SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning

Article *in* Applied Sciences · December 2021

DOI: 10.3390/app12010292

CITATIONS
3

READS
42

2 authors:

Yunyong Ko
Chung-Ang University
18 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Sang-Wook Kim
Hanyang University
412 PUBLICATIONS **4,166** CITATIONS
SEE PROFILE

applied sciences

MDPI

Article

SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning

Yunyong Ko and Sang-Wook Kim *

Department of Computer Science, Hanyang University, Seoul 04763, Korea; koyunyong@hanyang.ac.kr
* Correspondence: wook@hanyang.ac.kr

Abstract: The recent unprecedented success of deep learning (DL) in various fields is underlied by its use of large-scale data and models. Training a large-scale deep neural network (DNN) model with large-scale data, however, is time-consuming. To speed up the training of massive DNN models, data-parallel distributed training based on the parameter server (PS) has been widely applied. In general, a synchronous PS-based training suffers from the synchronization overhead, especially in heterogeneous environments. To reduce the synchronization overhead, asynchronous PS-based training employs the asynchronous communication between PS and workers so that PS processes the request of each worker independently without waiting. Despite the performance improvement of asynchronous training, however, it inevitably incurs the difference among the local models of workers, where such a difference among workers may cause slower model convergence. Fro addressing this problem, in this work, we propose a novel asynchronous PS-based training algorithm, SHAT that considers (1) the scale of distributed training and (2) the heterogeneity among workers for successfully reducing the difference among the local models of workers. The extensive empirical evaluation demonstrates that (1) the model trained by SHAT converges to the higher accuracy up to 5.22% than state-of-the-art algorithms, and (2) the model convergence of SHAT is robust under various heterogeneous environments.

Keywords: distributed deep learning; data parallelism; PS-based distributed training; heterogeneous environments

Citation: Ko, Y.; Kim, S.-W. SHAT: A Novel Asynchronous Training Algorithm That Provides Fast Model Convergence in Distributed Deep Learning. *Appl. Sci.* **2022**, *12*, 292. https://doi.org/10.3390/app12010292

Academic Editors: Jinho Kim and Young-ho Park

Received: 3 November 2021
Accepted: 23 December 2021
Published: 29 December 2021

1. Introduction

With the increasing size of training data and models, deep learning (DL) techniques have successfully solved the problems that traditional AI techniques did not solve well in various research fields such as computer vision, speech recognition, and natural language processing. The recent deep neural network (DNN) models often consist of millions to billions of parameters [1–4]. For instance, VGG-16 [2] is with 138 M parameters, BERT [3] with 345 M parameters, and GPT-3 [4] with 175 B parameters. Though the great advance of a computational accelerator such as GPU has dramatically improved the performance of training DNN models, training such a large model with a large amount of data often requires days and weeks [3–5].

To speed up such training, *data-parallel distributed training based on the parameter server (PS) framework* has been widely adopted [6–11]. Figure 1 shows the architecture of the PS-based distributed training. In the PS-based distributed training, the training data are split and distributed into multiple workers, each of which has its local model with the same DNN architecture. The training process of the PS-based distributed training is as follows: (1) each worker computes the loss of its local model based on its local data (i.e., forward pass), (2) each worker computes the gradients based on the loss (i.e., backward pass), (3) each worker sends its gradients to PS, (4) PS averages the gradients of all workers and applies the averaged gradients to the global model, and (5) PS synchronizes the models of all workers by broadcasting the updated global model to them. This process guarantees

that all workers always have the exactly same models across the entire training, which helps to speed up the model convergence.

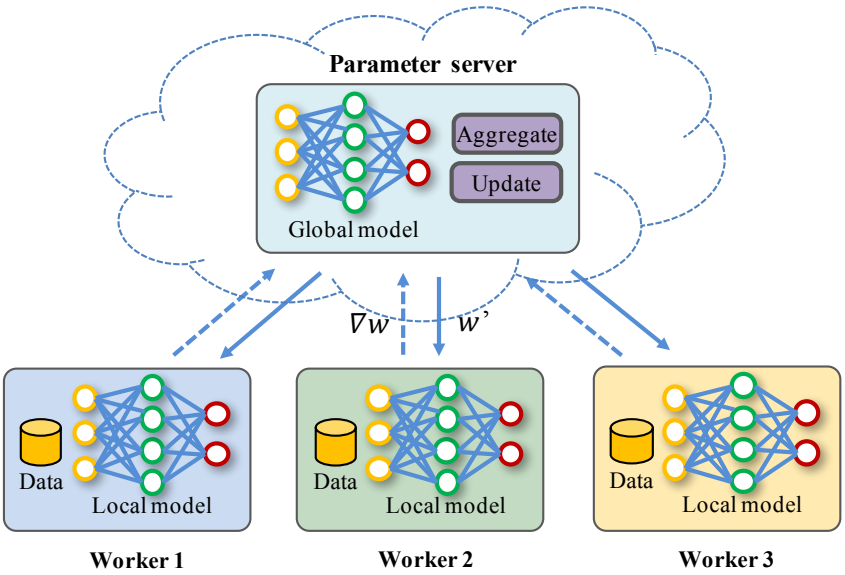


Figure 1. The architecture of a distributed training based on the parameter server framework.

While the synchronous PS-based distributed training [12,13], in general, has a good convergence rate thanks to its model synchronization across all workers, the synchronization overhead becomes larger as the number of workers and the size of a model increase, which may degrade the training performance. According to [14,15], the synchronization overhead is more than 70% of the entire training when the VGG-16 model training with more than 8 workers. This performance issue can be more serious in heterogeneous environments, where there are workers with different training speeds [16,17].

To reduce the synchronization overhead, a number of distributed training algorithms adopting asynchronous communication have been studied [6–8,10,11,18,19]. These algorithms aim to improve the training performance by reducing the synchronization overhead while maintaining the degradation of the model accuracy as small as possible. In general, the model accuracy and training performance are in a trade-off relationship; that is, reducing the overhead by asynchronous communication improves the training performance, but it inevitably results in a significant difference among the local models of workers. Such a difference can delay the convergence of the global model since the gradients computed from the local models of different workers may interfere with each other [7,11,14]. In addition, the difference among the local models of workers tends to become larger as the scale of distributed training increases (i.e., a larger number of workers), especially in heterogeneous environments. Therefore, in asynchronous distributed training, it is important to reduce the difference among the local models of workers by considering the scale of distributed training and the heterogeneity among workers. However, existing asynchronous training algorithms [6–10] do not consider the scale of distributed training and the heterogeneity among workers when they update the local model of each worker. By these limitations, in large-scale distributed heterogeneous environments, existing asynchronous training algorithms often have the problem of the delayed model convergence [11,14].

