IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 December 2023, accepted 26 December 2023, date of publication 5 January 2024,
date of current version 12 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3350193

RESEARCH ARTICLE

Pipeline Parallelism With Elastic Averaging

BONGWON JANG, IN-CHUL YOO, (Member, IEEE), AND DONGSUK YOOK, (Member, IEEE)
Artificial Intelligence Laboratory, Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea

Corresponding author: Dongsuk Yook (yook@korea.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Science, ICT and Future Planning, under Grant NRF-2017R1E1A1A01078157; and in part by the NRF under Project BK21 FOUR.

ABSTRACT To accelerate the training speed of massive DNN models on large-scale datasets, distributed training techniques, including data parallelism and model parallelism, have been extensively studied. In particular, pipeline parallelism, which is derived from model parallelism, has been attracting attention. It splits the model parameters across multiple computing nodes and executes multiple mini-batches simultaneously. However, naive pipeline parallelism suffers from the issues of weight inconsistency and delayed gradients, as the model parameters used in the forward and backward passes do not match, causing unstable training and low performance. In this study, we propose a novel pipeline parallelism technique called EA-Pipe to address the weight inconsistency and delayed gradient problems. EA-Pipe applies an elastic averaging method, which has been studied in the context of data parallelism, to pipeline parallelism. The proposed method maintains multiple model replicas to solve the weight inconsistency problem, and synchronizes the model replicas using an elasticity-based moving average method to mitigate the delayed gradient problem. To verify the efficacy of the proposed method, we conducted three image classification experiments on the CIFAR-10/100 and ImageNet datasets. The experimental results show that EA-Pipe not only accelerates training speed but also demonstrates more stable learning property compared to existing pipeline parallelism techniques. Especially, in the experiments using the CIFAR-100 and ImageNet datasets, EA-Pipe recorded error rates that were 2.58% and 2.19% lower, respectively, than the baseline pipeline parallelization method.

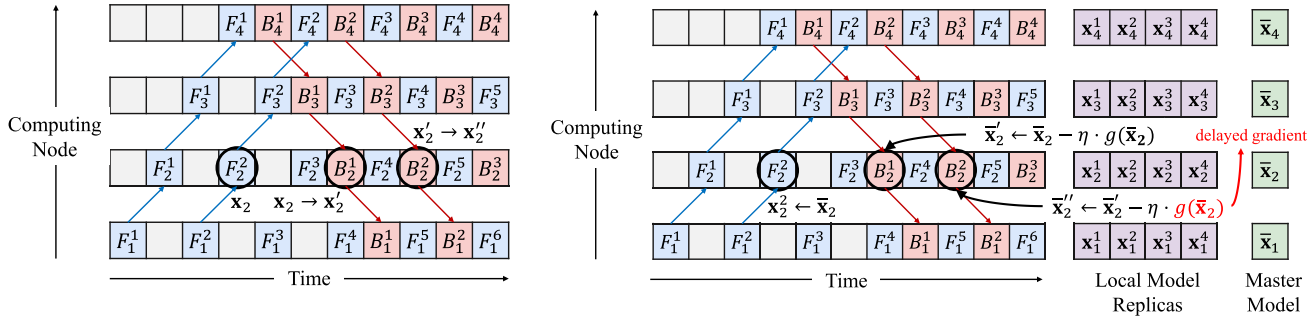
INDEX TERMS Deep learning, stochastic gradient descent (SGD), parallel processing, pipeline processing.

I. INTRODUCTION

In recent years, the increasing availability of advanced hardware support and large-scale datasets have led to the widespread adoption of massive deep neural network (DNN) models as the primary machine learning approaches. However, training massive DNN models on large-scale datasets takes a considerable amount of time, leading to an increasing demand for efficient model training. To efficiently train massive DNN models, for example, VGG [16] and ResNet [31] in an image processing task, or large language models [3], [4], distributed training has been extensively studied [5], [6], [7]. In general, distributed training is divided into two types: data parallelism and model parallelism.

Data parallelism partitions and distributes the training datasets across multiple computing nodes. In the most common data parallelism architecture, each computing node has a local copy of the master DNN model, and the synchronization between the local model and the master model takes place in the parameter server where the master model resides. Depending on the synchronization method, data parallelism is divided into two types: synchronous data parallelism [8], [9] (SDP) and asynchronous data parallelism [10], [11] (ADP). In SDP, the parameter server waits for all computing nodes to complete their works. Then, it synchronizes the master model with the local models in the computing nodes. Because all computing nodes wait for the synchronization to be completed, the training time of SDP is determined by the slowest computing node. In ADP, however, the parameter server does not wait for all computing nodes to

The associate editor coordinating the review of this manuscript and approving it for publication was Taous Meriem Laleg-Kirati.



(a) Weight inconsistency and delayed gradient problems **(b) Weight stashing and vertical sync methods in [15]**

FIGURE 1. (a) An example of pipeline parallelism scheduling timeline [15] having the weight inconsistency and delayed gradient problems. (b) An example of pipeline parallelism scheduling timeline [15] utilizing the weight stashing and vertical sync methods. $F_n^{t_f}$ and $B_n^{t_b}$ represent the forward and backward passes, respectively, at computing node n for mini-batch index t_f and t_b, respectively. In (a), $\mathbf{x}_n$ represents the partitioned model parameters at computing node n. In (b), $\mathbf{x}_n^i$ and $\bar{\mathbf{x}}_n$ represent the i-th replica of the partitioned local model parameters and the partitioned master model parameters, respectively, at computing node n. We can calculate the index i for the local model used in $F_n^{t_f}$ and $B_n^{t_b}$ by $i = \text{mod}(t_f - 1, N) + 1$ and $i = \text{mod}(t_b - 1, N) + 1$, respectively, where N is the number of computing nodes. The blue arrows represent the value propagated through the forward pass, while the red arrows represent the error propagated through the backward pass. $g(x)$ represents the gradient computed using model parameter $\mathbf{x}$. Though the weight inconsistency problem is resolved by using the weight stashing and vertical sync methods, the delayed gradient problem still remains in updating the master model.

finish their works. It synchronizes with the computing node that completes the computation first. Therefore, ADP can accelerate training speed more than SDP. However, since the parameter server accepts the gradients from the computing nodes in the order of their arrival and applies them to the master model, the late-arriving gradients cannot be correctly applied to the exact version of the master model parameters used to compute the gradients. Instead, they will be applied to the master model parameters that have already been updated by the early arrived gradients. These late-arriving gradients are called the *delayed gradients* (or *stale gradients*), and it is the major cause of the negative effect on training performance in ADP [12].

Model parallelism partitions the model parameters and distributes them across multiple computing nodes [11]. In naive model parallelism, the hardware utilization is low since only one computing node may be active at a time [13]. To mitigate this problem, pipeline parallelism has been proposed [13], [14], [15]. Pipeline parallelism can process multiple mini-batches simultaneously in all computing nodes, resulting in a significant acceleration in computation. Though pipeline parallelism can train DNN models efficiently in terms of training time, it suffers from two problems: the *weight inconsistency* and *delayed gradient* problems. The model parameters used in the forward pass and backward pass are different in naive pipeline parallelism, which is called the weight inconsistency problem. In addition, since the model parameters used during forward pass are not maintained until applying the gradients, it results in the delayed gradient problem as in ADP. These problems deteriorate model convergence, causing longer training time and higher error rates.

Fig. 1(a) illustrates an example of pipeline parallelism scheduling timeline [15] having the weight inconsistency and delayed gradient problems. Let $\mathbf{x}_2$ indicate the model parameters used in the forward pass of the 2nd mini-batch in the 2nd computing node (denoted as F_2^2). $\mathbf{x}_2$ is updated first in the backward pass of the 1st mini-batch in the 2nd computing node (denoted as B_2^1). Let $\mathbf{x}_2'$ indicate the updated model parameters after B_2^1. Then, $\mathbf{x}_2'$ is updated to $\mathbf{x}_2''$ in the backward pass of the 2nd mini-batch in the 2nd computing node (denoted as B_2^2). Since the model parameters used during B_2^2, that is, $\mathbf{x}_2'$, differ from the one used in F_2^2, that is, $\mathbf{x}_2$, the *weight inconsistency* problem occurs. Furthermore, the model parameters used for gradient computation are $\mathbf{x}_2$, whereas the parameters to which the gradients are applied are $\mathbf{x}_2'$. This causes the *delayed gradient* problem because the model parameters $\mathbf{x}_2$, which are used during the forward pass F_2^2, are not preserved until the gradients are applied.

Existing studies on pipeline parallelism have proposed various methods to address these problems by synchronously updating model parameters [13], [16], predicting correct model parameters [17], [18], or generating multiple model replicas [15]. However, these methods have drawbacks such as becoming less accurate as the number of computing nodes increases, or underutilizing hardware efficiency. For example, generating multiple model replicas is an approach to mitigate the weight inconsistency problem. However, delayed gradient problem may arise depending on the synchronization method (e.g., Applying ADP to synchronize multiple model replicas, as seen in [15], may result in the delayed gradient problem, which will be explained further in Section II).

