The Journal of Supercomputing (2022) 78:15663–15680
https://doi.org/10.1007/s11227-022-04466-8

# EP4DDL: addressing straggler problem in heterogeneous distributed deep learning

**Zeyu Ji[1] · Xingjun Zhang[1] · Jingbo Li[1] · Jia Wei[1] · Zheng Wei[1]**

Accepted: 17 March 2022 / Published online: 21 April 2022


## Abstract

Driven by big data, neural networks evolve more complex and the computing capacity of a single machine is often difficult to meet the demand. Distributed deep learning technology has shown great performance superiority for handling this problem. However, a serious issue in this field is the existence of stragglers, which significantly restricts the performance of the whole system. It is an enormous challenge to fully exploit the computing capacity of the system based on parameter server architecture, especially in a heterogeneous environment. Motivated by this, we designed a method named EP4DDL to minimize the impact of the straggler problem by load balance technique. In a statistical view, the approach introduces a novel metric named performance variance to give a comprehensive inspection of stragglers and employs flexible parallelism techniques for each node. We verify the algorithm on standard benchmarks and demonstrate that it can reduce training time to 57.46%, 24.8%, and 11.5%, respectively, without accuracy loss compared with the FlexRR, Con-SGD, and Falcon.

**Keywords** Distributed system · Heterogeneous environment · Stragglers · Deep learning · Flexible parallelism

Jingbo Li, Jia Wei and Zheng Wei have contributed equally to this work.

✉ Xingjun Zhang
xjzhang@xjtu.edu.cn

Zeyu Ji
zeyu.ji@stu.xjtu.edu.cn

Jingbo Li
lijingbo17@stu.xjtu.edu.cn

Jia Wei
weijia4473@stu.xjtu.edu.cn

Zheng Wei
frank.wei@stu.xjtu.edu.cn

[1] School of Computer Science and Technology, Xi'an Jiaotong University, West Xianning Road, Xi'an 710049, Shaanxi, China

## 1 Introduction

The diffusion of deep learning, which is based on statistics, has been made possible by an exponential growth in data, generally driven by the information explosion. The deep learning approaches have demonstrated impressive performance in various applications including facial recognition [1], natural language processing [2], computer game [3], real-time resource allocation [4]. To maximize the performance, the deep learning model often involves large data sets and increasingly complex networks, leading to the storage and computation resource demand far exceeding a single machine's capacity. Distributed deep learning technology delivers a prevalent scale-out solution by leveraging a cluster of machines to finish large-scale tasks under an acceptable time cost. The parameter server [5–7] is a general parallel framework supporting distributed deep learning, which focuses on data-parallel manner. However, due to straggler problems, iterative convergence often causes significant slowdowns.

The straggler problem [8, 9], which is an unforeseen phenomenon, will block the parallel running of the deep learning model and have a negative impact on program performance. The presence of stragglers in a distributed system is a serious problem that restricts the performance of the whole system. In particular, in heterogeneous environments [10, 11], the different performance of compute nodes and varying operating environments can lead to more serious straggler problems. It is an enormous challenge to fully exploit the computing capacity of the system.

Many techniques are proposed to mitigate the impact of stragglers. The method based on redundant job execution [12–14] is first proposed, which relies on the idempotent of the execution jobs in BSP model. However, they are inefficient in ASP and SSP models, which change the idempotent of shared state. Subsequently [9], proposes a dynamic task assignment approach. The main idea is that the fast workers can help the slowed workers to catch up [5]. Introduces new hyper-parameters to adjust the contribution of the gradient which is from each node to solve the straggler problem [7, 15, 16]. Use the idea of elastic parallelism to solve the straggler problem. Yet, the methods above do not fully exploit the computation resource in a heterogeneous environment where the stragglers are serious and hard to be detected. The dynamic task assignment approach [9] generates unavoidable data transfer overhead, the hyper-parameter approach [5] does not essentially solve the straggler problem, and the clustering approach used in falcon [7]'s elastic parallelism scheme groups compute nodes, eliminating the intra-group straggler problem, but the inter-group straggler problem still exists and has a tendency to expand.

In order to fully utilize the computational resources of nodes in a heterogeneous environment, we introduce a novel metric named performance variance to give a comprehensive inspection of stragglers and propose an approach—EP4DDL, which implements load balancing by searching for the optimal parallelism of each node to solve the performance degradation triggered by stragglers. Compared with flexRR [9], EP4DDL uses an elastic parallel strategy instead of dynamic task assignment to achieve load balancing and avoid additional

communication overhead. In contrast to falcon [7], EP4DDL considers all nodes as a group and treats the performance variance as a metric to limit the gap between the fastest and slowest nodes.

This method is a special form of load balancing optimized for data-parallel iterative deep learning:(1) it uses a customized gated recurrent units(GRUs) model based on time series to predict the performance of each node in the system instantaneously through real-time hardware parameters; (2) it uses multi-threading techniques to elastically adjust the computational resource of the straggler nodes; (3)on the top of the prediction model and an elastic parallel system, EP4DDL provides an optimized parallelism strategy for the straggler nodes. Overall, the approach can reduce the system heterogeneity and alleviate the impact of stragglers.

We evaluated the method EP4DDL in a cluster of 10 nodes with stragglers. We used 2 mainstream image classification models Alexnet [17] and VGG19 [18] on the data-sets MNIST [19] and Cifar-10 [20] as benchmarks. The experimental results show that the EP4DDL method achieves good performance in terms of training convergence speed, training accuracy and other metrics with little overhead. It also shows excellent performance in benchmark tests compared to previous solutions. Our main contributions can be summarized as follows:

1. We analyze and quantify the straggler problem from a statistical perspective, and propose to solve the straggler problem from two directions: performance expectation and performance variance;
2. We design a customized P-GRUs based on time series to predict the performance of each node in the system instantaneously through real-time hardware parameters;
3. We flexibly adjust the computational resources of the straggler nodes based on the above prediction model results and adopt multi-threading techniques to provide an optimized parallel strategy for the computation nodes;
4. We validated the EP4DDL method on the data-sets MNIST and CIFAR-10 with two general image classification models, Alexnet and VGG19.

