InGo: In-Network Aggregation Routing with Batch Size Adjustment for Distributed Training

Jianfeng Bao[1,2] Gongming Zhao[1,2] Hongli Xu[1,2] Haibo Wang[3] Peng Yang[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China
[2]Suzhou Institute for Advanced Research, University of Science and Technology of China
[3]University of Kentucky, USA

***Abstract*—Distributed training has emerged as a critical application in clusters due to the widespread adoption of AI technology across various domains. However, as distributed training continues to advance, it has become increasingly time-consuming. To address this challenge, researchers have explored leveraging In-Network Aggregation (INA) to expedite distributed model training. Specifically, by harnessing programmable hardware, such as Intel Tofino switches, INA can aggregate gradients within the network, thereby reducing the amount of gradient transmission and accelerating distributed training. However, previous works assume fixed routing selection and batch size, ignoring their impact on model convergence and resulting in extended completion time. To bridge this gap, we propose InGo, a pioneering approach that considers both in-network aggregation routing and batch size adjustment, and provide the rigorous convergence analysis. Then, we formally define the problem of in-network aggregation routing with batch size adjustment, and present an efficient algorithm with bounded approximation factors to solve this problem. Through extensive experiments on both physical platforms and simulated environments, we demonstrate that InGo significantly reduces the completion time by 25.2%-74.7% compared to state-of-the-art solutions.**

Index Terms*—*In-Network Aggregation, Batch Size Adjustment, Distributed Model Training, Programmable Switch

I. INTRODUCTION

Distributed training has made significant advancements in various domains, such as computer vision [1], recommendation systems [2], and natural language processing [3]. The widely used Parameter Server (PS) architecture allows workers to handle training samples while the PS manages global model updates. However, with the advancements in distributed training, the process has become increasingly time-consuming for training models. For instance, it takes 3.5 days to train a WMT 2014 English-to-French translation task on 8 NVIDIA P100 GPUs [4]. Thus, accelerating distributed training has emerged as a prominent research focus.

Driven by the rapid advancement of programmable networking, in-network aggregation (INA) [5] holds promise for accelerating distributed training. Specifically, INA utilizes programmable devices, such as smart network interface cards (NICs [6]) and programmable switches [7], to efficiently aggregate gradients from multiple workers within the network. The aggregated result is then sent to the PS. By doing so, INA effectively reduces communication overhead between workers and the PS, thereby reducing communication time and accelerating distributed training [5, 8].

Nevertheless, current INA solutions primarily concentrate on achieving efficient aggregation operations within the network while neglecting the significance of routing selection. For instance, ATP [5] designs a protocol to aggregate gradients on default top-of-rack switches, which may cause traffic imbalance on links. Though a few studies, such as GRID [8], consider the selection of aggregation nodes, they fail to contemplate the impact of link bandwidth capacity on communication time, potentially leading to excessive load on the link and consequently increasing communication time. Hence, it is crucial to devise an in-network aggregation routing that mitigates traffic imbalance and minimizes communication time.

Furthermore, existing INA solutions primarily focus on reducing communication time but often neglect optimizing the worker-side computation strategy. As a result, the computation/communication time ratio becomes unfavorable, hindering model convergence. In fact, there is a close coupling relationship between computation and communication [9]. Intuitively, if the proportion of communication time is too large, it means that the computation resources of workers are not fully utilized. Conversely, if the proportion of computation time is too large, it may mean that the global update is not frequent enough. Therefore, it is very important to maintain a reasonable computation/communication time ratio when using INA to accelerate distributed training.

In this paper, considering the linear relationship between batch size and the computation time [10], we aim to leverage INA routing to minimize communication time and adjust the batch size for achieving a reasonable computation/communication time ratio, thus accelerating distribute training. However, designing such a solution presents significant challenges. Firstly, INA routing involves numerous switches and thousands of feasible paths in a cluster, requiring decision-making on various variables while considering resource constraints such as bandwidth and switch resources, as well as the NIC processing capacity of the PS. Secondly, adjusting the batch size poses additional challenges as it impacts computation time, communication frequency, and gradient variance as illustrated in [11]. Larger batches stabilize gradient variance, aiding model convergence [12], but increase computation time. Conversely, smaller batches reduce computation time but result in less precise gradient information and increased communication frequency. Therefore, achiev-

ing a balance among these factors, including communication time, computation time, and gradient variance, is of utmost importance. To address these challenges, we propose InGo and highlight its main contributions as follows:

- We design InGo, the first-of-its-kind work on in-network aggregation routing with batch size adjustment for accelerating distributed training. Furthermore, we provide a theoretical analysis of the impact of INA routing and batch size on the model convergence bound.
- For accelerating distributed training, we design an algorithm based on randomized rounding which provides a method for INA routing with batch size adjustment. In addition, we analyze that the algorithm can achieve an approximate factor of $O(2\ln|S|+1)$, where S represents a set of switches in the network.
- We implement InGo on both hardware testbed and software simulation. Experimental results demonstrate that InGo can reduce the completion time by 25.2%-74.7% compared to state-of-the-art solutions.

The rest of this paper is organized as follows. Section II introduces the system model and workflow. Section III gives the convergence analysis. We give the problem definition in Section IV. The algorithm design and performance analysis is presented in Section V. The results of the testbed and large-scale simulations are presented in Section VI. We conclude this paper in Section VII.