To address the limitations, in this work, we propose a novel approach to asynchronous PS-based distributed training, the **S**cale and **H**eterogeneity aware **A**synchronous distributed **T**training algorithm (**SHAT**) that takes into account (1) the scale of distributed training and (2) the heterogeneity among workers in updating the local model of each worker, for effectively reducing the difference among the local models of workers in asynchronous distributed training. We demonstrate that (1) the model trained by SHAT converges to the higher accuracy up to 5.22% than the existing state-of-the-art algorithms,

and (2) the model convergence of SHAT is robust under various heterogeneous environments including the cluster with an extremely (×100) slower worker. To the best of our knowledge, this is the first work to consider the scale of distributed training and the heterogeneity among workers at the same time for fast model convergence in asynchronous training. SHAT can achieve fast model convergence regardless of the scale of distributed training and the heterogeneity among workers.

The main contributions of this work are as follows:

- Identifying the limitations of existing asynchronous distributed training in updating local models of workers—i.e., not considering both the scale of distributed training and the heterogeneity among workers.
- Proposing a novel asynchronous PS-based training algorithm, named as SHAT, successfully addressing the limitations of the existing asynchronous distributed training.
- Comprehensive evaluation verifying the effectiveness of SHAT in terms of the convergence rate and robustness under heterogeneous environments.

Organization. The rest of this paper is organized as follow. We describe data-parallel distributed training in detail and introduce literature related to data-parallel distributed training in Section 2. Then, we describe the limitations of existing asynchronous distributed training algorithms and present a novel algorithm for effectively addressing the limitations in Section 3. We empirically evaluate SHAT in Section 4. Finally, we conclude this paper in Section 5.

2. Related Work

With the increasing scale of models and data, a large number of studies on distributed training have been conducted [6–10,15,19–26]. They can be classified into data parallelism [6–8,10,11] and model parallelism [15,25]. This work focuses on data-parallel distributed training, where training data are split and distributed into multiple workers, while all workers have the model with the same DNN architecture. In data-parallel distributed training, each iteration consists of computation and communication stages. In a computation stage, a worker trains its local model based on its local data. In a communication stage, the training results (i.e., gradients) are aggregated via network communication. For this aggregation, either communication via a dedicated parameter server (PS) (i.e., PS-based), or peer-to-peer communication (i.e., P2P-based) can be used. On the other hand, the communication for the aggregation can be performed either synchronously or asynchronously. Table 1 shows the existing data-parallel distributed training algorithms. In this section, we introduce existing distributed training algorithms (i.e., PS-based and P2P-based) as shown in Table 1, and various techniques for improving distributed training.

Table 1. Existing data-parallel distributed training algorithms.

	PS-Based.	**P2P-Based.**
Synchronous.	Bulk Synchronous Parallel (BSP) [12]	AllReduce-SGD (AR-SGD) [27]
Asynchronous.	Asynchronous Parallel (ASP) [6] Stale Synchronous Parallel (SSP) [7] Elastic Averaging SGD (EASGD) [8] Dynamic SSP (DSSP) [9]	Gossip-SGD (GoSGD) [20] Decentralized Parallel SGD (D-PSGD) [21] Asynchronous D-PSGD (AD-PSGD) [22] Stochstic Gradient Push (SGP) [23]

2.1. PS-Based Distributed Training

In PS-based distributed training, a dedicated parameter server (PS) manages the global parameters by aggregating the training results of workers. In bulk synchronous parallel (BSP) [12,13], a synchronous centralized algorithm, the parameters of all workers are synchronized via PS that aggregates the gradients from all workers at once. Thus, all workers train the model in a consistent direction based on the same parameter values. However, the synchronization overhead can be significant, especially in a heterogeneous cluster, where some workers have relatively slow training speeds. To reduce the synchronization

overhead, many asynchronous distributed training algorithms [6–9,18,28,29] have been studied. In asynchronous parallel (ASP) [6], PS aggregates the gradients from workers in an asynchronous manner: PS does not wait for all workers; instead, PS processes the gradients from each worker individually. However, the parameter variance among workers could be large if there are workers with different training speeds, causing global model convergence delayed. Stale synchronous parallel (SSP) [7], positioned in the middle of BSP and ASP, relaxes the parameter synchronization by allowing workers to train with different parameter versions. However, SSP controls the difference among the versions to be always less than the predefined threshold. Dynamic SSP (DSSP) [9] further improves SSP by varying the threshold dynamically as training progresses. Elastic averaging SGD (EASGD) [8] reduces the communication overhead via periodic communication between PS and workers.

2.2. P2P-Based Distributed Training

In P2P-based distributed training, the training results of workers are aggregated via peer-to-peer (P2P) communication without PS. In AllReduce-SGD (AR-SGD) [27,30], a synchronous P2P-based distributed algorithm, the parameters of all workers are synchronized via AllReduce communication at every iteration. The synchronization overhead, however, can be a big problem like BSP. In decentralized parallel SGD (D-PSGD) [21], each worker exchanges its parameters with another worker and updates its parameters by averaging them at every iteration. D-PSGD proved that the convergence rate of P2P-based distributed training is comparable to that of SGD through the theoretical analysis, which promoted to study many P2P-based distributed training algorithms. AD-PSGD [22], the asynchronous version of D-PSGD, improves the training performance of D-PSGD by allowing workers to communicate with each other asynchronously. The stochastic gradient push (SGP) [23], a state-of-the-art P2P-based distributed training algorithm, adopts the push-sum gossip algorithm [31] to efficiently aggregate parameter aggregation of workers. To further accelerate the aggregation, SGP uses a directed exponential graph where a worker communicates with another worker placed in 2^i hops away at iteration i. The Cooperative-SGD [32] proposes a unified framework for generalizing existing distributed training algorithms and provides its convergence analysis within the generalized framework, where fully-synchronous SGD (BSP, AR-SGD), EASGD, and D-PSGD are represented as its special cases. Ref. [33] provides a general consistency condition that covers existing asynchronous distributed training algorithms.

