Received October 20, 2019, accepted November 24, 2019, date of publication November 28, 2019, date of current version December 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956632

Parameter Communication Consistency Model for Large-Scale Security Monitoring Based on Mobile Computing

RUI YANG[1,2], JILIN ZHANG[1,2,3], JIAN WAN[1,2,4], LI ZHOU[1,2], JING SHEN[1,2], YUNCHEN ZHANG[1,2], ZHENGUO WEI[5], JUNCONG ZHANG[5], 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 Dianzi University, Hangzhou 310018, China
[3]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
[4]School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou 310023, China
[5]Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China
[6]Zhejiang Dawning Information Technology Company, Ltd., Hangzhou 310051, China

Corresponding authors: Jilin Zhang (jilin.zhang@hdu.edu.cn) and Li Zhou (juliy26@hdu.edu.cn)

This work was supported in part by the National Key Technology Research and Development Program under Grant 2018YFB0204001, in part by the National Natural Science Foundation of China under Grant 61672200 and Grant 61572163, in part by the Key Technology Research and Development Program of the Zhejiang Province under Grant 2019C03135, in part by the Zhejiang Natural Science Funds under Grant LY17F020029 and Grant LY16F020018, in part by the State Key Laboratory of Computer Architecture under Project CARCH201712, and in part by the Hangzhou Dianzi University Postgraduate Research Innovation Fund Program under Grant CXJJ2018052.

ABSTRACT With the application of mobile computing in the security field, security monitoring big data has also begun to emerge, providing favorable support for smart city construction and city-scale and investment expansion. Mobile computing takes full advantage of the computing power and communication capabilities of various sensing devices and uses these devices to form a computing cluster. When using such clusters for training of distributed machine learning models, the load imbalance and network transmission delay result in low efficiency of model training. Therefore, this paper proposes a distributed machine learning parameter communication consistency model based on the parameter server idea, which is called the limited synchronous parallel model. The model is based on the fault-tolerant characteristics of the machine learning algorithm, and it dynamically limits the size of the synchronization barrier of the parameter server, reduces the synchronization communication overhead, and ensures the accuracy of the model training; thus, the model realizes finite asynchronous calculation between the worker nodes and gives full play to the overall performance of the cluster. The implementation of cluster dynamic load balancing experiments shows that the model can fully utilize the cluster performance during the training of distributed machine learning models to ensure the accuracy of the model and improve the training speed.

INDEX TERMS Mobile computing, security monitoring, distributed machine learning, limited synchronous parallel model, parameter server.

I. INTRODUCTION

The smart city is an inevitable trend of urban modernization and informatization development. The smart city plays a positive role in supporting social economic development, innovating economic development models, improving urban functional quality, and ensuring and improving people's livelihood. As an important basic resource of urban big data, security monitoring provides a large amount of image information. With the application of mobile computing, mobile computing makes full use of the computing power and communication capabilities of various sensing devices, greatly increasing data diversity and computing flexibility. The magnitude of data has shifted from terabytes to petabytes and is now shifting towards ZB, and traditional machine learning methods are facing severe challenges with this massive

The associate editor coordinating the review of this manuscript and approving it for publication was Ligang He.

amount of data. The distribution of traditional machine learning methods to adapt to the ever-increasing data scale has become a research hotspot in both academia and industry.

This paper focuses on the iterative convergence algorithm. This type of algorithm starts with the initial estimate and then continuously explores the approximate solution such that the problem is finally solved. The distributed implementation of the iterative convergence algorithm usually depends on the Bulk Synchronous Parallel model (BSP) [1]. In this model, each compute process performs the same iteration on the local model local replica generated by the previous iteration. After each iteration, each compute process enters the synchronization barrier for synchronization, which may significantly reduce the performance of these algorithms. This effect is observed because the time of each iteration is always determined by the worst-performing compute process. This problem is proportional to the degree of parallelism: with the increase in the number of compute processes, the probability that at least one's performance will lag in any given iteration also increases. In addition, potential causes of performance lag (such as hardware and network changes) are often unpredictable [2], [3].

Facing the performance lag problem of the bulk synchronous parallel model, Dean et al. proposed a distributed machine learning asynchronous iterative scheme [4] in which each compute process performs a full asynchronous calculation and each compute process synchronizes with the parameter server immediately after completing the iteration, greatly utilizing the performance of each computing node; however, because the model and the updated parameters exhibit uncontrollable delays, there is no guarantee regarding the model convergence speed. Ho et al. proposed a corresponding model for this problem, Stale Synchronous Parallel (SSP) [5], which takes into account the bulk synchronous and asynchronous characteristics: allowing some compute processes to perform a certain number of iterations that differs from other compute processes and iterating using local model parameters before global synchronization.

The two models described above have problems; thus, in this paper, a parameter communication consistency model based on a dynamic adaptive limited synchronization barrier is studied, which is called the limited synchronous parallel model (LSP). The model dynamically adjusts the size of the synchronization barrier according to the performance difference of each compute process, which effectively reduces the lag effect of slow compute processes on machine learning model training and considerably improves the training speed. The experiments of this paper show that, compared with the SSP, model training under LSP shows higher training accuracy; compared with BSP, the model training under LSP shows faster training speed.

This article describes some of the components of parameter communication in distributed machine learning in Section 2; Section 3 describes in detail the distributed machine learning parameter communication consistency model and provides the corresponding theoretical proof and the distributed machine learning framework for security monitoring based on the mainstream parameter communication consistency model; Section 4 describes experimental analysis on distributed machine learning using a limited synchronous model to verify the corresponding theory and functions; and Section 5 presents this study's conclusions.

II. RELATED WORK

The distributed machine learning parameter communication consistency model proposed in this paper is mainly related to distributed machine learning systems, parameter servers and parameter communication consistency models. Therefore, this section is divided into the above three parts to introduce related research work.

A. DISTRIBUTED MACHINE LEARNING SYSTEMS

Today, distributed systems are widely used, greatly improving the degree of machine learning distribution, and these systems have become a heavily researched topic in academic and industrial research.

Jia et al. of Berkeley University implemented a deep learning framework, Caffe [6], which has two training modes (CPU and GPU), for training specific neural networks. Chen et al. of the University of Washington developed a multilingual machine learning library, MXNet [7], which combines symbolic expressions with tensor calculations to maximize efficiency and flexibility. MXNet is lightweight and highly scalable for fast model training. Xing et al. of Carnegie Mellon University proposed the universal distributed system Petuum [8], providing users with a unified platform that can be used to build any large-scale machine learning or deep learning applications. Petuum can be deployed on any hardware, including workstations, data centers, and personal hosts, and offers a variety of standardized machine learning solutions for researchers and businesses. Coates et al. at Stanford University designed and implemented a GPU-based data-parallel and model-parallel framework COTS HPC [9], which uses Infiniband connections between GPU clusters and controls communication by MPI. Smith et al. proposed a general framework CoCoA [10] for machine learning and signal processing that covers general Non-Strongly convex regularizers, including some L1 regularization problems, such as sparse logistic regression, lasso regression, and elastic network regularization. The application programming interface MLI [11] proposed by Sparks et al. at the University of California, Berkeley, can be used to build a variety of common machine learning algorithms and then implement them in a distributed fashion. Collobert et al. of the Idiap Institute developed a multifunction digital computing framework and machine learning library Torch7 [12], which achieves high performance through CUDA with efficient OpenMP / SSE and low-level numerical routines. Huang et al. of the Chinese University of Hong Kong developed FlexPS [13], which supports changing parallelism at runtime, that is, mapping machine learning tasks to a series of stages through multi-stage abstraction and then setting the parallelism of stages according to their workloads.

Google has developed the large-scale distributed system DistBelief [4], which can scale to clusters with more than 10,000 CPU cores and can train neural networks with billions of parameters. This result greatly accelerates the training of neural networks for commercial voice recognition services. Based on DistBelief, TensorFlow [14], a large-scale open-source machine learning framework for second-generation heterogeneous distributed systems, was implemented by Google Research Institute. It transparently supports GPU acceleration, off-core operation, and multi-threading and can be extended across multiple nodes. It is one of the most popular machine learning frameworks at present. Microsoft has open-sourced the computational graph-based deep learning framework CNTK [15], and Microsoft created the Cortana speech model based on it; CNTK also has many other functions, especially serial-to-sequence modeling, which has been widely used. Keras has broad adoption in the industry and the research community, and Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Based on Spark, Databricks developed a distributed machine learning library, Mllib [16], which can simplify the development of machine learning pipelines through Spark. Liu et al. proposed the AFNDCAR scheme [17], which reduce the amount of redundant data and transmission delay in the body sensor networks (BSNs) by adjusting forwarder nodes and duty cycle. Gao *et al.* [18]–[20] studied the quality of service and quantitative verification in mobile device environments. Yin *et al.* [21]–[23] specifically target QoS prediction methods for mobile terminals. Alibaba has proposed a general distributed machine learning system KunPeng [26]. At present, KunPeng has been applied in many practical scenarios, such as Taobao "Double 11 shopping carnival" and Ant Financial Risk Assessment, and achieved good results. In addition, Baidu has built a multi-machine GPU training platform, Paddle [27], and Tencent has built a deep learning platform, Angel [28].