II. SYSTEM MODEL AND WORKFLOW

A. System Model

A typical parameter server architecture mainly consists of two components: a parameter server (PS) d and a set of workers $W = \{w_1, w_2, ...w_{|W|}\}$. The network comprises programmable switches represented by $S = \{s_1, s_2, ...s_{|S|}\}$ and links represented by $E = \{e_1, e_2, ...e_{|E|}\}$. Since both programmable switches and the PS have aggregation capabilities, we define the set of aggregation nodes as $A = S \cup \{d\}$. Therefore, we model the distributed training cluster as $G = (W, A, E)$. Our goal is to accelerate distributed training by designing an INA routing with batch size adjustment in a given time budget T^B.

B. Workflow

Before engaging in distributed training, it is crucial for the gradient packet to possess a dedicated packet header that instructs the switch to execute aggregation operations effectively. We call this header INA header, which mainly consists of $switch\_id$ and reg_id fields. Specially, $switch\_id$ is utilized to distinguish different switches, and reg_id indicates which register of the switch the gradient should be aggregated in. In addition, the INA header is encapsulated in the IP packet. In order to indicate whether the gradient in the packet has been aggregated, we distinguish it through using a pre-defined value in IP header protocol field.

We present the workflow of InGo in Alg. 1. On the worker side, the first step involves obtaininga batch with size b and then calculating the gradient (Line 3), . Then the gradient is converted to an integer by multiplying by a scaling factor (*e.g.* 10^8) like previous work [5]. After that, the gradient is flattened and divided into multiple fragments. Then these gradient fragments are encapsulated into the packet and sent to the PS (Lines 4-5). At last, after receiving the updated parameters sent by the PS, the worker updates its time budget T^B and starts the next iteration of training (Lines 6-8).

For the switch side, the switch mainly maintains two sets of registers: one set for recording the number of workers whose gradients have been aggregated (referred to as w_cnt), and another set for gradient aggregation (referred to as reg_val). In addition, we use max_worker on switch to indicate the sum of $worker\_id$ of workers whose gradient should be aggregated at the switch. When the switch receives a packet p, it checks the IP header and the INA header to decide whether the gradient need to be aggregated (Line 11). Then the switch updates w_cnt and reg_val (Lines 12-19) according to the INA header. At last, the switch judges whether the aggregation is completed according to w_cnt and only forwards the aggregated gradient packets (Lines 21-24).

For the PS side, the PS updates the model and sends the parameters back to all workers (Line 27).

Algorithm 1 Workflow of InGo

1: **Worker logic**
2: **while** $T^B > 0$ **do**
3: Fetch a batch of size b and calculate the gradient g
4: Convert floating-point g to an integer
5: Send the gradients to the PS
6: Receive the parameters sent back from the PS
7: Record the completion time of one iteration as t
8: $T^B = T^B - t$
9: **Switch logic**
10: upon receive $p(worker\_id, switch\_id, reg\_id, gradient)$
11: **if** p *need to be aggregated* **then**
12: **if** $w\_cnt! = max\_worker$ **then**
13: $w\_cnt = w\_cnt + worker\_id$
14: **else**
15: $w\_cnt = worker\_id$
16: **if** $w\_cnt == worker\_id$ **then**
17: $reg\_val = gradient$
18: **else**
19: $reg\_val = reg\_val + gradient$
20: $gradient = reg\_val$
21: **if** $w\_cnt == max\_worker$ **then**
22: Modify the packet header to represent that the aggregation has been completed
23: **else**
24: drop()
25: Forward the packet to the next hop
26: **PS logic**
27: Perform global update and broadcast the parameters

However, designing such a distributed training scheme is very challenging. Firstly, there are multiple resource con-

straints when deciding the INA routing, such as the bandwidth of links, the capacity of switches and the capacity of the PS. Additionally, the batch size not only affects computation time and communication frequency but also impacts the gradient variance, which can influence model convergence. Therefore, before devising an effective solution, it is essential to analyze the impact of both INA routing and batch size on the model convergence.

III. Convergence Analysis

A. Convergence Analysis

In this section, we analyze the impact of batch size and INA routing on the convergence in detail. In this paper, we use the mini-batch stochastic gradient descent (SGD) algorithm, which is widely used to solve distributed training problems [13]. Specifically, we focus on a synchronize scenario where all workers need to train a batch of samples and send gradients to the PS for global update in one iteration. In each iteration, SGD performs the following operations:

$$w(r+1) = w(r) - \alpha g(w(r), \mathcal{B}) \tag{1}$$

$$g(w(r), \mathcal{B}) = \frac{1}{b} \sum_{i=1}^{b} \nabla l(w(r), x_i) \tag{2}$$

where $w(r)$ is used to denote the parameters at the r-th iteration. $\mathcal{B}$ and b denote the mini-batch and its size. α denotes the learning rate, and $g(w(r), \mathcal{B})$ represents the gradient of $\mathcal{B}$ at the r-th iteration.

In addition, we use L to denote the loss function of the distributed training task, where L indicates the gap between the predicted value and the true value in the training task, and generally a smaller value indicates a better performance of the model [14]. Our objective is to minimize the loss function after the final iteration of training:

$$L(w(R)) = \frac{1}{n} \sum_{i=1}^{n} l(w(R), x_i) \tag{3}$$

where n refers to the total number of samples in the dataset, x represents one of the samples. In addition, $l(w(R), x_i)$ is used to denote the loss function of a single sample x_i at the R-th iteration and the loss function of the R-th iteration is calculated from the mean of the loss function of all samples as indicated in Eq. (3).