2.3. Other Techniques for Distributed Training

For efficient distributed training, many techniques have been studied from various perspectives. In general, the PS-based distributed training algorithms often suffer from the problem of PS being a bottleneck because PS aggregates training results from all workers. To mitigate this problem, parameter sharding [10,29,34] is generally applied to PS-based distributed training algorithms. This technique divides global parameters and distributes them into multiple PSs to process them in parallel. On the other hand, To speed up the model convergence in synchronous distributed training, various methods that control the learning rate in distributed training have been widely studied [26,27,35–39]. Linear scaling with warm-up (LSW) [27] uses the linear learning rate scaling rule that linearly increases the learning rate as the batch size increases, together with the gradual warm-up that gradually increases the learning rate from a small value. AdaScale SGD [26] further improves large-batch training by applying more reliable learning rate considering the variance of gradients. Eshraghi and Liang [17] and Yu et al. [16] consider distributed training over different types of networks. Distributed any-batch mirror descent (DABMD) [17] speeds up the training over a heterogeneous network, by varying the batch sizes across workers to minimize the waiting time by faster workers. Reliable parameter server (RPS) [16] assumes unreliable networks where the delivery of the messages between workers is not guaranteed. Through theoretical analysis, Yu et al. [16] shows the convergence rate of the PS-based distributed training algorithm over unreliable networks is comparable to that of reliable networks.

POSEIDON [10], iBatch [19], G-Pipe [25], and PipeDream [15] were proposed to maximize the training efficiency by overlapping the computation and communication.

3. The Proposed Method: SHAT

In this work, we consider the following problem:

$$\min_{w^1,\ldots,w^n,\tilde{w}} \sum_{i=1}^{n} \mathbb{E}[F(w^i, x^i)] + \frac{\rho}{2}||w^i - \tilde{w}||^2, \quad (1)$$

where n is the number of workers, w^i is the local model of worker i, x^i is the local data of worker i, $F(w^i, x^i)$ is the loss function of the local model w^i with data samples x^i, and $\tilde{x}$ is the global model. The goal of Equation (1) is twofold: (1) the first term minimizes the loss function (thus maximizing accuracy) and (2) the second term minimizes the difference between local and global models (thus preserving the consistency among parameters across workers). This work aims to achieve both objectives in Equation (1) and successfully train a DNN model in asynchronous PS-based distributed training.

3.1. Asynchronous Distributed Training

In this section, we review asynchronous distributed training [6,7] and identify the limitation in updating local models of workers. Figure 2 illustrates the difference between synchronous and asynchronous distributed training. In synchronous distributed training, the local models of all workers are synchronized by PS aggregating the gradients from all workers (see Figure 2a). In each iteration, each worker performs computing operations (i.e., forward/backward passes), sends its computed gradients to PS, and receives the updated global model from PS. Then, each worker replaces its local model with the global model and proceeds to the next training iteration. Here, all workers have the exactly same model since they receive the same global model from PS at the same time. However, the synchronization overhead might be quite significant when some workers fall behind other workers for computing operations (e.g., worker 1 in Figure 2a), which may degrade the overall training performance significantly.

To improve the performance of synchronous training, asynchronous distributed training aims to reduce the synchronization overhead by adopting asynchronous communication between PS and workers. In asynchronous distributed training, PS processes the gradients from each worker in an asynchronous manner, that is, PS receives the gradients of each worker, updates the global model using them, and then sends the updated global model right back to the worker immediately. Thus, each worker can proceed to the next iteration without waiting for other workers. As clearly illustrated in Figure 2b, in asynchronous distributed training, there is no synchronization barrier.

However, each worker receives a different version of the global model at a different point since it communicates with PS independently. Such an inevitable difference would increase as the scale of distributed training and the heterogeneity among workers get larger, which would adversely affect the global model convergence (thus, delaying the model convergence). This is because the gradients computed based on the different local models of workers lead to different learning directions, that is, the larger the difference among local models of workers is, the more different directions of gradients are. When the gradients with different directions are applied to the global model in PS, they are likely to interfere with each other in the global model convergence [7,11,14]. More specifically, when there is an extremely slow worker (i.e., straggler) in a distributed cluster, other normal workers can update the global model many times while the straggler computes its gradient locally. Thus, the gradient computed by the straggler becomes stale and is highly likely to lead the global model in PS to the wrong direction. Therefore, it is important to reduce the difference among local models of workers by accurately considering the scale of distributed training and the heterogeneity among workers in asynchronous distributed training.

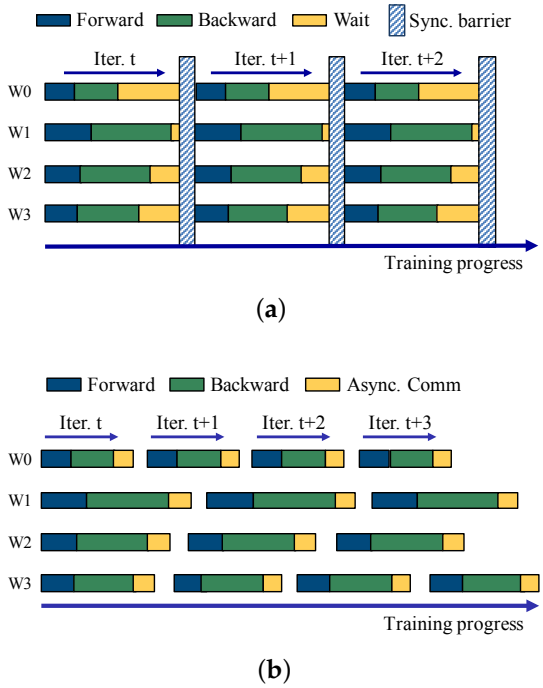


Figure 2. The difference between synchronous and asynchronous distributed training; (**a**) synchronous distributed training; (**b**) asynchronous distributed training.

In existing asynchronous distributed training, however, each worker just replaces its local model with the global model. Thus, when updating the local model of each worker, it does not consider (1) how many workers are joining in distributed training (i.e., the scale of distributed training) and (2) their performance difference (i.e., the heterogeneity among workers), which may cause a serious problem in the model convergence as we explained. Motivated from these limitations, we propose an update strategy for the local model of each worker to speed up the model convergence in asynchronous distributed training by considering the scale of distributed training and the heterogeneity among workers together.