The rest of the paper is organized as follows. Section 2 presents background and relevant theory, Sect. 3 describes design details and algorithm implementation, Sect. 4 shows experimental results and analysis, Sect. 5 introduces the related works and the last part concludes the paper.

## 2 Background

### 2.1 Parallel model and consistency protocol

In order to migrate the deep neural network model to a distributed computing platform, researchers propose the corresponding parallel models. There are three types of popular parallel models for deep neural networks: model parallelism [21], data parallelism [22] and hybrid parallelism [23, 24].

Model parallelism involves different hardware (CPU/GPU) in a distributed system being assigned to different parts of the neural network model, as shown in Fig. 1, where different network layers in the neural network model are assigned to different hardware for execution, or different parameters within the same layer are assigned to different hardware. Data parallelism means that each node in the distributed system has a copy of the same model, as shown in Fig. 2, where each node is assigned different training data, and then the parameter server merges the computation results of all machines in a certain way, and finally completes the parameter exchange. Hybrid parallelism is the trade-off between model parallelism and data parallelism. The network overhead in model parallelism starts to dominate as the number of devices increases compare with data parallelism. Therefore, most of the current researchers focus on data parallelism or hybrid parallelism.

Bulk synchronous parallel(BSP) is a typical method for parameter synchronization in distributed deep learning, which guarantees consistent results in both the stand-alone and distributed cases [25]. With this method, the host simply divides and maps the data to the nodes. By using the consistent model, it ensures that all nodes use the same parameters, but is vulnerable to stragglers. This incurs a substantial overhead on the overall system, which hinders training scaling. To solve the problem, researchers relax the synchronization restriction, creating an inconsistent model called asynchronous parallel (ASP) model [26, 27], which overcomes the drawbacks by eliminating the displayed fence between nodes. A well-known

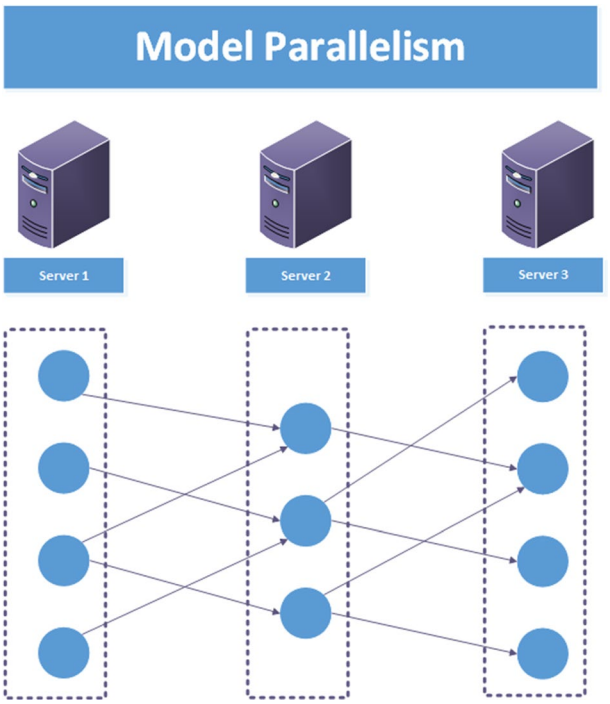


**Fig. 1** Model parallelism

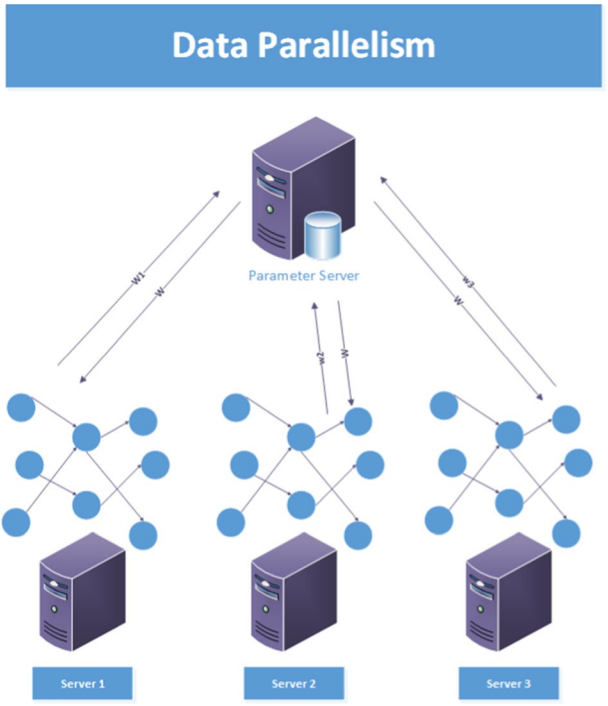


**Fig. 2** Data parallelism

instance of inconsistent SGD is the HOGWILD shared-memory algorithm [28], which allows training agents to read parameters and update gradients at will, overwriting existing progress. To alleviate the interference effect of updating $w$ at each step in the distributed system, the method transfers the gradient $\Delta w$ from the training process instead of $w$. However, this asynchronous behavior inevitably introduces new troubles called "gradient staleness" [29, 30] to the whole system. Therefore, the model with ASP tends to be much less effective than the BSP, and severe gradient staleness can significantly slow down the speed of convergence. To provide correctness guarantees in spite of asynchrony, Stale-Synchronous Parallelism (SSP) [6] proposes a compromise between consistent and inconsistent models. In SSP, a global synchronization step is introduced after a maximal staleness may have been reached by one of the nodes in order to bound the gradient staleness enforcedly. This approach works especially well in heterogeneous environments, where stragglers are kept in check. The specific relationship is shown in Fig. 3.