For the sake of analysis, our subsequent analysis is based on several common assumptions for distributed model training as illustrated in [14], which are listed as follows:

Assumption 1. *We assume that the loss function is smooth, i.e., there exists a constant ρ, such that:*

$$L(u) - L(v) \leq \nabla L(v)^T (u - v) + \frac{\rho}{2} \|(u - v)\|^2 \tag{4}$$

for any u and v.

Assumption 2. *We assume that the loss function is convex, i.e., there exists a constant μ, such that:*

$$L(u) - L(v) \geq \nabla L(v)^T (u - v) + \frac{\mu}{2} \|(u - v)\|^2 \tag{5}$$

for any u and v.

Assumption 3. *The mean of the gradient is defined as follow:*

$$\mathbb{E}(g(w)) = \frac{1}{n} \sum_{i=1}^{n} \nabla l(w(r), x_i) \tag{6}$$

Assumption 4. *The variance of the gradient is defined as follow:*

$$Var(g(w)) = \frac{1}{n} \sum_{i=1}^{n} (\nabla l(w, x_i) - \nabla L(w))(\nabla l(w, x_i) - \nabla L(w))^T \tag{7}$$

Assumption 5. *Let $\nabla L(w)$ and $\mathcal{V}(w)$ represent the mean and the variance of gradient distribution. The sample gradient follows a Gaussian distribution:*

$$\nabla l(w, x) \sim \mathcal{N}(\nabla L(w), \mathcal{V}(w)) \tag{8}$$

Assumption 6. *There is an optimal model parameter w^* that satisfies:*

$$w^* \triangleq arg \min_{w} L(w) \tag{9}$$

In order to facilitate the proof of Theorem 1, we present two import corollary as follows:

Corollary 1. *The gradient of each training mini-batch follows the following distribution:*

$$g(w, \mathcal{B}) \sim \mathcal{N}(\nabla L(w), \frac{\mathcal{V}(w)}{b}) \tag{10}$$

where b denotes the size of mini-batch $\mathcal{B}$.

Proof. The gradient of an arbitrary sample follows a Gaussian distribution $\mathcal{N}(\nabla L(w), \mathcal{V}(w))$ according to Assumption 5, besides this, the gradients of different samples are independent of each other.

According to Eq. (2) and Assumption 3, the mean of the gradient of a mini-batch can be calculated as follow:

$$\mathbb{E}(g(w, \mathcal{B})) = \frac{1}{b} \sum_{i=1}^{b} \nabla L(w) = \nabla L(w) \tag{11}$$

Similarly, according to Assumption 4, the variance of the gradient of a mini-batch can be calculated as follow:

$$Var((g(w, \mathcal{B})) = \frac{1}{b^2} \sum_{i=1}^{b} \mathcal{V}(w) = \frac{\mathcal{V}(w)}{b} \tag{12}$$

Therefore, $g(w, \mathcal{B})$ follows a Gaussian distribution with mean $\nabla L(w)$ and variance $\frac{\mathcal{V}(w)}{b}$. □

Corollary 2. *In the case where batch size is much smaller than the total number of dataset, the decrease in the loss function per iteration is approximately positively correlated with the square root of the batch size:*

$$L(w(r)) - L(w(r+1)) = \sqrt{b} \cdot \xi \tag{13}$$

where ξ is a constant that depends on different models.

Proof. We use $w(r)$ and $w(r+1)$ to represent the model paramaters before the training of the r-th and $(r+1)$-th iteration. According to Eq. (5), we have:

$$L(w(r+1)) - L(w(r)) \tag{14}$$
$$\geq \nabla L(w(r))^T(w(r+1) - w(r)) + \frac{\rho}{2}\|(w(r+1) - w(r))\|^2 \tag{15}$$
$$> \nabla L(w(r))^T(w(r+1) - w(r)) \tag{16}$$
$$= \nabla L(w(r))^T(-\alpha g(w(r))) \tag{17}$$

We assume that we train two different mini-batches $\mathcal{B}_1$ and $\mathcal{B}_2$ of size b_1 and b_2 at the r-th iteration. We set b_1 and b_2 to satisfy $b_1 = k^2 b_2$, where k is an arbitrary nonzero positive number. In addition, we use $w_1(r+1)$ and $w_2(r+1)$ to represent the model parameters after training $\mathcal{B}_1$ and $\mathcal{B}_2$ respectively. According to Eq. (17), we have:

$$L(w_1(r+1)) - L(w(r)) \geq \nabla L(w(r))^T(-\alpha\|g(w(r), \mathcal{B}_1)\|) \tag{18}$$

$$L(w_2(r+1)) - L(w(r)) \geq \nabla L(w(r))^T(-\alpha\|g(w(r), \mathcal{B}_2)\|) \tag{19}$$

According to Corollary 1, we know:

$$g(w(r), \mathcal{B}_1) \sim \mathcal{N}(\nabla L(w), \frac{\mathcal{V}(w)}{b1}) \tag{20}$$

$$g(w(r), \mathcal{B}_2) \sim \mathcal{N}(\nabla L(w), \frac{\mathcal{V}(w)}{b2}) \tag{21}$$