In this paper, we propose a novel pipeline parallelism method, called EA-Pipe, that aims to address the delayed gradient problem when generating multiple model replicas to mitigate the weight inconsistency problem. To this end, we view the delayed gradient problem from a data parallelism perspective, and apply an optimization method previously studied in the area of data parallelism. Especially, EA-Pipe

utilizes the moving average based elastic force called elastic averaging [19]. The elastic averaging algorithm showed comparable performance with prior ADP synchronization methods, while alleviating the delayed gradient problem and achieving more stable learning property. Therefore, it can be expected that similar effects can be achieved when applying the algorithm to the pipeline parallelism. The contributions of this paper are as follows:

- Unlike existing methods, we approach the problems in pipeline parallelism through a data parallelism perspective. Thereby, we can utilize the advantages of the existing ADP optimization methods such as the elastic averaging algorithm for pipeline parallelism.
- We proposed a novel round-robin parallel training algorithm by combining the pipeline parallelism and asynchronous data parallelism to solve the weight inconsistency and delayed gradient problems.
- We analyzed the convergence property of the proposed method and found that the error bound is the same as the synchronous elastic averaging case. To the best of the author's knowledge, this is the first theoretical convergence analysis of the round-robin asynchronous data parallelism through the elastic averaging pipeline parallelism.
- The experimental results indicate that EA-Pipe not only enhances training speed but also trains DNN models as stably as SGD. In particular, in the experiments using the CIFAR-100 and ImageNet datasets, EA-Pipe demonstrated error rates that were 2.58% and 2.19% lower, respectively, compared to the baseline pipeline parallelization method, PipeDream.

The rest of the paper is organized as follows: In Section II, we review related works concerning EA-Pipe. In Section III, the work scheduling, algorithm, and convergence property of EA-Pipe are explained. In Section IV, we verify the proposed method through three image classification experiments using three different models and datasets. The conclusion of this study and suggestions for future research are discussed in Section V, followed by the Appendix including the detail proof of the convergence analysis.

II. RELATED WORK

A. PIPEDREAM

PipeDream [15] follows a scheduling strategy in which each computing node executes the forward and backward passes alternatively for different mini-batches, ensuring high hardware utilization. Fig. 1(a) shows the pipeline scheduling of PipeDream. However, this pipeline scheduling leads to the weight inconsistency and delayed gradient problems, since the versions of the model parameters used in the forward and backward passes are inconsistent. To mitigate these problems, PipeDream proposed the weight stashing and vertical sync methods. The weight stashing method generates as many model replicas as the number of active mini-batches for each computing node. Thereby each mini-batch ends up using the same model parameters in both the forward and backward passes. The vertical sync method generates as many model replicas as the maximum number of active mini-batches in the pipeline, ensuring the time synchronous weight versions are available throughout all computing nodes. The maximum number of active mini-batches is always equal to the number of computing nodes. Fig.1(b) shows PipeDream utilizing the weight stashing and vertical sync methods.

With these two methods, PipeDream can solve the weight inconsistency problem effectively. However, during the backward pass, the computed gradients are applied to the model parameters that have already been updated by the earlier gradients. Therefore, the delayed gradient problem still remains. For example, F_2^2 uses the local model parameters $\mathbf{x}_2^2$, which are initialized to $\bar{\mathbf{x}}_2$. However, $\bar{\mathbf{x}}_2$ is updated at B_2^1. Therefore, the gradients calculated at B_2^2 are applied to the master model parameters which have already been updated at B_2^1. As a result, the delayed gradient problem still remains unresolved.

EA-Pipe addresses the persistent delayed gradient problem remained even after resolving the weight inconsistency problem by generating multiple model replicas in PipeDream. To tackle this problem, EA-Pipe approaches the delayed gradient problem from the perspective of data parallelism and applies the elastic averaging algorithm previously studied in the context of data parallelism.

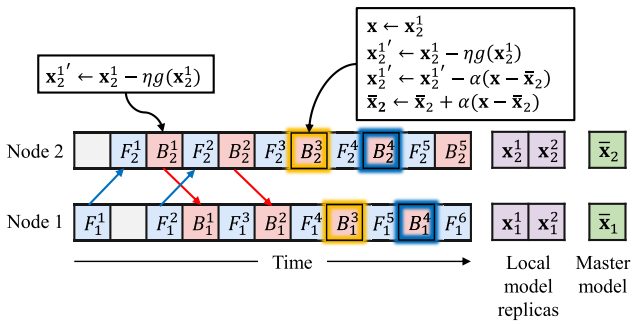
B. ELASTIC AVERAGING ALGORITHM

The elastic averaging algorithm [19] was proposed to reduce the communication cost in data parallelism, by enabling each computing node to conduct more local training computations and explore more extensively away from the master model before synchronization. Each local model is maintained as if it is connected with the master model by an elastic force, which constrains the distance between the local and master model parameters. The stronger the elastic force is, the more the distance between the local and master model parameters is constrained. While training, each local model fluctuates around the master model. Therefore, the risk of the master model falling into local optima may be reduced. The synchronization equation of the elastic averaging algorithm in data parallelism is as follows:

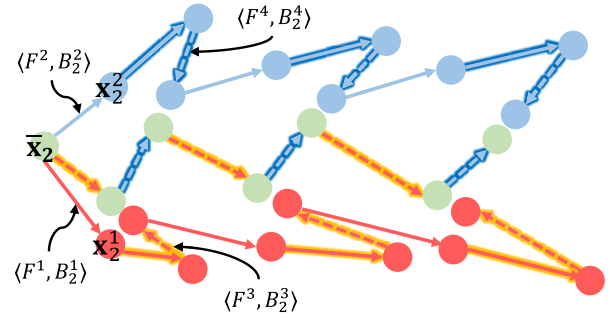
$$\begin{aligned} \mathbf{x} &\leftarrow \mathbf{x}_n \\ \mathbf{x}_n &\leftarrow \mathbf{x}_n - \alpha(\mathbf{x} - \bar{\mathbf{x}}) \\ \bar{\mathbf{x}} &\leftarrow \bar{\mathbf{x}} + \alpha(\mathbf{x} - \bar{\mathbf{x}}), \end{aligned} \tag{1}$$

where $\mathbf{x}_n$ and $\bar{\mathbf{x}}$ denote the n-th local and master model parameters, respectively. α indicates the strength of the elastic force between the local and master models.

EA-Pipe incorporates the elastic averaging algorithm into pipeline parallelism. As a result, the synchronization of model parameters occurs partially, unlike the elastic averaging algorithm in data parallelism. Further details on parameter synchronization in EA-Pipe will be discussed in Section III.



(a) EA-Pipe with two computing nodes



(b) Parameter update in the second computing node by EA-Pipe

FIGURE 2. An example workflow of EA-Pipe with two computing nodes when synchronization period is set to 2. In (a), the highlighted boxes represent the backward passes and synchronization with the master model. In (b), $\langle F^{t_f}, B_2^{t_b} \rangle = \langle F_1^{t_f}, F_2^{t_f}, B_2^{t_b} \rangle$.

III. PROPOSED METHOD

In this section, we propose a novel pipeline parallelism method, called EA-Pipe, which mitigates the weight inconsistency and delayed gradient problems at the same time. This section is divided into four subsections. Firstly, we introduce the pipeline scheduling of EA-Pipe. Secondly, we present the algorithm of EA-Pipe in pseudo-code. Thirdly, we analyze the convergence property of EA-Pipe. Lastly, we compare EA-Pipe with a recent work that uses similar approaches.

A. PIPELINE SCHEDULING OF EA-PIPE

EA-Pipe follows the same structure as PipeDream in Fig. 1(b). However, unlike PipeDream, the local models in EA-Pipe update their parameters locally without synchronizing with the master model parameters until a specific synchronization period is reached. Fig. 2 shows an example workflow of EA-Pipe with two computing nodes when synchronization period is set to 2. In Fig. 2(a), the local model parameters are updated locally without synchronization with the master model parameters. For example, at the backward pass B_2^1, the computed gradients cause the local model parameters to be updated from $\mathbf{x}_2^1$ to $\mathbf{x}_2^{1'}$. However, once the synchronization period is reached, the backward pass (e.g., B_2^3) not only updates the local model parameters but also synchronizes them with the master model parameters based on Eq. (1).

Algorithm 1 EA-Pipe: Executed by Computing Node n in Parallel With All Other Computing Nodes

Initialize partitioned master model $\bar{\mathbf{x}}_n$.
Copy $\bar{\mathbf{x}}_n$ to the replicas of the partitioned local models $\mathbf{x}_n^1, \mathbf{x}_n^2, \cdots, \mathbf{x}_n^N$.
Set t_f and t_b to 1.
repeat
  if not final phase **then** // forward pass
    if $(1 < n)$ **then**
      Wait for $F_{n-1}^{t_\mathrm{f}}$ to be finished.
    end if
    $i \leftarrow \mathrm{mod}(t_\mathrm{f} - 1, N) + 1$
    Execute $F_n^{t_\mathrm{f}}$ using $\mathbf{x}_n^i$.
    if $(n < N)$ **then**
      Send the result of $F_n^{t_\mathrm{f}}$ to node $n + 1$.
    end if
    $t_\mathrm{f} \leftarrow t_\mathrm{f} + 1$
  end if
  if not initial phase **then** // backward pass
    if $(n < N)$ **then**
      Wait for $B_{n+1}^{t_\mathrm{b}}$ to be finished
    end if
    Execute $B_n^{t_\mathrm{b}}$ as follows:
    $i \leftarrow \mathrm{mod}(t_\mathrm{b} - 1, N) + 1$
    $\mathbf{x} \leftarrow \mathbf{x}_n^i$
    $\mathbf{x}_n^i \leftarrow \mathbf{x}_n^i - \eta g(\mathbf{x})$
    if τ divides $\lceil t_\mathrm{b}/N \rceil$
      $\mathbf{x}_n^i \leftarrow \mathbf{x}_n^i - \alpha(\mathbf{x} - \bar{\mathbf{x}}_n)$
      $\bar{\mathbf{x}}_n \leftarrow \bar{\mathbf{x}}_n + \alpha(\mathbf{x} - \bar{\mathbf{x}}_n)$
    end if
    if $(n > 1)$ **then**
      Backpropagate the error to node $n - 1$ using $\mathbf{x}$.
    end if
    $t_\mathrm{b} \leftarrow t_\mathrm{b} + 1$
  end if
until (t_f and t_b equal T)