## 2.2 Straggler problem

Parameter servers provide a reliable solution for parallel training of deep learning in a distributed environment. Many parameter server-based studies make tradeoffs between its scale and convergence accuracy. But there is a hidden assumption

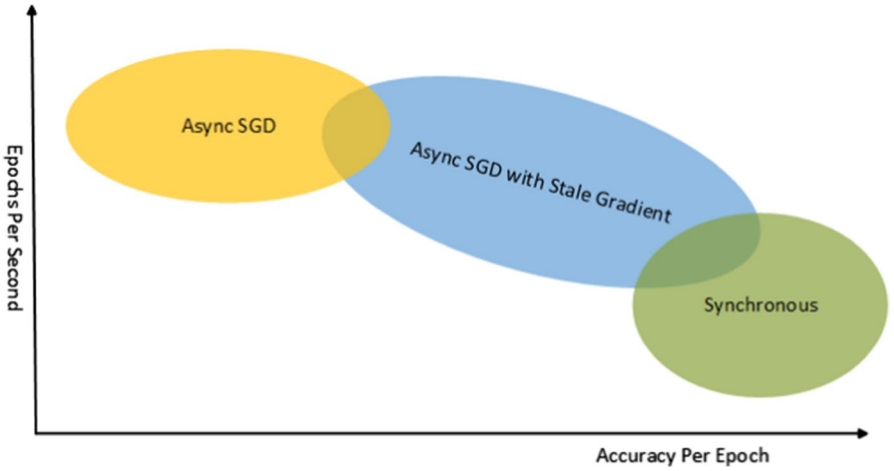


**Fig. 3** The relationship between BSP, ASP, and SSP

here: a homogeneously distributed environment, i.e., all nodes are in a similar computational and network environment. Therefore, the above studies are difficult to achieve the expected results in an environment with stragglers. A straggler is defined as a task which executes abnormally slow in comparison with the average task duration within a job. It brings the impact of a long-tail problem [8] in distributed systems.

To demonstrate how the straggler degrades the performance of the distributed deep learning system, we use a parameter server system containing four workers and a server with the same workload in the experimental section. As shown in Fig. 4, the training convergence time with stragglers is much longer than that without stragglers. For example, on the cifar10 data-set, the training time for the Alexnet model increases by 2.02 times, and the training time for the VGG model increases by 2 times. In conclusion, addressing the performance degradation caused by stragglers is a very critical issue in distributed deep learning. A number of previous efforts have been made to address the problem of stragglers, which overall can be categorized into 3 classes as follows:

1. *Solving the stragglers* [12, 13, 25] the main idea is redundant execution, where tasks assigned to stragglers are executed simultaneously on multiple nodes. The ensuing cost is that a huge amount of computation resources are wasted;
2. *Tolerating the stragglers* Accept the existence of stragglers and aim to eliminate their effects. By relaxing the strict of synchronization, [6, 26, 27] eliminate the impact of stragglers but introduce problems such as gradient staleness. As a further step, [5, 29, 30] introduce a new parameter to assign the gradient contribution of computational nodes;
3. *Improving the stragglers* [7, 15, 31] mitigates the impact of stragglers by the idea of task partitioning and load balancing.

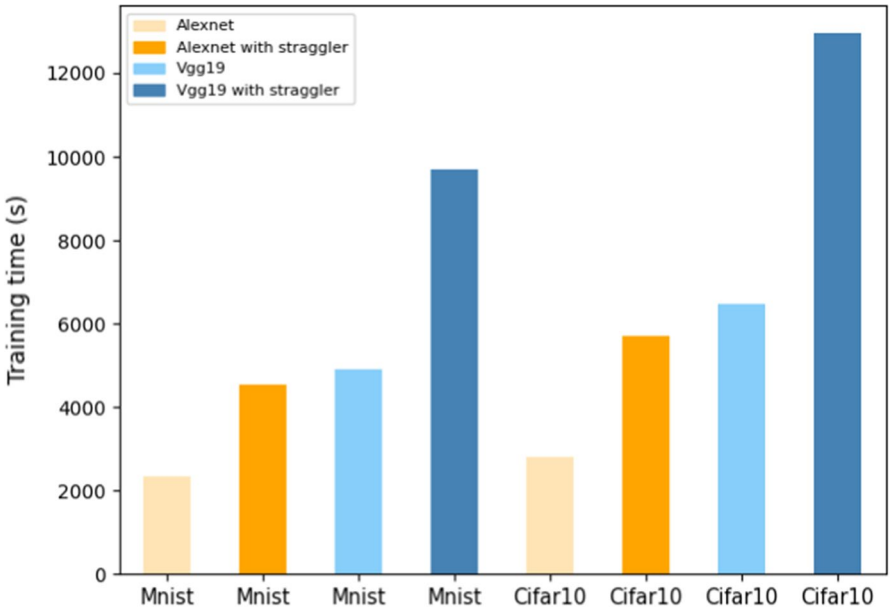


**Fig. 4** Stragglers degrade the system performance of training time

The first type of solution is based BSP approach, using redundant computation strategies. Cause of the bulk synchronous method, these types of solutions are inefficient in heterogeneous environments. To solve the straggler problem, the second type of solution introduces the stale gradient which results to slow convergence of the DNN model. Therefore, most recent research has concentrated on the third type of solution. The research [7] proposes a system called falcon which uses a time-based recurrent neural network for performance prediction of computational nodes. On top of the performance prediction results, falcon clusters the nodes with a density-based spatial algorithm and achieves intra-group performance balancing by using an elastic parallelism strategy. However, with this approach the performance gap between groups becomes insurmountable; In the research [15], FlexSlot adaptive changes the number of slots on each virtual node to promote the effective utilization of the resource pool; In research [31], firstly, a new model-based data partitioning algorithm is proposed based on extensive tests on the performance of compute nodes under different task volumes, which minimizes the execution time of computations in the parallel execution of the application. However, the algorithm cannot partition the data in real time and the computational overhead of the algorithm is unacceptable; [9] dynamically evaluates the performance of nodes and divides the nodes into help and helpee groups. The nodes in the help group can assist the nodes in the helpee group after completing their own tasks. But with this strategy, the communication overhead between nodes is huge.

It is obvious that the idea of load balancing is more applicable to distributed deep learning models in heterogeneous environments rather than the former two solutions. Therefore, our research is also based on it.

