NAST: In-Network Aggregation with Worker Selection for Accelerating Distributed Training

Jianfeng Bao[1,3] Peng Yang[1,3] Gongming Zhao[1,3] Huihui Tang[2] Hongli Xu[1,3] Qianpiao Ma[4]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Guangxi Key Laboratory of Digital Infrastructure, Guangxi Zhuang Autonomous Region Information Center
[3]Suzhou Institute for Advanced Research, University of Science and Technology of China
[4]Nanjing University of Science and Technology

2026 IEEE/ACM International Symposium on Quality of Service (IWQoS) | 979-8-3195-1703-6/26/$31.00 ©2026 IEEE | DOI: 10.1109/IWQoS70441.2026.11661119

***Abstract*—The rising scale of distributed training (DT) leads to significant communication overhead in clusters. The vast majority of communication loads are generated based on the all-reduce communication primitive. To optimize the all-reduce communication, in-network aggregation (INA) has emerged as a promising solution. Specifically, INA utilizes programmable switches to aggregate gradients within the network, thereby reducing the communication overhead. Existing DT with INA schemes primarily aggregate gradients from all workers in each iteration. However, the system environment is subject to dynamic resource competition and network congestion. As a result, the completion time of DT depends on the slowest worker, *i.e.*, the straggler. To tackle this challenge, we propose a framework called NAST, the first-of-its-kind framework that combines INA with worker selection to accelerate DT. Specifically, we formulate the problem of INA with worker selection and theoretically analyze the convergence of NAST. Based on the convergence result, we design an efficient algorithm to dynamically determine the number of participating workers in each iteration. Then, NAST selects participants for INA based on the arrival order of workers' gradients. We implement NAST on a testbed comprising 2 Intel Tofino switches and 9 machines. Experimental results demonstrate that NAST reduces the training time by 16.3%-29.8% and traffic consumption by 25.8%-42.4% compared to the state-of-the-art solutions.**

***Index Terms*—Programmable Network, In-Network Aggregation, Worker Selection, Distributed Training**

I. Introduction

In recent years, deep neural networks (DNNs) have been extensively adopted in various domains such as natural language processing [1], computer vision [2], and recommendation systems [3]. However, training these advanced models (*e.g.*, GPT-4 with 1.8 trillion parameters) necessitates distributed training across multiple GPUs, which inevitably incurs significant communication overhead. According to [4], when training DeepLight with only 8 workers and 100Gbps link bandwidth, the communication time accounts for approximately 79% of the total training time.

Since distributed training necessitates synchronizing model parameters or gradients across all workers at every iteration, all-reduce has become the primary source of communication overhead [5, 6]. Traditionally, this synchronization relies on the Parameter Server (PS) architecture [7], where a central server aggregates updates and broadcasts results. However, this centralized design inherently suffers from a severe bandwidth bottleneck at the server side. To alleviate this, modern frameworks adopt decentralized schemes like ring all-reduce [8] to fully utilize the bandwidth of every node; nevertheless, this approach comes at the cost of transmitting a significantly larger total volume of data.

To alleviate these bottlenecks in both centralized and decentralized architectures, some researchers propose in-network aggregation (INA) schemes [4, 9]. Specifically, INA usually requires the GPU nodes participating in all-reduce to send their intermediate data to a programmable switch, which, utilizing its high bandwidth and computing capabilities, processes the data at line rate on the device and sends back the complete results. For example, typical work like SwitchML [4] offloads the entire aggregation task to programmable switches, but still requires waiting for all gradients to finish aggregation. This design introduces dependencies for the reason that the duration of each iteration depends on the slowest worker (*i.e.*, straggler). It is apparent that the presence of stragglers results in an unacceptable prolongation of the training process. Moreover, in case of worker failures, the training process is forced to pause until corresponding workers recover from the failure status, which also leads to an extended training completion time. In practice, the straggler problem is inevitable because of system dynamics like load imbalance, resource competition, or network congestion [10, 11]. To mitigate the negative impact of stragglers on the performance of DT, an appropriate solution tailored for INA is very necessary.

Some studies [9, 12] propose different solutions, such as timeout strategy or congestion control, to address the straggler problem. For example, authors in Trio [12] set a timeout threshold to wait for gradients in each iteration. Therefore, gradients from slow workers will be directly discarded by programmable switches. However, Trio adopts static threshold settings but lacks flexibility for dynamic scenarios. Moreover, this approach wastes the computation resources of straggling workers. ATP [9] transfers gradients of stragglers directly to the PS and designs a congestion control mechanism to synchronize gradient transmission rates of workers, alleviating network bottlenecks on the PS. As a result, it reduces the probability of network-induced stragglers. Nonetheless, when encountering stragglers caused by dynamic resource competition,

979-8-3195-1703-6/26/$31.00 © 2026 IEEE

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

ATP still suffers from a long completion time. Consequently, to efficiently alleviate the straggler issue, seeking alternative solutions as supplementary methods is imperative.

To this end, we hope to select an appropriate number of fast workers in each iteration to participate in gradient aggregation. Besides, we allow slower workers to participate in gradient aggregation in subsequent iterations. Actually, it is not a trivial task to perform in-network aggregation with worker selection for accelerating DT. First, system dynamics result in different training times of one iteration for each worker. To this end, an appropriate number (represented by K_t) of participating workers needs to be dynamically determined during each iteration t, which is challenging. To eliminate the impact of calculating K_t, we basically overlap the execution time of calculation with the training time of the previous iteration. Second, we also need to ensure that programmable switches aggregate gradients from globally fast workers in each iteration. Specifically, each programmable switch faces challenges in accurately perceiving the order of gradient arrivals from all workers due to the fact that some workers are not directly connected to it. Consequently, we need to explicitly instruct each programmable switch to aggregate gradients from specific workers in each iteration. To handle these challenges, we propose NAST, and the main contributions of this paper are as follows:

1) We design the framework called NAST, the first-of-its-kind work that combines in-network aggregation with worker selection for accelerating DT. We also provide the problem formulation for INA with worker selection and theoretically analyze the convergence of NAST.
2) Upon the convergence result, an efficient algorithm is proposed to determine the number of participating workers in each iteration. Moreover, NAST provides a mechanism to explicitly instruct switches to aggregate gradients from those fast workers in each iteration.
3) We conduct the testbed with 2 Intel Tofino switches and 9 machines to validate the effectiveness of NAST. Experimental results show that NAST reduces the training time by 26.8%-42.1% and traffic consumption by 28.2%-40.3% for achieving similar test accuracy, compared to the state-of-the-art solutions.

II. System Model and NAST Workflow

In this section, we first analyze the communication bottlenecks of existing all-reduce architectures to highlight the necessity of In-Network Aggregation (INA). We then formalize the system model for distributed training and detail the workflow of our proposed NAST framework. Finally, we present a motivating example to intuitively illustrate how dynamically combining INA with worker selection effectively mitigates the straggler problem and accelerates the distributed training process.

A. Analysis of All-reduce

All-reduce is the dominant communication primitive in distributed training. In this section, we analyze the communication costs of the Parameter Server (PS) versus decentralized architectures. We assume n GPUs participate in the process, with each holding intermediate data of size V.