Fig. 2(b) shows how the parameters are updated in the second computing node by EA-Pipe in a virtual parameter space. For example, the second computing node updates the local model parameters $\mathbf{x}_2^2$ during B_2^2. As the synchronization period of 2 has not been reached yet, synchronization with the master model does not occur. Then, the second computing node updates $\mathbf{x}_2^2$ again during B_2^4. At this point, as the synchronization period of 2 has been reached, the second computing node synchronizes $\mathbf{x}_2^2$ with the master model parameters $\bar{\mathbf{x}}_2$. The highlighted arrow lines indicate the backward passes for each local model when it reaches the synchronization period, while the dashed arrow lines represent the synchronization process where the local model and master model pull each other.

B. ALGORITHM

A pseudo-code of EA-Pipe is shown in Algorithm 1, which is executed by all computing nodes simultaneously. τ and

T indicate the synchronization period and the total number of mini-batches, respectively. N represents the total number of computing nodes, hence the number of partitioned master models. The size of each partitioned model is, therefore, one $1/N$ of the original model size, if it is divided evenly. The term *initial phase* refers to the beginning of the pipeline scheduling timeline when only forward passes are executed to fill the pipeline. Similarly, the term *final phase* refers to the ends of the pipeline scheduling timeline when only backward passes are left to be executed to flush the pipeline. The forward pass $F_n^{t_f}$ is executed followed by the backward pass $B_n^{t_b}$ at each iteration. The model parameters $\bar{\mathbf{x}}_n$ and $\mathbf{x}_n^i$ are synchronized at the end of the backward pass whenever $\lceil t_b/N \rceil$ is divided by τ. $g(\mathbf{x})$ represents the gradient computed using model parameter $\mathbf{x}$.

C. CONVERGENCE ANALYSIS

In this section, we will discuss the convergence property of EA-Pipe. Previous studies have limited analysis on the elastic averaging algorithm. For example, the analysis in [19] covered only one local update for quadratic objective functions. The convergence analysis presented in [20] is only for a synchronous elastic averaging algorithm. Therefore, the convergence analysis of an asynchronous elastic averaging algorithm remains an open question. In EA-Pipe, the local models are synchronized with the master model sequentially, which can be considered as a round-robin data parallelism [21]. As shown in Fig. 2(b), we can observe that the synchronization order between the master model $\bar{\mathbf{x}}$ and the set of local models $\{\mathbf{x}^1, \mathbf{x}^2\}$ is always round-robin (i.e., $\mathbf{x}^1 \rightarrow \mathbf{x}^2 \rightarrow \mathbf{x}^1 \rightarrow \cdots$). Based on this observation, we will discuss the convergence property of EA-Pipe from the perspective of asynchronous elastic averaging algorithm in round-robin data parallelism.

The objective function $F(\mathbf{x})$ that we are interested in is defined as follows:

$$F(\mathbf{x}) := \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}_i}[\mathcal{L}(\mathbf{x}, \mathbf{s})], \tag{2}$$

which is commonly used for convergence analysis in the synchronous or asynchronous data parallelism [20], [22], [23]. Here $\mathbf{x} \in \mathbb{R}^d$, N, and $\mathcal{D}_i$ denote the model parameters, the number of local models, and the local data distribution for the i-th local model, respectively. $\mathcal{L}$ denotes the loss function.

Based on the analysis in [20] and [22] which proves the convergence rate of distributed SGD algorithms with local updates on non-convex objectives, we can derive the following theorem, which guarantees that EA-Pipe converges to stationary points of non-convex objective functions with the same error bound as a synchronous elastic averaging algorithm.

Theorem 1: Let L, η, τ, σ^2, ζ, and K be the Lipschitz constant, learning rate, synchronization period, bound of gradient variance, magnitude of the second largest eigenvalue (see Appendix A), and the total number of iterations, respectively. Then, the convergence rate of EA-Pipe is as follows:

$$\begin{aligned} \frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}[\|\nabla F(\mathbf{y}_k)\|^2] &\leq \frac{2(F(\mathbf{y}_0) - F_{\text{inf}})}{\eta_{\text{eff}} K} + \frac{\eta_{\text{eff}} L \sigma^2}{N} \\ &\quad + \eta_{\text{eff}}^2 L^2 \sigma^2 \left(\frac{1+\zeta^2}{1-\zeta^2} \tau - 1 \right) \left(1 + \frac{1}{N} \right)^2, \end{aligned} \tag{3}$$

where $\mathbf{y}_k$, F_{inf}, and η_{eff} denote the average of local and master model parameters at k-th time-step, the lower bound of the objective function, and an effective learning rate $\frac{N}{N+1}\eta$, respectively.

Theorem 1 demonstrates that if the learning rate η is chosen properly and the total number of iterations K is large enough, the error bound of the EA-Pipe algorithm is equal to that of the synchronous elastic averaging algorithm in [20]. The detailed proof of the theorem is provided in Appendix A.

D. COMPARISON TO SIMILAR WORKS

Recently, AvgPipe is proposed to enhance the throughput of pipeline parallelism by combining elastic averaging algorithm into pipeline parallelism [24]. Although AvgPipe and EA-Pipe shares similarities, we highlight some distinctions between their approach and ours. First, while AvgPipe is designed to operate at micro-batch level which has a constraint on the size of mini-batch, EA-Pipe is devised to operate at the mini-batch level without any size constraint. Second, while AvgPipe introduced elastic averaging algorithm to increase the throughput of pipeline parallelism, our approach proposes integrating the elastic averaging algorithm to address the delayed gradient problems. Last but not least, in contrast to [24], we conducted an analysis of the convergence property of pipeline parallelism with the elastic averaging algorithm.

IV. EXPERIMENTS

In this section, we explain three types of experiments conducted to verify the performance of the proposed method: small-scale, mid-scale, and large-scale experiments. The small-scale experiment evaluates EA-Pipe for an image classification task using CIFAR-10 dataset with VGG-16 [16] as the DNN model. The CIFAR-10 dataset has 60,000 32×32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. To prevent overfitting during model training, we employed an image augmentation technique. VGG-16 is a convolutional neural network architecture with 16 layers, including 13 convolutional layers and 3 fully connected layers.

We compared four training methods: the conventional sequential SGD (denoted as SGD in the Tables and Figures), PipeDream [15], SpecTrain [17], and the proposed EA-Pipe. Since SGD with momentum is commonly employed in

TABLE 1. Image classification error rates (%) and 95% confidence interval of the four training methods using the CIFAR-10 dataset and the VGG-16 model with varying number of GPUs (small-scale experiment).

	1-GPU	2-GPU	4-GPU	8-GPU
SGD	7.36 (±0.11)	-	-	-
PipeDream	-	7.95 (±0.40)	7.94 (±0.41)	8.09 (±0.50)
SpecTrain	-	10.05 (±0.54)	9.74 (±0.72)	9.91 (±0.16)
EA-Pipe	-	8.19 (±0.42)	8.05 (±0.36)	8.25 (±0.58)

TABLE 2. Computational overhead of the four training methods in the small-scale experiment, measured by the average training time (in seconds) per epoch.

	1-GPU	2-GPU	4-GPU	8-GPU
SGD	43.56	-	-	-
PipeDream	-	28.08	25.56	22.32
SpecTrain	-	25.56	22.32	20.52
EA-Pipe	-	28.44	25.92	23.04

TABLE 3. Statistical efficiency of the four training methods in the small-scale experiment, measured by the total training time (in hours) to reach the lowest error rates.

	1-GPU	2-GPU	4-GPU	8-GPU
SGD (1-GPU)	1.13	-	-	-
PipeDream	-	0.73	0.67	0.59
SpecTrain	-	0.57	0.54	0.54
EA-Pipe	-	0.79	0.67	0.63

computer vision tasks, we chose SGD as the baseline of the sequential training method [25]. We included SpecTrain, which addresses the weight inconsistency and delayed gradient problems through model parameter prediction. All methods were implemented in CUDA/C++. The performance of each training method was measured on a server with an Intel Xeon Silver 4110 CPU and eight NVIDIA GeForce RTX 2080 Ti GPUs.

The performances of the models trained with each algorithm were measured by the ratio of the misclassified images in the test data. We ran three repetitions of the experiments each with different random initial weights, and the average performance with 95% confidence interval for each method is reported here. The best case results of the three repetitions can be found in Appendices B, C, and D. The same evaluation protocol was consistently maintained throughout the subsequent experiments.