## 3 Design and implementation

### 3.1 Problem analysis

The quantitative and technical demands of distributed deep learning applications lead to an increase in system scale and system complexity for a single cluster that contains several servers. However, the increased complexity leads to a series of phenomena which are unexpected at the beginning of the system design. One of these phenomena is that some of the working nodes compute at a much lower speed compared to the normal nodes, which we call the stragglers. To better investigate how the stragglers are throughout the training cycle in a distributed system, we observe the straggler problem from a statistical perspective. We introduce the expectation to represent the overall performance of the system, then the variance can describe the dispersion of the system performance, which is a good fit with the straggler phenomenon. The above analysis shows that the performance of the slow nodes needs to be brought closer to the average performance of the whole system to better solve the straggler problem. Inspired by the theory of [7, 15], we adopt the idea of load balancing based on elastic parallelism.

First, we experimentally tested the status of CPU utilization during the training process of deep learning. It can be observed from Fig. 5 that the CPU utilization is not fully utilized. For training Alexnet and VGG19 on the MNIST data-set, the CPU utilization is only 33.4% and 40.7%. A large portion of the CPU resource is idle and the CIFAR 10 data-set has similar results. These results indicate that computing resources are not exploited. Then elastic controlling the parallelism for each worker can be a good solution to the straggler problem caused by the imbalance of computing capacity between workers. Here, the concept of parallelism is defined as the number of basic training units on a node. For example, In a CPU-based platform, we apply multi-threading techniques to ensure that a worker

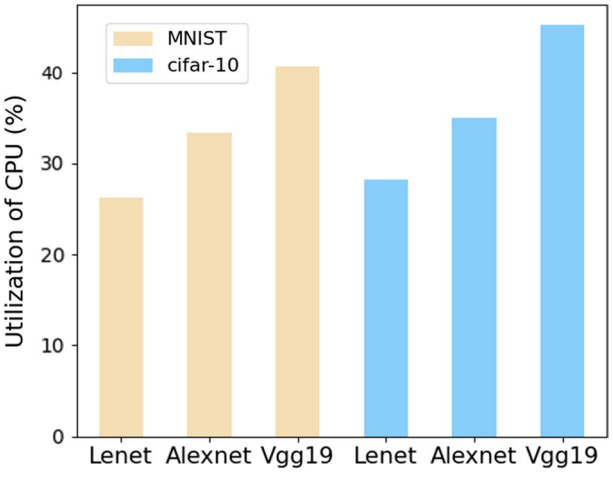


**Fig. 5** Average utilization of CPU computation capacity

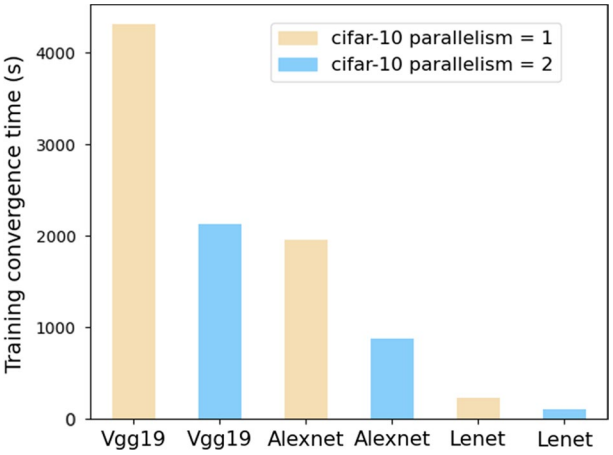


**Fig. 6** Training time on Cifar 10 with different threads

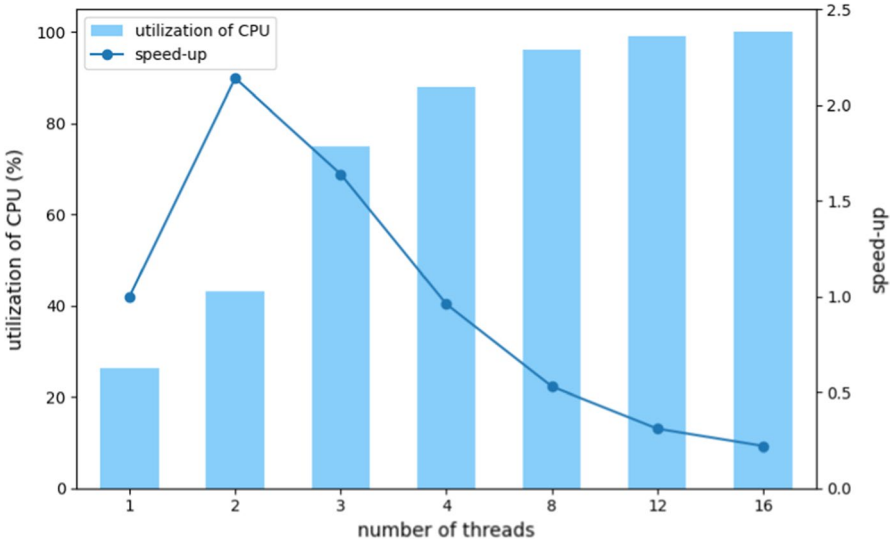


**Fig. 7** The utilization of CPU and speedup with different threads

can run multiple workloads simultaneously. As shown in Fig. 6, by using multiple threads, the model Alexnet is trained 2.2 times faster on the CIFAR 10 data-set, showing great potential. In Fig. 7, we can observe that the performance curve of the hardware shows an upward convex shape, and the optimal performance still requires a reasonable parallelism selection.

The above experiments and analysis show the feasibility of load balancing by adjusting multiple parallelisms to solve the straggler problem. We will then use algorithms to ensure how to assign reasonable parallelism to each node.