In the PS architecture, each GPU sends its local data to a central server, which sums and broadcasts the global result. Thus, each GPU transmits and receives a total data volume of V. However, the PS creates a critical bottleneck, as the central server's bandwidth is saturated by simultaneous incoming and outgoing traffic from all workers.

Conversely, ring all-reduce, a popular decentralized scheme, organizes GPUs in a logical ring. The operation requires $2 \cdot (n-1)$ communication steps, where each GPU sends a data chunk of size $\frac{V}{n}$ to its neighbor per step. This results in a total per-GPU communication volume of $\frac{2\cdot(n-1)\cdot V}{n}$. Although this volume exceeds that of the PS architecture and grows with n, ring all-reduce remains the mainstream choice as it fully leverages the bandwidth of all GPUs.

Nevertheless, In-Network Aggregation (INA) offers a promising solution to the PS bandwidth limitation. By leveraging programmable switches, different GPUs can send data to distinct switch ports, enabling parallel processing and aggregation at line rate. Motivated by the optimal per-worker communication volume of the PS architecture compared to ring all-reduce, we propose accelerating all-reduce by offloading the aggregation logic to programmable switches, thereby combining the efficiency of the PS model with high-performance networking hardware.

B. System Model for Distributed Training

We consider a cluster consisting of a compute node set and a programmable switch set.

1) Compute nodes include the PS Ω and a worker set $W = \{w_1, w_2, \ldots, w_{|W|}\}$ for global aggregation and model training, respectively. The number of all workers is n, *i.e.*, $|W| = n$.
2) Programmable switches are responsible for in-network gradient aggregation. Let $S = \{s_1, s_2, \ldots, s_{|S|}\}$ denote the programmable switch set. We assume that programmable switches are top-of-rack (ToR) switches in the cluster, and the set of workers connected to switch s_j is denoted as $W^j \subseteq W$.

We first assume that multiple workers train models on training data with independent and identical distribution. The training objective is to minimize the loss function, which can be formulated as follows:

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi^i\sim\mathbb{D}^i}\left[f(x,\xi^i)\right] \tag{1}$$

where $x \in \mathbb{R}^d$ is the model parameters shared by each worker, $\mathbb{D}^i$ represents the local dataset of worker w_i, and $f(x,\xi^i)$ is the loss function associated with model parameters x and a random sample (or a mini-batch) ξ^i from the local dataset $\mathbb{D}^i$.

For each worker, it is required to transmit gradients to the PS during each iteration. We use $g_t^i = \nabla f(x_t^i, \xi^i)$ to denote the gradient of local model parameters x_t^i on worker w_i in iteration t. Assume that the PS receives gradients from a set

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Workflow of NAST

1: Initialize model parameters x_0, the value K_0
2: **for** iteration $t \in \{0, 1, ..., T-1\}$ **do**
3: **Processing at Worker** w_i
4: **if** Receive model parameters from the PS **then**
5: Perform local training
6: Send available bandwidth and request for pushing g_t^i
7: Get a response and push g_t^i to the switch
8: Send computation time for updating K_{t+1}
9: **Processing at Programmable Switch** s_j
10: Initialize $W^{j'} \leftarrow \emptyset$ and receive K_t^j
11: **while** $|W^{j'}| \leq K_t^j$ **do**
12: Perform INA when receiving g_t^i from w_i
13: Update $W^{j'} \leftarrow W^{j'} \cup \{w_i\}$
14: **if** $|W^{j'}| = K_t^j$ **then**
15: Push the aggregated gradient to the PS
16: Update $W^{j'} \leftarrow \emptyset$
17: **Processing at the PS**
18: Response to the fastest K_t workers
19: Send K_t^j to each switch s_j
20: Receive aggregated gradients from switches and perform global aggregation
21: Update global model by Eqs. (2) and (3)
22: Distribute the updated model to K_t workers
23: Collect computation time from the K_t workers
24: Calculate K_{t+1} by Alg. 2

of fast workers W_t in iteration t, where the size of W_t is K_t. We define $\bar{g}_t$ as the aggregated gradient of the K_t workers in iteration t, and $\bar{g}_t$ is achieved as follows:

$$\bar{g}_t = \frac{1}{K_t} \sum_{w_i \in W_t} g_t^i \tag{2}$$

Then, the PS can leverage the mini-batch stochastic gradient descent (SGD) algorithm to update parameters. We use x_t to indicate model parameters on the PS in iteration t and set η as the learning rate. Consequently, new parameters x_{t+1} is achieved by Eq. (3).

$$x_{t+1} = x_t - \eta \bar{g}_t \tag{3}$$

Finally, the PS sends new model parameters back to workers in W_t for the next iteration.

C. Workflow of NAST

We propose the workflow of our DT framework called NAST, which is formally described in Alg. 1. Since the cluster primarily consists of a PS, a set of workers, and a set of switches, we describe the workflow on these three sides.

On the worker side (Lines 3-8), worker w_i starts training when receiving model parameters from the PS. After gradient computation is completed, worker w_i first notifies the PS with available bandwidth resources b_t^i of w_i and asks for pushing gradient g_t^i. We determine b_t^i by calculating the ratio of gradient traffic volume to concurrent communication volume of worker w_i. Once the fastest K_t workers are determined by the PS, they will receive a response containing the sending rate $\widehat{b}_t$ which helps synchronize the rate of gradient transmission. Note that, rate synchronization is a common technique to optimize the utilization of memory resources in programmable switches for INA schemes [4, 9]. Simultaneously, another thread on each worker sends computation time when gradient transmission is ready, which can assist in calculating the optimal value of K_{t+1} in Alg. 2.

On the switch side (Lines 9-16), switch s_j aggregates gradients from K_t^j workers in iteration t. Specifically, switch s_j receives the value of K_t^j from the PS at the beginning of each iteration. Then, using a temporary variable $W^{j'}$ to record the number of participating workers in the current iteration, switch s_j performs in-network aggregation on gradients from K_t^j workers, *i.e.*, $|W^{j'}| = K_t^j$. After that, each switch sends the aggregated gradient to the PS.

On the PS side (Lines 17-24), the PS responds to the fastest K_t workers and notifies them to transmit gradients at rate $\widehat{b}_t$, which is the minimum bandwidth among the fastest K_t workers. Simultaneously, the PS sends the corresponding value of K_t^j to each switch s_j. On receiving aggregated gradients from switches, the PS performs global aggregation and updates the global model. After that, the PS sends updated parameters to the fastest K_t workers. Similarly, another thread on the PS is responsible for receiving computation time and updating the value of K_t for each iteration t.