In the second experiment (mid-scale experiment), we scaled up from the first experiment, assessing EA-Pipe for an image classification task using CIFAR-100 dataset with ResNet-34 [31] as the DNN models. The CIFAR-100 dataset closely resembles CIFAR-10, except that it has 100 classes containing 600 images each. ResNet-34 is a variant of the residual network architecture with 34 layers, utilizing skip connections to address training challenges in deep neural networks. We compared four training methods, as in the previous experiment.

The third experiment compares EA-Pipe and PipeDream to further verify the effect of the delayed gradients on a larger scale of parallelism. We conducted the experiment using the ResNet-50 model and the ImageNet dataset, simulating PipeDream and EA-Pipe on a virtual 50-GPU environment. The ImageNet dataset contains 1,281,167 training images, 50,000 validation images and 100,000 test images in 1,000 classes. The main objective of the third experiment is to evaluate the effectiveness of EA-Pipe in addressing the delayed gradient problem for large-scale environments. The simulation experiment was conducted on a server with an Intel i5-10600K CPU and an NVIDIA RTX 2080 Ti GPU.

A. EXPERIMENTAL RESULTS ON CIFAR-10 WITH VGG-16 (SMALL-SCALE EXPERIMENT)

We conducted small-scale experiments using the CIFAR-10 dataset and the VGG-16 model with varying number of GPUs (one, two, four, and eight). For all four methods, we tried batch sizes of 128, 64, 32, and 16, and learning rates of 0.1 and 0.01. The number of training epochs was set to 100. Learning rates were reduced to one tenth after every 30 epochs. For SpecTrain training, however, it did not converge with the previously chosen learning rate candidates. Thus, we conducted additional SpecTrain training using learning rates of 0.001, 0.0005, 0.0003, 0.0001, paired with the batch sizes of 128, 64, 32, and 16, respectively. For EA-Pipe, we set the communication period to 1 and measured the best performance by varying the elastic force (0.1, 0.3, and 0.5).

Table 1 summarizes the results of the small-scale experiment. It was found that a batch size of 16 and a learning rate of 0.01 were the optimal hyperparameters for SGD, PipeDream, and EA-Pipe, while a batch size of 32 and a learning rate of 0.0003 were the optimal hyperparameters for SpecTrain. EA-Pipe showed the best performance at an elastic force of 0.3. SGD showed the lowest error rate of 7.36%. SpecTrain succeeded in training only when the learning rate was set to a small value (0.0005), while reaching higher error rates compared to PipeDream and EA-Pipe. PipeDream and the proposed method, EA-Pipe, did not exhibit significant differences. Therefore, additional validation of the proposed method was necessary through larger scale experiments, which will be discussed shortly.

Table 2 shows the computational overhead for the four training methods in the small-scale experiment. The computational overhead was calculated by dividing the total training time (in seconds) by the total number of epochs. SpecTrain showed a lower computational overhead compared to PipeDream and EA-Pipe since it used a batch size of 32, while PipeDream and EA-Pipe used a batch size of 16. These were the optimal values for each training method. Though SpecTrain was the fastest, it showed the worst image classification accuracy. Both PipeDream and EA-Pipe showed a reduction in computational overhead as the number of GPUs increased. EA-Pipe exhibited a slightly higher computational overhead compared to PipeDream.

Table 3 shows the statistical efficiency of the four training methods measured by the total training time to reach the

TABLE 4. Image classification performance (average error rates in % and 95% confidence interval) of the four training methods using the CIFAR-100 dataset and the ResNet-34 model on 8 GPUs (mid-scale experiment).

batch size	16			32			64		
learning rate	0.1	0.01	0.0003	0.1	0.01	0.0005	0.1	0.01	0.001
SGD	24.71 (±0.94)	23.29 (±0.93)	-	23.08 (±1.25)	24.60 (±0.58)	-	23.28 (±0.22)	27.18 (±0.32)	-
PipeDream	33.42 (±6.65)	23.80 (±1.47)	-	28.33 (±2.89)	26.70 (±1.58)	-	30.63 (±1.18)	33.46 (±5.97)	-
SpecTrain	-	-	26.72 (±0.93)	-	-	28.79 (±1.16)	-	-	33.02 (±0.44)
EA-Pipe	37.31 (±1.93)	**23.20 (±0.29)**	**-**	27.73 (±0.57)	25.33 (±0.87)	-	26.15 (±0.50)	30.34 (±3.78)	-

TABLE 5. Computational overhead of the four training methods in the mid-scale experiment, measured by the average training time (in seconds) per epoch.

	1-GPU	8-GPU
SGD	84.24	-
PipeDream	-	41.76
SpecTrain	-	43.21
EA-Pipe	-	43.20

TABLE 6. Statistical efficiency of the four training methods in the mid-scale experiment, measured by the total training time (in hours) to reach the lowest error rates.

	1-GPU	8-GPU
SGD	2.29	-
PipeDream	-	1.15
SpecTrain	-	1.12
EA-Pipe	-	1.06

lowest error rates. As can be seen in Tables 2 and 3, we doubt that EA-Pipe is appropriate for small scale parallelism.

B. EXPERIMENTAL RESULTS ON CIFAR-100 WITH RESNET-34 (MID-SCALE EXPERIMENT)

We conducted a mid-scale experiment using the CIFAR-100 dataset and the ResNet-34 model on 8 GPUs. For all four methods, we employed batch sizes of 64, 32, and 16, and learning rates of 0.1 and 0.01. The number of training epochs was set to 100. Learning rates were reduced to one tenth after every 30 epochs. For SpecTrain, however, it did not converge with the previously chosen learning rate candidates. Thus, we conducted additional SpecTrain training using learning rates of 0.001, 0.0005, 0.0003, paired with the batch sizes of 64, 32, and 16, respectively. For EA-Pipe, we set the communication period to 1 and measured the best performance by varying the elastic force (0.1, 0.3, and 0.5).

Table 4 presents the results of the mid-scale experiment. Both PipeDream and EA-Pipe reached at the lowest error rates at a batch size of 16 and a learning rate of 0.01. Except for when using a batch size of 16 and a learning rate of 0.1, EA-Pipe achieved lower error rates than PipeDream in all cases. SpecTrain failed to be trained at batch sizes of 64, 32, and 16, and learning rates of 0.1 and 0.01. Similar to the first experiment (Section IV-A), SpecTrain only succeeded in training when the learning rate was set to a smaller value (0.0003), while reaching higher error rates compared to PipeDream and EA-Pipe. We can suspect that using the weight prediction of SpecTrain to address the weight inconsistency and delayed gradient problems might not be effective.

Fig. 3 depicts the error rate curves for the three methods (SGD, PipeDream, and EA-Pipe). We did not include SpecTrain, since it showed the worst error rates. The robustness of EA-Pipe can be observed in the error rate graphs. At high learning rates (e.g., between 0 and 30 epochs), PipeDream exhibits slow convergence speed and unstable model training trends. On the other hand, EA-Pipe shows stable model training trends similar to SGD.

In Table 5, we compare the computational overhead of the four training methods in the mid-scale experiment. Similar to the small-scale experiment, EA-Pipe exhibited a slightly higher computational overhead compared to PipeDream. However, as can be seen in Table 6, in terms of the total training time taken to reach the lowest error rate, EA-Pipe is the most efficient one among the four training methods. We suspect that the effciency of EA-Pipe will be more evident as the size of parallelism gets larger, which is discussed in the next section.

C. EXPERIMENTAL RESULTS OF LARGE-SCALE PARALLELISM

In the previous section, we observed statistical efficiency and stability in training for EA-Pipe. However, it was not enough to confirm the adverse effect of the delayed gradient problem. Therefore, a large-scale experiment was conducted. Due to the lack of available computing accelerators, we conducted a large-scale parallelism experiment of EA-Pipe and PipeDream running in a 50-GPU simulated environment. To carry out the evaluation, we chose an image classification task on the ImageNet dataset using the ResNet-50 model.

Since an ImageNet experiment requires a significant amount of time for a training method to complete the whole training process, we selected a batch size of 64, which is the maximum size that the GPU memory can accommodate. To determine the optimal learning rate, we first evaluated 10% of the ImageNet dataset and found out that 0.06 was the best learning rate among the candidates of 0.1, 0.06, 0.03, and 0.01. Then, we evaluated the entire ImageNet dataset using the learning rate of 0.06, as well as two adjacent values (0.08 and 0.04). As a result, we identified 0.04 as the optimal learning rate. For the sake of fairness, PipeDream and EA-Pipe utilizes the same batch size and learning rate values. The number of training epochs was set to 100. Learning rates