We denote $g(w(r), \mathcal{B}_1)$ and $g(w(r), \mathcal{B}_2)$ as $\nabla L(w) + d_1$ and $\nabla L(w) + d_2$, where d_1 and d_2 follow:

$$d_1 \sim \mathcal{N}(0, \frac{\mathcal{V}(w)}{b_1}) \tag{22}$$

$$d_2 \sim \mathcal{N}(0, \frac{\mathcal{V}(w)}{b_2}) \tag{23}$$

Therefore, we can rewrite Eq. (20) and Eq. (21) in the following form:

$$L(w(r)) - L(w_1(r+1)) \leq L(w(r))^T\alpha\|\nabla L(w) + d_1\| \tag{24}$$

$$L(w(r)) - L(w_2(r+1)) \leq L(w(r))^T\alpha\|\nabla L(w) + d_2\| \tag{25}$$

We multiply both sides of inequality Eq. (19) by k:

$$k(L(w(r)) - L(w_2(r+1))) \leq \rho\alpha\|k\nabla L(w) + kd_2\| \tag{26}$$

Since $b_1 = k^2 b_2$, we know that d_1 and kd_2 have the same distribution. According to [15], the magnitude of variance is much greater than that of the mean, so the mean can be ignored. Therefore, we obtain the following conclusion:

$$k(L(w(r)) - L(w_2(r+1))) \approx L(w(r)) - L(w_1(r+1))) \tag{27}$$

Therefore, increasing the batch size by a factor of k^2 will increase the rate of decrease of the loss function by a factor of k, *i.e.*, the decrease in the loss function per iteration is approximately positively correlated with the square root of the batch size. □

We use T^c and T^t to denote the computation time and the communication time per iteration, and summarize the convergence bounds related to computation time and communication time in Theorem 1.

Theorem 1. *After training for R rounds, the convergence has an upper bound:*

$$L(w(R)) - L(w^*) \leq L(w(0)) - \frac{T^B}{T^c + T^t} \cdot \sqrt{|W|b} \cdot \xi_m \tag{28}$$

where $L(w^*)$ represents the minimum value of loss function according to Assumption 6.

Proof. According to the conclusion of Corollary 2, we can obtain the cumulative sum of R rounds' results by adding them up:

$$\begin{aligned} &L(w(R)) - L(w^*) \leq L(w(R)) - L(w(0)) \\ &= \sum_{i=1}^{R} L(w(i)) - L(w(i-1)) \\ &\leq \sum_{i=1}^{R} -\rho\alpha\|g(w(i-1))\| \\ &\leq -\frac{T^B}{T^c + T^t} \cdot \sqrt{|W|b} \cdot \xi_m \end{aligned} \tag{29}$$

where $|W|$ represents the number of workers, and b represents the batch size. In addition, ξ_m denotes $\min_{i\in[1,R]}\{\xi_i\}$ where ξ_i means a constant depends on the model parameters at the i-th iteration, as indicated in Corollary 2. □

IV. PROBLEM DEFINITION

This section gives the problem formulation of in-network aggregation with batch size adjustment for distributed training. During gradient transmission, InGo needs to decide which aggregation node (*e.g.*, a switch or the PS) each gradient should be aggregated at based on the network state. For each worker $w \in W$, we use $x_w^a \in \{0, 1\}$ to represent whether $a \in A$ is the aggregation node of worker w. We aim to minimize the value of loss function for a given time budget T^B.

Total Time Budget Constraint: We assume that the model can be trained for a total of R iteration in T^B. We use T_w to denote the amount of time worker w needs to complete one iteration and T_r to denote the completion time of the r-th iteration. T_w is composed of conmunication time T_w^t and computation time T_w^c. Furthermore, we use M to denote the model size and f to denote the sending rate. The total time budget constraint is represented as follows:

$$\sum_{r=1}^{R} T_r \leq T^B, \tag{30}$$
$$T_r = \max_{w\in W} T_w, \tag{31}$$
$$T_w = T_w^t + T_w^c, \forall w, \tag{32}$$
$$T_w^c = b \cdot c_w + o_w, \forall w, \tag{33}$$

$$T_w^t = \frac{M}{f}, \forall w, \tag{34}$$

where Eq. (30) presents the total time budget constrain. Eq. (31) indicates that T_r depends on the slowest worker. The time T_w it takes for a worker w to complete an iteration is reliant on both the computation time T_w^c and the communication time T_w^t as shown in Eq. (32). In Eq. (33), c_w and o_w are introduced to capture the linear relationship between batch size b and computation time [10], these variables are influenced by the specific hardware devices employed by the worker, such as CPU and GPU, as well as the characteristics of the model that the worker is being trained on. The value of T_w^t is calculated by dividing the model size M by the sending rate f as indicated in Eq. (34).

Aggregation Constraint: We use $x_w^a \in \{0,1\}$ to represent whether the gradient of worker w is aggregated by aggregation node $a \in A$ or not. Additionally, we use $y^s \in \{0,1\}$ to denote whether the programmable switch s is assigned to aggregate gradients or not. The aggregation constraints are as follows:

$$\sum_{a \in S \cup \{d\}} x_w^a = 1, \forall w, \tag{35}$$

$$x_w^s \leq y^s, \forall s, \tag{36}$$

Due to the limited availability of programmable switches in the cluster, aggregating the gradient across multiple switches may lead to an increased number of forwarding hops and higher utilization of bandwidth resources. In line with [5], we assume that only one aggregation can be performed on a gradient, as shown in Eq. (35). Furthermore, Eq. (36) ensures that if an arbitrary gradient is aggregated by switch s, then s is considered as the selected aggregation node.