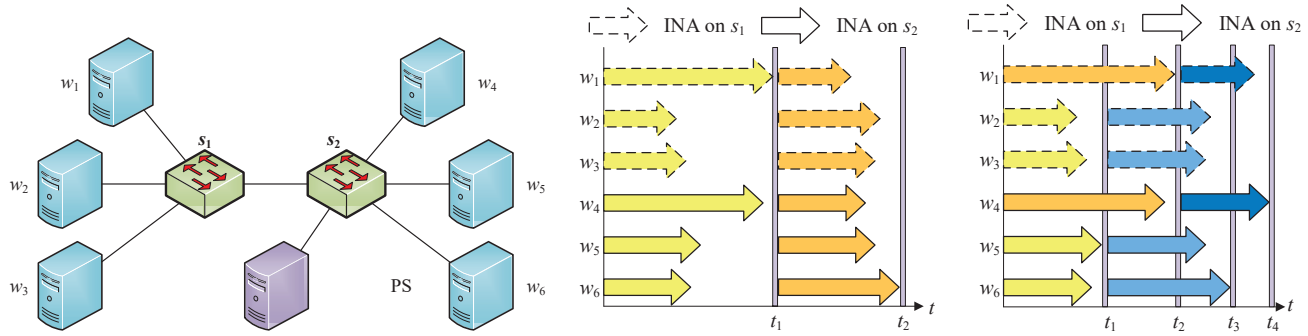
D. A Motivating Example

This section presents an example to illustrate the advantages of in-network aggregation with worker selection. Assuming a DT task that utilizes 1 PS, 2 programmable switches (denoted by s_1, s_2), and 6 workers (represented by $w_1, .., w_6$), the topology is visualized in Fig. 1(a). Specifically, each programmable switch is connected to three workers, and the PS is linked to one programmable switch. Then, we depict the motivating example in Fig. 1. The dotted arrows represent that workers w_1, w_2, and w_3 send gradients to switch s_1 for in-network aggregation. Similarly, the solid arrows denote that switch s_2 can aggregate gradients from w_4, w_5, and w_6. Besides, we use different colors to distinguish different training iterations in Figs. 1(b)-1(c).

The DT process of the traditional INA scheme is shown in Fig. 1(b). When local training on w_1, w_2, and w_3 is completed, these workers first transmit their gradients to switch s_1. On receiving gradients from 3 workers, switch s_1 performs in-network aggregation and sends the aggregated gradient to the PS. Similarly, switch s_2 performs the same operation on gradients from workers w_4, w_5, and w_6. Once the PS receives gradients from s_1 and s_2, it performs global aggregation and updates model parameters. Finally, the PS transfers the updated model back to all workers for the next iteration. Since the traditional INA scheme needs to aggregate gradients from all workers including straggling workers, it leads to the increasing completion time of one iteration.

We show the DT process of NAST in Fig. 1(c). With a determined value of K_t, the PS allows the fastest K_t workers to transmit their gradients. Specifically, in the first iteration, workers w_2, w_3, w_5, and w_6 are the fastest 4 workers. Given

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.



(a) Topology (b) Traditional INA (c) NAST

Fig. 1: The motivating example contains 1 PS, 2 switches (*i.e.*, s_1, s_2), and 6 workers (*i.e.*, $w_1, ..., w_6$). Fig. 1(a) shows the network topology. Figs. 1(b)-1(c) show the traditional INA and NAST, respectively. The dotted arrows represent INA on s_1, and the solid arrows denote INA on s_2. We use different colors to distinguish different iterations. In Fig. 1(b), the PS waits for gradients from all workers to perform global aggregation, where the completion time of one iteration is determined by the straggler. In Fig. 1(c), the PS aggregates gradients from the fastest K_t workers (*e.g.*, $K_1=4$, $K_2=2$, $K_3=4$, and $K_4=2$). Compared to the traditional INA, NAST performs two additional iterations, which indicates the potential for accelerating DT.

$K_1 = 4$, the PS responds to the 4 workers. Meanwhile, the PS notifies each switch to individually aggregate gradients from two workers because the PS knows the respective connections between the fastest workers and switches. Thus, switch s_1 performs in-network aggregation on gradients from workers w_2, w_3 and sends the aggregated gradient to the PS. Switch s_2 runs a similar operation when receiving gradients from w_5 and w_6. After both switches s_1 and s_2 push the aggregated gradient to the PS, the PS globally aggregates gradients and performs a model update. Finally, the PS transfers the new model back to workers w_2, w_3, w_5, and w_6 for the next iteration. Besides, considering system dynamics like resource competition or network congestion, NAST dynamically determines the value of K_t in different iterations, *e.g.*, $K_2 = 2$, $K_3 = 4$, and $K_4 = 2$ in the following iterations. In our example, NAST will perform 4 iterations within the same time while the traditional INA scheme only performs 2 iterations, which indicates the potential for accelerating DT. This paper focuses on how to dynamically determine the optimal number of fast workers in each iteration, thereby accelerating DT.

III. Problem Formulation and Convergence Analysis

A. Problem Formulation

We define the problem of in-network aggregation with worker selection as below. Given a training task, we will dynamically determine the number K_t and aggregate gradients from the fastest K_t workers in each iteration t to minimize the completion time of the training task. To facilitate in-network aggregation on switches, the PS explicitly specifies that each switch s_j aggregates gradients from K_t^j workers during iteration t. Besides, we use W_t to denote the set of workers that participate in the gradient aggregation in iteration t. With W^j consisting of workers lined to switch j, j aggregates gradients from the worker set $W^j \cap W_t = W_t^j$ in iteration t. The completion time of iteration t for worker w_i consists of two components: computation time c_t^i for calculating gradients until gradient transmission is ready and communication time for transmitting gradients. Specifically, we utilize the available bandwidth resources b_t^i to evaluate the communication time. Assuming the size of the traffic in each iteration is B, the communication time for worker w_i during iteration t is set to $\frac{B}{b_t^i}$. In practice, to fully utilize the switch for in-network aggregation, all selected workers maintain a consistent sending rate $\hat{b}_t$ within iteration t. Therefore, we set the sending rate constraint as $\hat{b}_t \leq b_t^i, \forall w_i \in W_t, \forall t \in [T]$. Consequently, the completion time for iteration t is represented as follows:

$$h_t = \max\{c_t^i + \frac{B}{\hat{b}_t}\}, \forall w_i \in W_t, \forall t \in [T] \tag{4}$$

where we overlap model distribution time and gradient transmission time like [9, 13]. Then, the completion time of the training task is achieved by $\sum_{t=1}^{T} h_t$. Finally, the problem can be formulated by:

$$\min \sum_{t=1}^{T} h_t$$

$$S.t. \begin{cases} f(x_T) \leq \mathcal{F} & \\ K_t^j = \sum_{w_i \in (W^j \cap W_t)} 1, & \forall t \in [T], \forall s_j \in S \\ K_t = \sum_{s_j \in S} K_t^j, & \forall t \in [T] \\ \hat{b}_t \leq b_t^i, & \forall w_i \in W_t, \forall t \in [T] \\ h_t = \max\{c_t^i + \frac{B}{\hat{b}_t}\}, & \forall w_i \in W_t, \forall t \in [T] \\ K_t \in \{1, 2, 3, ..., n\}, & \forall t \in [T] \\ K_t^j \in \{1, 2, ..., |W^j|\}, & \forall t \in [T], \forall s_j \in S \end{cases} \tag{5}$$

The first set of inequalities expresses the convergence requirement, where $\mathcal{F}$ is the convergence threshold of the value of loss function $f(x_T)$ in iteration T. The second set of equations represents that switch s_j will aggregate gradients from K_t^j workers in iteration t. Similarly, the third set of equations shows that the PS aggregates gradients from K_t workers in iteration t. The fourth set of inequalities means

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

the sending rate constraint to fully utilize the programmable switch for in-network aggregation. The fifth set of equations denotes the completion time h_t of iteration t. Our optimization goal is to minimize the completion time of all iterations, *i.e.*, $\sum_{t=1}^{T} h_t$.