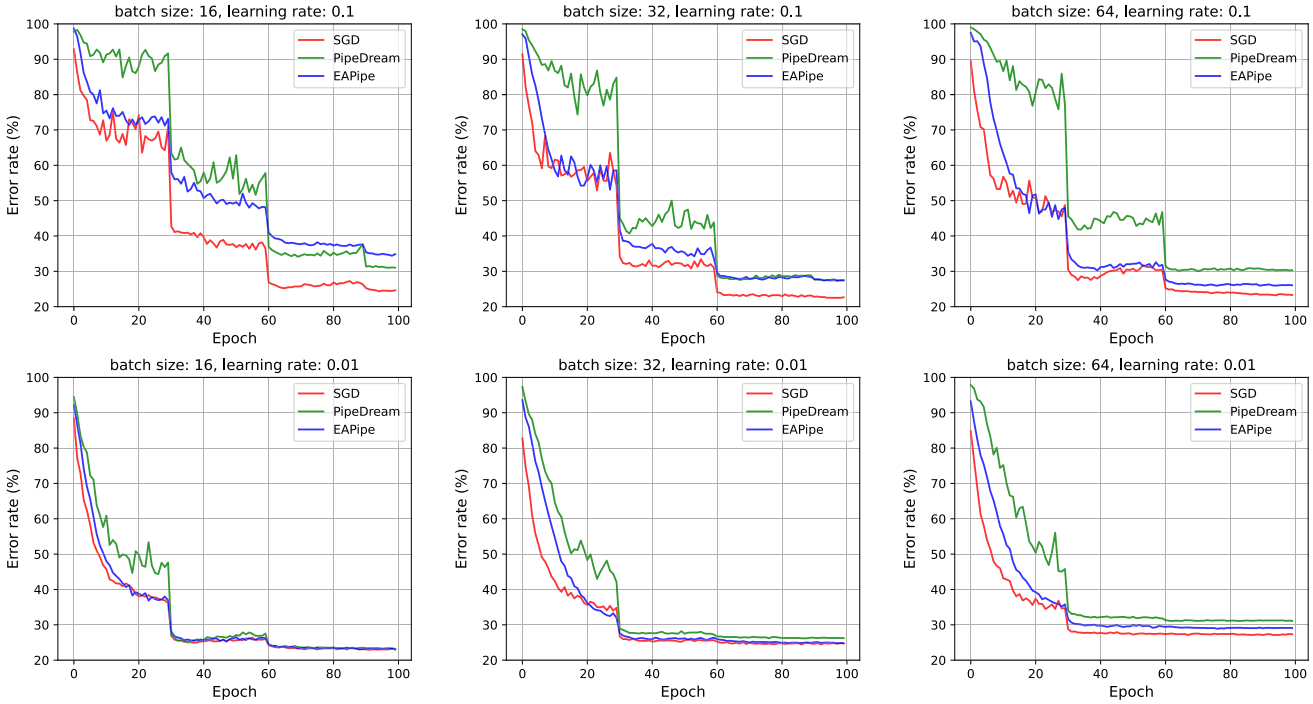


FIGURE 3. Training curves for the three training methods using the CIFAR-100 dataset with the ResNet-34 model on 8 GPUs.

TABLE 7. Image classification error rates (%) and 95% confidence intervals of the three training methods using the ImageNet dataset and the ResNet-50 model on 50 GPUs (large-scale experiment).

	1-GPU	50-GPU
SGD	23.55 (±0.26)	-
PipeDream	-	26.53 (±0.94)
EA-Pipe	-	**25.95 (±0.36)**

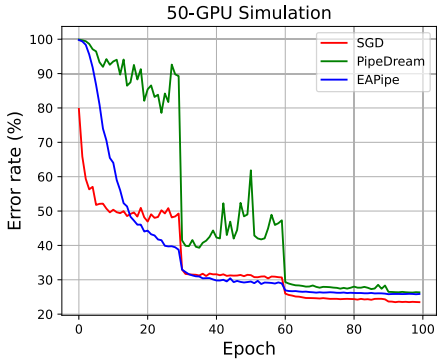


FIGURE 4. Training curves of the three training methods using the ImageNet dataset and the ResNet-50 model on 50 GPUs.

were reduced to one tenth after every 30 epochs. For EA-Pipe, we set the communication period to 1 and the elastic force to 0.1.

Table 7 shows the results of the three training methods using the ImageNet dataset and the ResNet-50 model on 50 GPUs. In the large-scale setting, EA-Pipe achieved lower error rate than PipeDream. As we suspected earlier, the delayed gradients become more problematic in large-scale parallelism, validating the need for a method solving the delayed gradient problem efficiently, such as EA-Pipe.

Fig. 4 depicts the error rate curves for these methods. We can observe that EA-Pipe shows a more stable learning property compared to PipeDream at large learning rates as observed in the previous section. This is due to the following fact. Since the effective weight change (i.e. gradients multiplied by the learning rate) is relatively large compared to small learning rate cases, the delayed gradients become more problematic in PipeDream.

V. CONCLUSION AND FUTURE WORK

In this study, a novel pipelined parallel SGD algorithm, EA-Pipe, has been proposed to mitigate the delayed gradient problem that occurs in pipeline parallelism. It utilizes the multiple model replicas and synchronizes them based on an elastic averaging scheme. Some conventional approaches reduce the batch size and learning rate to alleviate the delayed gradient problem to some extent, which can reduce the GPU hardware utilization and/or increase the training time. Our proposed method does not need to adjust the batch size and learning rate, thereby reducing the hyperparameter optimization time. The experimental results confirmed that the proposed method can achieve comparable error rates to SGD and show the efficacy of parallel training in large-scale environments. In addition, we analyzed the convergence property of EA-Pipe, and confirmed that the error bound

is the same as the synchronous elastic averaging algorithm. However, the proposed method could have a disadvantage in terms of memory utilization because it creates multiple model replicas. This disadvantage could constrain the training of very large models, which can be left to a future work.

APPENDIX A
PROOF OF THEOREM 1

In EA-Pipe, the local models are synchronized with the master model in round-robin manner, which makes the model parallelism of EA-Pipe as the round-robin data parallelism. At each time step k, all local computing nodes update their model parameters, and at every synchronization period, they are synchronized with the master model in a round-robin manner. The typical round-robin elastic averaging algorithms for the local models and the master model are Algorithm 2 and Algorithm 3, respectively.

K, η, α and τ denote the total number of time steps (iterations), learning rate, elastic force value, and synchronization period, respectively. $g(\mathbf{x}^{i,k}, \mathbf{s}^{i,k})$ indicates the stochastic gradients computed on a randomly sampled mini-batch $\mathbf{s}^{i,k} \sim \mathcal{D}_i$, where i and k indicate the indices for the local model and time-step, respectively. Note that the EA-Pipe in Algorithm 1 is a special case of these algorithms, which effectively implement the round-robin elastic averaging through the pipeline parallelism.

These algorithms optimize the following objective function.

$$F(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}_i}[\mathcal{L}(\mathbf{x}, \mathbf{s})] + \rho\|\mathbf{x}^i - \bar{x}\|^2 \tag{4}$$

The equivalence of Eq. (4) and the following objective function that we are interested in is studied in the literature and it is known as the global variable consensus problem [26].

$$F(\mathbf{x}) := \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\mathbf{s}\sim\mathcal{D}_i}[\mathcal{L}(\mathbf{x}, \mathbf{s})] \tag{5}$$

Therefore, we will be focusing on the convergence analysis of Eq. (5).

In order to analyze the convergence property of the round-robin elastic averaging algorithms, we utilize the Theorem 1 in [20], which is based on the following assumptions.

Assumption 1 (L-Smoothness): We assume that each local objective function $F_i(\mathbf{x}) := \mathbb{E}_{\mathbf{s}\sim\mathcal{D}_i}[\mathcal{L}(\mathbf{x}, \mathbf{s})]$ is L-smooth such that

$$\|\nabla F_i(\mathbf{x}) - \nabla F_i(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \tag{6}$$

where $i \in \{1, 2, \cdots, N\}$ and $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$.

Assumption 2 (Lower Bound): We assume that $F(\mathbf{x})$ has a lower bound F_{inf} such that:

$$F(\mathbf{x}) \geq F_{\text{inf}}. \tag{7}$$

Algorithm 2 Round-Robin Elastic Averaging Algorithm for Local Model Parameter $\mathbf{x}^i$

Initialize $\mathbf{x}^i$
$k = 0$
repeat
  if $k \mod \tau = 0$ **then**
    Wait until $\bar{\mathbf{x}}^k$ is synchronized with $\mathbf{x}^{i-1,k}$.
    $\mathbf{x}^{i,k+1} \leftarrow \mathbf{x}^{i,k} - \eta g(\mathbf{x}^{i,k}, \mathbf{s}^{i,k}) - \alpha(\mathbf{x}^{i,k} - \bar{\mathbf{x}}^k)$
  else
    $\mathbf{x}^{i,k+1} \leftarrow \mathbf{x}^{i,k} - \eta g(\mathbf{x}^{i,k}, \mathbf{s}^{i,k})$
  end if
  $k \leftarrow k + 1$
until k equals K

Algorithm 3 Round-Robin Elastic Averaging Algorithm for Master Model Parameter $\bar{\mathbf{x}}$

Initialize $\bar{\mathbf{x}}$
$k = 0$
repeat
  if $k \mod \tau = 0$ **then** // round-robin elastic averaging
    for $i = 1$ to N **do**
      Wait until $\mathbf{x}^{i,k}$ is ready.
      $\bar{\mathbf{x}}^k \leftarrow \bar{\mathbf{x}}^k + \alpha(\mathbf{x}^{i,k} - \bar{\mathbf{x}}^k)$
    end for
    $\bar{\mathbf{x}}^{k+1} \leftarrow \bar{\mathbf{x}}^k$
  end if
  $k \leftarrow k + 1$
until k equals K

Assumption 3 (Unbiased Gradients): We assume that the stochastic gradients are unbiased estimators of local objectives gradients such that

$$\mathbb{E}_{\mathbf{s}\sim\mathcal{D}_i}[g(\mathbf{x}, \mathbf{s})] = \nabla F(\mathbf{x}), \quad g(\mathbf{x}, \mathbf{s}) = \nabla\mathcal{L}(\mathbf{x}, \mathbf{s}). \tag{8}$$