### 3.2 Algorithm description

#### 3.2.1 Predict model

In particular, for clusters with shared resources, the computational capacity of a node varies with run-time. Consequently, a prerequisite for solving the load balancing problem is to obtain real-time performance of each worker. The performance of nodes may be affected by the available resources including CPU state and memory resources. Therefore, we introduce a gated recurrent units(GRUs) [32], an extended recurrent neural network that makes performance predictions on each node by using a set of feature vectors $v^e_{<i>}$CPU, memory, parallelism, time per epoch describing the computing nodes as input. To achieve this goal, we elaborate an online time-series-based long short-term memory model, which is inspired by [7, 33].

As shown in Algorithm 1, the input is the real-time state of the node (CPU, memory, time per epoch and parallelism). At first, the algorithm is to initiate the model with pre-trained parameters. Then it normalizes the input data, predicts the result with the model and anti-normalizes the output. Next the predicted result is used to calculate the current speed of the node. Finally, the algorithm inserts the input data into the history data and updates the parameters $w$.

This GRUs-based prediction model can provide a mapping function between parallelism and performance for each node with a small overhead. Then we need to minimize the variance of the performance of all nodes through the results provided by the above function, while also ensuring that the overall performance of the nodes remains in a high-performance interval.

#### 3.2.2 Formulation of performance optimization problem

Recall the analysis in Sect. 3.1, we should search the corresponding parallelism $p^e_{<i>}$ for each worker, which is constrained by the following limits:

$$p^e_{<i>} = \max\left(\min_{i=0,e=1}^{i,e}\left(\frac{\text{batch size}}{t^e_{<i>}}\right)\right) \tag{1}$$

If we simply solve for the combination with the smallest variance in node performance, the overall performance of the nodes will likely fall into a lower interval despite the small performance gap in each node, which will fail to meet our original intention of using distributed techniques. If all nodes use the best performance parallelism, i.e., only solving Eq. 2

$$p^e = \max_{e=1}^{e}\left(\frac{\text{batch size}}{t^e}\right) \tag{2}$$

which makes the nodes run the fastest, however, the performance gap between nodes is still not solved. The performance gap leads to the waste of resources on fast computing nodes with BSP. And on the case of ASP or SSP, the performance gap between nodes causes the problem of unstable gradients of staleness [29, 30]. Therefore, firstly, the method should find the lower bound $s^e_{<i>}$ among all nodes with

Eq. 1. Then it minimizes the performance variance of the nodes by using the threshold $s^e_{<i>}$ as a constraint. As a result, the method maintains the performance of each node in a high interval.

According to the above description, the problem of minimizing the variance can be represented as an integer nonlinear programming problem. The input to the performance model is a discrete sequence. To achieve optimal parallelism, we adopt the idea of branch and bound. As shown in Fig. 8, for example, there are four workers in a distributed system. On each node, a task can be executed by 1 thread, 2 threads, 3 threads, or 4 threads (parallelism represents the number of threads when a task is running on a worker). Then the Solution Tree is constructed from the root, which has four child nodes that represent four different statuses of parallelism in turn, and the values on the edges represent the performance values of the node with current parallelism. When the tree reaches the leaf node at the fourth level, the value of the fourth level leaf node is the performance variance value of the four nodes under that path. If a path is not explored to the fourth level, it means that the path does not meet the conditions and has no solution. Finally, we search all the paths and return the path with the smallest performance variance, which is the optimal parallelism solution for the current system node.

**Algorithm 1** Performance-aware Gate Recurrent Units

**Input: Workers status** $v^e_{<i>}$

**Output: Estimated speed of workers (Pictures/sec)**

**procedure** P-Gate Recurrent Units($v^e_{<i>}$)
    Initiate neural network with pre-trained $w$
    Normalize each input element $v^e_{<i>}$
    Calculate output $t^e_{<i>}$
    Anti-normalize $t^e_{<i>}$
    insert $v^e_{<i>}$ into history data and update $w$
    $s^e_{<i>} \leftarrow \frac{batch\ size}{t^e_{<i>}}$
    **return** $s^e_{<i>}$
**end procedure**

## 4 Experiment

We will test the EP4DDL method from three aspects:

1. How accurate is the performance prediction model? Experiments show that the accuracy of the prediction model for the performance of nodes in a CPU cluster reaches more than 89.3%. Therefore, the model can provide good support for the parallelism control algorithm and thus alleviate the straggler problem;

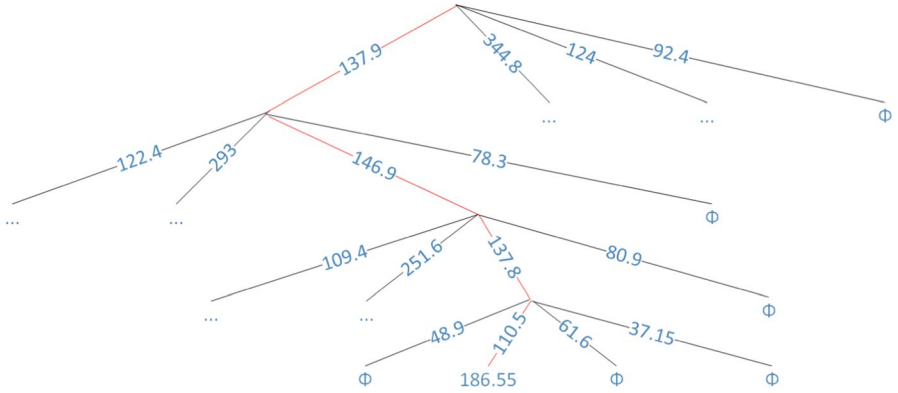


**Fig. 8** Examine all combinations with threshold and search an optimal parallelism solution

2. What is the computational overhead of the parallelism control algorithm? Experimental tests on two models with two data sets show that the overhead of the parallelism control model does not exceed 3% of the overall overhead;
3. What is the final performance of the EP4DDL method? We benchmark the EP4DDL method on three models with two data sets, and the results demonstrate the EP4DDL method has stable convergence efficiency and good scalability. It is also compared with three related methods(FlexRR, Con-SGD and falcon), and can reduce the training convergence time by 57.46%, 24.8%, and 11.5%, respectively.