B. Convergence Analysis

To analyze the feasibility of in-network aggregation with worker selection, we prove the convergence analysis of NAST. Our subsequent analysis is based on the following assumptions.

Assumption 1 (Smoothness)**.** *The loss function f is smooth with a constant $L > 0$, which implies that $\forall x, y$, we have:*

$$f(y) - f(x) \leq \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2 \tag{6}$$

Assumption 2 (Unbiased Local Gradient)**.** *For each worker with a random data sample in t-th iteration, the local gradient is unbiased as follows:*

$$\mathbb{E}_{\xi^i \sim \mathbb{D}^i} \left[\nabla f(x, \xi^i) \right] = \nabla f(x) \tag{7}$$

Assumption 3 (Bounded Variance)**.** *With constants σ and μ, the variance of the stochastic update is bounded by:*

$$\mathbb{E}_{\xi^i \sim \mathbb{D}^i} \left[\left\| \nabla f(x, \xi^i) - \nabla f(x) \right\|^2 \right] \leq \sigma^2 + \mu \|\nabla f(x)\|^2 \tag{8}$$

Assumption 4 (Staleness Bound)**.** *Considering high and rare latency, we use a general staleness bound. With a constant γ, it is achieved by:*

$$\mathbb{E} \left[\left\| \nabla f(x_t) - \nabla f(x_t^i) \right\|^2 \right] \leq \gamma \mathbb{E} \left[\|\nabla f(x_t)\|^2 \right] \tag{9}$$

Lemma 1. *Assuming that τ_t is the conditional probability that $x_t^i = x_t$ (i.e., any worker w_i is selected in iteration t) and a lower bound $\tau \leq \tau_t$ for any iteration t, we obtain:*

$$\mathbb{E} \left[\left\| \nabla f(x_t^i) \right\|^2 \right] \geq \tau \mathbb{E} \left[\|\nabla f(x_t)\|^2 \right] \tag{10}$$

Proof. Detailed proofs are shown in Appendix A. □

Lemma 2. *Under Assumptions 2 and 3, the variance of the sum of gradients from all workers in W_t during iteration t is also bounded by:*

$$\mathbb{E} \left[\left\| \sum_{w_i \in W_t} g_t^i \right\|^2 \right] \leq K_t \sigma^2 + \sum_{w_i \in W_t} (\mu + K_t) \left\| \nabla f(x_t^i) \right\|^2 \tag{11}$$

Proof. Detailed proofs are shown in Appendix B. □

Theorem 1. *Assume that the PS aggregates gradients from workers in W_t during iteration t ,and we adopt a fixed learning rate $\eta \leq \frac{1}{2L(\frac{\mu}{K_m}+1)}$, $K_m = \min_{t \in \{1,\dots,T\}} K_t$, $\alpha = 1 - \gamma + \frac{\tau}{2}$ and specify x_* as the optimal model. Thus, the convergence result of NAST is as follows:*

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E} \left[\|\nabla f(x_t)\|^2 \right] \leq \frac{2(f(x_0) - f(x_*))}{\alpha \eta T} + \frac{L \eta \sigma^2}{\alpha K_m} \tag{12}$$

Proof. Detailed proofs are shown in Appendix C. □

Corollary 1. *Let $\eta = \frac{\sqrt{n}}{\sqrt{T}}$, then the convergence result can be transformed as:*

$$\frac{1}{T} \mathbb{E} \left[\|\nabla f(x_t)\|^2 \right] \leq \frac{2(f(x_0) - f(x_*))}{\alpha \sqrt{nT}} + \frac{L \sigma^2 \sqrt{n}}{\alpha K_m \sqrt{T}} \tag{13}$$

Furthermore, the NAST algorithm exhibits better convergence performance as both K_m and T become larger. However, increasing the values of both K_m and T will increase the training time. Thus, it is a challenge to determine the optimal value of K_m and T to maintain a similar convergence performance with acceptable training time.

IV. NAST Design

A. Problem Approximation

To ensure the convergence, we set the upper bound of the mean square gradient to be less than a certain threshold (ϵ is a positive number), which is equivalent to meeting $f(x_T) \leq \mathcal{F}$. Subsequently, we have:

$$\frac{2(f(x_0) - f(x_*))}{\alpha \sqrt{nT}} + \frac{L \sigma^2 \sqrt{n}}{\alpha K_m \sqrt{T}} \leq \epsilon \tag{14}$$

Then, we rearrange the items and obtain:

$$T \geq \frac{1}{\alpha^2 \epsilon^2} \left(\frac{2(f(x_0) - f(x_*))}{\sqrt{n}} + \frac{L \sigma^2 \sqrt{n}}{K_m} \right)^2 \tag{15}$$

Here, we define:

$$\psi(K_m) = \frac{1}{\alpha^2 \epsilon^2} \left(\frac{2(f(x_0) - f(x_*))}{\sqrt{n}} + \frac{L \sigma^2 \sqrt{n}}{K_m} \right)^2 \tag{16}$$

In order to maintain the similar convergence performance and build the relationship between K_m and T, we set $T = \psi(K_m)$. Therefore, the objective can be reformulated as follows:

$$\min \sum_{t=1}^{\psi(K_m)} \bar{h} \tag{17}$$

where $\bar{h}$ represents the average completion time of finished iterations. With $K_m = \min_{t \in \{1,\dots,T\}} K_t$, the problem of Eq. (5) can be approximated as:

$$\min \sum_{t=1}^{\psi(K_m)} \bar{h}$$

$$S.t. \begin{cases} K_t^j = \sum_{w_i \in (W^j \cap W_t)} 1, & \forall t \in [T], \forall s_j \in S \\ K_t = \sum_{s_j \in S} K_t^j, & \forall t \in [T] \\ \hat{b}_t \leq b_t^i, & \forall w_i \in W_t, \forall t \in [T] \\ K_m \leq K_t, & \forall t \in [T] \\ h_t = \max\{c_t^i + \frac{B}{\hat{b}_t}\}, & \forall w_i \in W_t, \forall t \in [T] \\ \bar{h} = \frac{\sum_{t \in [T]} h_t}{T} & \\ K_t \in \{1, 2, 3, ..., n\}, & \forall t \in [T] \\ K_t^j \in \{1, 2, ..., |W^j|\}, & \forall t \in [T], \forall s_j \in S \end{cases} \tag{18}$$