Assumption 4 (Bounded Variance): We assume that the variance of stochastic gradients is bounded by some constants $\beta, \sigma \geq 0$ such that

$$\mathbb{E}_{\mathbf{s}\sim\mathcal{D}_i}[\|\nabla F(\mathbf{x}) - g(\mathbf{x}, \mathbf{s})\|^2] \leq \beta\|\nabla F(\mathbf{x})\|^2 + \sigma^2. \tag{9}$$

Assumption 5 (Mixing Matrix): We assume that the mixing matrix $\mathbf{W}$ satisfies $\mathbf{W}\mathbf{1} = \mathbf{1}$, $\mathbf{W}^\top = \mathbf{W}$. Besides, the magnitudes of all eigenvalues except the largest one are strictly less than 1 such that

$$\max\{|\lambda_2(\mathbf{W})|, \cdots\} < \lambda_1(\mathbf{W}) = 1. \tag{10}$$

In order to utilize the proof techniques in [20] and [22], we build a matrix-form update rule. Let matrices $\mathbf{X}_k$ and $\mathbf{G}_k \in \mathbb{R}^{(d\times(N+1))}$ be the stacks of all model parameters and stochastic gradients respectively, as follows:

$$\mathbf{X}_k = [\mathbf{x}^{1,k}, \cdots, \mathbf{x}^{N,k}, \bar{\mathbf{x}}^k], \tag{11}$$

$$\mathbf{G}_k = [g(\mathbf{x}^{1,k}, \mathbf{s}^{1,k}), \cdots, g(\mathbf{x}^{N,k}, \mathbf{s}^{N,k}), \mathbf{0}]. \tag{12}$$

Then, we can write the update rule of EA-Pipe as follows:

$$\mathbf{X}_{k+1} = (\mathbf{X}_k - \eta \cdot \mathbf{G}_k) \cdot \mathbf{S}_k, \tag{13}$$

where $\mathbf{S}_k \in \mathbb{R}^{(N+1)\times(N+1)}$ is the synchronization matrix which represents the mixing pattern between the local models and the master model. $\mathbf{S}_k$ is defined as follows:

$$\mathbf{S}_k = \begin{cases} \mathbf{W} & k \bmod \tau = 0 \\ \mathbf{I} & \text{Otherwise} \end{cases}. \tag{14}$$

The identity matrix $\mathbf{I}$ means that synchronization does not occur between the local models and the master model. On the other hand, $\mathbf{W}$ is called a *mixing matrix* which includes the synchronization operation between the local models and the master model based on the round-robin elastic averaging algorithm.

Remark 1: Instead of Eq. (13), one can use an alternative rule: $\mathbf{X}_{k+1} = \mathbf{X}_k \cdot \mathbf{S}_k - \eta \cdot \mathbf{G}_k$. However, according to [20], the convergence analysis on Eq. (13) can be extended to the alternative rule. Therefore, we will choose the update rule Eq. (13) to prove Theorem 1.

The mixing matrix $\mathbf{W}$ of the EA-Pipe algorithm does not satisfy Assumption 5. Nonetheless, in order to apply the proof in [20] without Assumption 5, we will first verify whether the mixing matrix $\mathbf{W}$ in the round-robin elastic averaging algorithm satisfies the following conditions.

1) $\mathbf{W}$ is a doubly-stochastic matrix.
2) $\mathbf{W}$ is a primitive and irreducible matrix.
3) $\mathbf{W}^\top\mathbf{W}$ is a positive matrix.

A. PROOF OF CONDITION 1

Lemma 1: Let $\mathbf{M}^l \in \mathbb{R}^{(N+1)\times(N+1)}$ denote the linear map between the l-th local model and the master model. For instance, when $N = 2$, we have $\mathbf{M}^1$ and $\mathbf{M}^2$ given by:

$$\mathbf{M}^1 = \begin{pmatrix} 1-\alpha & 0 & \alpha \\ 0 & 1 & 0 \\ \alpha & 0 & 1-\alpha \end{pmatrix}, \ \mathbf{M}^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1-\alpha & \alpha \\ 0 & \alpha & 1-\alpha \end{pmatrix},$$

where $0 < \alpha < 1$. Then, the mixing matrix $\mathbf{W} = \mathbf{M}^1 \times \cdots \times \mathbf{M}^N$ becomes a doubly-stochastic matrix.

Proof: Since $\mathbf{M}^l$ satisfies the following conditions, $\mathbf{M}^l$ is a doubly-stochastic matrix.

$$\sum_{i=1}^{N+1} \mathbf{M}^l_{ij} = \sum_{j=1}^{N+1} \mathbf{M}^l_{ij} = 1,$$

where $l \in \{1, 2, \cdots, N\}$, $\mathbf{M}_{ij} \geq 0$, and $i, j \in \{1, 2, \cdots, N+1\}$. According to Lemma 2, the product of two doubly-stochastic matrix is also a doubly-stochastic matrix. Therefore, the mixing matrix $\mathbf{W} = \mathbf{M}^1 \times \cdots \times \mathbf{M}^N$ is a doubly-stochastic matrix. □

Lemma 2: Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$ be doubly-stochastic matrices. Then $\mathbf{C} = \mathbf{AB}$ is also a doubly-stochastic matrix.

Proof: Let a_{ij}, b_{ij}, and c_{ij} denote the element in the i-th row and j-th column of matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, respectively. We can see that the sum of the element in each row of $\mathbf{C}$ is 1.

$$\begin{aligned} \sum_{i=1}^{n} c_{ij} &= \sum_{i=1}^{n} \Big(\sum_{k=1}^{n} a_{ik} b_{kj}\Big) \\ &= \sum_{k=1}^{n} \Big(b_{kj} \sum_{i=1}^{n} a_{ik}\Big) \\ &= \sum_{k=1}^{n} b_{kj} \qquad \because \sum_{i=1}^{n} a_{ik} = 1 \\ &= 1 \end{aligned}$$

Similarly, we can see that the sum of the element in each column of C is 1. Therefore, $\mathbf{C} = \mathbf{AB}$ is also a doubly stochastic matrix. Moreover, because $a_{ij}, b_{ij} \geq 0$ for all $1 \leq i, j \leq n$, one can ensure $c_{ij} \geq 0$. □

B. PROOF OF CONDITION 2

According to Lemma 3, we can see that $\mathbf{W}$ is a primitive and irreducible matrix.

Lemma 3: The mixing matrix $\mathbf{W} \in \mathbb{R}^{(N+1)\times(N+1)}$ in the round-robin elastic averaging algorithm can be written as follows:

$$\mathbf{W} = \begin{pmatrix} & & w_{1(N+1)} \\ & \mathbf{A} & \vdots \\ & & w_{N(N+1)} \\ w_{(N+1)1} & \cdots\ w_{(N+1)N} & w_{(N+1)(N+1)} \end{pmatrix}, \tag{15}$$

where $\mathbf{A}$ denotes the non-negative matrix with size of $\mathbb{R}^{N\times N}$ and w_{ij} is the entry of $\mathbf{W}$ at i-th row and j-th column.

Since all the local models are synchronized with the master model, the elements in the last row and the last column of $\mathbf{W}$ are always positive. Therefore, no matter what $\mathbf{A}$ is, $\mathbf{W}^2 = \mathbf{WW}$ always becomes a positive matrix. Consequently, $\mathbf{W}$ is a primitive and irreducible matrix.

C. PROOF OF CONDITION 3

According to Eq. (15), the entries in the last row and the last column of $\mathbf{W}$ is always positive, which means the entries in the last row and the last column of $\mathbf{W}^\top$ is also always positive. Therefore, $\mathbf{W}^\top\mathbf{W}$ becomes a positive matrix.

D. PROOF OF THEOREM 1

Now that we have confirmed that the mixing matrix $\mathbf{W}$ satisfies the required conditions, we can apply the convergence proof technique used in [20] and [22] without Assumption 5. Recall the update rule of the round-robin elastic averaging algorithm.

$$\mathbf{X}_{k+1} = (\mathbf{X}_k - \eta \cdot \mathbf{G}_k) \cdot \mathbf{S}_k$$
$$\mathbf{S}_k = \begin{cases} \mathbf{W} & k \bmod \tau = 0 \\ \mathbf{I}_k & \text{Otherwise} \end{cases}$$

Let $\mathbf{v} = [\frac{1}{N+1}, \cdots, \frac{1}{N+1}] \in \mathbb{R}^{N+1}$. Then, since $\mathbf{W}$ is a doubly-stochastic matrix, $\mathbf{Wv} = \mathbf{v}$ and hence $\mathbf{S}_k\mathbf{v} = \mathbf{v}$.

By multiplying $\mathbf{v}$ on both sides of the update rule, we have:

$$\mathbf{X}_{k+1}\mathbf{v} = \mathbf{X}_k\mathbf{v} - \eta \cdot \mathbf{G}_k\mathbf{v} \tag{16}$$

$$= \mathbf{X}_k\mathbf{v} - \frac{\eta}{N+1}\sum_{i=1}^{N} g(\mathbf{x}^{i,k}, \mathbf{s}^{i,k}). \tag{17}$$

To simplify the equation, we define an averaged variable $\mathbf{y}_k := \mathbf{X}_k\mathbf{v} = \frac{1}{N+1}\sum_{i=1}^{N}\mathbf{x}^{i,k} + \frac{1}{N+1}\bar{\mathbf{x}}^k$ and an effective learning rate $\eta_{\text{eff}} := \frac{N}{N+1}\eta$. Using these definitions, the update rule (17) becomes

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \frac{\eta_{\text{eff}}}{N}\sum_{i=1}^{N} g(\mathbf{x}^{i,k}, \mathbf{s}^{i,k}). \tag{18}$$

Following [27], [28], and [29], we analyze the convergence property of the round-robin elastic averaging algorithm with respect to $\mathbf{y}_k$.