However, most of the known distributed machine learning systems use compulsory global synchronization for each iteration as a training method, which loses the advantage of large-scale cluster performance. Several of them use a fully asynchronous model to update parameters, which cannot guarantee the convergence of the model. Aiming for the algorithm characteristics of dense iterative convergence distributed machine learning, the distributed machine learning system with loose consistency from the perspective of machine learning fault tolerance has not made a big breakthrough.

B. PARAMETER SERVER

Faced with the scalability issues of LDA and related topic models, Smola et al. of Carnegie Mellon University proposed a parallel LDA architecture [29], which is called the first-generation parameter server. Fig. 1 shows its structure.

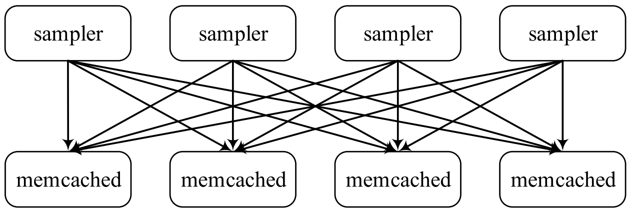

FIGURE 1. First-generation parameter server.

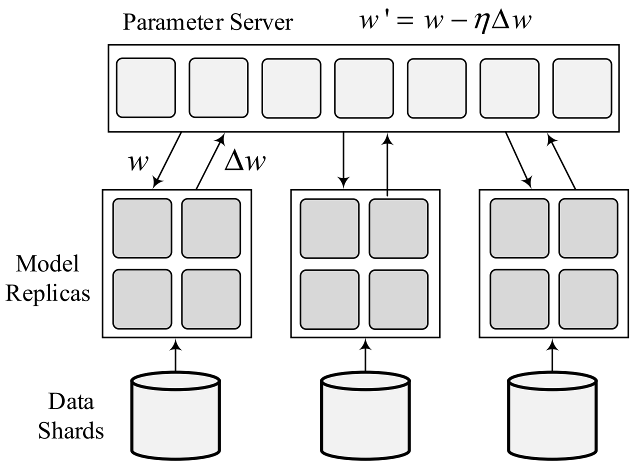

FIGURE 2. Second-generation parameter server.

The parallel LDA architecture adopts distributed Memcached, and the storage parameters adopt key-value pairs. It has excellent performance in the synchronization of worker nodes and reduces the storage of parameters to a certain extent. However, the communication overhead problem in today's large-scale distributed clusters is becoming increasingly prominent. Frequent parameter data exchange with key-value pairs as units will inevitably lead to excessive communication costs, which seriously affects the computational efficiency of the cluster.

The second generation of parameter servers emerged. YahooLDA [30] used a more standardized load balancing algorithm and a proprietary server with custom base units: get/set/update. DistBelief [4], a software framework for the first generation of Google Brain, was proposed by Dean of Google Research Institute. DistBelief stores the neural network model in the global parameter server. The parameter server communicates data with each worker node and supports data parallelism and model parallelism. It effectively solves the distributed problem of the SGD and L-BFGS algorithms. Fig. 2 shows the framework of the second generation of parameter servers.

Compared with the first-generation parameter servers, the data magnitude and training efficiency have been greatly improved, but the utilization of distributed cluster nodes is still low because the second-generation parameter servers do not fully consider the performance differences and stability of worker nodes.

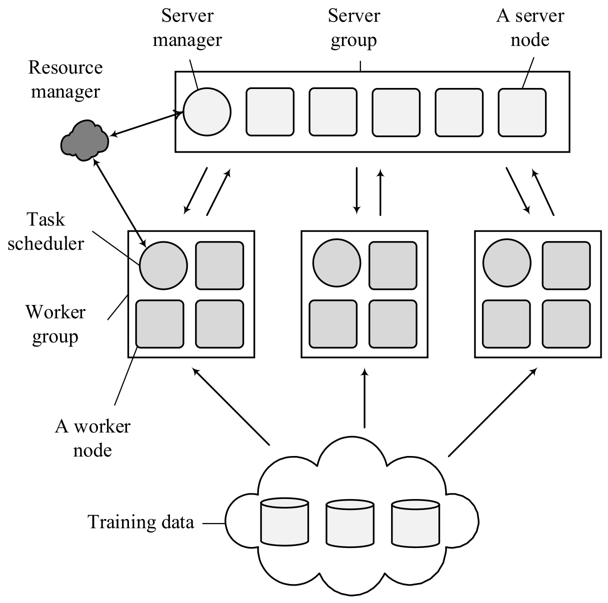

FIGURE 3. **Third-generation parameter server.**

Li et al. of Carnegie Mellon University proposed the third-generation parameter server system [31]. The worker node cluster and the parameter server cluster are the core components of the third-generation parametric server system, as shown in Figure 3. Global model parameters are distributed on each parameter server, which are managed by the management node. Many parallel algorithms can be executed in parallel on a group of worker nodes composed of multiple worker nodes. As with the second-generation parameter server, the primary role of the third-generation parameter server is to pass parameters. There is a scheduler in each worker node group, which is used to assign tasks and monitor the status. If the worker node is unresponsive or joins a new worker node, the scheduler can continue to assign tasks without interrupting training. Overall, the scheduler can solve the problem of worker node performance differences resulting in low computational efficiency, but it can only be handled by adding or removing worker nodes. This solution is singular and does not take into account factors such as external disturbances.

Considerable improvements in the parameter server have been found by both academia and industry. Harlap et al. focused on transient resource changes and implemented machine learning data processing and workload scheduling strategies based on parameter servers [32]. Li *et al.* [33] and Zhong *et al.* [34] use a parameter server to form a large-scale distributed factorization machine system. Zhou et al. implemented a multi-additive regression tree system based on a parameter server [35]. Yut et al. developed LDA* [36] based on an asymmetric parameter server architecture, which found a new balance between communication and computation and pushed some calculations to the parameter server. Cui et al. implemented GeePS [37], which optimizes the parameter server system for GPUs, enabling them to perform well on large-scale machine learning with GPUs.

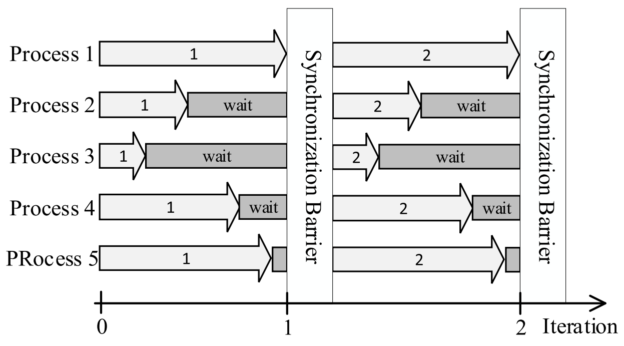

FIGURE 4. **Bulk synchronous parallel model.**

C. CONSISTENCY MODEL

If the scale of distributed machine learning is very large, then we must consider the consistency of the model parameters to increase the parameter read throughput, thereby speeding up the algorithm execution to ensure that the machine learning algorithm finally reaches the global optimal solution. Machine learning algorithms are usually implemented based on iterative convergence algorithms and are fault tolerant [38]. This makes it possible to design a loose consistency model.

We first introduce the classic bulk synchronous parallel model [1], as shown in Fig. 4, which is a strong consistency model that clearly separates the computational phase and the communication phase. Parallel machine learning programs running under the BSP are serializable. That is, their effects are equivalent to serial machine learning programs. Serialized machine learning programs ensure the correctness of all sequential queues, making BSP the most commonly used consistency model for distributed machine training.

Above all, there are two obvious defects in the bulk synchronous parallelism. The first drawback is that a large amount of communication overhead is required for each iteration. Ho *et al.* [5] conducted an experiment in which he ran the LDA topic model on a cluster and found that the parameter communication time was six times the computation time. The second drawback is that the next iteration will continue until all nodes have completed an iterative calculation. In this manner, we need to ensure that the performance of each worker node is similar. Chilimbi *et al.* [39] performed similar experiments and found another problem: even if the worker nodes have the same CPU, their calculation speed will change, and this change is unpredictable.

