale threshold s of 1 (this SSP is the same as BSP). Moreover, model training with ASP requires less training time. However, compared to the experiment in section 4.2, the communication cost is increased due to the addition of new computational nodes,

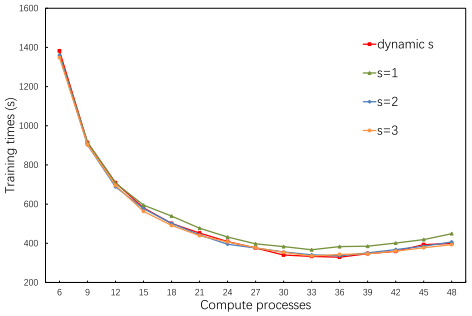


FIGURE 14. Verification of the extensibility of ASP based on the training time of the ML model.

and the gap between model training cost and communication cost is reduced, so the training times for both strategies are similar. In contrast, SSP with a delayed threshold s of 1 is requires a lot of training time.

According to the results above, ASP has good expansibility and can be used in larger clusters. However, the communication cost is worth considering. It is necessary to balance communication costs and training expenses to achieve better accuracy and training time.

V. CONCLUSION

Training the distributed ML model with SSP can reduce the communication and synchronization costs and improve the utilization rate of the computational nodes and the efficiency of calculation. However, the accumulated error can sometimes seriously damage the convergence rate of the distributed ML model. In this work, we propose a new communication strategy called ASP, which adds a weak threshold and adjusts the stale threshold dynamically based on the monitoring data that are obtained by Ganglia to balance the training time and the model accuracy, and improve the performance of the distributed ML algorithm. The experimental results show that ASP can guarantee better accuracy and achieve higher convergence speed than SSP, and it has good expansibility.

Currently, using ASP as a communication strategy to train a large-scale distributed ML model will yield low model accuracy. In the future, we will improve the applicability of ASP in large-scale clusters.

REFERENCES

[1] E. P. Xing, Q. Ho, P. Xie, and D. Wei, "Strategies and principles of distributed machine learning on big data," *Engineering*, vol. 2, no. 2, pp. 179–195, 2016.

[2] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 873–881.

[3] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed GraphLab: A framework for machine learning and data mining in the cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[4] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proc. Int. Conf. World Wide Web*, 2013, pp. 37–48.

[5] A. Ahmed, M. Aly, J. Gonzalez, S. Narayanamurthy, and A. J. Smola, "Scalable inference in latent variable models," in *Proc. Int. Conf. Web Search Data Mining*, 2012, pp. 123–132.

[6] J. K. Kim *et al.*, "STRADS: A distributed framework for scheduled model parallel machine learning," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, Art. no. 5.

[7] T.-Y. Liu, W. Chen, and T. Wang, "Distributed machine learning: Foundations, trends, and practices," in *Proc. 26th Int. Conf. World Wide Web Companion*, 2017, pp. 913–915.

[8] B. Randell, "System structure for software fault tolerance," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 2, pp. 220–232, Jun. 1975.

[9] W. F. McColl, "Bulk synchronous parallel computing," in *Abstract Machine Models for Highly Parallel Computers*. Oxford, U.K.: Oxford Univ. Press, 1995, pp. 41–63.

[10] A. V. Gerbessiotis and L. G. Valiant, "Direct bulk-synchronous parallel algorithms," *J. Parallel Distrib. Comput.*, vol. 22, no. 2, pp. 251–267, 1994.

[11] J. Dean *et al.*, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc, 2012, pp. 1223–1231.

[12] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[13] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.

[14] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, "On model parallelization and scheduling strategies for distributed machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2834–2842.

[15] H. Ma, F. Mao, and G. W. Taylor, "Theano-MPI: A theano-based distributed training framework," in *Proc. Eur. Conf. Parallel Process.*, 2016, pp. 800–813.

[16] Y. Zou, X. Jin, Y. Li, Z. Guo, E. Wang, and B. Xiao, "Mariana: Tencent deep learning platform and its applications," *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1772–1777, 2014.

[17] J. Jiang, J. Jiang, B. Cui, and C. Zhang, "TencentBoost: A gradient boosting tree system with parameter server," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 281–284.

[18] K. Yu, "Large-scale deep learning at baidu," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2013, pp. 2211–2212.

[19] J. Zhou *et al.*, "KunPeng: Parameter server based distributed learning systems and its applications in alibaba and ant financial," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1693–1702.

