Hindawi
International Journal of Intelligent Systems
Volume 2023, Article ID 2663115, 15 pages
https://doi.org/10.1155/2023/2663115

Research Article

Modeling the Training Iteration Time for Heterogeneous Distributed Deep Learning Systems

Yifu Zeng,[1,2] **Bowei Chen**,[2] **Pulin Pan**,[2] **Kenli Li**,[2] **and Guo Chen**[2]

[1]*College of Computer Science and Engineering, Changsha University, Changsha 410022, Hunan, China*
[2]*College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, Hunan, China*

Correspondence should be addressed to Guo Chen; guochen@hnu.edu.cn

Received 24 August 2022; Revised 16 September 2022; Accepted 14 October 2022; Published 21 February 2023

Academic Editor: Tao Li

Distributed deep learning systems effectively respond to the increasing demand for large-scale data processing in recent years. However, the significant investment in building distributed learning systems with powerful computing nodes places a huge financial burden on developers and researchers. It will be good to predict the precise benefit, i.e., how many times of speedup it can get compared with training on single machine (or a few), before actually building such big learning systems. To address this problem, this paper presents a novel performance model on training iteration time for heterogeneous distributed deep learning systems based on the characteristics of the parameter server (PS) system with bulk synchronous parallel (BSP) synchronization style. The accuracy of our performance model is demonstrated by comparing real measurement results on TensorFlow when training different neural networks with various kinds of hardware testbeds: the prediction accuracy is higher than 90% in most cases.

1. Introduction

In recent years, we have witnessed the boom of deep learning which has been successfully applied in numerous areas (also changed these areas), including image processing[1], speech processing [2], natural language processing [3], human-machine gaming [4], autonomous driving [5], and health care [6]. As the complexity of application scenarios and user requirements grow fast, deep learning models scale larger and larger, having massive data as training input, which exceed the processing capability of one single machine. Therefore, large-scale computer clusters are used to speedup the training of such big neural network models, driving the rapid development of distributed deep learning systems [7–10].

It requires significant investment to build such distributed learning systems, which consist of multiple high-end servers connected between each other with high speed networks. For example, the Google Brain project, which began exploring the use of very large-scale deep learning systems in 2011, costs an average of more than $100 million per year. For researchers in general universities, a distributed learning system often contains several computing nodes, each node has several GPUs/CPUs/FPGAs/TPUs, and requires a network connection of at least 10 Gbps, which cost up to $100,000. For such large investment, it will be good to predict the precise benefit, i.e., how many times of speedup it can get compared with training on single machine (or a few), before actually building such big learning systems. However, above question is still hard to answer.

Although several recent works [11–18] have tried mathematically modeling (some combined with experimental methods) the computation and communication part in distributed training, they are still not accurate enough in practice. Particularly, when modeling the communication time in multimachine training environment, they simply assume that all worker nodes (machines) have the same processing speed and are synchronized during communication (passing parameters/gradients), fairly sharing the network bandwidth. However, such communication model is too idealistic for large distributed learning systems in practice with worker nodes potentially being very heterogeneous with each other [19, 20]. Under such practical scenarios, it is neglected by previous models that worker machines may be complexly overlapped or interleaved with each other on both computation and communication during each training iteration. For example, the worker with the best computing capability pushes gradients as soon as it finishes calculation, which overlaps with the calculation of the other workers.

To address the above problem, we present a novel performance model on training iteration time for heterogeneous distributed deep learning systems. By carefully analyzing the steps of computing and communication during a training iteration, we can model the iteration time by inductively calculating the overlap and interference among multiple worker nodes. With our model, one can easily predict the speedup for different distributed hardware platforms only if knowing the target neural network and its configurations, before actually building the system and training on it.

We have evaluated the accuracy of our performance model by comparing real measurement results on TensorFlow [21] when training different neural networks on various kinds of hardware testbeds. Experimental results show that our model can predict the training iteration time in an average accuracy over 95%, with the worst accuracy being 78.3%, under various conditions (training alexnet and vgg11/16/19 using various number of machines with or without GPU connected with 1 Gbps or 10 Gbps networks).

The rest of the paper is organized as follows. Section 2 introduces the background and contributions of our research. Section 3 generally explained the input and output of our prediction model. The floating-point traffic statistics formula is derived in Section 4. Training Time Modelings of standalone/distributed platforms with CPU/GPU are performed in Section 5. The experimental results are explained in Section 6.

1.1. Notes. The performance model presented in this paper only focuses on (one of) the most widely used architecture of distributed deep learning systems, i.e., data-parallel parameter server (PS) system with bulk synchronous parallel (BSP) synchronization style [22, 23], which is shown to have better performance in both practice and theory [24, 25]. However, our performance model is easy to be extended using the same methods to different architectures such as all-reduce [26] and stale synchronous parallel (SSP) synchronization style [27, 28]. The extension is omitted in this paper. As previous works (i.e., [15]), we model the computation part based on the number of float-point operations and the communication part based on the network bandwidth and delay. We assume that distributed deep learning systems can fully utilize the hardware processing capability (which is the case in most of the current implementations), so implementation issues are not considered in our performance model. Moreover, this paper focuses on predicting the training speedup when using various hardware resources; hence, we do not model the convergence time for training a neural network to a desired accuracy, since the required number of iterations to converge will not change when using different number or different kinds of machines when the other configurations of the neural network remain unchanged.

2. Background, Related Works, and Problems

2.1. Background. In March 2017, Jeff Dean, head of Google Brain, gave a speech titled "Building Intelligent Systems with Large Scale Deep Learning" at UC Santa Barbara [29], where he predicted that machine learning expertise could be replaced by super computing power. Distributed machine learning systems are positioned to provide greater computing power. In distributed machine learning systems, the main parallel models include model parallelism (different machines (GPU/CPU, etc.) in a distributed system are responsible for different parts of the network model) and data parallelism (different machines have multiple copies, each machine is assigned a different data, and the computation results of all the machines are then merged in some way). The main system architecture includes PS (this architecture isolates the calculation of each worker node, and each worker node only interacts with the server) and All-Reduce (this architecture integrates data from different training nodes, and distributes the results to all training nodes after the integration is complete). The main parameter update methods include BSP (Bulk Synchronous Parallel), ASP (Asynchronous Parallel), and SSP (Stale Synchronous Parallel).

Among these combinations of system architecture and parameter update methods, PS + BSP is the most popular one that is demonstrated to have better performance in both practice and theory [24, 25]. With PS architecture, the bottleneck bandwidth is utilized up to twice more efficient than All-Reduce. Furthermore, PS + BSP can be used as asynchronous.

Therefore, the performance model presented in this paper only focuses on data-parallel parameter server (PS) system with bulk synchronous parallel (BSP) synchronization style.

2.2. Related Work. Previous work of predicting the training iteration time include [11–15, 17]. They build pure mathematical models or combined with experimental measurements to model the computation part and communication part. All of them assume all workers start to pull and push in a synchronized way, and calculate the time as parameter_size/gradient_size/bandwidth.

In computation part, the references [11, 13, 17] combine mathematical models and experimental measurements, while [12, 14, 15] predict the training iteration time with mathematical models alone.

2.3. Problems and Our Contributions. The iteration time modeling is not accurate in previous work, since distributed learning systems have heterogeneous computation hardware and each computing unit (GPUs/CPUs/FPGAs/TPUs) may have different computing power. As shown in Figure 1(a), the modeling in previous work is modeled as synchronized, which requires multiple computing units in the system to have the same processing speed. However, as shown in Figure 1(b), the computing process is generally unsynchronized in practice, since heterogeneous hardwares have different computation time. Therefore, the accuracy of total iteration time modeling can be improved by further considering the effect of heterogeneous computation hardware.

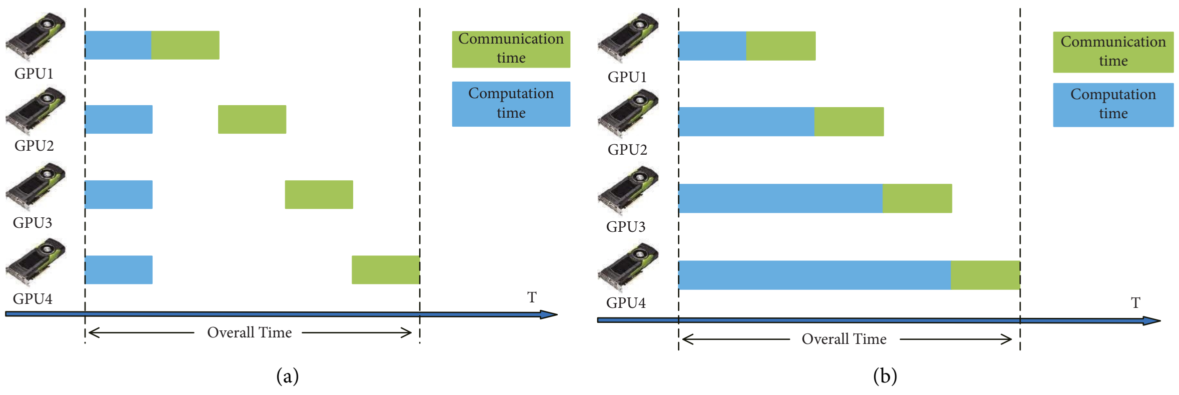


Figure 1: Comparison of the communication time modeled in previous works and in practice. (a) Modeled as synchronized, (b) but unsynchronized in practice.