For the synchronous waiting problem, Dean *et al.* [4] proposed a distributed machine learning asynchronous iterative scheme (ASP). As shown in Fig. 5, after each compute process performs an iteration asynchronously, the parameter server reads the local parameters and updates the global model.

Ideally, the acceleration of ASP is linearly related to the number of compute processes [40]. Although ASP can greatly improve the utilization of a cluster, asynchronous

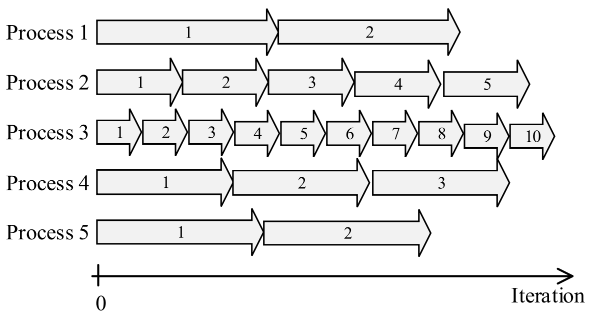

FIGURE 5. Asynchronous parallel model.

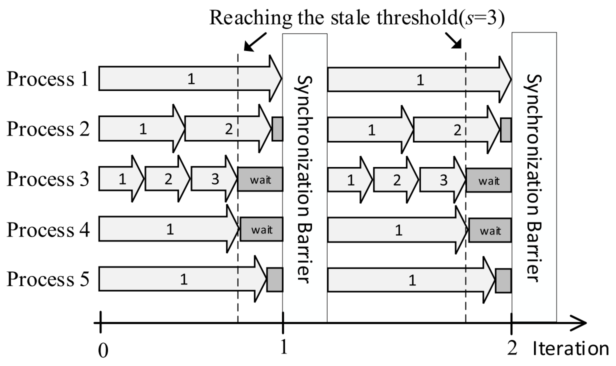

FIGURE 6. Stale synchronous parallel model.

communication will make the model information stale, which results in the updating of model errors, and the size of errors increases with the staleness of model information (the performance difference of compute processes increases). If you do not strictly constrain this stale information, it will cause the model to converge extremely slowly.

Ho *et al.* [5] proposed a stale synchronous parallel model based on ASP. As shown in Fig. 6, the SSP gives a stale threshold s and guarantees that the iterative progress of the compute process is maintained within the stale threshold s. That is, only when $t_{max} - t_{min} \geq s$ will all processes perform one synchronization, so that the error will not increase without limit.

The SSP does not fully consider its limited fault tolerance. If the worker node does not reach the stale threshold during the training process, the accuracy cannot be guaranteed. Our previous work improved it and proposed a dynamic synchronous parallel model based on dynamic finite fault tolerance [41], [42]. The DSP adds an optional condition for entering the synchronization barrier. When the minimum number of iterations in the compute process reaches the weak threshold w, the synchronization barrier can also be entered, thereby preventing the stale threshold from being set too large and preventing the parameter server from being unable to reach the synchronous condition. As shown in Fig. 7, the DSP dynamically adjusts s according to the state of multiple rounds of synchronization. If multiple rounds of iterations use the weak threshold w to enter the synchronization barrier, it indicates that the performance of each compute

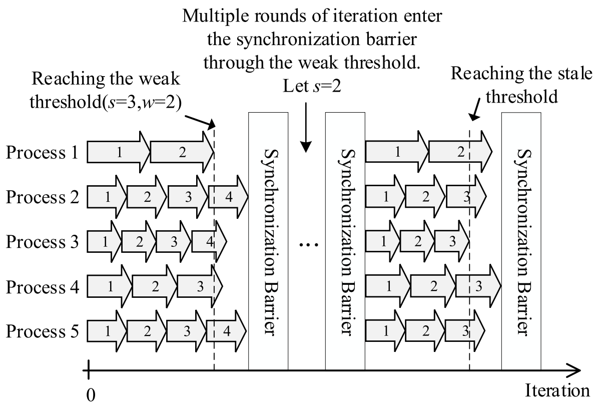
Multiple rounds of iteration enter the synchronization barrier through the weak threshold. Let s=2
Reaching the weak threshold(s=3,w=2)
Reaching the stale threshold
Process 1
Process 2
Process 3
Process 4
Process 5
Synchronization Barrier
...
0
Iteration

FIGURE 7. Dynamic synchronous parallel model.

process is similar, and the DSP will slightly lower the value of the stale threshold s. If multiple rounds of iterations use the stale threshold s to enter the synchronization barrier, it indicates that the performance difference of each worker node is large, and the DSP will slightly increase the value of s.

In addition, Khunayn *et al.* [43] and Harlap *et al.* [44] proposed their respective dataset dynamic allocation strategies based on the BSP mode. However, as the amount of data continues to increase, computing clusters are isomerized and distributed on a large scale, and dynamically allocating data sets will inevitably lead to communication congestion problems.

III. LIMITED SYNCHRONOUS PARALLEL MODEL

In this section, the detailed model design of the limited synchronous parallel model is first carried out, and then the stochastic gradient descent algorithm (SGD) widely used in neural network model training is taken as an example to theoretically analyze the feasibility of the limited synchronous parallel model. Finally, the distributed machine learning framework for Caffe implemented by this model and the parameter server idea is introduced.

A. MODEL DESIGN

Unlike the bulk synchronous parallel model, during each iteration of the LSP, the parameter server reads the local parameters of multiple, but not all, compute processes and updates the global model. Similar to the stale synchronous parallel model, the use of the LSP model must be aware of some of the key differences in the consistency model that may affect the design and performance of the algorithm. These differences can be described by the following characteristics.

1) LIMITED SYNCHRONIZATION BARRIER

To minimize the lag effect of the backward compute process, the LSP implements a limited barrier with a limited threshold $l \in [1, P]$. LSP makes the fastest l process of each iteration complete the calculation task and then synchronizes without waiting for the slow process.

Algorithm 1 Optimal Clustering Algorithm Based on K-Means

```
Input: performance data set $\mathbb{Z} = \{z_1, z_2, \ldots, z_P\}$; cluster cluster number $n$.
Output: optimal performance cluster $C^*$.
1.  Randomly select $n$ samples from $\mathbb{Z}$ as the initial mean vector $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$
2.  REPEAT
3.    Let $C_i = \emptyset\ (1 \le i \le n)$
4.    FOR $j = 1, 2, \ldots, P$ DO
5.      $d_{ji} = \|z_j - \mu_i\|_2$
6.      $\lambda_j = \arg min_{i\in\{1,2,\ldots,n\}} d_{ji}$
7.      $C_{\lambda_j} = C_{\lambda_j} \cup \{z_j\}$
8.    END FOR
9.    FOR $i = 1, 2, \ldots, n$ DO
10.     $\boldsymbol{\mu}_i' = \frac{1}{|C_i|}\sum_{z\in C_i} z$
11.     IF $\boldsymbol{\mu}_i' \neq \boldsymbol{\mu}_i$ THEN
12.       Update($\boldsymbol{\mu}_i$, $\boldsymbol{\mu}_i'$)
13.     END IF
14.   END FOR
15. UNTIL Current mean vector is not updated
16. $C^* = \arg min_{i\in\{1,2,\ldots,n\}} \|\mu_i\|_2$
```

2) LIMITED ASYNCHRONY

LSP are not only synchronized with good-performing compute processes. For each iteration, fast processes are relative, and even the worst-performing processes can still be fast processes. The finiteness of the asynchronousness depends on the monitoring frequency of the performance monitoring module, and the LSP performs global synchronization each time the performance detection data are obtained.

3) LOCAL UPDATE

After the compute process completes a single iteration, if it is identified as a slow process by the parameter server at this time, it is updated with a local model copy.

Fig. 8 illustrates the above characteristics of the LSP, assuming that the three compute processes enter the limited synchronization barrier after completing the computational task. In the first iteration, the best-performing compute process 3 first blocked the limited synchronization barrier, then entered the limited synchronization barrier with compute processes 2 and 4, and started the next iteration after obtaining the new global model, while compute process 5 was identified as a slow process and updated locally. In the second iteration, the compute processes 1, 2, and 3 enter the synchronization barrier, and there are no relatively slow nodes. It can be seen that the number of iterations of each compute process is not necessarily the same as the number of iterations of the parameter server, which reflects the asynchronous nature of the LSP.