By utilizing the intermediate result from the proof of Lemma 2 in [20] (specifically, Eq. (61) in [20]), we get the following equation (when $\eta_{\text{eff}}L\left(1+\frac{\beta}{N}\right) \le 1$):

$$\begin{aligned}
&\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}[\|\nabla F(\mathbf{y}_k)\|^2] \\
&\le \frac{2[F(\mathbf{y}_0) - F_{\text{inf}}]}{\eta_{\text{eff}}K} + \frac{\eta_{\text{eff}}L\sigma^2}{N} \\
&\quad + \frac{L^2}{KN}\sum_{k=0}^{K-1}\sum_{i=1}^{N}\mathbb{E}[\|\mathbf{y}_k - \mathbf{x}^{i,k}\|^2] \\
&\quad - \left[1 - \eta_{\text{eff}}L\left(\frac{\beta}{N}+1\right)\right]\frac{1}{KN}\sum_{k=0}^{K-1}\sum_{i=1}^{N}\mathbb{E}[\|\nabla F(\mathbf{x}^{i,k})\|^2].
\end{aligned} \tag{19}$$

We can derive an upper bound for the third term of the right hand side of Eq. (19) as follows:

$$\sum_{i=1}^{N}\|\mathbf{y}_k - \mathbf{x}^{i,k}\|^2 \le \sum_{i=1}^{N}\|\mathbf{y}_k - \mathbf{x}^{i,k}\|^2 + \|\mathbf{y}_k - \bar{\mathbf{x}}^k\|^2 \tag{20}$$

$$= \|\mathbf{X}_k(\mathbf{I} - \mathbf{v}\mathbf{1}^\top)\|_{\text{F}}^2, \tag{21}$$

where $\|\cdot\|_{\text{F}}$ is the Frobenius matrix norm.

According to the update rule (13) and repeatedly using the fact $\mathbf{W}\mathbf{v} = \mathbf{v}$, $\mathbf{1}^\top\mathbf{W} = \mathbf{1}^\top$ and $\mathbf{v}^\top\mathbf{1} = 1$, we have:

$$\mathbf{X}_k(\mathbf{I} - \mathbf{v}\mathbf{1}^\top) = (\mathbf{X}_{k-1} - \eta\mathbf{G}_{k-1})\mathbf{S}_{k-1}(\mathbf{I} - \mathbf{v}\mathbf{1}^\top) \tag{22}$$

$$= -\eta\sum_{j=0}^{k-1}\mathbf{G}_j\left(\prod_{s=j}^{k-1}\mathbf{S}_s - \mathbf{v}\mathbf{1}^\top\right). \tag{23}$$

Therefore,

$$\sum_{i=1}^{N}\|\mathbf{y}_k - \mathbf{x}^{i,k}\|^2 \le \eta^2\left\|\sum_{j=0}^{k-1}\mathbf{G}_j\left(\prod_{s=j}^{k-1}\mathbf{S}_s - \mathbf{v}\mathbf{1}^\top\right)\right\|_{\text{F}}^2, \tag{24}$$

where

$$\prod_k \mathbf{S}_k = \prod_k \mathbf{W}_k. \tag{25}$$

In order to keep utilizing the proof sequence of [20], we need to ensure that $\|\mathbf{W}^n - \mathbf{v}\mathbf{1}^\top\|_{\text{op}}$ must be strictly less than 1, where $\|\cdot\|_{\text{op}}$ is the operator norm. The following lemmas guarantees that $\|\mathbf{W}^n - \mathbf{v}\mathbf{1}^\top\|_{\text{op}} < 1$.

Lemma 4: Let $\mathbf{W}$ *and* $\mathbf{J} \in \mathbb{R}^{n\times n}$ *be a asymmetric doubly stochastic matrix with non-negative element and* $\mathbf{1}\mathbf{1}^\top/\mathbf{1}^\top\mathbf{1}$, *respectively. Then, the operator norm on* $\mathbf{W} - \mathbf{J}$ *is always less than 1.*

$$\|\mathbf{W} - \mathbf{J}\|_{\text{op}} = \zeta < 1 \tag{26}$$

Proof: The operator norm of $\mathbf{A}$ *is defined as* $\sqrt{\lambda_{max}(\mathbf{A}^\top\mathbf{A})}$, *where* $\lambda_{max}(\mathbf{A}^\top\mathbf{A})$ *is the maximum eigenvalue of* $\mathbf{A}^\top\mathbf{A}$. *That is,* $\|\mathbf{W} - \mathbf{J}\|_{\text{op}}$ *is the square root on the maximum eigenvalue of* $(\mathbf{W} - \mathbf{J})^\top(\mathbf{W} - \mathbf{J})$. $(\mathbf{W} - \mathbf{J})^\top(\mathbf{W} - \mathbf{J})$ *can be expanded as follows*:

$$\begin{aligned}
(\mathbf{W} - \mathbf{J})^\top(\mathbf{W} - \mathbf{J}) &= (\mathbf{W}^\top - \mathbf{J}^\top)(\mathbf{W} - \mathbf{J}) \\
&= \mathbf{W}^\top\mathbf{W} - \mathbf{J}^\top\mathbf{W} - \mathbf{W}^\top\mathbf{J} + \mathbf{J}^\top\mathbf{J} \\
&= \mathbf{W}^\top\mathbf{W} - \mathbf{J} - \mathbf{J} + \mathbf{J} \\
&= \mathbf{W}^\top\mathbf{W} - \mathbf{J}.
\end{aligned} \tag{27}$$

Here, we can observe the followings:

1) *Both* $\mathbf{W}$ *and* $\mathbf{W}^\top$ *are doubly stochastic matrices.*
2) $\mathbf{W}^\top\mathbf{W}$ *is a symmetric real matrix, which is diagonalizable [30]. Also, according to Lemma 3, it is a positive matrix.*
3) *According to Lemma 2,* $\mathbf{W}^\top\mathbf{W}$ *is a doubly stochastic matrix.*
4) *Let* $\mathbf{W}^\top\mathbf{W} = \mathbf{Z}$, *then* $\mathbf{Z}\mathbf{J} = \mathbf{J}\mathbf{Z}$.
5) *Since all elements of* $\mathbf{W}$ *are non-negative, all elements of* $\mathbf{Z}$ *are also non-negative.*

Based on the above observations, we can diagonalize $\mathbf{Z}$ *and* $\mathbf{J}$ *simultaneously. We decompose* $\mathbf{Z}$ *as follows*:

$$\mathbf{Z} = \mathbf{Q}\Lambda\mathbf{Q}^\top, \text{ where } \Lambda = diag\{\lambda_1(\mathbf{Z}), \lambda_2(\mathbf{Z}), \cdots, \lambda_n(\mathbf{Z})\}. \tag{28}$$

Since $\mathbf{Z}$ *is a doubly stochastic matrix, the maximum eigenvalue of* $\mathbf{Z}$, $\lambda_1(\mathbf{Z})$, *is 1. Similarly, the matrix* $\mathbf{J}$ *can be decomposed as* $\mathbf{Q}\Lambda_\mathbf{0}\mathbf{Q}^\top$ *where* $\Lambda_\mathbf{0} = diag\{1, 0, \cdots, 0\}$. *Then, we have*:

$$\mathbf{Z} - \mathbf{J} = \mathbf{Q}(\Lambda - \Lambda_\mathbf{0})\mathbf{Q}^\top. \tag{29}$$

Now, the maximum eigenvalue of $\mathbf{Z} - \mathbf{J}$ *is* $max\{0, \lambda_2(Z), \cdots, \lambda_n(Z)\}$. *Here, we consider two cases*:

- *If all* $\lambda_2(\mathbf{Z}), \cdots, \lambda_n(\mathbf{Z})$ *are negative, the maximum eigenvalue of* $\mathbf{Z} - \mathbf{J}$ *is 0.*
- *If one of* $\lambda_2(\mathbf{Z}), \cdots, \lambda_n(\mathbf{Z})$ *is positive, the maximum eigenvalue of* $\mathbf{Z} - \mathbf{J}$ *is positive. Furthermore, according to Perron-Frobenius theorem [31], [32], since* $\mathbf{Z}$ *is a positive primitive stochastic matrix, there is only one eigenvalue 1. All other eigenvalues are smaller than 1.*

TABLE 8. Image classification error rates (%) of the four training methods using the CIFAR-10 dataset and the VGG-16 model with varying number of GPUs.