## 4.1 Experiment setting

### 4.1.1 Experimental environment

We build a 10-node parameter server consisting of 9 compute nodes and 1 server node. Each node is a 12-core CPU (4 of them are configured with i7-8700 and 16G RAM, the remaining 6 nodes are configured with i5-10500 and 4G RAM). All the machines, which are interconnected using gigabit routing, work with CentOS Linux release 7.8.2003 and Docker containers. We can observe that this cluster is naturally heterogeneous.

### 4.1.2 Workload

We train the image classification models Lenet, Alexnet, and VGG19 on two data sets MNIST and CIFAR-10 as the benchmark test. Empirically, the batch size in the model parameters is set to 256, the initial learning rate is set to 0.01, and the number of epochs for model training is set to 50.

#### 4.1.3 Baseline

We organize 7 baselines, which can be divided into two groups. (1)BSP-based baseline: BSP on PyTorch, and sync-opt [25] (2)ASP(including SSP)-based baseline: ASP [26] on PyTorch, SSP [6] on PyTorch, FlexRR, ConSGD [5] and falcon [7]. To simulate the occurrence of stragglers in our experiments, we manually injected stragglers into the task.

### 4.2 Predict model

We verified the prediction accuracy of P-GRUs by comparing the predicted time for epoch with its real completion time. As shown in Fig. 9, P-GRUs provides speed prediction with large errors at the beginning. Yet, as the training data grows, the prediction accuracy gradually improves, eventually reaching an average accuracy of 89.3%.

### 4.3 Overhead

In order to alleviate the straggler time delay by adjusting the worker's iteration speed, we need to dynamically execute the EP4DDL method to deliver optional parallelism for each worker. As shown in Fig. 10, the execution frequency of the algorithm will have a different impact on the accuracy and overhead. We also observe that when the EP4DDL method is executed once per epoch, the prediction accuracy is low (61.2%), and the time overhead is small. When the frequency is increased

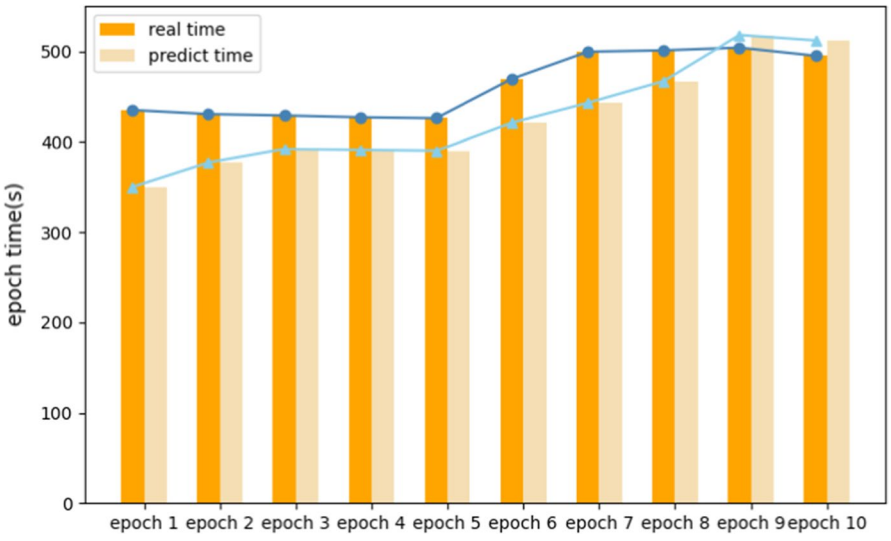


**Fig. 9** Predict epoch time of P-GRUs

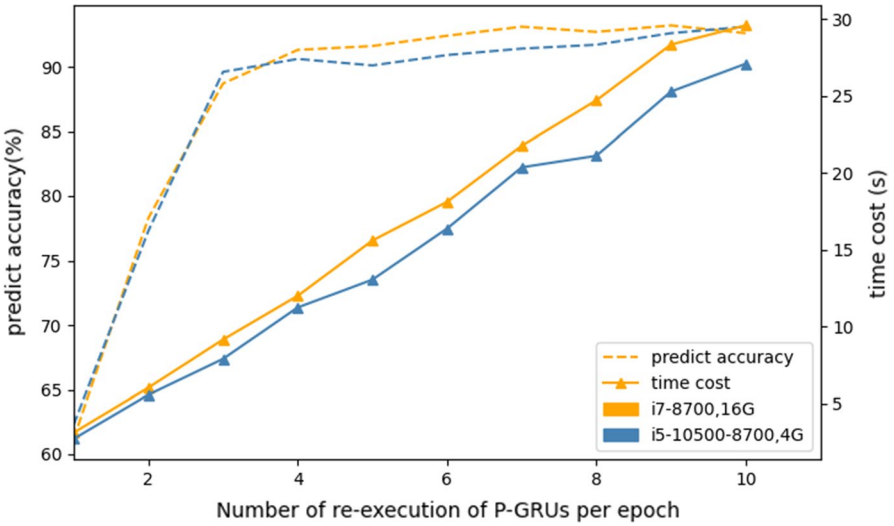


**Fig. 10** Predict accuracy and time cost of P-GRUs

to 3 times per epoch, the prediction accuracy improves significantly (88.7%), and the time cost is not very high. When the frequency is determined to be 8 times per epoch, the prediction accuracy reaches 92.7%, and the corresponding time overhead becomes unacceptable(24.72s).

### 4.4 Convergence

To verify the convergence of the EP4DDL method, we tested it with different node sizes on two data sets. As shown in Fig. 11, the models converge to acceptable accuracy under different data sets.

### 4.5 Baseline comparison

#### 4.5.1 Iteration time

The variance of the node performance in the system is reduced overall due to the effective improvement of the straggler's performance. Figure 12 shows that in the BSP group, the iteration time of the EP4DDL method is reduced by 76.7% and 60.48% compared to the BSP algorithm and sync-opt. However, it is still relatively high due to the explicit synchronization fence. In the ASP group, all algorithms have reduced the iteration time compared to the methods in the BSP group. Further, the EP4DDL method has a corresponding reduction of 59.5%, 10.6%, and 7.25% in iteration time compared to FlexRR, ConSGD, and Falcon due to its real-time characteristics and better search algorithm.