How to set a limited threshold for a limited synchronization barrier is an important issue, and it is necessary to consider the inherent performance gap between nodes and network delays

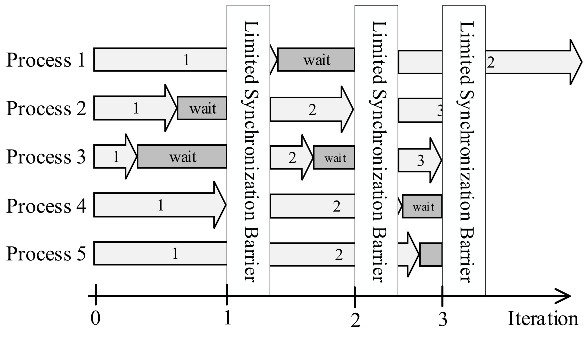

FIGURE 8. Limited synchronous parallel model.

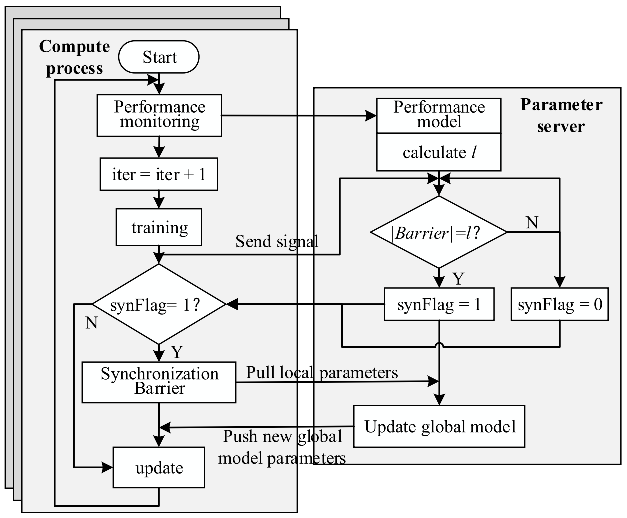

FIGURE 9. Flow chart of limited synchronous parallel model execution.

and their dynamic changes. In this paper, the performance monitoring module pulls the single iteration time (including computing time and communication time) of each compute process, then divides it based on the K-Means clustering algorithm (Algorithm 1), and takes the size of the best performance cluster as the limited threshold in the next iteration to cope with the dynamic changes of cluster performance.

The following describes the implementation of the LSP policy process, as shown in Fig. 9.

The improvement of the limited synchronous parallel model is embodied in the parameter server. The barrier implemented by LSP waits for the fastest l local parameter $u_{p,c}$ based on the principle of first come first served. The compute process that preempts the synchronization resources will enter the synchronization barrier, and other nodes will update locally.

The flow of each compute process is similar, so the flow of a compute process and a parameter server is taken as an example in the flowchart.

For a compute process, its single iteration is as follows:

1) At the beginning of this iteration, the performance monitoring process monitors the iteration time of the compute process and pushes it to the parameter server.

2) A synchronization barrier preemption signal is sent to the parameter server to determine whether the condition for entering the synchronization barrier is satisfied.
3) The compute process receives the synchronization signal *synFlag*. If *synFlag* is 0, each compute process will update the local model parameters using local gradients; if *synFlag* is 1, each compute process enters the synchronization barrier, sends a local gradient to the parameter server, and receives new global model parameters.

For the parameter server, to clearly show its role, the operation part of the parameter server multi-thread is simplified in the figure. The execution process of the parameter server is as follows:

1) The performance monitoring system pulls performance data of each compute process.
2) The limited threshold l is adaptively adjusted using the performance data calculation.
3) A synchronization barrier preemption signal is sent after the completion of the iteration of the compute process.
4) The parameter server determines whether the compute process is successful in preemption according to the state of the synchronization barrier. If the preemption fails and does not enter the synchronization barrier, the parameter server sets the signal *synFlag* to 0 and sends it to the compute process; if the condition of the synchronization barrier is reached, the parameter server sets the signal *synFlag* to 1 and sends it to each compute process.
5) The parameter server receives the local gradients sent by the compute processes after the synchronization barrier and, after aggregating the local gradients, calculates a new global model parameter and sends the global model parameters to the compute process entering the synchronization barrier.

B. THEORETICAL ANALYSIS

In machine learning program execution, the data set D is prestored on the computing cluster and is composed of $p = 1 \ldots P$ index. Let D_p be the p-th data partition and $A^{(t)}$ be the model of clock t (a clock represents some units of work in a machine learning algorithm and iteration in a common iteration-convergence algorithm). Data parallel computation executes the following update equation until some convergence conditions are satisfied:

$$A^{(t)} = F\left(A^{(t-1)}, \sum_{p=1}^{P} \Delta\left(A^{(t-1)}, D_p\right)\right) \tag{1}$$

where $\Delta()$ performs the calculation using the complete model $A^{(t-1)}$ on the data partition D_p. The next model $A^{(t)}$ is generated by the sum of the intermediate results produced by the $F\Delta()$ polymerization $\Delta()$ calculation and the current model $A^{(t-1)}$. For the stochastic gradient descent algorithm, $\Delta()$ calculates the gradient, updated to

$$A^{(t)} = A^{(t-1)} + \eta \sum_{p=1}^{P} \Delta\left(A^{(t-1)}, D_p\right) \tag{2}$$

where η is the step size. To implement machine learning calculations, all compute processes have a partial copy of model A, and the parameter server will aggregate the local parameters of the compute process. The global model A is stored on the server, which is distributed and therefore not subject to stand-alone memory. The compute process accesses the entire model state on the server through the messaging interface.

In this instance, the limited synchronous parallel model is formally expressed, assuming that the limited threshold is $l \in [1, P]$ and the global synchronization threshold is $m > 1$, that is, the number of synchronizations of each limited synchronization interval is m. Then, for a compute process p with a number of iterations c, it can access the noise model $\tilde{A}_{p,c}$ consisting of the initial model A_0 and the update $\boldsymbol{u}_{p,c}$. Then, the limited synchronous parallel model is as shown in (3).

$$\tilde{A}_{p,c} = A_0 + \left[\sum_{c'=1}^{c-m-1} \sum_{p'=1}^{l} u_{p',c'}\right] + \left[\sum_{\left(p',c'\right) \in \mathcal{U}_{p,c}} u_{p',c'}\right] \tag{3}$$

where $\mathcal{U}_{p,c}$ represents an updated subset of all compute processes during the $[c-m, c+m-1]$ iteration; $\left[\sum_{c'=1}^{c-m-1} \sum_{p'=1}^{l} u_{p',c'}\right]$ indicates the global update of the previous limited synchronization interval; $\left[\sum_{(p',c') \in U_{p,c}} u_{p',c'}\right]$ indicates the update of this limited synchronization interval.

Theorem 1(SGD Expectation Convergence Theorem Under the Limited Synchronous Parallel Model): For the convex function $f(A) = \frac{1}{T}\sum_{t=1}^{T} f_t(A)$, assuming that the component ∇f_t is also convex, the model minimum value A^* is now solved. There are P compute processes with a limited threshold of $l \in [1, P]$ and a global synchronization threshold of $m > 1$. Let $u_t := -\eta_t \nabla f_t(\tilde{A}_t)$, where $\eta_t = \frac{\sigma}{\sqrt{t}}$, $\sigma = \frac{F}{L\sqrt{2(m+1)l}}$, and L and F are constants. Assuming that f_t satisfies the Lipschitz continuous condition and the distance function $D(A \| A^*) \leq F^2$, then:

$$R[A] := \frac{1}{T}\sum_{t=1}^{T} f_t\left(\tilde{A}_t\right) - f\left(A^*\right) \leq 4FLP\sqrt{2(m+1)lT} \tag{4}$$

Equation (4) shows that the noise model $\tilde{A}_t$ using the limited synchronous parallel model converges to the real model A_t, i.e., the limited synchronous parallel model has a convergence guarantee in the training of distributed machine learning models. In addition, the limited threshold is dynamically adjusted based on cluster performance, ensuring the dynamics of machine learning fault tolerance.

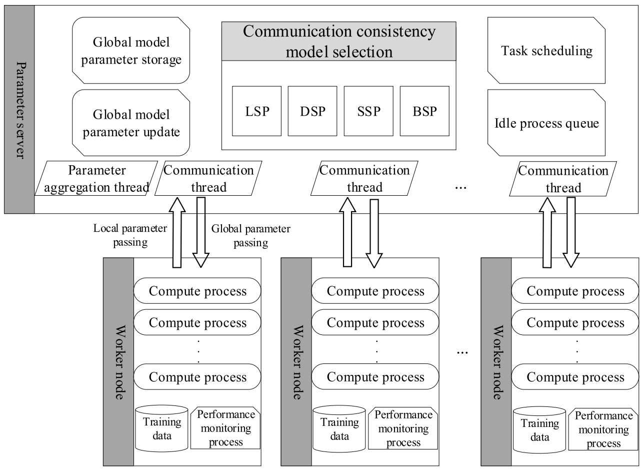

FIGURE 10. Distributed machine learning framework for security monitoring.