Therefore, we combine the performance and working principle of training equipment, neural network structure, and network bandwidth to establish a mathematical model of one round of iterative time for distributed training under PS + BSP. Our analysis is based on stochastic gradient descent (SGD) since SGD and its variants are the main algorithms for training DNN models. What is more, sigmoid is one of the famous activation functions because of continuity and differentiability. Since it is proposed and widely used earlier, it is used as the first step of our modeling of activation function. Other activation functions will be further modeled in future work. Moreover, we measured the iteration time of a variety of deep neural network (DNN) models and compared it with the predicted iteration time; the experimental results that demonstrate our prediction model is highly accurate.

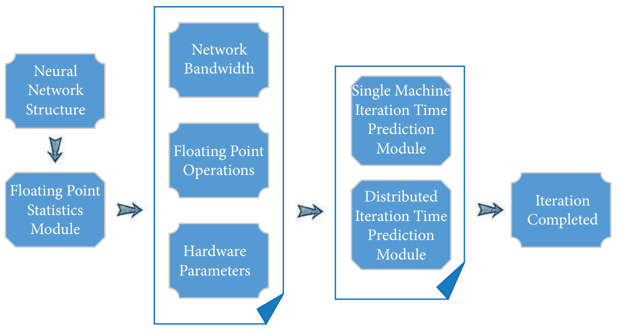

Neural Network Structure
Floating Point Statistics Module
Network Bandwidth
Floating Point Operations
Hardware Parameters
Single Machine Iteration Time Prediction Module
Distributed Iteration Time Prediction Module
Iteration Completed

Figure 2: One round of iterative time modeling framework.

3. Performance Model Overview

According to the floating-point calculation formula of neural networks that derived in Section 4, the floating-point statistics module in Figure 2 output the specific floating-point operations of various neural networks to the next module. Afterwards, combined with the floating-point operations, and the learning system's network bandwidth and hardware parameters, an iterative time prediction module for both single machine and distributed system can be established in Section 5. Prediction times output by the iterative time prediction module are demonstrated to be accurate in Section 6.

4. Derivation of Floating Point Operations Calculation Formula

According to our investigations, although some floating-point calculation statistics of neural networks have been proposed on some academic or technical platforms, there are some errors after our verification. Therefore, out of scientific rigor, the floating-point traffic statistics formula is derived in this section by combining the specific calculation process of forward-propagation and back propagation.

4.1. Formula Derivation of Convolutional Floating Point Operations. The matrix multiplication operation is the most basic calculation in the forward-propagation convolution operation. The premise of the matrix multiplication is that the size of the two multiplied matrices must be the same. After the two matrices are multiplied by a moment, the output is a natural number. The calculation of the moment multiplication can be expressed by formula (1). Among them, a_{ij} and b_{ij} are the elements of n -order matrices A and B, respectively.

$$\text{output} = \sum_{i=0,j=0}^{n-1} a_{ij}b_{ij}. \quad (1)$$

Now, we use $5 \times 5 \times 3$ samples and two $2 \times 2 \times 3$ convolution kernels $F0$ and $F1$ to illustrate the process of forward-propagation. The $5 \times 5 \times 3$ samples are expanded to obtain three 5×5 matrices, which is shown in Figure 3. First of all, the first three 2×2 matrix among the three sample matrices and the three 2×2 matrices of the convolution kernel $F0$ are, respectively, subjected to moment multiplication (marked in red in Figure 3). Secondly, the results of the three moment multiplication plus the $F0$'s bias value 1 equals

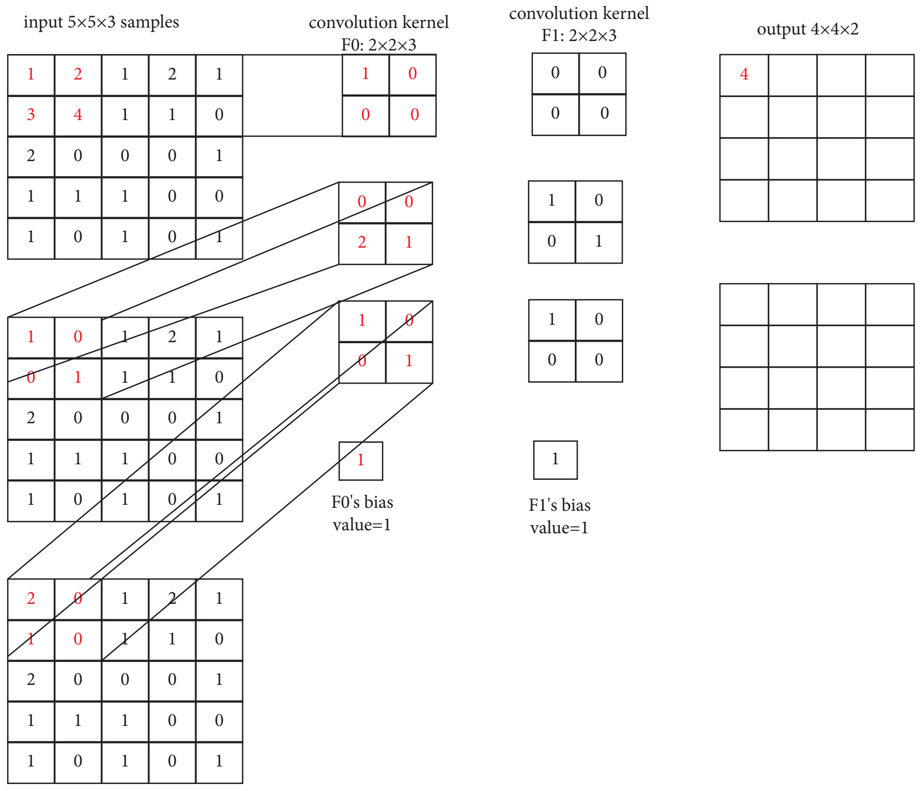


Figure 3: One round of iterative time modeling framework.

to the first matrix value 4 in the output matrix (also marked in red in Figure 3). The specific calculation process is as follows:

$$\begin{aligned}1\times1+2\times0+3\times0+4\times0&=1,\\1\times0+0\times0+0\times2+1\times1&=1,\\2\times1+0\times0+1\times0+0\times1&=2,\\1+1+2+1&=4.\end{aligned}\tag{2}$$

The calculation process has a total of 24 floating-point operations. It is noted that all parameters other than the bias value correspond to one floating-point operation. So further, the calculation of the number of floating-point operations f in this part can be generalized to formula (3). Among them, *inch* represents the input third dimension value, and kw and kh, respectively, represent the width and height of the convolution kernel.

$$f=\text{inch}\times\text{kh}\times\text{kw}\times2.\tag{3}$$

Third of all, move the "matrix to be multiplied" in the input sample to the left by one position. Similarly, perform a moment multiplication with the convolution kernel $F0$, and finally add a bias value of 1 to obtain the second value 7 of the output. According to the above method, the first 4×4 matrix output can be calculated.

Next, we calculate the value of each element of the second 4×4 matrix output. The only difference between this process and the previous steps is that the convolution kernel $F0$ is replaced with $F1$. After the convolution operation, the second two-dimensional tensor in the output tensor can be obtained, so that the complete $4\times4\times2$ matrix tensor can be obtained.

It can be observed from this example that each element in the output $4\times4\times2$ matrix tensor corresponds to 24 floating-point operations, that is, the amount of floating-point operations consumed by this example is $4\times4\times2\times24=576$. That is, outn × outh × outw × 24, where the value of 24 is obtained by formula (3), and outn, outh, and outw are the length, width, and height of the output tensor, respectively. Therefore, we can combine formula (3) to deduce the statistics of floating-point operations of each layer of convolution as formula (4).

$$\text{FLOPcov}=\text{inch}\times\text{kh}\times\text{kw}\times\text{outn}\times\text{outh}\times\text{outw}\times2.\tag{4}$$

The actual neural network is usually composed of multilayer convolution and a fully connected neural network, such as Alexnet and VggNet. The parameters of the fully connected layer usually occupy more than 90% of the entire neural network, although unlike the convolutional layer, the same parameter will not be recalculated. But its floating-point capacity cannot be ignored. The fully connected layer is usually a one-dimensional tensor. Therefore, the calculation of floating-point operations can also use formula (4), where kh, kw, outh, and outw are all set to 1, and inch and outn are, respectively, the number of elements contained in the two one-dimensional tensors before and after the layer.

4.2. Formula Derivation of Back-Propagation Floating Point Operations. Back-propagation is the process of calculating the gradient, and the calculation of the gradient is performed based on the calculation result of the forward-propagation. According to the calculation of forward-propagation, (5) and (6) can be derived as follows:

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)}, \tag{5}$$

$$a^{(l)} = \sigma\left(z^{(l)}\right), \tag{6}$$

where σ is the activation function; here, we choose Sigmod function as the activation function. $z^{(l)}$ is the result obtained by calculating the convolution of the forward-propagation of the l -th layer, and adding the bias value $b^{(l)}$, $a^{(l)}$ is the output processed by the activation function σ of the forward-propagation training result of the l -th layer.