3.2. Update Strategy for Model Convergence

By updating the local model of each worker using the global model, the local models of all workers could be mixed indirectly through the global model. Thus, we aim to reduce the difference between the global model and the local model of each worker by effectively mixing the local models of workers. To this end, inspired by [8,11], we define the generalized update rule for local model of each worker w^i in asynchronous distributed training, based on the problem formulation represented in Equation (1). By taking the gradient descent with respect to w^i on Equation (1), we get:

$$w^i = w^i - \eta \cdot (g^i + \rho(w^i - \tilde{w})) \tag{2}$$

Denote $\alpha = \eta \cdot \rho$ and $w^i = w^i - \eta \cdot g^i$, which can be considered equal to the local parameter update of worker i. Then, we get the following update rule for w^i:

$$\begin{aligned} w^i &= w^i - \alpha(w^i - \tilde{w}) \\ &= (1 - \alpha)w^i + \alpha\tilde{w} \end{aligned} \tag{3}$$

Equation (3) implies that a worker updates its local model by taking the weighted average between its local model w^i and the global model $\tilde{x}$ where α is an oscillating weight factor between local and global models. We note that this formulation can generalize diverse parameter update rules in distributed training. For example, if $\alpha = 0$, it is the same as local training only (i.e., ensemble), while, if $\alpha = 1$, it degenerates to the update method

in existing asynchronous distributed training (e.g., Hogwild [6]), where the local model of each worker is completely replaced by the global model.

With the generalized update rule represented in Equation (3), we can transform the problem of asynchronous distributed training to the problem of setting the oscillating weight factor α. Now, let us describe the update strategy for the local model of each worker (i.e., how to set α) in SHAT. The proposed update strategy is based on the following intuition: the less the amount of the gradients of a worker is applied to the global model in PS, the larger the difference between the local model of the worker and the global model becomes. By this intuition, the difference may increase more as the number of workers gets larger (i.e., the larger scale of distributed training) and workers have more different training speeds (i.e., the larger heterogeneity among workers). Then, we take into account (1) the scale of distributed training and (2) the heterogeneity among workers in updating the local model of each worker, for effectively reducing the difference. Given the number of workers n and the degree of the staleness of each worker s^i, we set the weight of each worker α^i as follows.

$$\alpha^i = 1 - \frac{s^i}{\log n}, \quad s^i = \frac{n}{c^i}. \tag{4}$$

Here, we define the degree of the staleness of each worker as $s^i = \frac{n}{c^i}$, where c^i is the number of updates by other workers in the global model since the previous update by worker i has been performed (i.e., $s^i \approx 1, c^i \approx n$ in a homogeneous cluster, where all workers have almost same training speed). Thus, in SHAT, the weight of the global model for updating the local model of each worker, α^i is determined by the scale of distributed training and the heterogeneity among workers. The weight of the global model increases with a more number of workers and a less degree of heterogeneity among workers, helping (1) the local models of workers to be combined more effectively and (2) the local model of each worker to catch up quickly the learning direction of the global model. We note that α^i is not a hand-tuned hyperparmeter but automatically decided by the number of workers n and the degree of the staleness of each worker s^i. As a result, in SHAT, each worker updates its local model by considering the scale of distributed training and the heterogeneity among workers (i.e., with the staleness of each worker), not just replacing its local model by the global model.

In summary, existing asynchronous training algorithms such as ASP [6], SSP [7], and DSSP [9] mainly focus on improving the training performance (i.e., throughput) via an asynchronous communication strategy. They adopt a simple update method for a local model of each worker: the local model of a worker is completely replaced with the global model (i.e., $\alpha^i = 1$). Thus, the existing asynchronous training algorithms do consider neither the scale of distributed training nor the heterogeneity among workers when they update the local model of each worker, which would result in delayed model convergence. On the other hand, this work aims to speed up the model convergence by reducing the difference among local models of workers. To this end, we define our generalized update rule (Equation (3)) for the local model of each worker in asynchronous distributed training and propose a new update strategy that considers the scale of distributed training and the heterogeneity among workers together. We will empirically verify the effectiveness of SHAT on the model convergence in Section 4.

3.3. Algorithm and Performance Consideration

Algorithm 1 shows the whole training process of SHAT. At iteration t, a worker i computes its gradients g_t^i based on the data sampled from X, sends g_t^i to PS, and receives the updated global model $\tilde{w}$ and c^i (lines 3–6 in Algorithm 1). Then, the worker updates its local model using our strategy represented in Equation 4 (lines 7–8 in Algorithm 1). While, whenever PS receives the gradients g^i from each worker, PS computes the degree of the heterogeneity of the other workers $c^k (k \neq i)$ (lines 13–19 in Algorithm 1). Then, PS updates the global model by applying the received gradients and sends the updated global model

$\tilde{w}$ and the degree of the staleness c^k to the worker. We highlight again that in SHAT, each worker updates its local models by taking into account (1) the scale of distributed training and (2) the heterogeneity among workers.

Algorithm 1 Training processes of a worker and PS in SHAT

Function SHAT_WORKER(w_0, X, f, i):
  for $t = 0, 1, \ldots$ **do**
    $b \leftarrow \text{sampleBatch}(X)$
    $g_t^i \leftarrow \frac{1}{|b|} \sum_{x \in b} \nabla F(w_t, x)$
    $\text{sendGradientsToPS}(g_t^i)$
    $\tilde{w}, c^i \leftarrow \text{receiveModelFromPS}()$
    $\alpha^i \leftarrow 1 - \frac{s^i}{\log n}, \quad s^i \leftarrow \frac{n}{c^i}$ // *Scale and heterogeneity aware strategy*
    $w^i \leftarrow (1 - \alpha^i) w^i + \alpha^i \tilde{w}$ // *Generalized update rule*
  end for

Function SHAT_PS(n, η):
  $C \leftarrow \{0, 0, \ldots, 0\}, C \in \mathbb{N}^n$
  $c^i \in C$
  for $s = 0.1, \ldots$ **do**
    Receive g^i from worker i
    $g^i \leftarrow \text{receiveGradientFromWorker}()$
    for $k = 0, 1, \ldots, n-1$ **do** // *Computing the staleness of each worker*
      if $k \neq i$ **then**
        $c^k \leftarrow c^k + 1$
      end if
    end for
    $\tilde{w} \leftarrow \tilde{w} - \eta \cdot g^i$
    $\text{sendUpdateModelToWorker}(\tilde{w}, c^i)$
    $c^i \leftarrow 0$
  end for
  return $\tilde{w}$

In PS-based distributed training, PS may become the bottleneck of the entire training since a single PS is in charge of aggregating the training results (i.e., gradients) from all workers and sending back the updated global model to each worker. To alleviate this problem, parameter sharding [10,28,29,34,40] is generally applied to PS-based distributed training. Parameter sharding divides the parameters in the global model and distributes them into multiple PSs to process them in parallel as illustrated in Figure 3. Via parameter sharding, the communication and computation overhead are distributed to multiple PSs, thus improving the overall training performance.