C. DISTRIBUTED MACHINE LEARNING FRAMEWORK FOR SECURITY MONITORING

This paper implements a distributed machine learning framework for security monitoring by using the idea of a parameter server, which supports multiple parameter communication consistency models. Including novel LSP and DSP and traditional BSP and SSP, and its structure is shown in Fig. 10.

The framework divides nodes into parameter servers and worker nodes. A parameter server includes a global model parameter storage module, a global model parameter update module, a task scheduling module, a compute process idle queue, a parameter aggregation thread, and a communication thread group. A worker node includes a compute process group, a performance monitoring process, a copy of the model parameters, and a subdata set. No communication is performed between the worker nodes.

The global model parameter storage module on the parameter server stores the latest global model parameters, namely, the initial global model parameters and the global model parameters for each round of global iteration updates.

The global model parameter update module on the parameter server is responsible for global model parameter updates and has different operations under different parameter communication consistency models. For example, in the bulk synchronous parallel model, the stale synchronous parallel model, and the dynamic synchronous parallel model, the global model parameter updating module waits for the parameter aggregation thread to aggregate the local parameters of all of the compute processes and then performs an update operation; in the limited synchronous parallel model, the global model parameter update module only needs to wait for the local parameters aggregated by the parameter aggregation thread to reach a limited threshold and then starts the update operation.

To achieve an asynchronous parallel compute process, the framework implements an idle process queue. After each process completes an iteration, it will be pushed into the idle process queue. The task scheduling module performs different scheduling operations on the idle process queue according to different communication consistency models with different parameters. In the limited asynchronous parallel model, dynamic asynchronous parallel model and stale asynchronous parallel model, the task scheduling module will be pushed into the idle process queue immediately and sequentially before the limited threshold, stale threshold and weak threshold are reached, and the next iteration task will be allocated; under the bulk synchronous parallel model, the task scheduling module waits for the global model parameter update module to perform the update operation before launching all the spatial compute processes and assigning the next iteration's calculation task.

There is a set of transmission threads on the parameter server for receiving local gradients transmitted by the worker nodes and transmitting the latest global model parameters to the worker nodes. To be able to receive and transmit in parallel, each worker node has a unique transport thread corresponding to it.

The data subset on the worker node is initialized before the model training, and the parameter server divides the data set evenly and distributes it to each worker node. The data subset is generally stored on the disk. Each iteration extracts part of the data from the middle batch for distributed machine learning training.

The performance monitoring process on the worker node is used to collect and send performance metrics for worker nodes. The compute process on the worker node trains the distributed machine learning model in a data-parallel manner and uses the global model parameters to calculate the local gradient through the gradient descent algorithm and the backpropagation algorithm. Each compute process performs an iteration independently.

IV. EXPERIMENT AND RESULT ANALYSIS

A. EXPERIMENTAL ENVIRONMENT SETTINGS

This article deploys the distributed machine learning framework introduced in Section 3.3 on a cluster of three servers, one of which is a parameter server, and the other two are worker nodes. All servers are homogeneous nodes. Each server is equipped with two E5-2630v4 processors with a frequency of 2.2GHz and a total of 20 logic cores. And two NVDIDA K80 accelerators are configured for each server. There are 256 GB of memory and a total of 500 GB of disk arrays. Each server is connected via Gigabit Ethernet.

This article uses the CIFAR-100 dataset, which contains 100 categories of image sets. The CIFAR-100 training set contains 50,000 examples and the test set contains 10,000 examples. The trained model uses the deep residual network ResNet [45].

B. LIMITATION OF FAULT TOLERANCE

To test the finiteness of the fault-tolerant characteristics of machine learning algorithms and verify the performance of

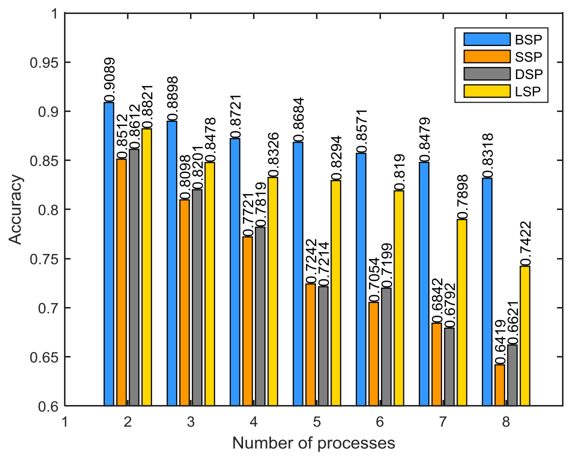

FIGURE 11. Comparison of the accuracy of the parameter communication consistency model.

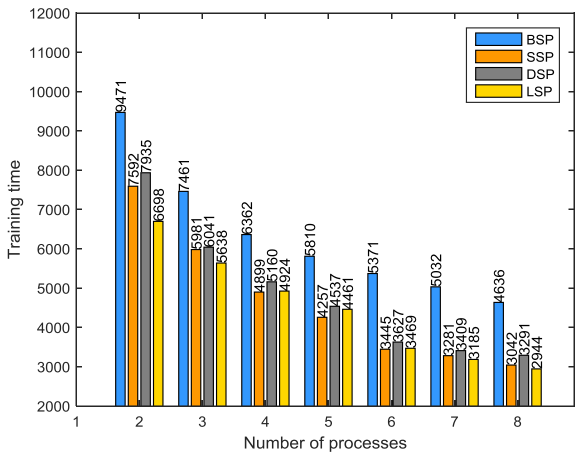

FIGURE 12. Comparison of training time of the parameter communication consistency model.

different parameter communication consistency models, this section presents statistics regarding the time and accuracy of training using machine learning models under the Limited Synchronous Parallel model (LSP), Bulk Synchronous Parallel model (BSP), Stale Synchronous Parallel model (SSP) and Dynamic Synchronous Parallel model (DSP). At the same time, to verify the scalable rows of each model, the experiment also counts the experimental results using different numbers of processes. The limited threshold of the LSP is preset to the number of computing processes; the stale threshold of the SSP and the DSP is preset to 3, and the weak threshold of the DSP is preset to 2. Figure 11 and Figure 12 show the accuracy and training time for training with different models.

It can be seen from Fig. 11 that regardless of what parameter communication consistency model is used for the distributed machine learning model training, since the total number of iteration tasks is constant, as the compute process increases, the iterative tasks shared by each compute process are reduced. The global synchronization of the parameter server is also relatively reduced, and the accuracy of the machine learning model trained using the different parameter communication consistency model is also reduced. As seen from Fig. 12, no matter what communication consistency model is adopted, the increase in the number of compute processes means that the parallel computing power is enhanced, and the training time is reduced accordingly. However, as the compute processes increase, the parameter server will communicate with more compute processes, and the parameter communication overhead will gradually be greater than the iterative computation overhead, resulting in no further reduction of the training time of the machine learning model.

Referring to Fig. 11 and Fig. 12, the parameter server under the BSP model waits for all compute processes to complete the iteration and then performs global synchronization in each iteration calculation. Therefore, BSP guarantees strong consistency of model training with the highest accuracy. However, the iterative calculation time of the BSP model depends on the slowest computation process, which results in considerable time being spent training with the BSP model. LSP, SSP and DSP all use the fault-tolerant characteristics of machine learning algorithms for limited asynchronous training, which shortens the synchronization waiting time of the parameter server; therefore, the training time is significantly lower than that of BSP. However, due to the abuse of the fault-tolerant feature of the SSP, in the case where the performance of each worker node is similar, the worker nodes each cause too many local iterations, which easily fall into the local optimal solution. As the compute processes increase, the number of local iterations also increases, and the number of global model parameter updates decreases, resulting in a continuous decrease in accuracy, and the model cannot meet the usage requirements. The DSP increases the communication overhead and sets the weak threshold, which reduces the number of local updates of the compute process, and it thus has higher accuracy than the SSP but also takes more training time. The LSP implements a limited synchronization barrier, which ensures the number of updates of the global model, prevents the global model from approaching the local optimal solution after local iteration, and thus has higher accuracy than SSP and DSP. At the same time, because the LSP reduces the communication overhead during global synchronization, the LSP also has a faster training speed.