The mean square error is the most commonly used loss function, so that the loss function is derived as (7) based on the mean square error. Among them, 1/2 is to offset the coefficients obtained after the derivation, and has no effect on the surface calculation.

$$C(w,b) = \frac{1}{2}\left(a^{(l)} - y\right)^2. \tag{7}$$

The gradient of the weight w and and bias value b of the last layer of the neural network (the output layer) $\partial C(w,b)/\partial w^{(l)}$ and $\partial C(w,b)/\partial b^{(l)}$ is calculated as

$$\frac{\partial C(w,b)}{\partial w^{(l)}} = \frac{\partial C(w,b)}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial w^{(l)}} = \left(a^{(l)} - y\right)\sigma'\left(z^{(l)}\right)a^{(l-1)}, \tag{8}$$

$$\frac{\partial C(w,b)}{\partial b^{(l)}} = \frac{\partial C(w,b)}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial b^{(l)}} = \left(a^{(l)} - y\right)\sigma'\left(z^{(l)}\right). \tag{9}$$

The common part of (8) and (9) can be written as

$$\delta^{(l)} = \frac{\partial C(w,b)}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}} = \left(a^{(l)} - y\right)\sigma'\left(z^{(l)}\right). \tag{10}$$

The value of $\sigma'(z^{(l)})$ is shown in equation (16).

The gradient of the output layer (l-th layer) has been calculated in (8) and (9); in the same way, the gradient of the $l-1$-th layer can be calculated. Since the error of the output layer l has been calculated above, according to the back-propagation theory, the error of the current layer is the composite function of all the neuron errors of the previous layer, that is, the error of the previous layer can be used to express the error of the current layer.

Assume that $\delta^{(l+1)}$ of the $l+1$-th layer is calculated, then the $\delta^{(l)}$ of the l-th layer is (the l-th layer here is not the output layer but any non-last layer) as follows:

The common part of (8) and (9) can be written as

$$\delta^{(l)} = \frac{\partial C(w,b)}{\partial z^{(l+1)}}\frac{\partial z^{(l+1)}}{\partial z^{(l)}} = \delta^{(l+1)}\frac{\partial z^{(l+1)}}{\partial z^{(l)}}. \tag{11}$$

Among (11), $z^{(l+1)}$ is derived by

$$z^{(l+1)} = w^{(l+1)}a^{(l)} + b^{(l+1)} = w^{(l+1)}\sigma\left(z^{(l)}\right) + b^{(l+1)}. \tag{12}$$

Then, we have (13) and (14):

$$\frac{\partial z^{(l+1)}}{\partial z^{(l)}} = w^{(l+1)}\sigma'\left(z^{(l)}\right), \tag{13}$$

$$\delta^{(l)} = w^{(l+1)}\sigma'\left(z^{(l)}\right)\delta^{(l+1)}. \tag{14}$$

Derived from the Sigmod function, Formula (15) and (16) can be obtained.

$$\sigma\left(z^{(l)}\right) = \frac{1}{1+e^{-x}}, \tag{15}$$

$$\sigma'\left(z^{(l)}\right) = \sigma\left(z^{(l)}\right)\left(1 - \sigma\left(z^{(l)}\right)\right). \tag{16}$$

Therefore, for the calculation of the intermediate layer gradient, there is a recursive formula (17).

$$\delta^{(l)} = w^{(l+1)}\sigma\left(z^{(l)}\right)\left(1 - \sigma\left(z^{(l)}\right)\right)\delta^{(l+1)}. \tag{17}$$

Finally, new parameters w and b can be obtained by (18) and (19) Among them, the η is the learning rate (generally a fixed value).

$$w = w - \eta\frac{\partial C(w,b)}{\partial w^{(l)}}, \tag{18}$$

$$b = b - \eta\frac{\partial C(w,b)}{\partial b^{(l)}}. \tag{19}$$

In summary, it can be seen from (8) that each parameter of the last layer of parameters has $4 + \text{sig}$ floating-point operations, where sig is the number of floating-point operations required for the calculation of the Sigmod function. The value of sig is explained in section 4.3. Combined with 2 floating-point operations in (18), the total number of floating-point operations is $6 + \text{sig}$. The gradient corresponding to each parameter of the middle layer can also be obtained as $6 + \text{sig}$ times by formula (17) and equation (18). The gradient ($\partial C(w,b)/\partial b^{(l)}$) of the bias value b corresponding to w is calculated as in equation (9), and this value has been calculated in equation (8). Therefore, the calculation of equation (9) has no additional floating-point operations, and there are 2 floating-point operations in equation (19). From the above, the statistical formulas for back-propagation floating-point numbers are shown in equations (22)–(24) in section 4.3.

4.3. Derivation of Total Floating Point Operations. The total floating-point number of forward-propagation needs to be calculated separately according to the number of layers of the neural network. According to formula (4), the floating-point number of the i-th layer is shown in formula (20). Among them, inch represents the number of input channels of the i-th layer, kh and kw, respectively, represent the length and width of the i-th layer convolution kernel, outn, outh, and outw, respectively represent the matrix number, height, and width of the output tensor of the i-th layer. The total FLOPS of forward-propagation is shown in formula (21).

$$\text{FLOPS}_i = \text{inch} \times \text{kh} \times \text{kw} \times \text{outn} \times \text{outh} \times 2, \tag{20}$$

$$\text{FLOPS}_{\text{forward}} = \sum_{i=1}^{n} \text{FLOPS}_i. \tag{21}$$

The sum of the number of parameters w and b is the sum of the parameters of the neural network (para). The number of b(parab) is the result of length × width × height of the tensor size output by each layer. The number of w is the number of all parameter (para and parab). Therefore, the FLOPS of back-propagation is shown in equations (22)–(24). Among them, FLOPS_w represents the number of floating-point calculations required for a weight in a back-propagation, and FLOPS_b represents the number of floating-point calculations required for a bias value in a back-propagation.

$$\text{FLOPS}_{\text{backward}} = \text{para} \times \text{FLOPS}_w + \text{parab} \times \text{FLOPS}_b, \tag{22}$$

$$\text{FLOPS}_w = 6 + \text{sig}, \tag{23}$$

$$\text{FLOPS}_b = 2. \tag{24}$$

From the formula (15), the value of sig is $2 + x$, where x represents the number of floating-point calculations consumed to calculate the exponential function e^x. Since the calculation of the exponential function e^x in *Python*3 is essentially an approximation obtained by calculating the Taylor expansion of e^x (equation (25)), we can get equation (26). Among them, i is Taylor's expansion series, and this value is uncertain in the underlying implementation of *Python*3. *Python*3 is dynamically adjusted according to the actual calculated value, and the calculation accuracy is increased by 3 digits each time. Due to the complexity and uncertainty of the parameters in deep learning, we will take the value of i based on the results of experiments, and through the results of multiple experiments, i is approximately equal to 20.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + 0\left(x^6\right), \tag{25}$$

$$x = 1 + 4i. \tag{26}$$

In summary, in a round of iteration with a batch size of bs, the number of floating-point numbers (FLOPS) is expressed as formula (27).

$$\text{FLOPS} = bs \times \text{FLOPS}_{\text{forward}} + \text{FLOPS}_{\text{backward}}. \tag{27}$$

5. Predictive Modeling of Distributed Training Time Based on CPU and GPU

5.1. Training Time Modeling of Stand-Alone CPU Platform. The Single Instruction Multiple Datastream (SIMD) technology enables multiple data to be calculated in parallel within the same CPU clock cycle with only one instruction. In general, the parallel data calculation of Single Instruction to Multiple Execution depends on the number of CPU registers. If it is 64 bits, it can be split into 8 8 bit registers, and 8 8 bit data operations can be completed at the same time. The efficiency is increased by 8 times. Similarly, it can be divided into 2 32 bit or 4 16 bit registers as well. According to the operating principle of the CPU, the mathematical model of the single-core cpu's one round iteration time can be established as formula (28). Among them, Tcpu represents the training prediction time. The value of FLOPs is obtained by formula (27) in Section 4.3. cpuhz represents the CPU frequency, and simd represents the number of data that can be operated in parallel after optimization with SIMD instructions.

$$T_{\text{cpu}} = \frac{\text{FLOPs}}{\text{cpuhz} \times \text{simd}}. \tag{28}$$

5.2. Training Time Modeling of Distributed CPU Platform. Most of the 64 bit processors on the server market now have a clock speed between 1.6 Ghz and 2.6 Ghz, so the computing power for 32 bit floating-point numbers is between 3.2 and 5.2 GFLOPs. Commonly used neural networks such as Alexnet and Vgg have parameters ranging from tens to hundreds of megabytes. Specifically, Alexnet has 63 M parameters, which is equivalent to 2 Gbit data, while Vgg has 133 M parameters, which is 4.25 Gbit data. If the computing power of each training node is not far apart, and the total bandwidth is below 1 Gb, then the communication overhead will inevitably seriously affect the training performance. The communication overhead will become a performance bottleneck for training when the time for each node to execute push is in conflict, which will cause network congestion. The purpose of our modeling is to help relevant personnel obtain the most cost-effective overall system architecture solution by predicting what kind of equipment constitutes a system architecture that can avoid the performance bottleneck caused by communication.