Layerwise parameter sharing is normally used because a layer is represented as a single data structure in many DNN frameworks (e.g., Tensor in Tensorflow) so that it can be processed sequentially, which means that the parameters in the same layer are stored in the same PS. In some DNN models, however, a single layer may contain the majority of the entire model parameters. For instance, in the VGG-16 model [2], the last two fully-connected layers are in charge of more than 90% of all parameters. In this case, the PS in charge of the extremely large layers suffers from a big burden, thereby becoming a bottleneck in the overall training. To address this issue, we apply memorywise parameter sharding to SHAT. Memorywise sharding defines the maximum size of sharded parameters and has multiple PSs process the parameters in a layer together if it is larger than the maximum size.

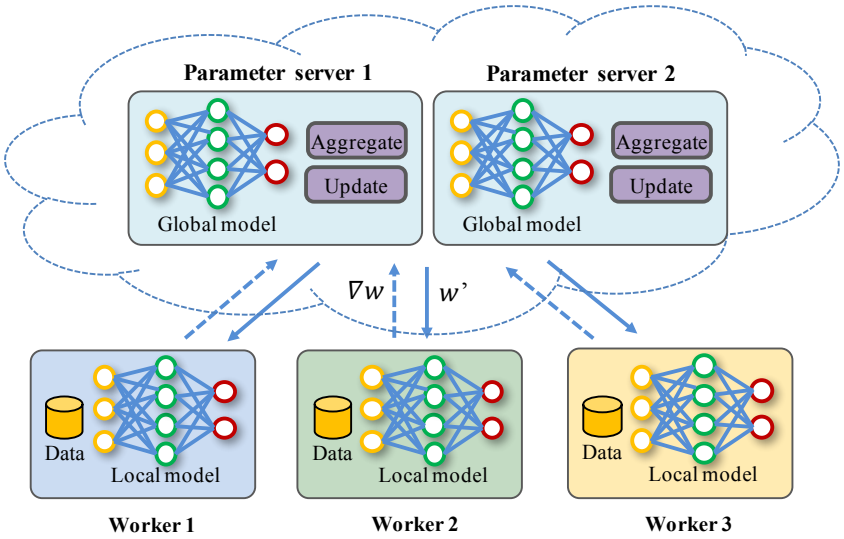


Figure 3. Parameter sharding.

4. Experimental Validation

In this section, we evaluate SHAT by answering the following evaluation questions:

- EQ1. Does the model trained by SHAT achieve the accuracy higher than those of the state-of-the-art methods?
- RQ2. How robust is SHAT to various heterogeneous environments in terms of the model convergence?
- RQ3. How effective are parameter sharding techniques on SHAT in terms of training performance?

4.1. Experimental Setup

4.1.1. Datasets and Models

We evaluate SHAT with two widely used CNN models, ResNet-50 [1] and VGG-16 [2]. ResNet-50 with 23 M parameters is a computation-intensive model, while VGG-16 with 138 M parameters is a communication-intensive model. As the training datasets for both models, we use the CIFAR dataset that consists of 50 K training images and 10 K test images with 10 labels (https://www.cs.toronto.edu/~kriz/cifar.html, accessed on 22 December 2021).

4.1.2. Competing Methods

We compare SHAT with the following three baseline methods, including the two existing asynchronous PS-based training methods and a variation of SHAT.

- ASP (i.e., HOGWILD) [6]: this baseline is a widely-recognized data-parallel distributed training method, where each worker updates its local model by replacement with the global model (i.e., $\alpha = 1$).
- ENSEMBLE: this baseline is an ensemble learning method, where each worker trains its local model only locally without receiving the global model (i.e., $\alpha = 0$).
- SHAT^{Root}: this baseline is a variation of SHAT with $\alpha^i = 1 - \frac{h^i}{\sqrt{n}}$ for each worker.
- SHAT: this method is the original version of SHAT, the asynchronous PS-based training with $\alpha^i = 1 - \frac{h^i}{\log n}$ for each worker.

4.1.3. Hyperparameter Settings

We set batch size B as 128 for ResNet-50 and 96 for VGG-16 to fully utilize the GPU memory. We use momentum SGD and set momentum as 0.9, weight decay factor as 0.0001, and learning rate η as $0.01 \times n$ for CIFAR-10, based on the learning rate scaling rule [27].

We apply the learning rate warm-up for the first 20 epochs for the CIFAR dataset, and decay η by $\frac{1}{10}$ at epoch 150 for CIFAR-10.

4.1.4. System Configuration

We use TensorFlow 1.15 and MPICH 3.1.4 to implement all algorithms including SHAT on Ubuntu 18.04 OS. We evaluate SHAT on the cluster with four machines, where each machine has two NVIDIA RTX 2080 Ti GPUs and an Intel i7-9700k CPU with 64 GB memory. All machines are interconnected by 10 Gbps Ethernet (Mellanox ConnectX-4Lx).

4.2. EQ1. Model Accuracy and Convergnece

First, we evaluate the model accuracies of all competing methods. We train ResNet-50 and VGG-16 on CIFAR-10 (250 epochs) using all competing methods and measure Top-1 test accuracy (%). Table 2 shows the results. The results demonstrate that the models trained by SHAT converge to higher accuracies than those of the baseline methods. In particular, SHAT achieves the higher accuracy up to 5.22% than the existing asynchronous distributed training (i.e., ASP). This result implies that SHAT successfully reduces the difference among the local models of workers in asynchronous distributed training.

Table 2. The model accuracy of asynchronous training with different update methods.