Path Constraint: The notation $q_p \in \{0,1\}$ denotes whether the path p is selected. We express the path constraint as follows:

$$\sum_{p \in P_w^a} q_p = x_w^a, \forall w, a, \tag{37}$$

$$\sum_{p \in P_s^d} q_p = y^s, \forall s, \tag{38}$$

As shown in Eq. (37), each worker $w \in W$ should choose a forwarding path $p \in P_w^a$ between worker and its corresponding aggregation node ($x_w^a = 1$). Similarly, if programmable switch $s \in S$ is selected as an aggregation node ($y^s = 1$), there should be a path $p \in P_s^d$ selected between it and the PS, *i.e.*, Eq. (38).

Bandwidth Constraint: Let $I(e,p) \in \{0,1\}$ denote whether the path p goes through the link e or not. The bandwidth of the link e is represented by $B(e)$ and the bandwidth constraint is represented as follows:

$$\sum_{p \in P_w^a \cup P_s^d} q_p \cdot I(e,p) \cdot f \leq B(e), \forall e, \tag{39}$$

where Eq. (39) ensures that the total bandwidth occupied by the gradients must not exceed the link's available bandwidth.

Switch Aggregation Capacity Constraint: We use $C(s)$ to denote the processing capacity of a switch. Therefore we have the following contraints:

$$\sum_{w \in W} x_w^s \cdot f \leq C(s), \forall s, \tag{40}$$

The total bandwidth required for aggregating the gradients must not exceed the processing capacity constraint of the switch as emphasized in Eq. (40).

PS Capacity Constraint: We use $N(d)$ to represent the ingress bandwidth of the PS and formulate this constraint as follow:

$$\sum_{p \in P_s^d \cup P_w^d} q_p \cdot f \leq N(d), \tag{41}$$

where Eq. (41) guarantees that the total bandwidth used for sending gradients to the PS cannot exceed the PS's ingress bandwidth capacity.

Our objective is to minimize the loss function at the final iteration, *i.e.*, $L(w(R))$, while considering the limited available resources within the given time budget. We formalize this problem as follows:

$$\min L(w(R))$$
$$S.t. \ \ (30) \sim (41)$$

We use $h(b)$ to represent the right-hand side of Eq. (29), and combining Eq. (33) and Eq. (34), we gain:

$$h(b) \leq L(w(0)) - \frac{\sqrt{|W|} b T^B \xi_m}{(k \cdot b + o) + \frac{M}{f}} \tag{42}$$

Thus, we found that as the increasing of sending rate f, the convergence bound of loss function is lower. Therefore, we minimize the loss function by designing an in-network aggregation routing algorithm in the next section to optimize the sending rate as much as possible.

V. Algorithm Design and Theoretical Analysis

A. Routing Selection Algorithm Design for INA

In this section, we propose an algorithm based on randomized rounding, called R-InGo, to maximize the sending rate and minimize the loss function, the algrothm is sumrized in Alg. 2. According to the analysis in Section III-IV, we abstract the problem into the following optimization problem as follows:

$$\max f$$
$$S.t. \ \ (35) \sim (41)$$

We design the algorithm via four major steps: 1) Relax the variables of the original problem, construct it as a linear programming problem, and then solve the optimal solution of this linear equation group; 2) Determine INA routing scheme, including the aggregation nodes and path selection of each gradient according to the optimal solution of the linear equation group; 3) Calculate the sending rate based on the INA routing scheme; 4) Calculate the optimal batch size selection according to Eq. (28).

1) **Construct Linear Programming Problem:** We relax the constraints by replacing $x_w^a \in \{0,1\}$, $y^s \in \{0,1\}$ and $q_p \in \{0,1\}$ with $x_w^a \in [0,1]$, $y^s \in [0,1]$ and $q_p \in [0,1]$. Then we replace f with $g = \frac{1}{f}$ and move it to the other side of the inequality to perform equivalent

transformations and convert the nonlinear equation into a linear equation Eq. (43). We bring the inequality group into the linear programming solver (*e.g.*, PuLP) and represent the optimal solution as $\widetilde{x}_w^a, \widetilde{y}^s, \widetilde{q}_p, \widetilde{f}$.

$$\max f$$
$$\begin{cases} \sum\limits_{a \in S \cup \{d\}} x_w^a = 1, & \forall w \in W \\ x_w^s \le y^s, & \forall s \in S \\ \sum\limits_{p \in P_w^a} q_p = x_w^a, & \forall w, a \\ \sum\limits_{p \in P_s^d} q_p = y^s, & \forall s \in S \\ \sum\limits_{w \in W} x_w^s \le C(s) \cdot g, & \forall s \in S \\ \sum\limits_{p \in P_s^d \cup P_w^d} q_p \le N(d) \cdot g, & \\ \sum\limits_{p \in P_w^a \cup P_s^d} q_p \cdot I(e,p) \le B(e) \cdot g, & \forall e \\ g = \frac{1}{f}, & \\ x_w^a \in [0,1], & \forall w \in W, a \\ y^s \in [0,1], & \forall s \in S \\ q_p \in [0,1], & \forall p \in P \end{cases} \tag{43}$$