Based on the PS architecture and combined with the synchronization communication algorithm, mathematical models for training nodes with equal computing power and nonequal computing power is established. The one-round iteration time modeling on platforms with equal computing power are illustrated in formulas (29)–(31). Among them, T

represents the total time of one-round iteration, T_{cpu} is obtained by formula (28). T_{pull} is the total time for all workers to pull parameters, and T_{push} is the total time for all workers to send gradients. T_{pull} and T_{push} are both communication time, and the communication performance is mainly manifested in T_{push}. n is the number of workers, P and G are, respectively, the amount of parameters and gradient (need to be converted to bit), and B is the bandwidth of the parameter server (bit/s).

$$\begin{aligned} T = T_{\text{cpu}} + T_{\text{push}} \times n \\ + T_{\text{pull}} \times n, \end{aligned} \tag{29}$$

$$T_{\text{pull}} = \frac{P}{B}, \tag{30}$$

$$T_{\text{push}} = \frac{G}{B}. \tag{31}$$

The one-round iteration time modeling on platforms with nonequal computing power and equal communication power are illustrated in Algorithm 1. The main scenario considered in this paper is the case where multiple workers as well as PSs are connected to the same switch. In this case, it is common that each machine have equal communication power. And the link between the PS and the switch is the main communication bottleneck. Figure 4 gives a brief explanation of Algorithm 1. In the figure, nodes $w0w3$ have the same push time and pull time, but the calculation time is different and gradually increases. To calculate one round of iteration time T, first initialize T to the sum of the calculation time of $w0$ and the push time. Second, since $T = T_{\text{cpu0}} + T_{\text{push}} > = T_{\text{cpu1}}$ and $T_{cpu0} > T_{cpu1}$, the push of $w0$ has not ended when $w1$ is calculated. Thanks to the communication bottleneck in the link between the PS and the switch, the gradient of $w1$ will enter the sending queue and wait for the gradient of $w0$ to be sent before pushing, and the push of $w1$ will finish after the push of $w0$ ended with T_{push}, which can be shown in yellow. Therefore, the total time consumption from the start of local calculation of $w0$ to the end of the push of $w1$ is equal to the sum of calculation time of $w0$, the push time of $w0$ and the push time of $w1$, which represented as T. As shown in Figure 4, the current value of T can be represented by moving the push legend of $w1$ directly behind the push legend of $w0$. Third, since the calculation time of $w2$ is larger than the current value of T, so T is directly set to the sum of the calculation time of $w2$ and the push time of $w2$. Fourth, similar to the second step, the current value of T is the calculation time of $w2$ plus the push time of $w2$ and the push time of $w3$. Finally, add the pull time to T, that is the result of one round of iteration time (By default, the parameter server will use a balanced algorithm to ensure that each computing node can receive new parameters at the same time).

In our analysis, the ratio of computation time to communication time and the local computation time for each worker is uncertain. When the ratio of computation time to communication time is different, the one-round iteration time will be calculated differently according to Algorithm 1 and Algorithm 2. In particular, when the communication time is greater than the computation time of each worker, the one-round iteration time is simply summarized as

$$T = n * T_{\text{pull}} + T_{\min} + n * T_{\text{push}}, \tag{32}$$

where $T_{\min}$ is the local computation time of the fastest worker(with the best computing capability in n workers). In this case, the bottleneck push process starts as soon as the fastest worker completes its computation and the iteration time is not related to the computation time of other workers. In other words, under the communication bottleneck, replacing some of the workers with those with worse computational performance does not change the iteration training time, but it significantly saves computational resources compared with equal computing power. And it will be better to make the communication time as equal to the computation time as possible. When there is a communication bottleneck, replacing a higher bandwidth network device can result in a better performance improvement than replacing a better computing device. The reverse is also true.

5.3. Training Time Modeling of Stand-Alone GPU Platform. The floating-point computing power of GPU is usually expressed in FLOPs/s per second. The maximum GPU floating-point computing power for computing with stream processor is gpuhz × sp × fc, where gpuhz represents the GPU clock speed, sp represents the number of stream processors, and fc represents the number of floating-point operations that can be performed per stream processor in a single clock cycle. From this, the single iteration time mathematical model of a single-GPU computer can be established as formula (33). In this formula, the value of FLOPs is obtained by formula (27). In addition, the value of fc depends on whether the Fused-Multiply-Add(FMA) instruction set is used. If it is used, fc = 2. Otherwise, fc = 1.

$$T_{\text{gpu}} = \frac{\text{FLOPs}}{\text{gpuhz} \times \text{sp} \times \text{fc}}. \tag{33}$$

5.4. Training Time Modeling of Distributed GPU Platform. Since sending data is executed by the CPU, a training device that uses GPU for distributed training can execute the two tasks of training and sending data in parallel. In this scenario, the first push of the training node occurs after the back-propagation completes the calculation of the gradient of the first layer. In addition, since most of today's GPUs are clocked at more than 1.5 Ghz, even if there is only one stream processor, its calculation speed is 1.5 × 32 Gbit/s. Therefore, if and only if the sending speed is greater than 48 Gbit/s, the sending time will be less than the calculation time. However, the number of stream processors of current GPUs is usually more than hundreds, so the calculation speed is obviously much faster than the sending speed. In this way, the calculation time of one round of iterative time modeling for distributed multimachines using GPUs can be regarded as the forward-propagation time plus the time to complete the first layer of back propagation. This time can be

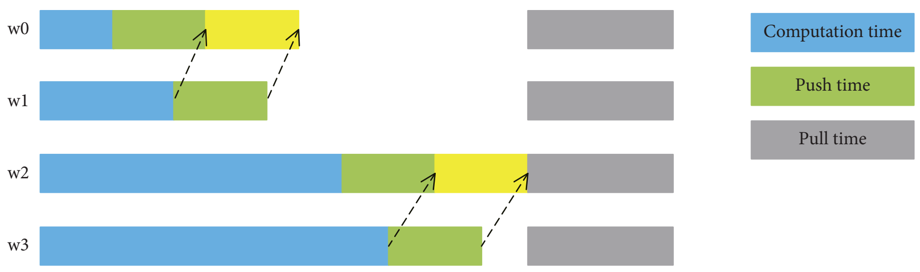


Figure 4: One-round iteration time modeling algorithm with nonequal computing power CPUs.

```
Require: FLOPs, CPUhz1, CPUhz2 . . . CPUhzn, simd = 2, $B, G, P, n$.
Ensure: One-round iteration time $T$.
1: Calculate the computing time of each device by formula (28), and arrange the CPUs in ascending order as Tcpu1, Tcpu2 . . . Tcpun;
2: Calculate $T_{\text{pull}}$ and $T_{\text{push}}$ by formulas (30) and (31);
3: $T = T_{\text{cpu1}} + T_{\text{push}}$;
4: for $i$ in $(2, n)$ do
5:    if $T \geq T_{\text{cpui}}$ then
6:       $T+ = T_{\text{push}}$;
7:    else
8:       $T = T_{\text{cpui}} + T_{\text{push}}$;
9:    $T+ = n \times T_{\text{pull}}$;
```

Algorithm 1: One-round iteration time modeling Algorithm with nonequal computing power CPUs and equal communication power.

obtained by dividing the total time of back propagation by the number of pushes. The number of pushes depends on the experimental results, which are shown in Table 1. The one-round iteration time modeling algorithm of distributed GPU platform is shown in Algorithm 2.

Now, we give a briefly explanation of Algorithm 2 in conjunction with Figure 5. In the figure, $w0$ $w3$ are four training nodes, and the red square represents the forward-propagation time. The blue square and the green square represent the back-propagation time and push time, respectively, and both are divided into several small segments according to the number of pushes. As mentioned above, the calculation speed is greater than the sending speed, so each green segment is slightly larger than the blue segment. First, when $w0$ finishes calculating the first blue segment, it starts to send the gradient. When the second blue segment is calculated, since the first gradient has not yet been sent, it enters the sending queue and waits. That is to say, the green segment representing the push of second gradient needs to be placed behind the green segment representing the push of first gradient, and same for subsequent operations. Second, when $w1$ calculates the first gradient (the first blue segment) and starts sending, since the gradient of $w0$ has not been sent yet (green square), the gradient of $w1$ enters the waiting queue. The corresponding push time is directly added to the push time of $w0$. At this time, the total time T equals to the forward-propagation time of $w0$ (red square), plus the calculation time of the first gradient of $w0$ (the first blue segment), plus the push time of $w0$ and $w1$ (green square and yellow square). In the third step, we can see that the forward-propagation time of $w2$ plus the back-propagation time of the first layer is greater than T, so the value of T is set to the forward-propagation time plus the back-propagation time of the first layer plus the push time of of $w2$. Fourth, it can be seen that the forward-propagation time of $w3$ plus its first-layer back-propagation calculation time is less than T. Therefore, the push time of $w3$ can be added to T as the current total time T. Finally, adding the total pull time pull × 4 is the iteration time T. Finally, by adding T to the total pull time (pull × 4, the four gray squares in the figure), one round of iteration time T can be obtained.

Similarly, when the communication time is greater than the computation time of each worker, the one round iteration time can also be summarized as ??. The iteration time is only related to the computation time of the fastest worker.