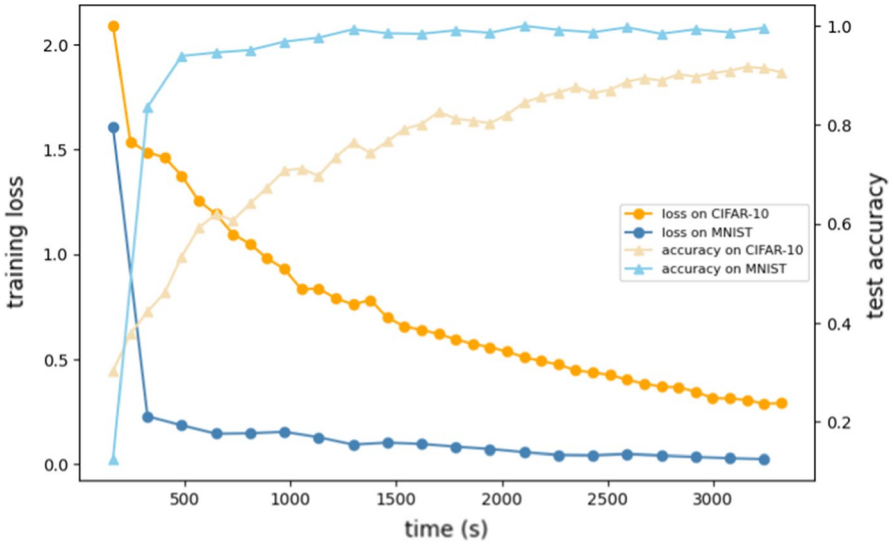


**Fig. 11** Training loss and test accuracy on Alexnet model with CIFAR-10 and MNIST

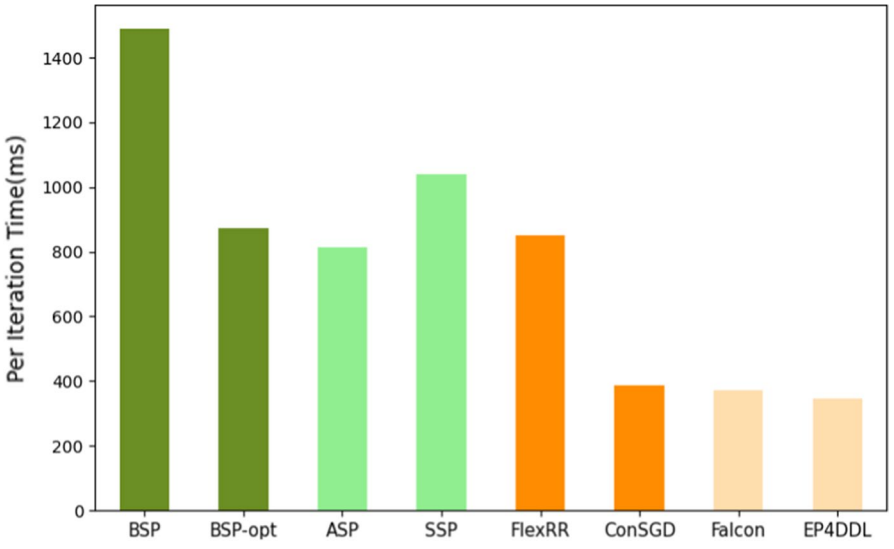


**Fig. 12** Training on Alexnet model with CIFAR-10

### 4.5.2 Convergence time

We tested the two data sets in different baseline tests. As shown in Fig. 13, both the BSP-based and SSP-based algorithms eventually converge with very low performance in heterogeneous environments. The ASP algorithm is very adaptable to the

heterogeneous environment but can lead to non-convergence of results. Compared to the state of art solutions, The EP4DDL method performs more iterations with higher accuracy and reduces the convergence time of the model by 57.46%, 24.8%, and 11.5% over FlexRR, Con-SGD, and falcon.

## 5 Conclusion

The straggler problem is a primary matter in distributed deep learning training under a heterogeneous environment. Therefore, we propose the EP4DDL method, which introduces the idea of load balancing to solve the straggler problem of iterative convergent data-parallel distributed deep learning. By using a reasonable adjustment of the elastic parallelism of nodes to achieve load balancing of nodes, the EP4DDL method fully exploits the computational capacity of nodes and avoids the performance degradation problem caused by stragglers. And on the top of experiment with various benchmarks, the EP4DDL method demonstrates reliability, excellent performance. It reduces training time by 7.5–59.5% and iteration time by 11.5–57.4% without accuracy loss over Falcon, Con-SGD, and FlexRR.

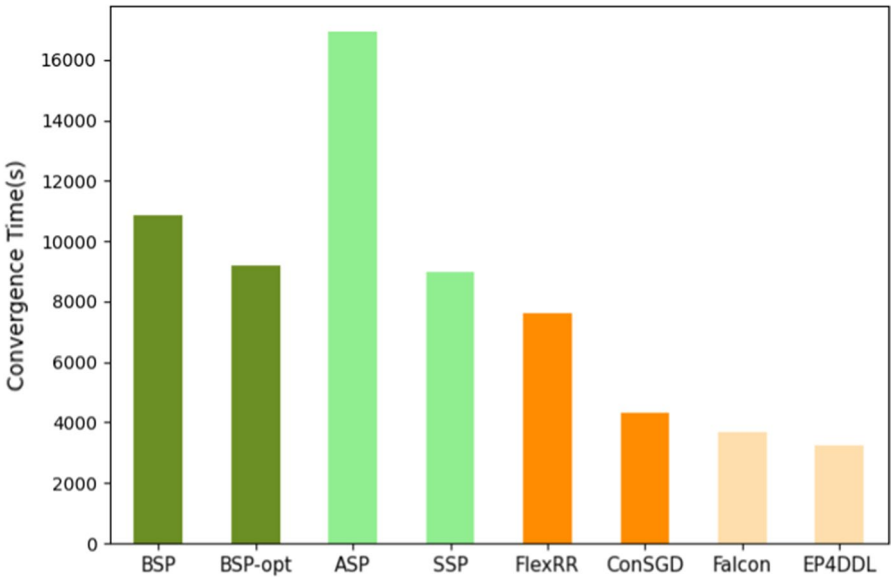


**Fig. 13** Training on VGG19 model with CIFAR-10

**Acknowledgements** We would like to thank the anonymous reviewers, whose insightful comments greatly improved the quality of this paper. The work described in this paper was supported in part by the Key Basic Research Program of the China Basic Strengthening Program (2019-JCJQ-ZD-041) and the National Key Research and Development Program of China (2016YFB0200902).