B. Algorithm Description

A key challenge in algorithm design is determining the values of $\bar{h}$ and K_m in the objective of Eq. (18). To address this challenge, we determine them as outlined in Alg. 2. Basically, the main idea is that according to the information of the system dynamics like computation time and communication time from workers, we introduce the concept of preparation time to assist in estimating the value of $\bar{h}$. Besides, we leverage a prediction process to estimate $\bar{h}$ for each given value of K_{t+1} (*i.e.*, $K_{t+1} = 1, 2, 3, ..., n$) in each iteration. With Eqs. (16) and (17), the optimal value of K_{t+1} is determined when the minimum completion time of the training task is achieved. Finally, we describe our algorithm in the following three steps:

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Optimal Selection of Participating Worker Numbers in Different Iterations

Input: computation time c_t^i, sending rate $\hat{b}_t$
Output: OPT_K_{t+1}

1: **if** Alg. 2 runs for the first time **then**
2: Initialize $p'_i \leftarrow 0, p_i \leftarrow 0, \mathcal{Q}_i \leftarrow \emptyset, \forall w_i \in W$
3: Set $h_t \leftarrow \max\{c_t^i + \frac{B}{\hat{b}_t}\}, \forall w_i \in W_t$.
4: **for** each $w_i \in W$ **do**
5: Update $p'_i \leftarrow p'_i + h_t$
6: **if** $w_i \in W_t$ **then**
7: Add p'_i to Q_i and update $p_i \leftarrow \frac{\sum_{z \in \mathcal{Q}_i} |z|}{|\mathcal{Q}_i|}$
8: Update $p'_i \leftarrow 0$
9: $H_{min} \leftarrow +\infty$
10: **for** each $K_{t+1} \in \{1, 2, 3, ..., n\}$ **do**
11: **for** each $w_i \in W$ **do**
12: Set $a_i \leftarrow p_i$
13: **for** each $t \in \{1, 2, 3, .., T^*\}$ **do**
14: Set $h_t \leftarrow a_{(K_{t+1})}$
15: Set $\bar{h} \leftarrow \frac{\sum_{0 \le t' < t} h_{t'}}{t}$
16: **for** each $w_i \in W$ **do**
17: **if** $a_i \le h_t$ **then**
18: Update $a_i \leftarrow p_i$
19: **else**
20: Update $a_i \leftarrow a_i - h_t$
21: Set $T \leftarrow \psi(K_{t+1})$
22: Set $H \leftarrow T\bar{h}$
23: **if** $H < H_{min}$ **then**
24: Update $H_{min} \leftarrow H$
25: Set $OPT_K_{t+1} \leftarrow K_{t+1}$
26: **Return** OPT_K_{t+1}

1) Collect Information, 2) Measure Preparation Time, and 3) Select Optimal K_{t+1}.

1) Collect Information: In essence, dynamic factors like computation time and communication time can both affect the completion time of one iteration. To measure the dynamics, we leverage the records of historical values to predict computation time and communication time in following iterations. Each worker records the time spent training until gradient transmission is ready as computation time. Note that, due to the memory constraints in programmable switches, only a limited number of aggregated gradient fragments (a part of a complete gradient) will be temporarily stored. If there are significant time intervals between the arrival of the same gradient fragments from different workers at the switch, the in-network aggregation may encounter failures. Consequently, we utilize the minimum available bandwidth $\hat{b}_t$ among the fastest K_t workers as the cost of bandwidth resources in iteration t (Line 3), which indicates that the communication time can be achieved by $\frac{B}{\hat{b}_t}$.

2) Measure Preparation Time: The mean time interval from when the PS distributes its updated model to worker w until it receives gradients from w is referred to as the gradient preparation time. It encompasses the duration of model distribution, local training, and gradient transmission. By gathering information from workers, the PS can calculate the gradient preparation time $\mathcal{P} = \{p_1, p_2, ..., p_n\}$. Since the system environment is dynamic, the values of elements within $\mathcal{P}$ vary as each worker participates in different iterations. To determine the number of workers participating in the subsequent iteration, we log the historical gradient preparation time for each worker and take the average time, which serves as a reference for selecting the participating workers in the next iteration.

To record the time elapsed since a worker's last participation in global aggregation, the PS utilizes the variable p'_i. The value of p'_i increases over time if worker w_i still keeps training without pushing gradients (Line 5). When worker w_i takes part in global aggregation, we record the current p'_i and insert it in a queue Q_i. By computing the average value of all elements in queue Q_i, the PS estimates the gradient preparation time for worker w_i (Lines 6-7). Next, we need to recalculate the value of p'_i and reset it to zero (Line 8). Leveraging the collection of all workers' gradient preparation times $\mathcal{P}$, the PS starts to estimate the optimal number of workers to be involved in the subsequent iteration.

3) Select Optimal K_{t+1}: The vector $\mathcal{P}$ and the value of $\bar{h}$ can determine the value of OPT_K_{t+1} for the next iteration, where we need to estimate the $\bar{h}$. However, it is difficult to establish a mathematical connection between these variables. Hence, a prediction method is proposed to estimate the value of $\bar{h}$ first (Lines 11-20). Using a strategy similar to a greedy approach, the prediction method simulates the arrival of gradients and global aggregation by considering different gradient preparation time $p_i \in \mathcal{P}$ for each $K_{t+1} = \{1, 2, ..., n\}$. The prediction method simulates T^* iterations to achieve a stable $\bar{h}$. In practice, we set T^* as the number of total workers to ensure fast execution of our prediction method.

We use arrival time a_i to represent the time interval from the current moment until the PS receives the gradients from worker w_i in the next iteration. Given that the PS has collected the information about each worker's gradient preparation time $\mathcal{P}$, we use $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ to denote the arrival time that the PS will receive the gradient from each worker. Consequently, we construct a set $\mathcal{A}$ and sort the elements in $\mathcal{A}$ as $a_{(1)} \le a_{(2)} \le ... \le a_{(n)}$, representing their arrival time in ascending order. Similarly, we sort members of $\mathcal{P}$ as $p_{(1)} \le p_{(2)} \le ... \le p_{(n)}$.

Arrival time a_i for each worker w_i is initialized as its gradient preparation time p_i for each value of K_{t+1} (Line 12). Since the PS aggregates gradients from K_{t+1} workers, we set the time that the gradient from the K_{t+1}-th fastest worker as the completion time of iteration t (Line 14). Obviously, the fastest K_{t+1} workers will all reach the PS during this iteration, so we set arrival time a_i as p_i for these workers. Otherwise, we reduce a_i by h_t, which means the gradient from worker w_i is expected to take additional time (*i.e.*, $a_i - h_t$) to reach the PS (Lines 16-20). Once the average completion time $\bar{h}$ of one iteration is estimated, we compute the value of iterations T with selected K_{t+1}. Finally, we determine the optimal value

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

of OPT_K_{t+1} by achieving the minimum value of training time $H = T\bar{h}$ (Lines 21-25).