6. Performance Evaluation

6.1. Experimental Preparation and Experimental Environment Introduction. Existing neural network-related papers generally do not count the amount of floating-point operations, and there is no reliable statistical work on the amount of floating-point operations in neural networks. As a prework for performance evaluation, we have performed statistics on the forward-propagation and back propagation of a number of commonly used convolutional neural networks according to formulas in Section 4. In addition, we also obtain the number of push gradients in one round of iteration of some neural networks according to related distributed experiments. The result is shown in Table 1.

For the experimental environment, Ubuntu18.04 and *Python*3.7 are selected to implement the functional modules of the distributed machine learning performance modeling system. The construction of the distributed system is based

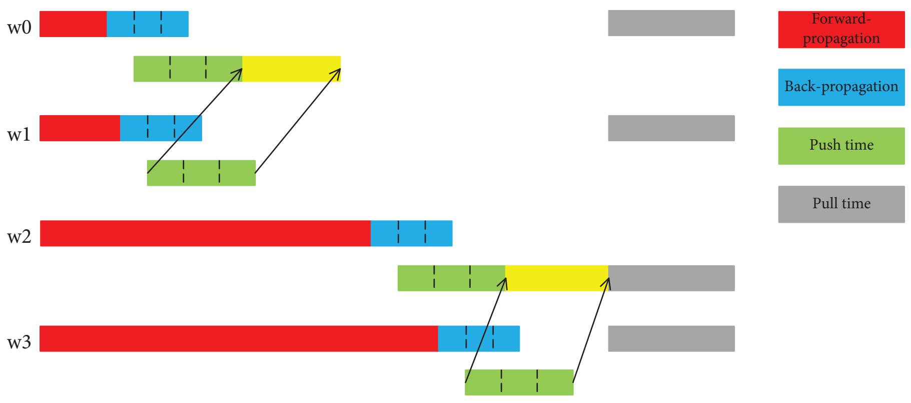


FIGURE 5: One-round iteration time modeling with GPUs.

TABLE 1: Statistics of FLOPs and number of push.

Convolutional neural network	Forward-propagation	Back-propagation	Number of push
Alexnet	$1.6776 \times 10^{+09}$	$5.2554 \times 10^{+09}$	19
Overfeat	$5.8584 \times 10^{+09}$	$1.2402 \times 10^{+10}$	20
Vgg11A	$1.5224 \times 10^{+10}$	$1.1292 \times 10^{+10}$	24
Vgg13B	$2.2623 \times 10^{+10}$	$1.1308 \times 10^{+10}$	—
Vgg16C	$2.3548 \times 10^{+10}$	$1.1358 \times 10^{+10}$	34
Vgg16D	$3.0947 \times 10^{+10}$	$1.1759 \times 10^{+10}$	—
Vgg19E	$3.9270 \times 10^{+10}$	$1.2210 \times 10^{+10}$	40
Resnet18	$3.5924 \times 10^{+9}$	$9.7808 \times 10^{+8}$	—
Resnet34	$7.2898 \times 10^{+9}$	$1.8366 \times 10^{+9}$	—
Resnet50	$6.9943 \times 10^{+9}$	$1.8015 \times 10^{+9}$	269
Resnet101	$1.4421 \times 10^{+10}$	$3.4114 \times 10^{+9}$	524
Resnet152	$2.1845 \times 10^{+10}$	$4.7372 \times 10^{+9}$	779

on TensorFlow2.1, TensorFlow/benchmark, and CUDA-10.2. The CPUs are Intel ® Xeon™ E5-2660 @ 2.2 GHz and Intel ® Xeon™ E5-2620 @ 2.0 GHz. The GPUs are NVIDIA Quadro RTX 4000 8 GB @ 1.545Ghz and GeForce GTX 1060 6 GB @ 1.506 Ghz. The end network bandwidth is 1 Gb/s. Alexnet, Vgg11A, Vgg16D, and Vgg19E are selected as representatives of neural networks. The dataset is ImageNet, and the size of each sample is $224 \times 224 \times 3$. The source code is available at https://github.com/bocway/Distributed-DNN-Modeling.

6.2. Experimental Results of CPU and GPU in Stand-Alone Platform. The experimental results of prediction time and actual measurement time of one round of iteration on stand-alone CPU platform are shown in Table 2 and Figure 6. It can be seen that the experimental results in this section are relatively ideal with an average accuracy of more than 90%, which demonstrates the effectiveness of the proposed prediction model. The experiments prove that our modeling of the computation time is essentially correct. There are two main reasons for the error, including the randomness of system scheduling, and the limit of floating calculation accuracy of *Python* 3.7. Due to these two reasons, even if the same experiment is performed on the same device multiple times, the results will not be exactly the same.

The experimental results of prediction time and actual measurement time of one round of iteration on stand-alone GPU platform are shown in Table 3 and Figure 7. Since the FMA instruction are not used in these experiments, the value of fc is 1. There are two reasons for the errors in stand-alone GPU training. The first one is the computing power is too strong, which finishes training too fast that the time management function cannot be accurately captured. The second one is some training models such as Alexnet are too small and the GPU's stream processors are not all used. It means the GPU's computing power is not fully utilized, which result in errors in the prediction. It can be seen that the accuracy of single-machine GPU training increases as the scale of the neural network increases. Vgg19 is the largest of these four neural networks, and the accuracy is above 96% under the actual measurement with $bs = 32$ and $bs = 64$. The scale of Vgg16 is slightly inferior to that of Vgg19, and the accuracy can also be stabilized at around 90%. However, Alexnet and Vgg11 have relatively large fluctuations in the accuracy between $bs = 32$ and $bs = 64$ due to the smaller neural network scale, which verifies the above-mentioned reasons for the errors.

6.3. Experimental Results of CPU and GPU in Distributed Platform. The CPU frequency of the two training nodes is 2 Ghz (Intel ® Xeon™ E5-2620), and the CPU frequency of the parameter service node is 2.2 Ghz (Intel ® Xeon™ E5-2660). The experimental results of prediction time and actual measurement time of one round of iteration on distributed CPU platform are shown in Table 4 and Figure 8.