2) **Determine INA Routing Scheme:** R-InGo takes the optimal solution $\widetilde{x}_w^a$ as the probability of selecting the aggregation node and picks out one among all nodes as the aggregation node. Based on the result of the aggregation node selection, we obtain integer solutions $\widehat{x}_w^a$ and $\widehat{y}^s$ (Lines 4-9). Similarly, the path p is choosed as the forwarding path with the probablity $\frac{\widetilde{q}_p}{\widetilde{x}_w^a}$, then the integer solution $\widehat{q}_p$ is obtained (Lines 10-16). Based on the routing scheme, we obtain the maximum sending rate $\widehat{f}$ which satisfies all constraints.
3) **Calculate the Optimal Batch Size Selection:** We record the computation time of different batch size. Specially we set b_{min} to 16 and b_{max} to 256, and train batches of different sizes from b_{min} to b_{max} to record their computation time. Then, we fit o_w and c_w satisfying Eq. (33) by the least squares method through the measured computation time. Finally, we compute the optimal batch size according to Eq. (42) (Lines 19-22).

B. Performance Analysis

Theorem 2. *R-InGo guarantees the uniqueness of aggregation node selection for each worker.*

Proof. The selection of aggregation nodes can be abstracted as a multinomial distribution model. Specially, for each worker $w \in W$ and $a_i \in A$, $x_w^{a_1}, x_w^{a_2}, ..., x_w^{a_{|A|}}$ can be viewed as a set of 0-1 random variables, where $P(x_w^{a_i} = 1) = \widetilde{x}_w^{a_i}$. Thus we have the following equation:

$$Pr(x_w^{a_1} = \widehat{x}_w^{a_1}, ..., x_w^{a_{|A|}} = \widehat{x}_w^{a_{|A|}}) = \frac{\sum\limits_{a_i \in A} \widehat{x}_w^{a_i}}{\prod\limits_{a_i \in A} \widehat{x}_w^{a_i}!} \prod_{a_i \in A} (\widetilde{x}_w^{a_i})^{\widehat{x}_w^{a_i}} \tag{44}$$

where $\widehat{x}_w^{a_i}$ represents the possible values of the random variable $x_w^{a_i}$. In line 5 of Alg. 2, the selection of aggregation nodes can be viewed as a sampling in this distribution, and therefore guarantees the uniqueness. □

Algorithm 2 R-InGo: Routing Selection Algorithm for In-network Aggregation

1: **Step 1: Solving the Relaxed Problem**
2: Derive the optimal solutions $\left\{\widetilde{x}_w^a, \widetilde{y}^s, \widetilde{q}_p, \widetilde{f}\right\}$
3: **Step 2: Choose Aggregation Nodes and Routing Paths**
4: **for** each worker $w \in W$ **do**
5: Choose the aggregation node $a \in S \cup \{d\}$ with the probability $\frac{\widetilde{x}_w^a}{\sum_{a \in A} \widetilde{x}_w^a}$
6: If a node a is chosen as a aggregation node, we set $\widehat{x}_w^a = 1$
7: A_w is used to denote the aggergation node of w
8: **for** each switch $s \in S$ **do**
9: We set $\widehat{y}^s = 1$ if there exists any workers choose s as the aggregation node
10: **for** each worker $w \in W$ **do**
11: Choose a path $p \in P_w^{A_w}$ with the probability $\frac{\widetilde{q}_p}{\widetilde{x}_w^{A_w}}$
12: Set $\widehat{q}_p = 1$
13: **for** each switch $s \in S$ **do**
14: **if** $\widehat{y}^s = 1$ **then**
15: Choose a path $p \in P_s^d$ with the probability $\frac{\widetilde{q}_p}{\widetilde{y}^s}$
16: Set $\widehat{q}_p = 1$
17: Compute the value of the maximum sending rate $\widehat{f}$
18: **Step 3: Compute the Value of the Batch Size**
19: **for** b in $[b_{min}, b_{max}]$ **do**
20: Record the computation time of a batch with size of b
21: Obtain the relation between computation time and batch size of the form Eq. (33)
22: b = $arg \min\limits_{b \in \mathbb{Z}^+} h(b)$ according to Eq. (42)

Output:
$\left\{\widehat{x}_w^a, \widehat{y}^s, \widehat{q}_p, \widehat{f}\right\}$

Theorem 3. *R-InGo guarantees the uniqueness of the link selection.*

The proof of Theorem 3 is similar to that of Theorem 2. Due to limited space, the detailed proof is omitted here.

Lemma 1. ***Chernoff Bound****: Given n independent variables: $z_1, z_2, ..., z_n, \forall z_i \in \{0,1\}$. Let $\tau = \mathbb{E}[\sum_{i=1}^n z_i]$. Then, we have:*

$$Pr[\sum_{i=1}^{n} z_i \ge (1+\rho)\tau] \le e^{\frac{-\rho^2 \tau}{2+\rho}} \tag{45}$$

Theorem 4. *R-InGo will not exceed the Aggregation Constraint by an approximation factor of $O(2\ln|S|+1)$. Under the proper assumption (i.e., $C(s) \ge 10\widehat{f}$), the bound can be tightened to 2.28.*