Remark 1. *As shown in **Exp#3** of Section V-C, the execution of Alg. 2 takes a short time and has negligible impact on DT.*

C. Policy Update for Programmable Switches

Although the PS should aggregate gradients from the fastest K_t workers by Alg. 2, each switch is unaware of the specific number of workers from which it needs to aggregate gradients. Without additional design, each switch only perceives the arrival sequence of gradients from workers directly connected to it. In order to ensure that switches aggregate gradients of the globally fastest K_t workers, we introduce a request-response mechanism in Alg. 1. Since each worker inserts the ID numbers of directly connected switches in the request, PS will calculate the value of K_t^j for each switch j while responding. Finally, after responding to K_t requests, PS will send the value of K_t^j to each switch j and update the policy accordingly in iteration t.

V. Performance Evaluation

A. Experimental Setup

Benchmarks: NAST is compared to three INA solutions: ATP [9], SwitchML [4], and INA-CWS. Specifically, ATP performs dynamic and best-effort gradient aggregation at multiple racks of programmable switches. As a result, if the network environment exhibits high dynamics, some gradients from workers will fail to be aggregated by the programmable switch. Since ATP utilizes the PS for global aggregation, it allows gradients to be directly aggregated by the PS. The second one is an in-network aggregation service called SwitchML, which focuses primarily on a rack-scale architecture, *i.e.*, using a single programmable switch to aggregate gradients from all workers. For a fair evaluation, we additionally enhance the training of SwitchML by transmitting the aggregated gradients from several programmable switches to the PS for supporting multiple racks. The third one is INA with cluster-based worker selection, where each switch is treated as a cluster header and a cluster comprises a switch and workers directly connected to it. The workflow of INA-CWS is as follows: each cluster header aggregates gradients from workers in the same cluster, and the PS updates the model on receiving aggregated gradients from one cluster header. Then the PS distributes the updated model back to the corresponding workers.

Performance Metrics: We primarily adopt the following four metrics to evaluate the performance of NAST: (1) the test accuracy; (2) the training loss; (3) the total training time; and (4) the traffic consumption.

During the evaluation, we compute the ratio between the number of the samples correctly classified by the model and the number of all samples in the test set as *the test accuracy*. *The training loss* is the quantification difference between the predicted output of the model and the actual target output during the training phase. The time duration from the PS distributing the initial model to all workers until the PS finishing the final model update is referred to as *the total training time*. Finally, the total traffic of gradients sent by workers and gradients received by the PS during the training task is accumulated as the *traffic consumption*.

B. Testbed Settings

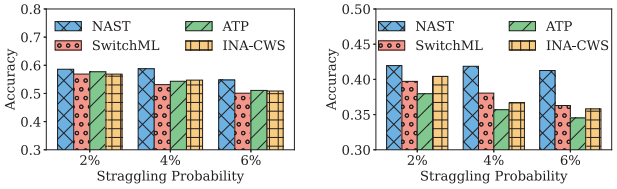
Platform Setup: We implement a testbed with a similar size, where there are 8 workers, 1 PS, and 2 Wedge100BF-32x programmable switches [14]. Specifically, one switch is connected to 4 workers and the PS, and the other switch is linked to the remaining workers. Additionally, both switches are connected to each other. Each worker is equipped with one NVIDIA GeForce RTX 3080Ti GPU with NVIDIA driver version 515.48.07 and CUDA 11.7.

Workloads: We choose two CNN models: ResNet50 with the size of 97MB [2] and VGG19 with the size of 548MB [15] as benchmark models on the dataset Cifar-100 [16]. Similar to [17], two models are trained with PyTorch, and the batch sizes of ResNet50 and VGG19 are both 64 per GPU. Each training task performs 200 epochs, and each epoch includes 98 iterations when training with 8 workers. That is, the iteration is the number of batches needed to complete one epoch, and an epoch refers to one cycle through the full training dataset. Specifically, in NAST and INA-CWS, we define one iteration as the model update on the PS.

Implementation: We implement NAST on workers and the PS by using two threads: a training thread and a logging thread. In training threads of workers, local training is performed by PyTorch. The training thread on the PS performs the model update and distribution. The logging threads on the workers are responsible for sending requests and computation time, while the logging thread on the PS responds and collects information from workers for running Alg. 2. After running the algorithm, the PS publishes the value of K_t^j to each programmable switch s_j. Specifically, the PS connects switches with the Secure Shell (SSH) protocol and leverages the Barefoot Runtime Interface (BRI) to publish the value of K_t^j. To perform in-network aggregation in programmable switches, we write the P4 program with 748 lines of code. Each programmable switch can aggregate 64 gradients per packet with one switch pipeline like [4]. After local training on each worker, we partition the complete gradient into a set of gradient fragments, with each fragment containing 64 gradients. We utilize the self-defined packet header to encode the gradient fragments in order to filter gradient packets corresponding to the NAST traffic.

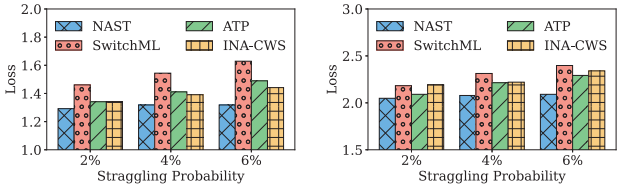
Straggler Generation: Following recent works [12], we simulate stragglers by executing the *sleep* command to estimate the impact of straggling workers on training performance. Specifically, during the training task execution of workers, we select three delay points to insert the *sleep* command. At each delay point, the worker decides to sleep with a straggling probability of ρ. If a worker is determined to straggle at any of the delay points, the training thread is slowed down by $0.5\times$ to $1.5\times$ the ideal completion time of one iteration, where the

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.



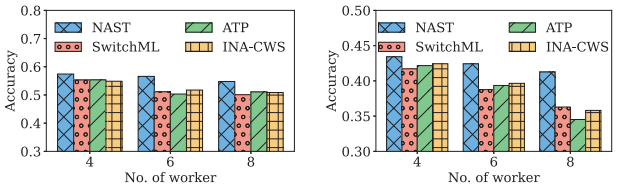
(a) ResNet50 (b) VGG19

Fig. 2: Impact of Straggling Workers on Accuracy



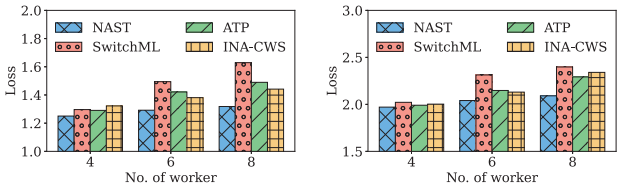
(a) ResNet50 (b) VGG19

Fig. 3: Impact of Straggling Workers on Loss



(a) ResNet50 (b) VGG19

Fig. 4: Accuracy under Different No. of Workers



(a) ResNet50 (b) VGG19

Fig. 5: Loss under Different No. of Workers

ideal completion time refers to the time taken for one iteration in the absence of any straggler problem.