This section of the experiment verifies the finiteness of the fault-tolerant nature of the iterative-convergence algorithm. When performing distributed machine learning model training on clusters with machines with similar performance, LSP is synchronized due to less synchronous communication overhead and low latency parameters., has (1) higher training accuracy than SSP, DSP; (2) faster training speed than BSP.

C. DYNAMICS OF FAULT TOLERANCE

Different from the experimental setup in Section 4.2, to simulate the real mixed load cluster environment, this section of

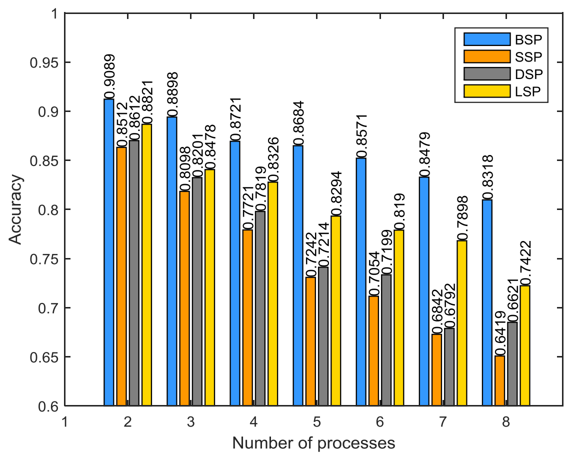

FIGURE 13. Scalability verification based on accuracy of the parameter communication consistency model.

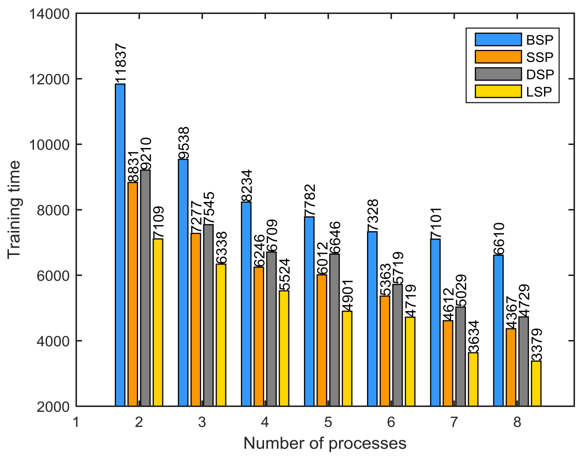

FIGURE 14. Scalability verification of training time based on the parameter communication consistency model.

the experiment adds random communication delay to one of the worker nodes during the execution of the model training task (the experimental environment of each model is the same). In this manner, the dynamics of the fault-tolerant characteristics of the machine learning iterative-convergence algorithm are verified, and the performance differences of the distributed machine learning model training of each parameter communication consistency model in the case of dynamic performance changes of the worker nodes are compared. Figures 13 and 14 show the accuracy and training time for training with different models, respectively.

It can be seen from Fig. 13 that because the BSP clearly separates the calculation phase and the communication phase, the accuracy of the machine learning model trained under the BSP model is not affected by the performance fluctuation of the worker node. The SSP and DSP can increase the synchronization condition faster as the node performance difference increases, which avoids the expansion of fault tolerance and improves the accuracy of the training model. However, the setting of the weak threshold in the DSP is therefore lost, and the accuracy is almost the same as that of the SSP. In addition, the LSP is reduced due to the difference in node performance, and the limited threshold is reduced, that is, the number of processes entering the limited synchronization barrier is reduced, so that the LSP still ensures a high accuracy and is higher than the SSP and the DSP.

It can be seen from Fig. 14 that because the single iteration time of the BSP is limited by the process with the longest iteration time, the BSP needs more time to iterate in the performance fluctuation of the worker node, which seriously affects the efficiency of the model training. In addition, because the LSP implements a limited synchronization barrier and realizes the asynchronous calculation of the worker node, the waiting problem caused by the performance lag process in the synchronization phase is weakened. Thus, LSP has significant advantages for the overall performance lag problem caused by the difference in performance of the worker node.

This part of the experiment verifies the dynamics of the fault-tolerant characteristics of the iterative-convergence algorithm. When the distributed machine learning model is trained under the mixed load of the cluster, the LSP can dynamically adjust the limited threshold and reduce the delay of the performance lag node to the model training. The impact causes LSP to have a significant advantage in computational efficiency.

V. CONCLUSION

Mobile computing takes full advantage of the computing power and communication capabilities of various sensing devices and uses these devices to form a computing cluster. When using such clusters for training of distributed machine learning models, the use of the bulk synchronous parallel model causes the machine learning model to have high accuracy. However, in the scene of large-scale security monitoring, because it is limited to the worst-performing nodes in the cluster, the overall computing performance of the cluster will be greatly wasted as the performance difference of the worker nodes increases. The stale synchronous parallel model (SSP) and the dynamic synchronous parallel model (DSP) utilize the fault-tolerant characteristics of the iterative-convergence algorithm, which realizes the asynchronous parallel computing of the worker nodes and improves the utilization of the computing cluster, but at the same time, the accuracy of the model training is greatly reduced due to the delay of the parameters.

This paper draws on the advantages of bulk synchronous parallel model (BSP) and asynchronous parallel model (ASP) and proposes a limited synchronous parallel model (LSP), which implements a limited synchronization barrier, which synchronizes a part of the worker process that is iteratively fast in each iteration, ensures frequent synchronization, and reduces communication overhead. In addition, LSP also implements finite asynchronous calculation of the worker process, which ensures the balance of computational efficiency and convergence speed. Finally, the experiments in this paper prove that the model is superior to the bulk synchronous parallel model, the stale synchronous parallel model and the dynamic synchronous parallel model in terms of computational efficiency and convergence speed.

Finally, in the finite experiment to verify the fault-tolerant characteristics of the machine learning iteration-convergence algorithm, the accuracy of the LSP is not as good as that of the BSP, but it is 11.12% higher than the SSP and 9.91% higher than the DSP. The training time of LSP is as good as the SSP and DSP and is 29.73% shorter than the BSP. In the dynamic experiment to verify the fault-tolerant characteristics of machine learning iterative-convergence algorithm, the accuracy of LSP performance is still less than that of BSP, and LSP is only improved by 7.83% and 5.09% compared with SSP and DSP because SSP and DSP tend to synchronize more times. The training efficiency of LSP has been improved, which is 16.89% and 22.35% shorter than SSP and DSP, and 39.53% shorter than BSP.

In the scenario of large-scale security monitoring, the bulk synchronous parallel model is more suitable for machine learning training tasks in offline scenarios, and the limited synchronous parallel model proposed in this paper is more suitable for scenarios with higher real-time requirements.

Although the distributed machine learning framework based on the limited synchronous parallel model proposed in this paper has certain advantages compared with other methods, it still has the potential to improve in terms of the convergence speed and accuracy of model training. Our future work will address these shortcomings to further improve the effectiveness of the limited parallel model.

APPENDIX: PROOF OF THEOREM 1

Theorem 1 (SGD Expectation Convergence Theorem Under the Limited Synchronous Parallel Model): For the convex function $f(A) = \frac{1}{T}\sum_{t=1}^{T} f_t(A)$, assuming that the component ∇f_t is also convex, the model minimum value A^* is now solved. There are P compute processes with a limited threshold of $l \in [1, P]$ and a global synchronization threshold of $m > 1$. Let $u_t := -\eta_t \nabla f_t(\tilde{A}_t)$, where $\eta_t = \frac{\sigma}{\sqrt{t}}, \sigma = \frac{F}{L\sqrt{2(m+1)l}}$, and L and F are constants. Assuming that f_t satisfies the Lipschitz continuous condition and the distance function $D(A \| A^*) \le F^2$, then:

$$R[A] := \frac{1}{T}\sum_{t=1}^{T} f_t\left(\tilde{A}_t\right) - f\left(A^*\right) \le 4FLP\sqrt{2(m+1)lT}$$

Proof:

$$\begin{aligned} R[A] :&= \frac{1}{T}\sum_{t=1}^{T} f_t\left(\tilde{A}_t\right) - f\left(A^*\right) \\ &\le \sum_{t=1}^{T}\langle \nabla f_t(\tilde{A}_t), \tilde{A}_t - A^*\rangle \\ &= \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{A}_t - A^*\rangle \end{aligned}$$

Lemma 2: Assume that model A is n-dimensional, i.e., $A = \mathbb{R}^n$, for all A^*:

$$\begin{aligned} \langle \tilde{g}_t, \tilde{A}_t - A^*\rangle &= \frac{1}{2}\eta_t \|\tilde{g}_t\|^2 + \frac{D(A^* \| A_t) - D(A^* \| A_{t+1})}{\eta_t} \\ &\quad + \left[\sum_{i\in\mathfrak{Q}_t} \eta_i \langle \tilde{g}_i, \tilde{g}_t\rangle - \sum_{i\in\mathfrak{R}_t} \eta_i \langle \tilde{g}_i, \tilde{g}_t\rangle\right] \end{aligned}$$

where $D(A \| A^*) = \frac{1}{2}\|A - A^*\|^2$, $\mathfrak{Q}_t$ represents the update set of $\tilde{A}_t$ compared to A_t, and $\mathfrak{R}_t$ represents the additional update set of $\tilde{A}_t$ compared to A_t. $|\mathfrak{Q}_t| + |\mathfrak{R}_t| \le 2m(P-1)$, $min(\mathfrak{Q}_t \cup \mathfrak{R}_t) \ge max(1, t-(m+1)l)$, $\max(\mathfrak{Q}_t \cup R_t) \le t+mP$.