Proof. Since each worker w chooses the switch as a aggregation node according to the probability $\widetilde{x}_w^s$ randomly and independently, we have $\mathbb{E}[\widehat{x}_w^s] = \widetilde{x}_w^s$. Let $z_1^s = \widehat{x}_{w_1}^s, z_2^s = \widehat{x}_{w_2}^s, ..., z_{|W|}^s = \widehat{x}_{w_{|W|}}^s$. Therefore, for an arbitrary switch s, we construct $|W|$ independent variables: $z_1^s, z_2^s, ..., z_{|W|}^s$. We

assume $\mathbb{E}[\sum_{i=1}^{|W|} z_i^s] = \tau_1$. Therefore we have:

$$\begin{cases} z_i^s \in [0,1], \forall z_i \\ \mathbb{E}[\sum_{i=1}^{|W|} z_i^s] = \tau_1 \end{cases} \tag{46}$$

Combined with $\mathbb{E}[\widehat{x}_w^s] = \widetilde{x}_w^s$, we have:

$$\mathbb{E}[\sum_{i=1}^{|W|} z_i^s] = \sum_{i=1}^{|W|} \mathbb{E}[z_i^s] = \sum_{i=1}^{|W|} \mathbb{E}[\widehat{x}_w^s] = \sum_{i=1}^{|W|} \widetilde{x}_w^s \tag{47}$$

Since the optimal solution of the linear equation ensures the validity of the constraint, we know $\sum_{i=1}^{|W|} \widetilde{x}_w^s = \tau_1 \leq \frac{C(s)}{\widetilde{f}}$. We assume that exists a constant $k_1 \geq 1$ such that $k_1\tau_1 = \frac{C(s)}{\widetilde{f}}$.

By applying Lemma 1, for an arbitrary switch s, we have:

$$Pr[\sum_{i=1}^{|W|} z_i^s \geq (1+\rho)\tau_1] \leq e^{\frac{-\rho^2\tau_1}{2+\rho}} \tag{48}$$

We assume that:

$$e^{\frac{-\rho^2\tau_1}{2+\rho}} \leq \frac{1}{|S|} \tag{49}$$

which means that the probability bound goes quickly to zero as the number of switches increases.

By solving Eq.(49), we have:

$$\rho \geq \frac{\ln|S| + \sqrt{\ln^2|S| + 8\tau_1 \ln|S|}}{2\tau_1} \tag{50}$$

Thus, the approximate factor of the aggregation constraint is as follow:

$$\frac{(\rho+1)}{k_1} = \frac{\ln|S| + 2\tau_1 + \sqrt{\ln^2|S| + 8\tau_1 \ln|S|}}{2k_1\tau_1} \tag{51}$$

Since k_1 is greater than 1, therefore we have:

$$\frac{(\rho+1)}{k_1} \leq \frac{\ln|S| + 2k_1\tau_1 + \sqrt{\ln^2|S| + 8k_1\tau_1 \ln|S|}}{2k_1\tau_1} \tag{52}$$

$$\Rightarrow \frac{(\rho+1)}{k_1} \leq \frac{\ln|S| + \sqrt{\ln^2|S| + 8\frac{C(s)}{\widetilde{f}} \ln|S|}}{2\frac{C(s)}{\widetilde{f}}} + 1 \tag{53}$$

Since $\sqrt{\ln^2|S| + 8\frac{C(s)}{\widetilde{f}} \ln|S|}$ is lower than $\ln|S| + \sqrt{8\frac{C(s)}{\widetilde{f}} \ln|S|}$, we have:

$$\frac{(\rho+1)}{k_1} \leq \frac{2\ln|S| + \sqrt{8\frac{C(s)}{\widetilde{f}} \ln|S|}}{2\frac{C(s)}{\widetilde{f}}} + 1 \tag{54}$$

Therefore, the approximate factor of the *Aggregation Constraint* is $O(2\ln|S| + 1)$.

A programmable switch can have a processing capacity of up to 3.2Tbps. The network interface card processing capacity of the PS can achieve 100Gbps. In addition, we assume that there are 50 switches [8]. Based on the above assumptions, the optimal solution for $\widetilde{f}$ can reach at most 12.5Gbps through our experiments. Since the aggregation capacity of the switch is much larger than the sending rate, we make a reasonable assumption, *i.e.*, $C(s) \geq 10\widetilde{f}$. By substituting the above values into the Eq. (54), the bound can be tightened to about 2.28. □

Theorem 5. *R-InGo will not exceed the Bandwidth Constraint and the PS Capacity Constraint by an approximation factor of $O(\ln|S|)$. Under the proper assumption (i.e., $\frac{B(e)}{\widetilde{f}} \approx 2\ln|S|$ and $\frac{N(d)}{\widetilde{f}} \approx 2\ln|S|$), the bound can be tightened to 2.26.*

The proof of Theorem 5 is similar to that of Theorem 4, thus we omit the detailed proof here.

VI. Performance Evaluation

In this section, we conduct a comprehensive evaluation of InGo. We begin by presenting the experimental setup, which includes the performance metrics, benchmarks, datasets and models. Subsequently, we construct a small-scale testbed consisting of 4 Wedge100BF-32x programmable switches to assess the efficiency of InGo. Furthermore, we perform large-scale simulations as a supplement to our evaluation.

A. Experiment Setup

Performance Metrics: We adopt the following performance metrics to evaluate the improvement of InGo for distributed training: (1) the gradient sending rate; (2) the aggregated tensor elements per second (denoted by ATEs); (3) the test accuracy; (4) the train loss.