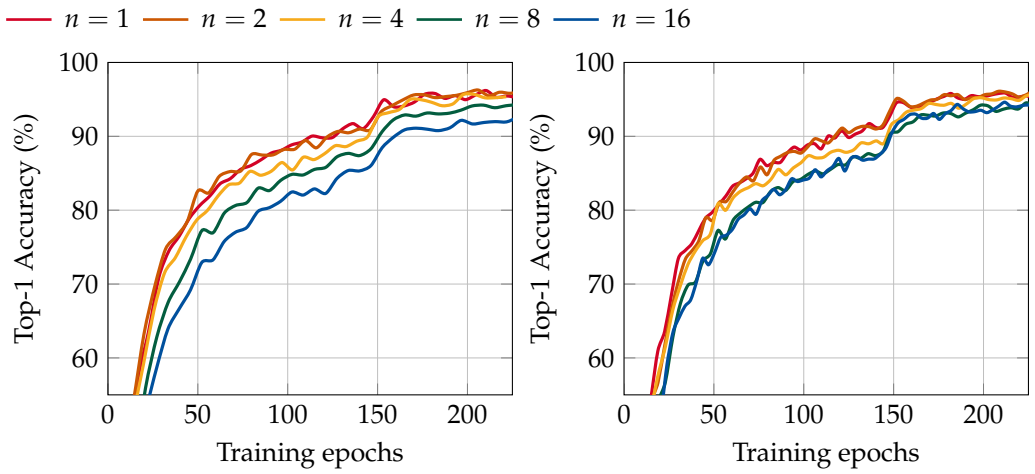
Model	# of Workers	Competing Methods			
		ENSEMBLE ($\alpha = 0$)	**ASP** ($\alpha = 1$)	**SHAT**^{Root} ($\alpha^i = 1 - \frac{h^i}{\sqrt{n}}$)	**SHAT** ($\alpha^i = 1 - \frac{h^i}{\log n}$)
ResNet-50 [1]	2	0.9319	0.9332	0.9384	**0.9399**
	4	0.9112	0.9315	0.9329	**0.9382**
	8	0.8910	0.9182	0.9293	**0.9311**
	16	0.8491	0.9019	0.9113	**0.9311**
VGG-16 [2]	2	0.9012	0.9209	**0.9242**	0.9238
	4	0.8821	0.9171	0.9223	**0.9228**
	8	0.8247	0.9071	0.9128	**0.9226**
	16	0.7002	0.8687	0.9081	**0.9209**

For more in-depth evaluation for model convergence of SHAT, we also measure the model accuracy of SHAT and ASP in terms of training epochs, with varying the number of workers n. Figure 4 shows the results on the training of ResNet-50 and VGG-16 on CIFAR-10, where the x-axis represents the training epoch and the y-axis represents the top-1 accuracy. SHAT always outperforms ASP in terms of the epochwise convergence rate, and the gap between SHAT and ASP tends to become larger as the number of workers increases. This is because the update strategy of SHAT carefully considers the scale of distributed training (i.e., $\log n$ in Equation (4)), and then update the local model of each worker precisely based on them. As a result, SHAT successfully improves the model convergence of asynchronous distributed training, which verifies that SHAT successfully addresses one of the limitations of the existing asynchronous distributed training—i.e., not considering the scale of distributed training—as we claimed.

4.3. EQ2. Robustness to Heterogeneous Environments

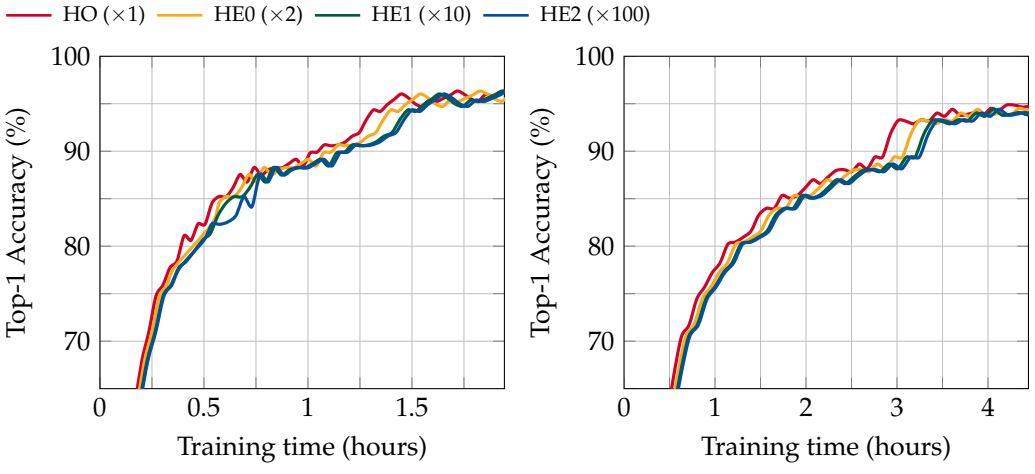
In a heterogeneous environment, there are workers with varying training speeds, which may adversely affect the model convergence of asynchronous distributed training. For example, when there are a few workers with slow training speeds, the gradients computed from the slow workers (i.e., staled gradients) are highly likely to cause the global model to be learned in a wrong direction, thereby delaying the model convergence. Thus, distributed training algorithms should be evaluated in this aspect. To evaluate SHAT in this aspect, we configure one homogeneous and three heterogeneous clusters: (1) **HO**: a homogeneous cluster with no slow workers, (2) **HE0**: a heterogeneous cluster with a $\times 2$

slower worker, (3) **HE1**: a heterogeneous cluster with a $\times 10$ slower worker, and (4) **HE2**: a heterogeneous cluster with a $\times 100$ slower worker. We train the ResNet-50 model using SHAT on the four clusters, and measure their accuracies with respect to the training time. Figure 5 shows the results, where the x-axis represents the training time and the y-axis represents the top-1 accuracy. Clearly, *SHAT is quite robust to all heterogeneous clusters in terms of the model convergence*. In particular, regardless of the degree of heterogeneity, SHAT always shows a consistent convergence rate. Note that the small loss is inevitable because of the slow worker. This result indicates that SHAT effectively reduces the difference among the local models of workers, by considering the heterogeneity among workers (i.e., the staleness of each worker c^i).



(**a**) (**b**)

Figure 4. Comparison of convergence rate with respect to training epochs for ResNet-50 on CIFAR-10; (**a**) ASP ($\alpha = 1$); (**b**) SHAT ($\alpha^i = 1 - \frac{h^i}{\log n}$).



(**a**) (**b**)

Figure 5. Convergence rate of SHAT under three different heterogeneous clusters; (**a**) ResNet-50; (**b**) VGG-16.