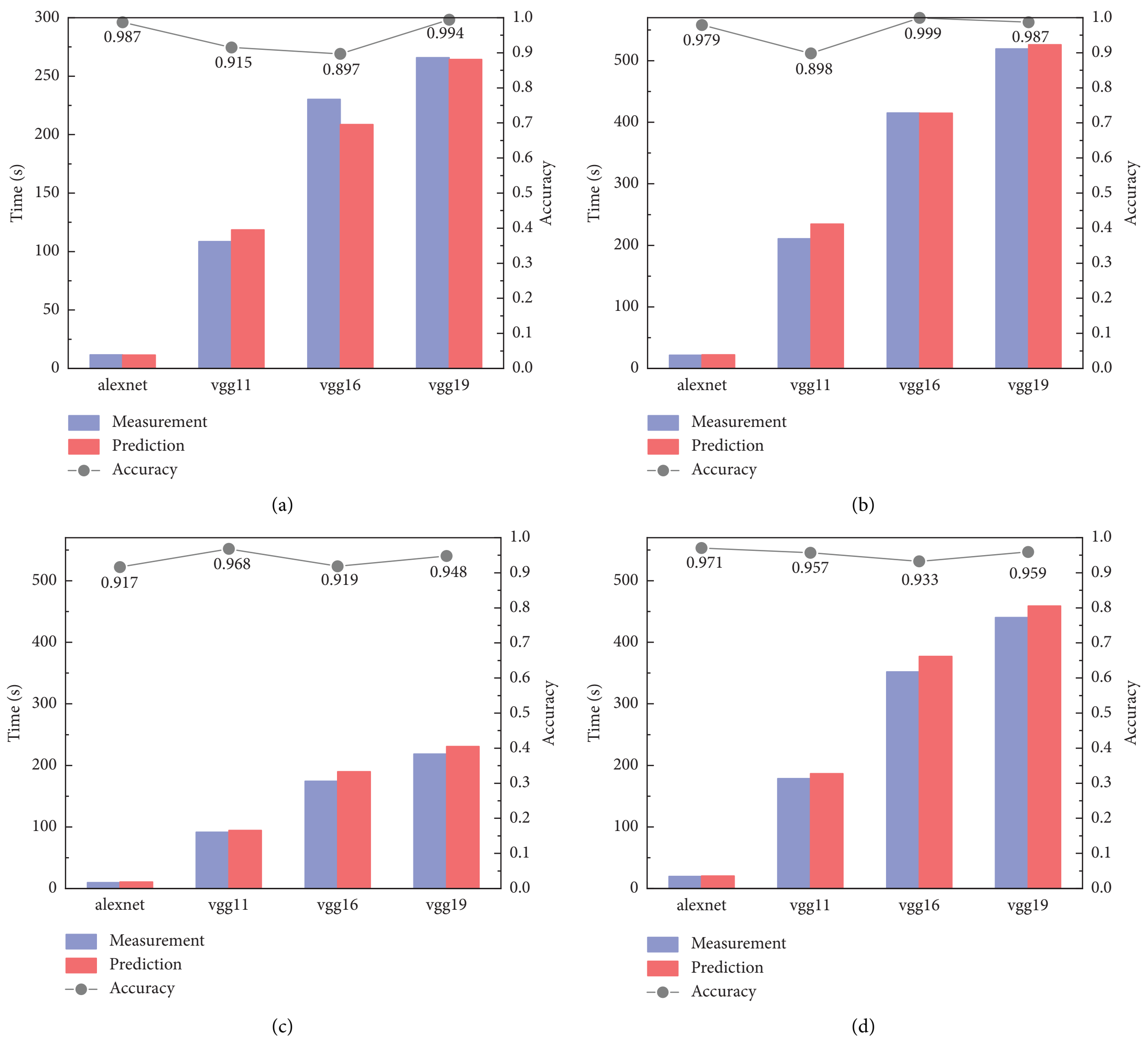


FIGURE 6: Comparison between prediction time and measurement time on stand-alone CPU platform. (a) 2.0 GhzCPU, bs = 32, (b) 2.0 GhzCPU, bs = 64, (c) 2.2 GhzCPU, bs = 32, and (d) 2.2 GhzCPU, bs = 64.

```
Require: forwardpropagationFLOPs, backpropagationFLOPs, numberofpushes, the gpuhz, sp, fc, $B, G, P$ of each node, batch − size set by each node, number of nodes $n$.
Ensure: One-round iteration time $T$.
1: Calculate $T_{\text{pull}}$ and $T_{\text{push}}$ by formulas (30) and (31);
2: for $i$ in $(1, n)$ do
3:     $\text{FLOPs}_i = (\text{forwardpropagationFLOPs} \times \text{batch} - \text{size}_i + \text{backpropagationFLOPs} \div \text{numberofpushes})$;
4:     $\text{Tgpu}_i = \text{FLOPs}_i \div (\text{gpuhz}_i \times \text{sp} \times \text{fc})$
5: Sort $(\text{Tgpu}_1, \text{Tgpu}_2 \ldots \text{Tgpu}_n)$ in ascending order;
6: $T = T\text{gpu}_1 + T\text{push}$;
7: for $i$ in $(2, n)$ do
8:     if $T \geq T\text{gpu}_i$ then
9:         $T + = T\,push$;
10:    else
11:        $T = T\text{gpu}_i + T\text{push}$;
12:    $T + = n \times T\text{pull}$;
```

ALGORITHM 2: Training time modeling algorithm of distributed GPU platform.

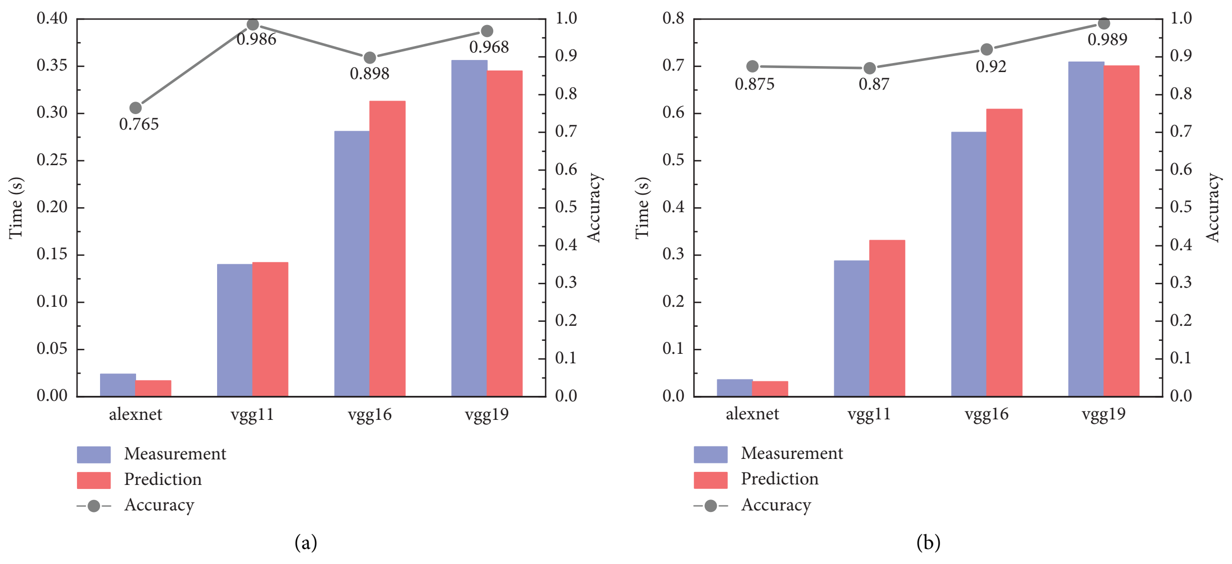


Figure 7: Comparison between prediction time and measurement time on stand-alone GPU platform. (a) bs = 32 and (b) bs = 64.

Table 2: Prediction time (PT), measurement Time (MT), and accuracy on stand-alone CPU platform.

Frequency (Ghz)	bs = 32	PT	MT	Accuracy	bs = 64	PT	MT	Accuracy
2.0	Alexnet	11.59	11.74	0.987	Alexnet	22.18	21.72	0.979
	Vgg11	118.68	108.62	0.915	Vgg11	234.68	210.76	0.898
	Vgg16	208.75	230.26	0.897	Vgg16	415.06	415.36	0.999
	Vgg19	264.35	265.87	0.994	Vgg19	526.15	519.27	0.987
2.2	Alexnet	10.54	9.66	0.917	Alexnet	20.17	19.58	0.971
	Vgg11	94.41	91.42	0.968	Vgg11	186.68	178.64	0.957
	Vgg16	189.78	174.32	0.919	Vgg16	377.33	351.99	0.933
	Vgg19	230.71	218.68	0.948	Vgg19	459.19	440.37	0.959

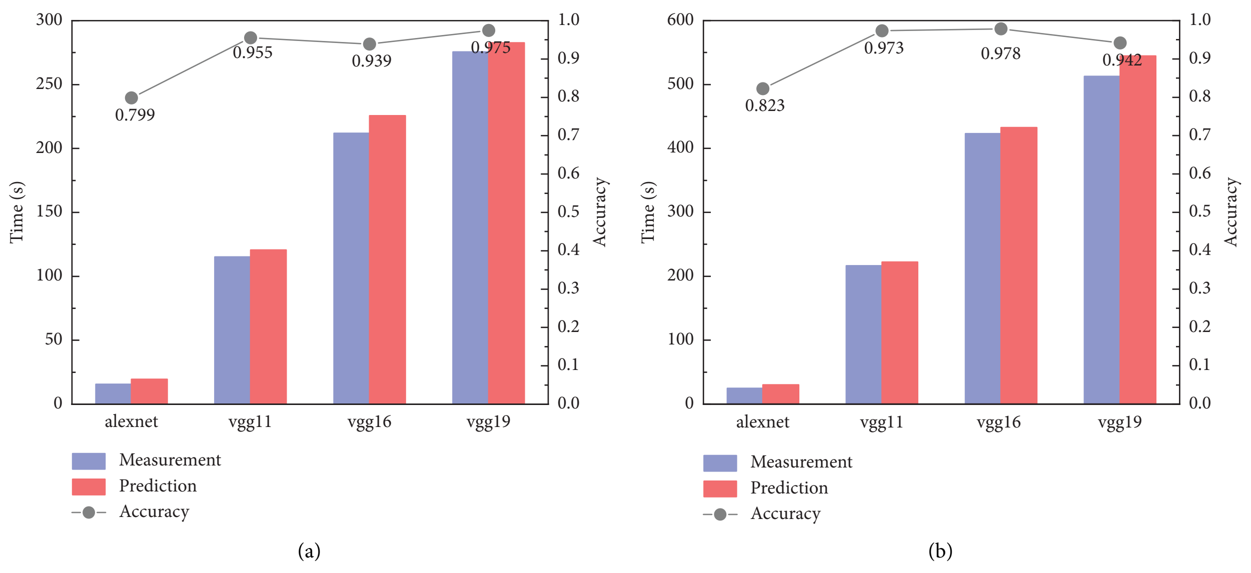
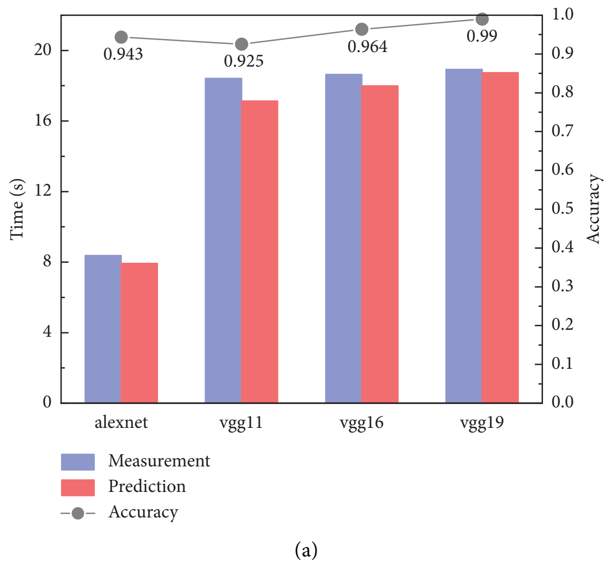


Figure 8: Comparison between prediction time and measurement time on distributed CPU platform. (a) bs = 32 and (b) bs = 64.