Proof:

$$\begin{aligned} & D\left(A^* \| A_t\right) - D\left(A^* \| A_{t+1}\right) \\ &= \frac{1}{2}\left\|A^* - A_t\right\|^2 - \frac{1}{2}\left\|A^* - A_t + A_t - A_{t+1}\right\|^2 \\ &= \frac{1}{2}\left\|A^* - A_t\right\|^2 - \frac{1}{2}\left\|A^* - A_t + \eta_t\tilde{g}_t\right\|^2 \\ &= \eta_t\langle \tilde{g}_t, A_t - A^*\rangle - \frac{1}{2}\eta_t \|\tilde{g}_t\|^2 \\ &= \eta_t\langle \tilde{g}_t, \tilde{A}_t - A^*\rangle + \eta_t\langle \tilde{g}_t, A_t - \tilde{A}^*\rangle - \frac{1}{2}\eta_t \|\tilde{g}_t\|^2 \end{aligned}$$

where

$$\begin{aligned} \langle \tilde{g}_t, A_t - \tilde{A}^*\rangle &= \langle -\sum_{i\in\mathfrak{Q}_t} \eta_i\tilde{g}_t + \sum_{i\in\mathfrak{R}_t} \eta_i\tilde{g}_i, \tilde{g}_i\rangle \\ &= -\sum_{i\in\mathfrak{Q}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle + \sum_{i\in\mathfrak{R}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle \end{aligned}$$

Lemma 2 certification.

Then, for $R[A]$ where:

$$\begin{aligned} R[A] &\le \sum_{t=1}^{T}\langle \tilde{g}_t, \tilde{A}_t - A^*\rangle \\ &= \sum_{t=1}^{T}\frac{1}{2}\eta_t \|\tilde{g}_t\|^2 + \sum_{t=1}^{T}\frac{D(A^* \| A_t) - D(A^* \| A_{t+1})}{\eta_t} \\ &\quad + \sum_{t=1}^{T}\left[\sum_{i\in\mathfrak{Q}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle - \sum_{i\in\mathfrak{R}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle\right] \\ &= \sum_{t=1}^{T}\frac{1}{2}\eta_t \|\tilde{g}_t\|^2 + \frac{D(A^* \| A_1)}{\eta_1} - \frac{D(A^* \| A_{T+1})}{\eta_T} \\ &\quad + \sum_{t=2}^{T}\left[D\left(A^* \| A_t\right)\left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right)\right] \\ &\quad + \sum_{t=1}^{T}\left[\sum_{i\in\mathfrak{Q}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle - \sum_{i\in\mathfrak{R}_t} \eta_i\langle \tilde{g}_i, \tilde{g}_t\rangle\right] \end{aligned}$$

In this instance, is the boundary of each item of the above formula. The first item:

$$\sum_{t=1}^{T}\frac{1}{2}\eta_t \|\tilde{g}_t\|^2 \le \sum_{t=1}^{T}\frac{1}{2}\eta_t L^2$$

$$\begin{aligned}&= \sum_{t=1}^{T} \frac{1}{2}\frac{\sigma}{\sqrt{t}} L^2 \\ &\le \sigma L^2 \sqrt{T}\end{aligned}$$

The second section:

$$\begin{aligned}&\frac{D\left(A^*\| A_1\right)}{\eta_1} - \frac{D\left(A^*\| A_T\right)}{\eta_T} + \sum_{t=2}^{T}\left[D\left(A^*\| A_t\right)\left(\frac{1}{\eta_t}-\frac{1}{\eta_{t-1}}\right)\right] \\ &\le \frac{F^2}{\sigma} + 0 + \frac{F^2}{\sigma}\sum_{t=2}^{T}\left[\sqrt{t}-\sqrt{t-1}\right] \\ &= \frac{F^2}{\sigma} + \frac{F^2}{\sigma}\left[\sqrt{T}-1\right] \\ &= \frac{F^2}{\sigma}\sqrt{T}\end{aligned}$$

The third item:

$$\begin{aligned}&\sum_{t=1}^{T}\left[\sum_{i\in\mathfrak{Q}_t}\eta_i\langle\tilde{g}_i,\tilde{g}_t\rangle - \sum_{i\in\mathfrak{R}_t}\eta_i\langle\tilde{g}_i,\tilde{g}_t\rangle\right] \\ &\le \sum_{t=1}^{T}\left[|\mathfrak{Q}_t| + |\mathfrak{R}_t|\right]\eta_{\mathrm{MAX}(1,t-(m+1)l)}L^2 \\ &= L^2\left[\sum_{t=1}^{(m+1)l}\left[|\mathfrak{Q}_t| + |\mathfrak{R}_t|\right]\eta_1 \right.\\ &\quad\left. + \sum_{t=(m+1)l+1}^{T}\left[|\mathfrak{Q}_t| + |\mathfrak{R}_t|\right]\eta_{t-(m+1)l}\right] \\ &= L^2\left[\sum_{t=1}^{(m+1)l}\left[|\mathfrak{Q}_t| + |\mathfrak{R}_t|\right]\sigma \right.\\ &\quad\left. + \sum_{t=(m+1)l+1}^{T}\left[|\mathfrak{Q}_t| + |\mathfrak{R}_t|\right]\frac{\sigma}{\sqrt{t-(m+1)\,l}}\right] \\ &\le \sigma L^2\left[\sum_{t=1}^{(m+1)P} 2m\left(P-1\right)\sigma \right.\\ &\quad\left. + \sum_{t=(m+1)P+1}^{T} 2m\left(P-1\right)\frac{\sigma}{\sqrt{t-(m+1)\,l}}\right] \\ &\le 2\sigma L^2\left(P-1\right)\left[\left(m+1\right)P + 2\sqrt{T-\left(m+1\right)l}\right] \\ &\le 2\sigma L^2\left(P-1\right)\left[\left(m+1\right)P + 2\sqrt{T}\right] \\ &\le 2\sigma L^2\left[\left(m+1\right)P\right]^2 + 4\sigma L^2\left(m+1\right)P\sqrt{T}\end{aligned}$$

Then:

$$\begin{aligned}R\left[A\right] &\le \sum_{t=1}^{T}\langle\tilde{g}_t, \tilde{A}_t - A^*\rangle \\ &\le \sigma L^2\sqrt{T} + \frac{F^2}{\sigma}\sqrt{T} + 2\sigma L^2\left[\left(m+1\right)P\right]^2 \\ &\quad + 4\sigma L^2\left(m+1\right)P\sqrt{T}\end{aligned}$$

Let $\sigma = \frac{F}{L\sqrt{2\alpha}}$, where $\alpha = (m+1)l$, then:

$$\begin{aligned}R\left[A\right] &\le \frac{FL\sqrt{T}}{\sqrt{2\alpha}} + FL\sqrt{2\alpha T} + \frac{\sqrt{2}FLP^2\alpha^{3/2}}{l^2} + \frac{2\sqrt{2}FLP\sqrt{\alpha T}}{l} \\ &\le \frac{FL\sqrt{T}}{\sqrt{2\alpha}} + FL\sqrt{2\alpha T} + \sqrt{2}\,FLP^2\alpha^{3/2} + 2\sqrt{2}\,FLP\sqrt{\alpha T} \\ &= FL\sqrt{2\alpha T}\left(\frac{1}{2\alpha} + 1 + \frac{P^2\alpha}{\sqrt{T}} + 2P\right) \\ &\le FL\sqrt{2\alpha T}\left(\frac{1}{2\alpha} + \frac{P^2\alpha}{\sqrt{T}} + 3P\right)\end{aligned}$$

When $T \to \infty$, $\frac{1}{2\alpha} + \frac{P^2\alpha}{\sqrt{T}} \le 1$, then:

$$\begin{aligned}R\left[A\right] &\le 4FLP\sqrt{2\alpha T} \\ &= 4FLP\sqrt{2(m+1)lT}\end{aligned}$$

Theorem 1 is complete.