4.4. EQ3. Effects of Parameter Sharding

As the number of workers in a distributed cluster increases, the communication overhead required in distributed training inevitably increases as well. With the increase of the communication overhead, the training throughput per unit time of each worker decreases, which may adversely affect the training performance of the entire distributed training. To efficiently process the communication overhead, as explained in Section 3.3, we apply the parameter sharding technique to SHAT. In this experiment, we evaluate

the scalability of SHAT with the increasing number of workers and the effectiveness of parameter sharding techniques. We compare the following three versions of SHAT. (1) **No-sharding**: SHAT without parameter sharding, where a single PS processes all parameters in the training model; (2) **Layerwise**: SHAT with the layerwise parameter sharding, where four PSs are in charge of the same number of layers (but different numbers of parameters). (3) **Memorywise**: SHAT with the memorywise parameter sharding, where four PSs are in charge of the same number of parameters. We measure the training throughput of each method with numbers of workers, in the ResNet-50 and VGG-16 training on CIFAR-10. Figure 6 shows the results, where the x-axis represents the number of workers and the y-axis represents the speedup results.

Clearly, for both models, the memorywise parameter sharding improves the training throughput of SHAT the most. This indicates that the memory-wise parameter sharding successfully splits large layers and distributes sharded parameters to PSs evenly, which helps to mitigate the PS bottleneck problem. In the case of the ResNet-50 training, the layerwise sharding improves the training throughput of SHAT well. This is because the numbers of parameters are similar to each other across layers in ResNet-50. On the other hand, in the case of the VGG-16 training, the layerwise sharding rarely improves the training performance since a few layers have more than 90% of all parameters which leads to the problem of PS being a bottleneck.

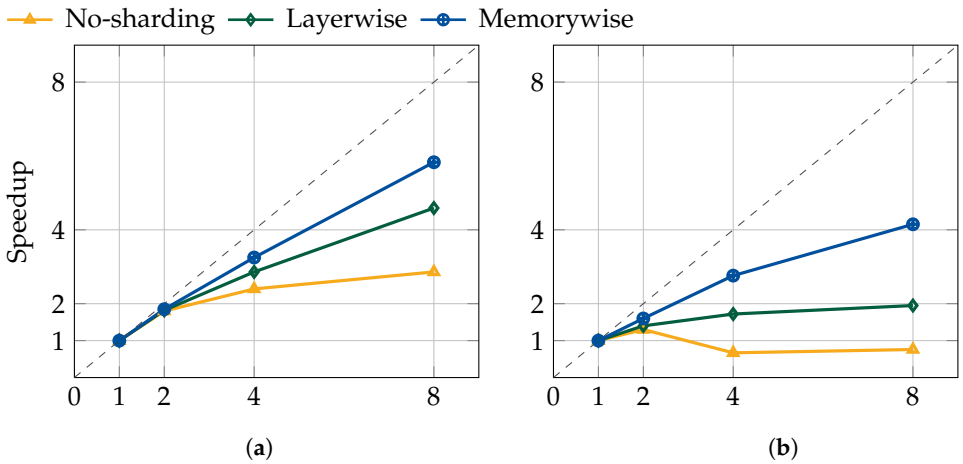


Figure 6. Effects of parameter sharding techniques on the training performance of SHAT; (**a**) ResNet-50; (**b**) VGG-16.

5. Conclusions

In asynchronous distributed training, for achieving the fast model convergence, it is critical to reduce the difference among local models of workers by considering the scale of distributed training and the heterogeneity among workers. This paper identified the limitations of existing asynchronous distributed training—i.e., not considering the scale of distributed training and the heterogeneity among workers, and proposed a novel asynchronous PS-based distributed training algorithm, named as SHAT that successfully addresses the limitations simultaneously. We defined the generalized update rule for the local model of each worker in asynchronous distributed training, and suggested a reasonable and intuitive strategy for updating the local model of each worker. Through comprehensive experiments, we showed that the models trained by SHAT converge to higher accuracies than the state-of-the-art methods. Furthermore, via the experiments on three heterogeneous environments, we demonstrated that the model convergence of SHAT is robust under various heterogeneous environments, where the models trained by SHAT converge at a consistent rate regardless of the degree of heterogeneity. Finally, we also verified that the memorywise parameter sharding effectively improves the training performance of SHAT in both computation-intensive and communication-intensive models. When we consider that real-world distributed systems generally consist of a large number

of heterogeneous workers, SHAT could be a good solution to scalable distributed training in practice. In order to strengthen the credibility of this work, in future work, we plan to conduct additional experiments using larger models/datasets or different types of models/datasets.