[20] J. Zhou, Q. Cui, X. Li, P. Zhao, S. Qu, and J. Huang, "PSMART: Parameter server based multiple additive regression trees system," in *Proc. 26th Int. Conf. World Wide Web Companion*, 2017, pp. 879–880.

[21] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[22] T. Chen *et al.* (2015). "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems." [Online]. Available: https://arxiv.org/abs/1512.01274

[23] E. P. Xing *et al.*, "Petuum: A new platform for distributed machine learning on big data," *IEEE Trans. Big Data*, vol. 1, no. 2, pp. 49–67, Jun. 2015.

[24] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 703–710, 2010.

[25] J. Langford, A. J. Smola, and M. Zinkevich, "Slow learners are fast," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 2331–2339.

[26] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, 2008.

[27] C. H. Teo, S. V. N. Vishwanthan, A. J. Smola, and Q. V. Le, "Bundle methods for regularized risk minimization," *J. Mach. Learn. Res.*, vol. 11, pp. 311–365, Jan. 2010.

[28] M. A. Zinkevich, M. Weimer, A. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2595–2603.

[29] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 19–27.

[30] M. Li, Z. Liu, A. J. Smola, and Y.-X. Wang, "Difacto: Distributed factorization machines," in *Proc. 9th ACM Int. Conf. Web Search Data Mining*, 2016, pp. 377–386.

[31] M. Li *et al.*, "Scaling distributed machine learning with the parameter server," in *Proc. Int. Conf. Big Data Sci. Comput.*, 2014, pp. 583–598.

[32] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an efficient and scalable deep learning training system," in *Proc. USENIX Conf. Oper. Syst. Design Implement.*, 2014, pp. 571–582.

[33] W. Dai, A. Kumar, J. Wei, Q. Ho, G. Gibson, and E. P. Xing, "High-performance distributed ML at scale through parameter server consistency models," in *Proc. 29th AAAI Conf. Artif. Intell.*, 2015, pp. 79–87.
[34] J. Wei *et al.*, "Managed communication and consistency for fast data-parallel iterative analytics," in *Proc. ACM Symp. Cloud Comput.*, 2015, pp. 381–394.
[35] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, Art. no. 4.
[36] J. Zhang *et al.*, "A parameter communication optimization strategy for distributed machine learning in sensors," *Sensors*, vol. 17, no. 10, p. E2172, 2017.
[37] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: Design, implementation, and experience," *Parallel Comput.*, vol. 30, no. 7, pp. 817–840, 2004.
[38] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1990, pp. 396–404.

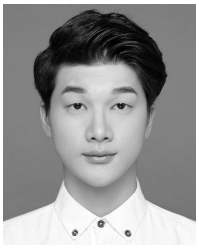
JILIN ZHANG received the Ph.D. degree in computer application technology from the University of Science Technology Beijing, Beijing, China, in 2009. He currently serves as an Associate Professor with the School of Computer Science and Technology, Hangzhou Dianzi University. His research interests include high-performance computing and cloud computing.

HANGDI TU is currently pursuing the M.S. degree with the School of Computer Science and Technology, Hangzhou Dianzi University, China. His research interests include parallel computing, machine learning, and high-performance computing.

YONGJIAN REN received the Ph.D. degree in engineering from Zhejiang University, Hangzhou, China, in 1989. He is currently a Distinguished Professor with Hangzhou Dianzi University. His research interests include mass storage and cloud computing.

JIAN WAN received the Ph.D. degree in computer application technology from Zhejiang University, Zhejiang, China, in 1989. He is currently a Professor in software engineering with Hangzhou Dianzi University, China. His research interests include grid computing, service computing, and cloud computing.

LI ZHOU received the master's degree from Hangzhou Dianzi University, Hangzhou, China, in 2003. She is currently an Associate Professor with the School of Computer Science and Technology, Hangzhou Dianzi University. Her current research interests include virtual storage system, cloud storage, cloud computing, and high-performance computing.



MINGWEI LI is currently pursuing the M.S. degree with the School of Computer Science and Technology, Hangzhou Dianzi University, China. His research interests include parallel computing, machine learning, and high-performance computing.

JUE WANG is currently an Associate Professor with the Supercomputing Center, Chinese Academy of Science. The motivation behind his work is to improve soft systems by increasing the productivity of programmers and by increasing software performance on modern architectures including many cores clusters and GPU.