	1-GPU	2-GPU	4-GPU	8-GPU
SGD	7.31	-	-	-
PipeDream	-	7.83	7.77	7.86
SpecTrain	-	9.90	9.49	9.84
EA-Pipe	-	8.02	7.88	8.09

As a result, $\|\mathbf{W}-\mathbf{J}\|_{\text{op}}=\zeta$ *is non-negative and strictly less than 1.* □

Lemma 5: Let $\mathbf{W}$ *and* $\mathbf{J}\in\mathbb{R}^{n\times n}$ *be an asymmetric doubly stochastic matrix with non-negative element and* $\mathbf{1}\mathbf{1}^\top/\mathbf{1}^\top\mathbf{1}$*, respectively. Then, the operator norm on* $\mathbf{W}^n-\mathbf{J}$ *is always less than or equal to* ζ^n*, where* ζ *is the operator norm on* $\mathbf{W}-\mathbf{J}$*.*

$$\|\mathbf{W}^n-\mathbf{J}\|_{\text{op}}\leq\zeta^n,\quad \textit{where}\quad \|\mathbf{W}-\mathbf{J}\|_{\text{op}}=\zeta \tag{30}$$

Proof: We can prove it by induction on n.

Base case ($n=1$*): We have* $\|\mathbf{W}^1-\mathbf{J}\|_{\text{op}}=\zeta^1\leq\zeta$.

Inductive hypothesis: Assume that the equation holds for $n=k$.

Inductive step: Let $n=k+1$*, then*

$$\begin{aligned}\|\mathbf{W}^{k+1}-\mathbf{J}\|_{\text{op}}&=\|(\mathbf{W}-\mathbf{J})(\mathbf{W}^k-\mathbf{J})\|_{\text{op}}\\&\leq\|\mathbf{W}-\mathbf{J}\|_{\text{op}}\|\mathbf{W}^k-\mathbf{J}\|_{\text{op}}\\&\leq\zeta\cdot\zeta^k\ \because \textit{Inductive hypothesis}\\&=\zeta^{k+1}.\end{aligned}$$

Now we complete the proof on Lemma 5. □

By Lemma 5, we do not need Assumption 5 in [20], and we can apply the same procedure of Appendix D.2 in [20] to get the following final result:

$$\begin{aligned}&\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}[\|\nabla F(\mathbf{y}_k)\|^2]\\&\quad\leq\frac{2[F(\mathbf{y}_0)-F_{\text{inf}}]}{\eta_{\text{eff}}K}+\frac{\eta_{\text{eff}}L\sigma^2}{N}\\&\qquad+\eta_{\text{eff}}^2L^2\sigma^2\left(\frac{1+\zeta^2}{1-\zeta^2}\tau-1\right)\left(1+\frac{1}{N}\right)^2.\end{aligned}\tag{31}$$

APPENDIX B
BEST CASE RESULTS OF THE SMALL-SCALE EXPERIMENT

Table 8 shows the best case results of the four training methods on the CIFAR-10 dataset using the VGG-16 model with varying number of GPUs. We observed the similar trends as in Table 1.

APPENDIX C
BEST CASE RESULTS OF THE MID-SCALE EXPERIMENT

Table 9 shows the best case results of the four training methods using the CIFAR-100 dataset and the ResNet-34 model on 8 GPUs. We observed the similar trends as in Table 4.

TABLE 9. Image classification error rates (%) of the four training methods using the CIFAR-100 dataset and the ResNet-34 model on 8 GPUs.

batch size	16			32			64		
learning rate	0.1	0.01	0.0003	0.1	0.01	0.0005	0.1	0.01	0.001
SGD	24.33	22.93	-	22.50	24.46	-	23.22	27.10	-
PipeDream	31.01	23.21	-	27.28	26.18	-	30.08	31.07	-
SpecTrain	-	-	26.49	-	-	28.50	-	-	32.85
EA-Pipe	36.43	**23.07**	**-**	27.48	24.93	-	25.93	29.13	-

TABLE 10. Image classification error rates (%) of the three training methods using the ImageNet dataset and the ResNet-50 model on 50 GPUs.

	1-GPU	50-GPU
SGD	23.49	-
PipeDream	-	26.28
EA-Pipe	**-**	**25.79**

APPENDIX D
BEST CASE RESULTS OF THE LARGE-SCALE EXPERIMENT

Table 10 shows the best case results of the three training methods using the ImageNet dataset and the ResNet-50 model on 50 GPUs. We observed the similar trends as in Table 7.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1. Minneapolis, MN, USA, 2019, pp. 4171–4186.

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. *Improving Language Understanding by Generative Pre-Training*. Accessed: 2018, [Online]. Available: https://www.mikecaptain.com/resources/pdf/GPT-1.pdf

[5] E. Yu, D. Dong, and X. Liao, "Communication optimization algorithms for distributed deep learning systems: A survey," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 12, pp. 3294–3308, Dec. 2023.

[6] X. Miao, Y. Wang, Y. Jiang, C. Shi, X. Nie, H. Zhang, and B. Cui, "Galvatron: Efficient transformer training over multiple GPUs using automatic parallelism," *Proc. VLDB Endowment*, vol. 16, no. 3, pp. 470–479, Nov. 2022.

[7] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, "Varuna: Scalable, low-cost training of massive deep learning models," in *Proc. 17th Eur. Conf. Comput. Syst.*, Rennes, France, Mar. 2022, pp. 472–487.

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

[9] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 23. Vancouver, BC, Canada, 1990, pp. 2595–2603.

[10] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 24. Granada, Spain, 2011, pp. 693–701.

[11] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25. Lake Tahoe, NV, USA, 2012, pp. 1223–1231.

[12] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Proc. Adv. Neural Inf. Proces. Syst.*, vol. 28. Montreal, QC, Canada, 2015, pp. 2737–2745.

[13] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32. Vancouver, BC, Canada, 2019, pp. 103–112.

[14] A. Petrowski, G. Dreyfus, and C. Girault, "Performance analysis of a pipelined backpropagation parallel algorithm," *IEEE Trans. Neural Netw.*, vol. 4, no. 6, pp. 970–981, Jun. 1993.

[15] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized pipeline parallelism for DNN training," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Huntsville, ON, Canada, Oct. 2019, pp. 1–15.

[16] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia, L. Diao, X. Liu, and W. Lin, "DAPPLE: A pipelined data parallel approach for training large models," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Feb. 2021, pp. 431–445.

[17] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel DNN training through model parallelism on multi-GPU platform," 2018, *arXiv:1809.02839*.

[18] A. Kosson, V. Chiley, A. Venigalla, J. Hestness, and U. Koster, "Pipelined backpropagation at scale: Training large models without batches," in *Proc. Mach. Learn. Sys.*, vol. 3, 2021, pp. 479–501.

[19] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging SGD," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28. Montreal, QC, Canada, 2015, pp. 685–693.

[20] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of local-update sgd algorithms," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 9709–9758, 2021.

[21] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 22. Vancouver, BC, Candada, 2009, pp. 2331–2339.

[22] J. Wang, H. Liang, and G. Joshi, "Overlap local-SGD: An algorithmic approach to hide communication delays in distributed SGD," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 8871–8875.

[23] A. Koloskova, S. U. Stich, and M. Jaggi, "Sharper convergence guarantees for asynchronous SGD for distributed and federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, New Orleans, LA, USA, 2022.

[24] Z. Chen, C. Xu, W. Qian, and A. Zhou, "Elastic averaging for efficient pipelined DNN training," in *Proc. 28th ACM SIGPLAN Annu. Symp. Princ. Pract. Parallel Program.*, Montreal, QC, Candada, Feb. 2023, pp. 380–391.

[25] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 12, pp. 10665–10673.

[26] M. R. Hestenes, *Optimization Theory: The Finite Dimensional Case*. Hoboken, NJ, USA: Wiley, 1975.

[27] X. Lian, C. Zhang, H. Zhang, C. J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30. Long Beach, CA, USA, 2017, pp. 5330–5340.

[28] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[29] K. Yuan, Q. Ling, and W. Yin, "On the convergence of decentralized gradient descent," *SIAM J. Optim.*, vol. 26, no. 3, pp. 1835–1854, Jan. 2016.

[30] M. Oliverira. *Some Facts on Symmetric Matrices*. Accessed: 2008. [Online]. Available: http://maecourses.ucsd.edu/~mdeolive/mae280a/lecture11.pdf

[31] E. Kani. *Primitive Stochastic Matrices*. Accessed: 2022. [Online]. Available: https://mast.queensu.ca/~math211/m211oh/m211oh98.pdf

[32] O. Knill. *Lecture 34: Perron Frobenius Theorem*. Accessed: 2011. [Online]. Available: https://people.math.harvard.edu/~knill/teaching/math19b_2011/handouts/lecture34.pdf

BONGWON JANG received the B.S. degree from Korea University, Seoul, South Korea, in 2021, where he is currently pursuing the degree with the Artificial Intelligence Laboratory. His research interests include parallel computing and machine learning.

IN-CHUL YOO (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Korea University, Seoul, South Korea, in 2006, 2008, and 2015, respectively. He is currently a Research Professor with the Artificial Intelligence Laboratory, Korea University. His research interests include robust speech recognition and speaker recognition.

DONGSUK YOOK (Member, IEEE) received the B.S. and M.S. degrees in computer science from Korea University, Seoul, South Korea, in 1990 and 1993, respectively, and the Ph.D. degree in computer science from Rutgers University, New Brunswick, NJ, USA, in 1999.

From 1999 to 2001, he was with the IBM Thomas J. Watson Research Center, USA, where he worked on speech recognition. He is currently a Professor with the Department of Computer Science and Engineering, Korea University. He is also the Director of the Artificial Intelligence Laboratory, Korea University. His research interests include machine learning and speech processing.

Prof. Yook is a member of the Acoustical Society of Korea and the Korean Institute of Information Scientists and Engineers.