It can be seen that the performance on Alexnet is only mediocre that the accuracy is 78.3% and 82.3% when the *bs* is 32 and 64, respectively. However, the accuracy of the other three convolutional neural networks can reach around 95%. Since the distributed platform is built on two virtual machines on a physical machine, we speculate that the parameter server only performs one pull operation on the physical machine, and the two virtual machines obtain

Table 3: Prediction time (PT), measurement time (MT), and accuracy on stand-alone GPU platform.

bs = 32	PT	MT	Accuracy	bs = 64	PT	MT	Accuracy
Alexnet	11.59	11.74	0.987	Alexnet	22.18	21.72	0.979
Vgg11	118.68	108.62	0.915	Vgg11	234.68	210.76	0.898
Vgg16	208.75	230.26	0.897	Vgg16	415.06	415.36	0.999
Vgg19	264.35	265.87	0.994	Vgg19	526.15	519.27	0.987



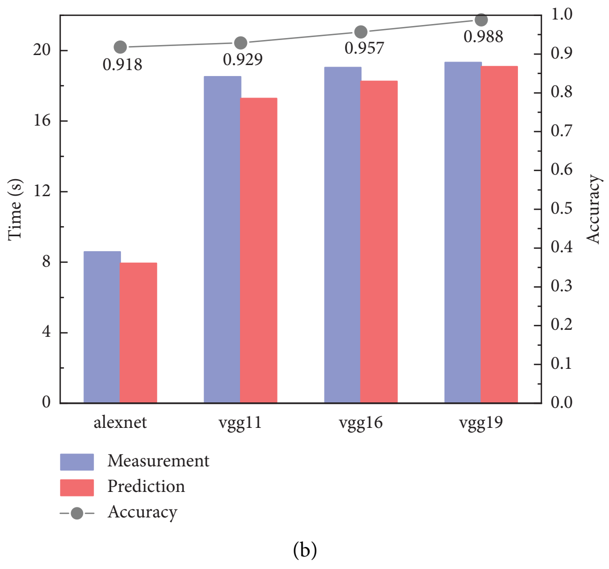


Figure 9: Comparison between prediction time and measurement time on distributed GPU platform. (a) bs = 32 and (b) bs = 64.

Table 4: Prediction time (PT), measurement time (MT), and accuracy on distributed CPU platform.

bs = 32	PT	MT	Accuracy	bs = 64	PT	MT	Accuracy
Alexnet	19.51	15.28	0.783	Alexnet	30.09	24.75	0.823
Vgg11	120.68	115.27	0.955	Vgg11	222.34	216.39	0.973
Vgg16	225.71	211.86	0.939	Vgg16	432.77	423.35	0.978
Vgg19	282.74	275.61	0.975	Vgg19	544.54	512.75	0.942

updated parameters through the shared space. Due to the small scale of Alexnet, the training time is short, so less counting of the time of a pull has a great impact on accuracy. If the measurement time is added to the time of a pull, the accuracy of can be increased from 78.3% and 82.3% to about 86% and 89%, respectively. The other three convolutional neural networks have a much larger scale than Alexnet, which means less counting of the time of a pull has a little impact on accuracy.

The experiment of distributed GPU platform uses two physical machines and each one has a GPU (NVIDIA quadro RTX4000). The CPU of the parameter service node is Intel ® Xeon™ E5-2660. It can be seen from Table 5 and Figure 9 that the accuracy is relatively stable and mainly around 95%. This is because the GPU has strong computing power, and its calculation time is negligible in one round of iteration time. Therefore, the measurement time is mainly composed of the communication time of parameters and gradient transmission, which can maintain a high level of accuracy and stability.

We also complete experiments of distributed heterogeneous GPU platform by adding a heterogeneous GPU (GeForce GTX 1060 6 GB), which only has close to half the computing performance of the other GPU. Since the communication time in our experimental environment is greater than the computation time, the iteration time can be formulated by formula (32). It can be seem from Table 6 and Figure 10 that the accuracy is relatively stable and over 93%. Similarly, this is because the GPU has strong computing power, and its calculation time is negligible in one round of iteration time. To further verify the modeling accuracy, we increased the network bandwidth to 10GbE, and the result is shown in Table 7 and Figure 11. It can be seem that the accuracy is relatively mainly over 80% and the iteration training time is reduced significantly by alleviating the network bottlenecks. However the accuracy decreases by

Table 5: Prediction time (PT), measurement time (MT), and accuracy on distributed GPU platform.

bs = 32	PT	MT	Accuracy	bs = 64	PT	MT	Accuracy
Alexnet	7.93	8.38	0.943	Alexnet	7.94	8.59	0.918
Vgg11	17.14	18.42	0.925	Vgg11	17.28	18.51	0.929
Vgg16	17.99	18.64	0.964	Vgg16	18.26	19.04	0.957
Vgg19	18.74	18.93	0.990	Vgg19	19.09	19.32	0.988

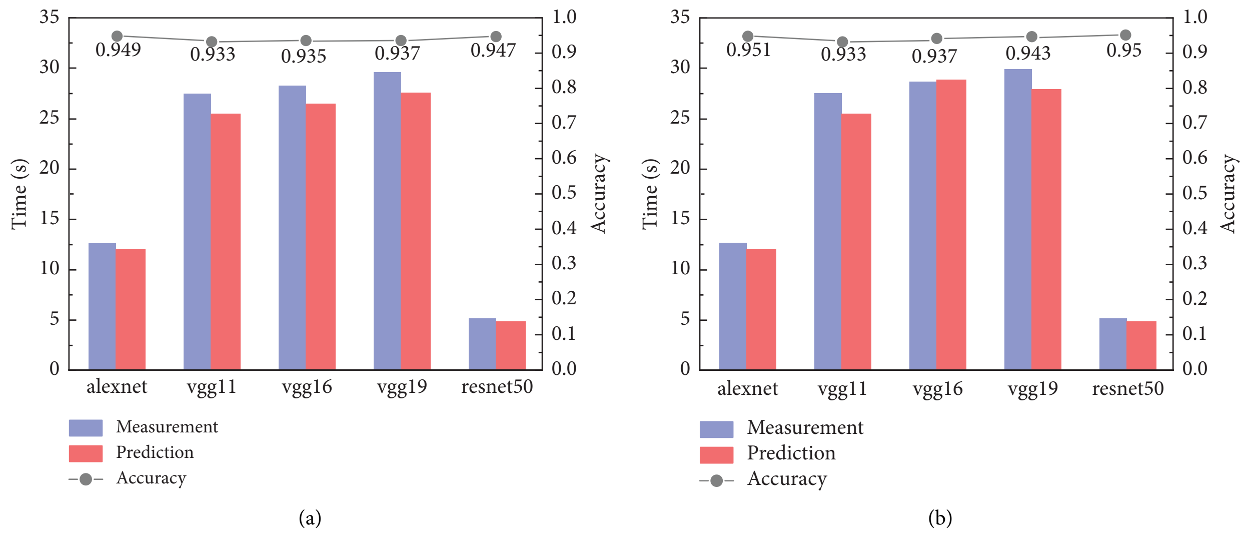


Figure 10: Comparison between prediction time and measurement time on distributed heterogeneous GPU platform with 1 GbE. (a) bs = 16 and (b) bs = 32.

Table 6: Prediction time (PT), measurement time (MT), and accuracy on distributed heterogeneous GPU platform which connected with 1GbE.

bs = 16	PT	MT	Accuracy	bs = 32	PT	MT	Accuracy
Alexnet	12.17	12.83	0.949	Alexnet	12.19	12.817	0.951
Vgg11	25.58	27.43	0.933	Vgg11	25.65	27.51	0.933
Vgg16	26.71	28.57	0.935	Vgg16	28.85	28.66	0.937
Vgg19	27.76	29.63	0.937	Vgg19	27.94	29.91	0.934
Resnet50	4.95	5.23	0.947	Resnet50	4.98	5.24	0.95

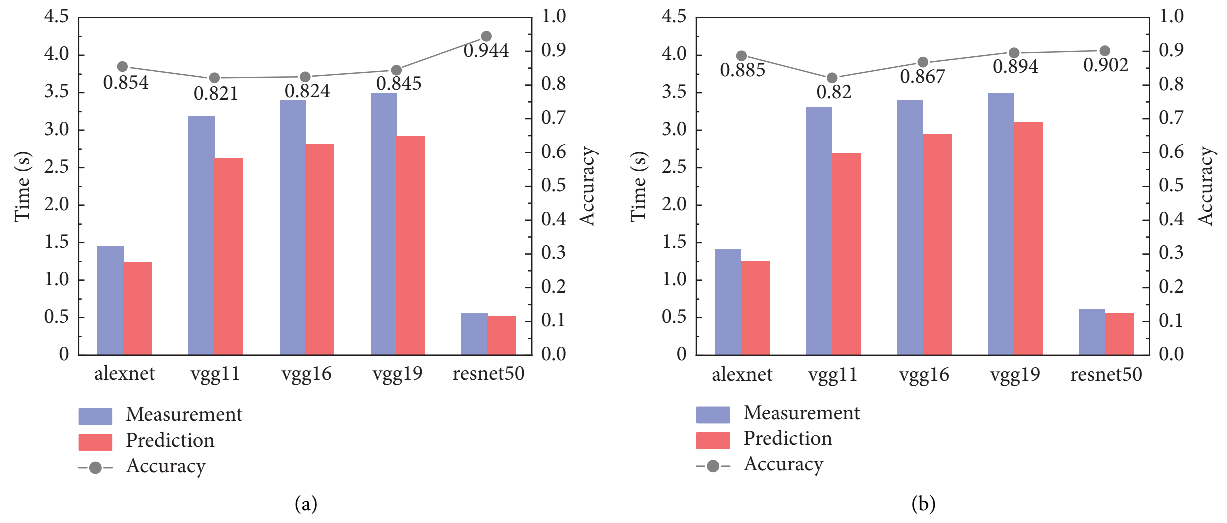


Figure 11: Comparison between prediction time and measurement time on distributed heterogeneous GPU platform with 10 GbE. (a) bs = 16 and (b) bs = 32.