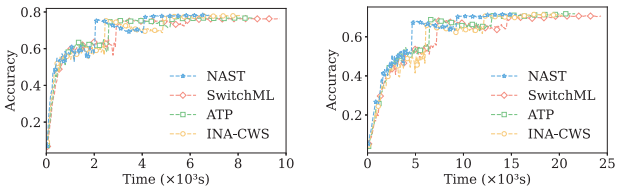
C. Testbed Results

(Exp#1) Impact of the straggling workers: To evaluate the impact of different straggling probabilities on training performance, we vary the probabilities and measure the test accuracy and training loss under different solutions within the same training period. We use 8 workers to perform the DT task. Figs. 2 and 3 illustrate that as the probability of straggling increases, NAST consistently achieves the best training performance and adapts well to system dynamics. Meanwhile, SwitchML, ATP, and INA-CWS experience a decrease in accuracy and an increase in training loss. For example, after 500s of training VGG19, the accuracy of SwitchML, ATP, INA-CWS, and NAST is 50.1%, 51.1%, 50.8%, and 54.8%, respectively, when the straggling probability is 6%. In terms of training loss, under the probability $\rho = 6\%$, NAST decreases the loss by 12.8%, 8.8%, and 10.7% compared to SwitchML, ATP, and INA-CWS in training VGG19. From these observations, it can be concluded that NAST suffers the least impact on performance under higher straggling probability. This is because our algorithm dynamically selects the number of participating workers based on the dynamic nature of the network, aiming to accelerate DT. Note that, we set the straggling probability as 6% by default for the following experiments.

(Exp#2) Comparison on the number of workers: We vary the number of workers to estimate the performance of four solutions. Specifically, we measure the test accuracy and training loss of different solutions under the same training time and straggling probability. As shown in Figs. 4 and 5, NAST outperforms other solutions in achieving better test accuracy and smaller training loss across different numbers of workers. For example, after 500s of training ResNet50 with 8 workers, the accuracy of SwitchML, ATP, INA-CWS, and NAST is 50.1%, 51.1%, 50.9%, and 54.8%, respectively. Thus, NAST increases the test accuracy by 9.4%, 7.2%, and 7.7% compared to SwitchML, ATP, and INA-CWS. As for the training loss, NAST still achieves a smaller loss compared to other solutions in training ResNet50 and VGG19. This is because as the number of workers increases, workers in SwitchML straggles at more different phases of training, resulting in a longer training time. Similarly, each cluster in INA-CWS experiences a longer iteration time for the same reason. ATP provides a best-effort approach that adapts to the dynamic network. However, this approach leads to more gradients that are aggregated on the PS, and the communication pressure of the PS becomes higher with an increasing number of workers. NAST aggregates as many gradients as possible on programmable switches and reduces the communication pressure on the PS to achieve a fast training speed.

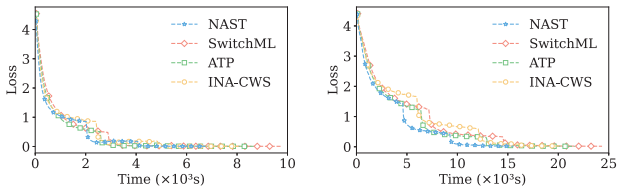
(Exp#3) Overall training performance: We measure the overall performance of the DT task by evaluating the test accuracy and training loss with the training time under different solutions. We utilize 8 workers to perform the DT task. Figs. 6 and 7 show that NAST outperforms other solutions by achieving faster convergence. For instance, when training ResNet50, the time of SwitchML, ATP, INA-CWS, and NAST to achieve above 70% accuracy is 2927.2s, 2611.3s, 2453.4s, and 2053.9s, respectively. It means that NAST can reduce the training time by 29.8%, 21.3%, and, 16.3%, respectively, compared to SwitchML, ATP, and, INA-CWS to achieve similar test accuracy. This can be attributed to the fact that SwitchML and ATP require synchronization with the slowest worker in each iteration. INA-CWS restricts the influence of the slowest worker by clustering, but the slowest worker still affects the workers in the same cluster, thus prolonging the training time. NAST aggregates gradients from fast workers according to Alg. 2 and guides stragglers to contribute gradients in one subsequent iteration, thereby accelerating training. Note that,

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.



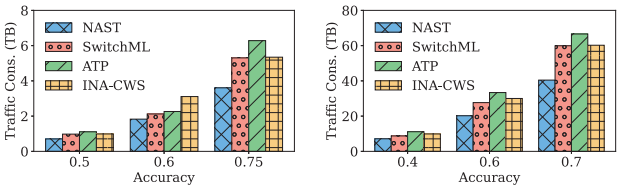
(a) ResNet50 (b) VGG19

Fig. 6: Accuracy vs. Time



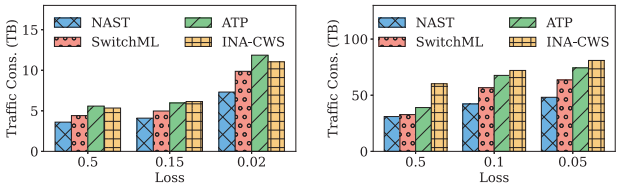
(a) ResNet50 (b) VGG19

Fig. 7: Loss vs. Time



(a) ResNet50 (b) VGG19

Fig. 8: Traffic Consumption when Achieving the Target Accuracy



(a) ResNet50 (b) VGG19

Fig. 9: Traffic Consumption when Obtaining the Same Loss

the execution time of Alg. 2 is around 57ms which is less than the completion time of one iteration. Overlapping with communication time in the previous iteration and computation time in the current iteration, the algorithm's execution time has a negligible impact on training.

(Exp#4) Comparison on traffic consumption: We evaluate the traffic consumption of four solutions by measuring the amount of data sent by workers and received by the PS. As shown in Figs. 8 and 9, NAST consumes less traffic to achieve the same test accuracy and training loss than other solutions. Under the same traffic consumption, NAST achieves the highest test accuracy and the lowest training loss. This is because SwitchML and ATP aggregate gradients from all workers in each iteration, while ATP further increases the data received by the PS due to its best-effort delivery design. INA-CWS performs each iteration with only half of the workers, leading to slow convergence. In contrast, NAST reduces the gradients sent by workers in each iteration while utilizing the programmable switch to reduce the communication pressure on the PS. As a result, NAST achieves the same training performance with lower traffic consumption.

From the above testbed results, we can draw some conclusions. Figs. 2-3 illustrate that NAST has the least impact on training performance under different straggling probabilities compared to other solutions. Second, as shown in Figs. 4-5, with the increasing number of workers, NAST consistently performs better than other solutions. Third, from Figs. 6-7, we will see that NAST is able to reduce the training time by 22.4% on average. At last, Figs. 8-9 show that NAST can reduce traffic consumption by about 36.8% on average to achieve the same test accuracy and training loss compared to other solutions.

VI. Conclusion

In this paper, we propose a novel mechanism called NAST, which combines in-network aggregation with worker selection to accelerate distributed training. We theoretically prove the convergence of NAST and provide an efficient algorithm to dynamically select an optimal number of workers that participate in in-network aggregation during different training iterations. Extensive testbed-based experimental results show that NAST is able to accelerate the distributed training process and reduce traffic consumption.

Acknowledgement

The corresponding author of this paper is Gongming Zhao. This work is supported by the National Natural Science Foundation of China (NSFC) under Grant 62372426, the Youth Innovation Promotion Association of the Chinese Academy of Science under Grant 2023481, Basic Research Program of Jiangsu under Grant BK20250124, the Fundamental Research Funds for the Central Universities under Grant WK2150250043, and the Open Project Program of the Guangxi Key Laboratory of Digital Infrastructure (Grant No. GXDIOP2024003).