## References

1. Zhong Y, Oh S, Moon HC (2021) Service transformation under industry 4.0: investigating acceptance of facial recognition payment through an extended technology acceptance model. Technol Soc 64:101515
2. Stewart R, Velupillai S (2021) Applied natural language processing in mental health big data. Neuropsychopharmacology 46(1):252
3. Lanctot M, Lockhart E, Lespiau JB, et al (2019) OpenSpiel: a framework for reinforcement learning in games. arXiv preprint arXiv:1908.09453
4. Peng Y, Bao Y, Chen Y et al (2021) Dl2: a deep learning-driven scheduler for deep learning clusters. IEEE Trans Parallel Distrib Syst 32(8):1947–1960
5. Jiang J, Cui B, Zhang C et al (2017) Heterogeneity-aware distributed parameter servers. In: Proceedings of the ACM International Conference on Management of Data, pp 463–478
6. Ho Q, Cipar J, Cui H et al (2013) More effective distributed ml via a stale synchronous parallel parameter server. Adv Neural Inf Process Syst:1223
7. Zhou Q, Guo S, Lu H et al (2020) Falcon: addressing stragglers in heterogeneous parameter server via multiple parallelism. IEEE Trans Comput 70(1):139–155
8. Gill SS, Ouyang X, Garraghan P (2020) Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres. J Supercomputing 76(12):10050–10089
9. Harlap A, Cui H, Dai W et al (2016) Addressing the straggler problem for iterative convergent parallel ML. In: Proceedings of the Seventh ACM Symposium on Cloud Computing, pp 98–111
10. Kishor A, Chakraborty C, Jeberson W (2021) A novel fog computing approach for minimization of latency in healthcare using machine learning. Int J Interact Multimed Artif Intell 6(Special Issue on Current Trends in Intelligent Multimedia Processing Systems):7–17
11. Benalla M (2016) A distributed intelligent system for emergency convoy. Int J Interact Multimed Artif Intell 4:1
12. Aktas MF, Peng P, Soljanin E (2017) Effective straggler mitigation: which clones should attack and when? ACM SIGMETRICS Perform Eval Rev 45(2):12–14
13. Zhang J, Simeone O (2020) LAGC: Lazily aggregated gradient coding for straggler-tolerant and communication-efficient distributed learning[J]. IEEE Trans Neural Networks Learn Syst 32(3): 962–974
14. Bitar R, Wootters M, El Rouayheb S (2020) Stochastic gradient coding for straggler mitigation in distributed learning. IEEE J Sel Areas Inf Theor 1(1):277–291
15. Guo Y, Rao J, Jiang C et al (2016) Moving hadoop into the cloud with flexible slot management and speculative execution. IEEE Tran Parallel Distrib Syst 28(3):798–812
16. Huang Y, Jin T, Wu Y et al (2018) Flexps: Flexible parallelism control in parameter server architecture. Proc VLDB Endow 11(5):566–579
17. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Adv Neural Inf Process Syst 25:1097–1105
18. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
19. LeCun Y, Cortes C, Burges CJC "THE MNIST DATABASE of handwritten digits". http://yann.lecun.com/exdb/mnist/
20. Krizhevsky A, Nair V, Hinton G CIFAR-10: cs.toronto.edu/~kriz/cifar.html
21. Huang Y, Cheng Y, Bapna A et al (2018) Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1811.06965
22. Dean J, Corrado GS, Monga R et al (2012) Large scale distributed deep networks
23. Wu X, Xu H, Li B et al (2020) Stanza: layer separation for distributed training in deep learning. IEEE Trans Serv Comput

24. Geng J, Li D, Wang S (2020) Fela: incorporating flexible parallelism and elastic tuning to accelerate large-scale DML. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). IEEE, pp 1393–1404
25. Chen J, Pan X, Monga R et al (2016) Revisiting distributed synchronous SGD. arXiv preprint arXiv:1604.00981
26. Zheng S, Meng Q, Wang T et al (2017) Asynchronous stochastic gradient descent with delay compensation. In: International Conference on Machine Learning. PMLR, pp 4120–4129
27. Costantini S, De Gasperis G, De Lauretis L (2021) An application of declarative languages in distributed architectures: ASP and DALI microservices. Int J Interact Multimed Artif Intell 6(Special Issue on Artificial Intelligence, Paving the Way to the Future):66–78
28. Niu F, Recht B, Re C et al (2011) HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent. Adv Neural Inf Process Syst 24:693–701
29. Zhang W, Gupta S, Lian X et al (2016) Staleness-aware async-SGD for distributed deep learning. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pp 2350–2356
30. Chen M, Mao B, Ma T (2021) FedSA: a staleness-aware asynchronous Federated Learning algorithm with non-IID data. Fut Gener Comput Syst 120:1–12
31. Khaleghzadeh H, Manumachu RR, Lastovetsky A (2018) A novel data-partitioning algorithm for performance optimization of data-parallel applications on heterogeneous HPC platforms[J]. IEEE Trans Parallel Distribut Syst 29(10):2176–2190
32. Cho K, Van Merriënboer B, Gulcehre C et al (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078
33. Chen, C et al (2018) Fast distributed deep learning via worker-adaptive batch sizing. In: Proceedings of the ACM Symposium on Cloud Computing