Table 7: Prediction time (PT), measurement time (MT), and accuracy on distributed heterogeneous GPU platform which connected with 10 GbE.

bs = 16	PT	MT	Accuracy	bs = 32	PT	MT	Accuracy
Alexnet	1.23	1.44	0.854	Alexnet	1.23	1.39	0.885
Vgg11	2.62	3.19	0.821	Vgg11	2.69	3.28	0.820
Vgg16	2.80	3.40	0.824	Vgg16	2.94	3.39	0.867
Vgg19	2.94	3.48	0.845	Vgg19	3.11	3.48	0.894
Resnet50	0.52	0.55	0.944	Resnet50	0.55	0.61	0.902

10% compared to the result connected with 1 GbE by the below reasons:

(1) Communication time error: we experimented and found that there is startup overhead a in the pull and push process, which is not related to batch size and the bandwidth we use. In our experimental environment, the startup overhead a is about 0.5s, which cannot be ignored in our 10 GbE experiments.

(2) Computation time error: in the GPU training process, it exits the process of memory copy from host to GPU, which cannot be ignored if the local computation time of worker is short.

As for Resnet50, the startup overhead can be ignored in our experiment and percentage of computation is much higher than the others. So, it still maintains high accuracy.

In this section, we built a distributed PS system based on the Linux system and TensorFlow/benchmark. Through the use of synchronous communication algorithms, distributed measurements were carried out on various types of CPUs and GPUs. Through the analysis of the results, it is known that although the cause of the error cannot be solved by mathematical modeling, the mathematical model established in this paper still has a high level of accuracy of around 90%.

7. Conclusions and Further Study

Based on the PS architecture, this paper deeply studies various key factors that affect the performance of distributed machine learning training, and combines these factors to establish the one round of iterative time prediction model for training on stand-alone/distributed CPU/GPU platforms. In addition, this paper also designed rigorous experiments to verify that the accuracy of the proposed performance prediction model is higher than 90% in most cases. In addition, the highest accuracy rate 99.4% is achieved in the prediction of stand-alone CPU Platform with Vgg19E. As for our future work, on the one hand, considering the increase of training nodes will lead to the emergence of communication bottlenecks, we will study one round of iterative time modeling of multiparameter servers. On the other hand, we are going to incorporate local SGD synchronization communication into the scope of modeling.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The research was partially funded by the National Key R&D Program of China (2020YFB2104000), the National Natural Science Foundation of China (No. 61872132, No. 62172148), the Scientific Research Foundation of Hunan Provincial Education Department (Grant no. 21A0535), and the Natural Science Foundation of Hunan Province (Grant no. 2021JJ40635), the Natural Science Foundation of Hunan Province for Excellent Young Scholars, and the Training Program for Excellent Young Innovators of Changsha.

References

[1] J. Juszczyk, P. Badura, J. Czajkowska et al., "Automated size-specific dose estimates using deep learning image processing," *Medical Image Analysis*, vol. 68, no. 101, Article ID 101898, 2021.

[2] T. Arias-Vergara, P. Klumpp, J. C. Vasquez-Correa, E. Nöth, J. R. Orozco-Arroyave, and M. Schuster, "Multi-channel spectrograms for speech processing applications using deep learning methods," *Pattern Analysis & Applications*, vol. 24, no. 2, pp. 423–431, 2021.

[3] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2021.

[4] H. T. Joo and K. J. Kim, "Visualization of deep reinforcement learning using grad-cam: how ai plays atari games?" in *Proceedings of the 2019 IEEE Conference on Games (CoG)*, pp. 1-2, London, UK, August 2019.

[5] Y. Deng, T. Zhang, G. Lou, X. Zheng, J. Jin, and Q. L. Han, "Deep learning-based autonomous driving systems: a survey of attacks and defenses," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 7897–7912, 2021.

[6] S. Yang, F. Zhu, X. Ling, Q. Liu, and P. Zhao, "Intelligent health care: applications of deep learning in computational medicine," *Frontiers in Genetics*, vol. 12, Article ID 607471, 2021.

[7] F. Guo, B. Xu, W. A. Zhang, C. Wen, D. Zhang, and L. Yu, "Training deep neural network for optimal power allocation in islanded microgrid systems: a distributed learning-based approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2057–2069, 2022.

[8] H. Jiang, J. Starkman, Y. J. Lee, H. Chen, X. Qian, and M. C. Huang, "Distributed deep learning optimized system over the cloud and smart phone devices," *IEEE Transactions on Mobile Computing*, vol. 20, no. 1, pp. 147–161, 2021.

[9] X. Tang, W. Cao, H. Tang et al., "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2079–2092, 2022.

[10] Y. Zhang, F. McQuillan, N. Jayaram et al., "Distributed deep learning on data systems: a comparative analysis of approaches," *Proceedings of the VLDB Endowment*, vol. 14, no. 10, pp. 1769–1782, 2021.

[11] S. Alqahtani and M. Demirbas, "Performance analysis and comparison of distributed machine learning systems," 2019, http://arxiv.org/abs/1909.02061.

[12] A. Castelló, M. F. Dolz, E. S. Quintana-Ortí, and J. Duato, "Analysis of model parallelism for distributed neural networks," in *Proceedings of the 26th European MPI Users' Group Meeting*, pp. 1–10, Zurich, Switzerland, September 2019.

[13] Y. Oyama, A. Nomura, I. Sato, H. Nishimura, Y. Tamatsu, and S. Matsuoka, "Predicting statistics of asynchronous sgd parameters for a large-scale distributed deep learning system on gpu supercomputers," in *Proceedings of the 2016 IEEE International Conference on Big Data (Big Data)*, pp. 66–75, Washington, DC, USA, December 2016.

[14] Z. Pei, C. Li, X. Qin, X. Chen, and G. Wei, "Iteration time prediction for cnn in multi-gpu platform: modeling and analysis," *IEEE Access*, vol. 7, pp. 64788–64797, 2019.

[15] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, April 2017.

[16] X. Tang, C. Shi, T. Deng, Z. Wu, and L. Yang, "Parallel random matrix particle swarm optimization scheduling algorithms with budget constraints on cloud computing systems," *Applied Soft Computing*, vol. 113, no. 107, Article ID 107914, 2021.

[17] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1355–1364, Sydney, Australia, April 2015.

[18] T. Zhu, Y. Lin, and Y. Liu, "Improving interpolation-based oversampling for imbalanced data learning," *Knowledge-Based Systems*, vol. 187, Article ID 104826, 2020.

[19] M. Grossman, M. Breternitz, and V. Sarkar, "Hadoopcl2: motivating the design of a distributed, heterogeneous programming system with machine-learning applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 762–775, 2016.

[20] H. Hu, D. Wang, and C. Wu, "Distributed machine learning through heterogeneous edge systems," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 5, pp. 7179–7186, 2020.

[21] M. Abadi, A. Agarwal, P. Barham et al., "Tensorflow: large-scale machine learning on heterogeneous distributed systems," 2016, http://arxiv.org/abs/1603.04467.

[22] Z. Han, H. Tan, S. H. C. Jiang et al., "Spin: bsp job scheduling with placement-sensitive execution," *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2267–2280, 2021.

[23] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[24] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed {DNN} training in heterogeneous GPU/CPU clusters," in *Proceedings of the 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 463–479, Carlsbad, CA, USA, November 2020.

[25] 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.

[26] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter, "Blueconnect: decomposing all-reduce for deep learning on heterogeneous network hierarchy," *IBM Journal of Research and Development*, vol. 63, no. 6, p. 1, 2019.

[27] H. Shi, Y. Zhao, B. Zhang, K. Yoshigoe, and A. V. Vasilakos, "A free stale synchronous parallel strategy for distributed machine learning," in *Proceedings of the 2019 International Conference on Big Data Engineering*, pp. 23–29, Hong Kong, China, June 2019.

[28] X. Zhao, A. An, J. Liu, and B. X. Chen, "Dynamic stale synchronous parallel distributed training for deep learning," in *Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1507–1517, Dallas, Texas, USA, July 2019.

[29] J. Dean, "Large-scale deep learning for building intelligent computer systems," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, San Francisco, CA, USA, February 2016.