Appendix

A. *Proof of Lemma 1*

Due to system dynamics, workers have different computation times. Thus, worker w_i is selected with conditional probability τ_t in iteration t. If the computation time follows an exponential distribution, we have $\tau_t = \frac{1}{n}$ [18]. For generality, we use a lower bound $\tau \leq \tau_t$ and obtain:

$$\mathbb{E}\left[\left\|\nabla f(x_t^i)\right\|^2\right] \geq \tau \mathbb{E}\left[\left\|\nabla f(x_t)\right\|^2\right]$$

which completes the proof.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

B. Proof of Lemma 2

For any $w_i, w_{i'} \in W_t$ and $w_i \neq w_{i'}$, we have:

$$\mathbb{E}\left[\left\langle g_t^i - \nabla f(x_t^i), g_t^{i'} - \nabla f(x_t^{i'})\right\rangle\right] = 0. \tag{19}$$

Considering Eq. (19) as a cross term, we obtain:

$$\mathbb{E}\left[\left\|\sum_{w_i \in W_t}(g_t^i - \nabla f(x_t^i))\right\|^2\right] \leq \sum_{w_i \in W_t}(\sigma^2 + \mu\left\|\nabla f(x_t^i)\right\|^2). \tag{20}$$

With Eqs. (19) and (20), Lemma 2 follows:

$$\mathbb{E}\left[\left\|\sum_{w_i \in W_t} g_t^i\right\|^2\right] \leq K_t\sigma^2 + \sum_{w_i \in W_t}\mu\left\|\nabla f(x_t^i)\right\|^2 + \sum_{w_i \in W_t} K_t\left\|\nabla f(x_t^i)\right\|^2. \tag{21}$$

C. Proof of Theorem 1

By Assumption 1, we have:

$$\begin{aligned} & f(x_{t+1}) - f(x_t) \\ \overset{④}{=} & -\frac{\eta}{2K_t}\sum_{w_i \in W_t}(\|\nabla f(x_t)\|^2 + \left\|g_t^i\right\|^2 - \left\|g_t^i - \nabla f(x_t)\right\|^2) \\ & + \frac{L\eta^2}{2}\left\|\bar{g}_t\right\|^2, \end{aligned} \tag{22}$$

where the first term is expanded by $2\langle a, b\rangle = \|a\|^2 + \|b\|^2 - \|a-b\|^2$ in ④. Next, we consider:

$$\begin{aligned} & \mathbb{E}\left[\left\|\nabla f(x_t) - g_t^i\right\|^2\right] \\ & = \mathbb{E}\left[\left\|\nabla f(x_t) - \nabla f(x_t^i)\right\|^2\right] - \mathbb{E}\left[\left\|\nabla f(x_t^i)\right\|^2\right] + \mathbb{E}\left[\left\|g_t^i\right\|^2\right]. \end{aligned} \tag{23}$$

Taking the expectation in Eq. (22), we obtain:

$$\begin{aligned} & \mathbb{E}\left[f(x_{t+1})\right] \\ & \leq \mathbb{E}\left[f(x_t)\right] + \frac{\eta(\gamma-1)}{2}\mathbb{E}\left[\|\nabla f(x_t)\|^2\right] + \frac{L\eta^2\sigma^2}{2K_t} \\ & \quad + \frac{\eta}{2K_t}\sum_{w_i \in W_t}(\frac{L\eta}{K_t}(\mu + K_t) - 1)\mathbb{E}\left[\left\|\nabla f(x_t^i)\right\|^2\right]. \end{aligned} \tag{24}$$

For the last term in Eq. (24), we choose

$$\eta \leq \frac{1}{2L(\frac{\mu}{K_m}+1)} \leq \frac{1}{2L(\frac{\mu}{K_t}+1)}, \tag{25}$$

where $K_m = \min_{t\in\{1,\ldots,T\}} K_t$. Thus, $\frac{L\eta}{K_t}(\mu + K_t) \leq \frac{1}{2}$, and the last term is bounded by:

$$\frac{\eta\sum_{w_i \in W_t}(\frac{L\eta}{K_t}(\mu+K_t)-1)\mathbb{E}\left[\left\|\nabla f(x_t^i)\right\|^2\right]}{2K_t} \leq \frac{-\eta\tau\mathbb{E}\left[\|\nabla f(x_t)\|^2\right]}{4}.$$

The last line follows from Lemma 1. Then, Eq. (24) becomes:

$$\mathbb{E}\left[f(x_{t+1})\right] \leq \mathbb{E}\left[f(x_t)\right] - \frac{\alpha\eta}{2}\mathbb{E}\left[\|\nabla f(x_t)\|^2\right] + \frac{L\eta^2\sigma^2}{2K_m}.$$

Summing over $t = 0, ..., T-1$, we obtain the convergence result for NAST:

$$\begin{aligned} & \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\nabla\left\|f(x_t)\right\|^2\right] \\ & \leq \frac{2(f(x_0) - f(x_*))}{\alpha\eta T} + \frac{L\eta\sigma^2}{\alpha K_m}, \end{aligned} \tag{26}$$

which completes the proof.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[3] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 191–198.

[4] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with {In-Network} aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[5] L. Hui, W. Yang, F. Wu, Y. Wang, F. Lyu, and Y. Zhang, "Directreduce: A scalable ring allreduce offloading architecture for torus topologies," *IEEE Internet of Things Journal*, 2025.

[6] S. Fan and Z. Zhang, "Pier: Efficient large language model pretraining with relaxed global communication," *arXiv preprint arXiv:2511.17849*, 2025.

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.

[8] Z. Cheng and Z. Xu, "Bandwidth reduction using importance weighted pruning on ring allreduce," *arXiv preprint arXiv:1901.01544*, 2019.

[9] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "{ATP}: In-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 741–761.

[10] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Redundancy techniques for straggler mitigation in distributed optimization and learning," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 2619–2665, 2019.

[11] Q. Zhou, S. Guo, H. Lu, L. Li, M. Guo, Y. Sun, and K. Wang, "A comprehensive inspection of the straggler problem," *Computer*, vol. 54, no. 10, pp. 4–5, 2021.

[12] M. Yang, A. Baban, V. Kugel, J. Libby, S. Mackie, S. S. R. Kananda, C.-H. Wu, and M. Ghobadi, "Using trio: juniper networks' programmable chipset-for emerging in-network applications," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 633–648.

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

[14] Intel tofino. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] Cifar-100 (canadian institute for advanced research). [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[17] Byteps performance when training cnn. [Online]. Available: https://github.com/bytedance/byteps/blob/master/docs/performance.md

[18] I. Mitliagkas, C. Zhang, S. Hadjis, and C. Ré, "Asynchrony begets momentum, with an application to deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2016, pp. 997–1004.

Authorized licensed use limited to: NANJING UNIVERSITY OF SCIENCE AND TECHNOLOGY. Downloaded on November 05,2026 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.