REFERENCES

[1] L. G. Valiant, “A bridging model for parallel computation,” *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.
[2] J. Dean and L. A. Barroso, “The tail at scale,” *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.
[3] S. Dutta, V. Cadambe, and P. Grover, “Short-Dot: Computing large linear transforms distributedly using coded short dot products,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2100–2108.
[4] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, “Large scale distributed deep networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.
[5] Q. Ho, J. Cipar, H. Cui, J. K. Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing, “More effective distributed ml via a stale synchronous parallel parameter server,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.
[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.
[7] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems,” Dec. 2015, *arXiv:1512.01274*. [Online]. Available: https://arxiv.org/abs/1512.01274
[8] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, “Petuum: A new platform for distributed machine learning on big data,” *IEEE Trans. Big Data*, vol. 1, no. 2, pp. 49–67, Jun. 2015.
[9] A. Coates, B. Huval, T. Wang, D. J. Wu, A. Y. Ng, and B. Catanzaro, “Deep learning with COTS HPC systems,” in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1337–1345.
[10] V. Smith, S. Forte, C. Ma, M. Takac, M. I. Jordan, and M. Jaggi, “CoCoA: A general framework for communication-efficient distributed optimization,” *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 8590–8638, 2018.
[11] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska, “MLI: An API for distributed machine learning,” in *Proc. IEEE 13th Int. Conf. Data Mining (ICDM)*, Dec. 2013, pp. 1187–1192.
[12] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A MATLAB-like environment for machine learning,” in *Proc. NIPS Workshop*, 2011, pp. 1–6.
[13] Y. Huang, T. Jin, Y. Wu, Z. Cai, X. Yan, F. Yang, J. Li, Y. Guo, and J. Cheng, “FlexPS: Flexible parallelism control in parameter server architecture,” *Proc. VLDB Endowment*, vol. 11, no. 5, pp. 566–579, 2018.
[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, “TensorFlow: A system for large-scale machine learning,” in *Proc. OSDI*, vol. 16, 2016, pp. 265–283.
[15] F. Seide and A. Agarwal, “CNTK: Microsoft’s open-source deep-learning toolkit,” in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, p. 2135.

[16] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. B. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar, "MLlib: Machine learning in apache spark," *The J. Mach. Learn. Res.*, vol. 17, no. 34, pp. 1–7, 2016.

[17] X. Liu, M. Zhao, A. Liu, and K. K. L. Wong, "Adjusting forwarder nodes and duty cycle using packet aggregation routing for body sensor networks," *Inf. Fusion*, vol. 53, pp. 183–195, Jan. 2020.

[18] H. Gao, W. Huang, and X. Yang, "Applying probabilistic model checking to path planning in an intelligent transportation system using mobility trajectories and their statistical data," *Intell. Automat. Soft Comput.*, vol. 25, no. 3, pp. 547–559, 2019.

[19] H. Gao, W. Huang, Y. Duan, X. Yang, and Q. Zou, "Research on cost-driven services composition in an uncertain environment," *J. Internet Technol.*, vol. 20, no. 3, pp. 755–769, 2019.

[20] H. Gao, H. Miao, L. Liu, J. Kai, and K. Zhao, "Automated quantitative verification for service-based system design: A visualization transform tool perspective," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 10, pp. 1369–1397, 2018.

[21] Y. Yin, J. Xia, Y. Li, Y. Xu, W. Xu, and L. Yu, "Group-wise itinerary planning in temporary mobile social network," *IEEE Access*, vol. 7, pp. 83682–83693, 2019.

[22] Y. Yin, L. Chen, Y. Xu, J. Wan, H. Zhang, and Z. Mai, "QoS prediction for service recommendation with deep feature learning in edge computing environment," in *Mobile Networks and Applications*. New York, NY, USA: Springer, 2019, doi: 10.1007/s11036-019-01241-7.

[23] Y. Yin, W. Zhang, Y. Xu, H. Zhang, Z. Mai, and L. Yu, "QoS prediction for mobile edge service recommendation with auto-encoder," *IEEE Access*, vol. 7, pp. 62312–62324, 2019.

[24] S. Pang, H. Chen, H. Liu, J. Yao, and M. Wang, "A deadlock resolution strategy based on spiking neural P systems," *J. Ambient Intell. Hum. Comput.*, pp. 1–12, 2019, doi: 10.1007/s12652-019-01223-3.

[25] T. Song, S. Pang, S. Hao, A. Rodríguez-Patón, and P. Zheng, "A parallel image skeletonizing method using spiking neural P systems with weights," *Neural Process. Lett.*, vol. 50, pp. 1485–1502, Oct. 2018.

[26] J. Zhou, X. Li, P. Zhao, C. Chen, L. Li, X. Yang, Q. Cui, J. Yu, X. Chen, Y. Ding, and Y. A. Qi, "KunPeng: Parameter server based distributed learning systems and its applications in Alibaba and ant financial," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2017, pp. 1693–1702.

[27] K. Yu, "Large-scale deep learning at Baidu," in *Proc. 22nd ACM Int. Conf. Inf. Knowl. Manage.*, Oct./Nov. 2013, pp. 2211–2212.

[28] J. Jiang, L. Yu, J. Jiang, Y. Liu, and B. Cui, "Angel: A new large-scale machine learning system," *Nat. Sci. Rev.*, vol. 5, no. 2, pp. 216–236, 2018.

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

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

[31] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. OSDI*, vol. 14, Oct. 2014, pp. 583–598.

[32] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: Agile ML elasticity through tiered reliability in dynamic resource markets," in *Proc. 12th Eur. Conf. Comput. Syst.*, Apr. 2017, pp. 589–604.

[33] 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*, Feb. 2016, pp. 377–386.

[34] E. Zhong, Y. Shi, N. Liu, and S. Rajan, "Scaling factorization machines with parameter server," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2016, pp. 1583–1592.

[35] 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*, Apr. 2017, pp. 879–880.

[36] L. Yut, C. Zhang, Y. Shao, and B. Cui, "LDA*: A robust and large-scale topic modeling system," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1406–1417, 2017.

[37] 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.*, Apr. 2016, Art. no. 4.

[38] 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. AAAI*, Jan. 2015, pp. 79–87.

[39] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. OSDI*, vol. 14, 2014, pp. 571–582.

[40] 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.

[41] J. Zhang, H. Tu, Y. Ren, J. Wan, L. Zhou, M. Li, J. Wang, L. Yu, C. Zhao, and L. Zhang, "A parameter communication optimization strategy for distributed machine learning in sensors," *Sensors*, vol. 17, no. 10, p. 2172, 2017.

[42] J. Zhang, H. Tu, Y. Ren, J. Wan, L. Zhou, M. Li, and J. Wang, "An adaptive synchronous parallel strategy for distributed machine learning," *IEEE Access*, vol. 6, pp. 19222–19230, 2018.

[43] E. B. Khunayn, S. Karunasekera, H. Xie, and K. Ramamohanarao, "Straggler mitigation for distributed behavioral simulation," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 2638–2641.

[44] A. Harlap, H. Cui, W. Dai, J. Wei, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Addressing the straggler problem for iterative convergent parallel ML," in *Proc. 7th ACM Symp. Cloud Comput.*, Oct. 2016, pp. 98–111.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

RUI YANG 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.

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

JING SHEN is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Hangzhou Dianzi University. Her areas of research interest include high performance computing and system reliability.

YUNCHEN ZHANG 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.

ZHENGUO WEI graduated in business administration from the Zhejiang University of Technology. In 2000, he was with Dawning Information Technology Company, Ltd. In 2018, he was the General Manager with Zhejiang Dawning Information Technology Company, Ltd. He was fully responsible for the business of the enterprise and responsible for the research and development and management of software products related to high-performance computing, big data, and artificial intelligence.

JUNCONG ZHANG graduated in computer science and technology from Xiangfan University. In 2018, he was the Manager of the Technology Center, Zhejiang Dawning Information Technology Company, Ltd. He was responsible for the research and development of software products and technical support related to high performance computing, big data, and artificial intelligence.

JUE WANG is currently an Associate Professor with the Supercomputing Center, Chinese Academy of Sciences. 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-core clusters and GPUs.