Author Contributions: Conceptualization, Y.K. and S.-W.K.; methodology, Y.K.; software, Y.K.; validation, Y.K.; formal analysis, Y.K. and S.-W.K.; investigation, Y.K.; writing—original draft preparation, Y.K. and S.-W.K.; writing—review and editing, Y.K. and S.-W.K.; supervision, S.-W.K.; project administration, S.-W.K.; funding acquisition, S.-W.K. All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported by the National Research Foundation of Korea (NRF) under Project Number 2020R1A2B5B03001960 and Institute of Information & Communications Technology Planning & Evaluation (IITP) under Project Number 2020-0-01373.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegasm, NV, USA, 26 June–1 July 2016; pp. 770–778.
2. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, USA, 7–9 May 2015.
3. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
4. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *arXiv* **2020**, arXiv:2005.14165.
5. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
6. Recht, B.; Re, C.; Wright, S.; Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Proceedings of the Advances in Neural Information Processing Systems (NIPS), Granada, Spain, 12–14 December 2011; pp. 693–701.
7. Ho, Q.; Cipar, J.; Cui, H.; Lee, S.; Kim, J.K.; Gibbons, P.B.; Gibson, G.A.; Ganger, G.; Xing, E.P. More effective distributed ml via a stale synchronous parallel parameter server. In Proceedings of the Advances in Neural Information Processing Systems, Tahoe, NV, USA, 5–10 December 2013; pp. 1223–1231.
8. Zhang, S.; Choromanska, A.E.; LeCun, Y. Deep learning with elastic averaging SGD. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015; pp. 685–693.
9. Zhao, X.; An, A.; Liu, J.; Chen, B.X. Dynamic stale synchronous parallel distributed training for deep learning. In Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–9 July 2019; pp. 1507–1517.
10. Zhang, H.; Zheng, Z.; Xu, S.; Dai, W.; Ho, Q.; Liang, X.; Hu, Z.; Wei, J.; Xie, P.; Xing, E.P. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In Proceedings of the USENIX Annual Technical Conference (ATC), Santa Clara, CA, USA, 12–14 July 2017; pp. 181–193.
11. Ko, Y.; Choi, K.; Jei, H.; Lee, D.; Kim, S.W. ALADDIN: Asymmetric Centralized Training for Distributed Deep Learning. In Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM), Gold Coast, QLD, Australia, 1–5 November 2021.
12. Gerbessiotis, A.V.; Valiant, L.G. Direct bulk-synchronous parallel algorithms. *J. Parallel Distrib. Comput.* **1994**, *22*, 251–267. [CrossRef]
13. Zhao, X.; Papagelis, M.; An, A.; Chen, B.X.; Liu, J.; Hu, Y. Elastic Bulk Synchronous Parallel Model for Distributed Deep Learning. In Proceedings of the IEEE International Conference on Data Mining (ICDM), Beijing, China, 8–11 November 2019; pp. 1504–1509.
14. Ko, Y.; Choi, K.; Seo, J.; Kim, S.W. An In-Depth Analysis of Distributed Training of Deep Neural Networks. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17–21 May 2021; pp. 994–1003.
15. Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N.R.; Ganger, G.R.; Gibbons, P.B.; Zaharia, M. PipeDream: generalized pipeline parallelism for DNN training. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), Huntsville, ON, Canada, 27–30 October 2019; pp. 1–15.
16. Yu, C.; Tang, H.; Renggli, C.; Kassing, S.; Singla, A.; Alistarh, D.; Zhang, C.; Liu, J. Distributed learning over unreliable networks. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 7202–7212.
17. Eshraghi, N.; Liang, B. Distributed Online Optimization over a Heterogeneous Network with Any-Batch Mirror Descent. In Proceedings of the International Conference on Machine Learning (ICML), Online, 13–18 July 2020; pp. 2933–2942.
18. Zhou, Z.; Mertikopoulos, P.; Bambos, N.; Glynn, P.; Ye, Y.; Li, L.J.; Fei-Fei, L. Distributed Asynchronous Optimization with Unbounded Delays: How Slow Can You Go? In Proceedings of the International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; pp. 5970–5979.

19. Wang, S.; Pi, A.; Zhou, X. Scalable distributed dl training: Batching communication and computation. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), New York, NY, USA, 7–12 February 2019; Volume 33, pp. 5289–5296.
20. Blot, M.; Picard, D.; Cord, M.; Thome, N. Gossip training for deep learning. In Proceedings of the Advances in Neural Information Processing Systems Workshop on Optimization for Machine Learning, Barcelona, Spain, 5–10 December 2016.
21. Lian, X.; Zhang, C.; Zhang, H.; Hsieh, C.J.; Zhang, W.; Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 5330–5340.
22. Lian, X.; Zhang, W.; Zhang, C.; Liu, J. Asynchronous decentralized parallel stochastic gradient descent. In Proceedings of the International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; pp. 3049–3058.
23. Assran, M.; Loizou, N.; Ballas, N.; Rabbat, M. Stochastic gradient push for distributed deep learning. In Proceedings of the International Conference on Machine Learning (ICML), Long Beach, CA, USA, 9–15 June 2019; pp. 344–353.
24. Li, Y.; Yu, M.; Li, S.; Avestimehr, S.; Kim, N.S.; Schwing, A. Pipe-SGD: A decentralized pipelined SGD framework for distributed deep net training. In Proceedings of the International Conference on Neural Information Processing Systems (NIPS), Montreal, QC, Canada, 3–8 December 2018; pp. 8056–8067.
25. Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q.V.; Wu, Y.; et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 8–14 December 2019; pp. 103–112.
26. Johnson, T.; Agrawal, P.; Gu, H.; Guestrin, C. AdaScale SGD: A User-Friendly Algorithm for Distributed Training. In Proceedings of the International Conference on Machine Learning (ICML), Vienna, Austria, 12–18 July 2020; pp. 4911–4920.
27. Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv* **2017**, arXiv:1706.02677.
28. Dean, J.; Corrado, G.; Monga, R.; Chen, K.; Devin, M.; Mao, M.; Senior, A.; Tucker, P.; Yang, K.; Le, Q.V.; et al. Large scale distributed deep networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1223–1231.
29. Jiang, J.; Cui, B.; Zhang, C.; Yu, L. Heterogeneity-aware distributed parameter servers. In Proceedings of the ACM International Conference on Management of Data, (SIGMOD), Chicago, IL, USA, 14–19 May 2017; pp. 463–478.
30. Sergeev, A.; Balso, M.D. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv* **2018**, arXiv:1802.05799.
31. Kempe, D.; Dobra, A.; Gehrke, J. Gossip-based computation of aggregate information. In Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS), Cambridge, MA, USA, 11–14 October 2003; pp. 482–491.
32. Wang, J.; Joshi, G. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *arXiv* **2018**, arXiv:1808.07576.
33. Nadiradze, G.; Markov, I.; Chatterjee, B.; Kungurtsev, V.; Alistarh, D. Elastic Consistency: A Practical Consistency Model for Distributed Stochastic Gradient Descent. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual Conference, 2–9 February 2021.
34. Chilimbi, T.; Suzue, Y.; Apacible, J.; Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI), Broomfield, CO, USA, 6–8 October 2014; pp. 571–582.
35. You, Y.; Gitman, I.; Ginsburg, B. Large batch training of convolutional networks. *arXiv* **2017**, arXiv:1708.03888.
36. You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; Hsieh, C.J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv* **2019**, arXiv:1904.00962.
37. You, Y.; Hseu, J.; Ying, C.; Demmel, J.; Keutzer, K.; Hsieh, C.J. Large-batch training for LSTM and beyond. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, CO, USA, 17–19 November 2019; pp. 1–16.
38. Huo, Z.; Gu, B.; Huang, H. Large batch training does not need warmup. *arXiv* **2020**, arXiv:2002.01576.
39. Smith, S.L.; Kindermans, P.J.; Ying, C.; Le, Q.V. Don't Decay the Learning Rate, Increase the Batch Size. In Proceedings of International Conference on Learning Representations (ICLR), Vancouver, BC, Canada, 30 April–3 May 2018.
40. Cui, H.; Zhang, H.; Ganger, G.R.; Gibbons, P.B.; Xing, E.P. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In Proceedings of the European Conference on Computer Systems (EUROSYS), London, UK, 18–21 April 2016; p. 4.