During the experiment run, we use iftop to monitor the bandwidth usage of each worker as the *sending rate*. *ATEs* is measured by the number of elements aggregated on switches per second during communication. Similar to previous studies [8, 16], this metric can be used to evaluate the efficiency of INA. Moreover, we assess the *test accuracy* as the ratio of correctly predicted samples, and the value of *train loss* is determined by the cross-entropy between predicted results and true labels.

Benchmarks: We compare InGo with three benchmarks. The first benchmark is ATP [5], which utilizes in-network aggregation through pre-defined routing paths in a two-layer topology. During communication, gradients are aggregated at the first available aggregation node encountered. The second benchmark is an extension of ATP, called GRID [8], that incorporates gradient routing scheduling to consider the impact of the aggregation node location on aggregation efficiency. The third benchmark is a simplified version of InGo with a fixed batch size of 64, denoted by InGo-64. We use InGo-64 to show the acceleration effect of batch size adjustment on distributed training.

Datasets and Models: We assess the performance of InGo by conducting model training on the CIFAR-10 and CIFAR-100 datasets. CIFAR-10 comprises 60,000 color images, with 50,000 images for training and 10,000 for testing, distributed across 10 different classes. In contrast, CIFAR-100 contains the same number of images as CIFAR-10, but with a more challenging classification task, as it consists of 100 classes. For our evaluations, we train two popular models, namely AlexNet and Inception-V3 , on both datasets. AlexNet consists of one 3 × 3 convolutional layer, one 5 × 5 convolutional layer, three 3 × 3 convolutional layers, two fully-connected hidden layers, and one fully-connected output layer. Inception-V3 comprises five convolutional layers and eleven inception blocks. To simplify references, we denote the training of

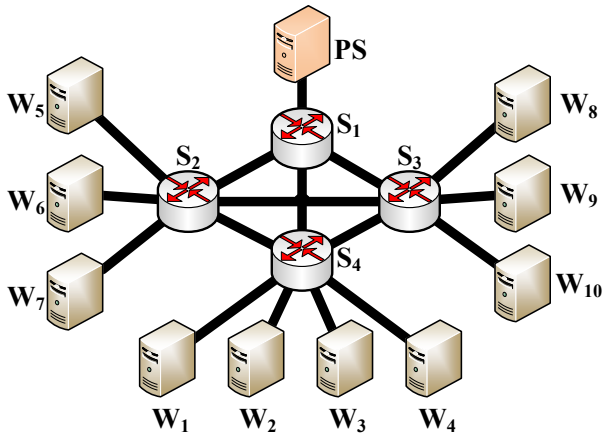


Fig. 1: Topology of the testbed consisting of 1 PS, 10 workers (W_1-W_{10}) and 4 programmable switches (S_1-S_4).

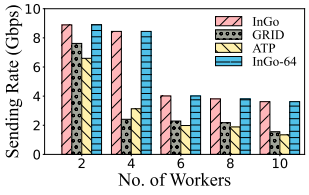


Fig. 2: Sending Rate of IncV3-C100 vs. No. of Workers

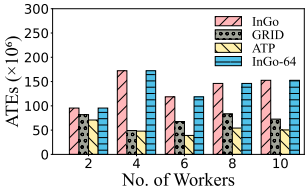


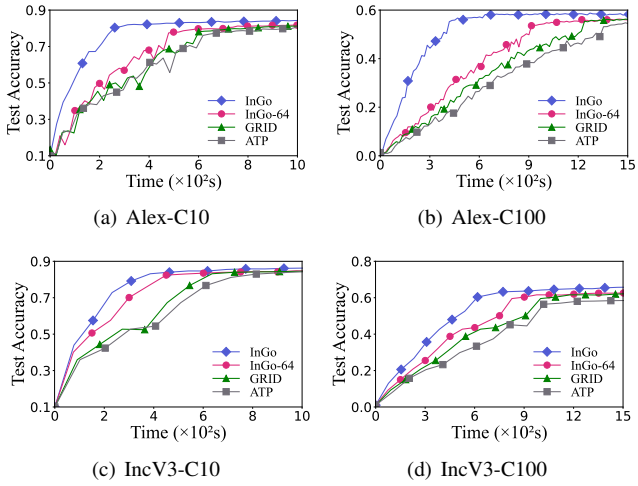
Fig. 3: ATEs of IncV3-C100 vs. No. of Workers

Inception-V3 on CIFAR-10 as IncV3-C10, and refer to the training of AlexNet on CIFAR-10, Inception-V3 on CIFAR-100, and AlexNet on CIFAR-100 as Alex-C10, IncV3-C100, and Alex-C100, respectively.

B. Testbed Evaluation

Testbed Settings: As illustrated in Fig. 1, the testbed comprises 11 servers and 4 Wedge100BF-32x programmable switches. One server acts as the PS, and the remaining 10 servers serve as workers (W_1-W_{10}). Four switches (S_1-S_4) are interconnected. Moreover, S_1 is connected to the PS, while S_2, S_3, and S_4 are connected to 3, 3, and 4 workers, respectively. Each link has a uniform bandwidth, 10Gbps. Each server is equipped with an RTX3090 GPU, a 22-core Intel Xeon 6152 processor, and a Mellanox ConnectX-6 100G dual-port NIC. All the programmable switches feature the Intel Tofino chip and are equipped with Software Development Environment (SDE) 9.7.0. We utilize the PyTorch framework for model training.

Comparision on Sending Rate: In this set of evaluations, we analyze the performance of the sending rate, and the results are presented in Fig. 2. Since different models and datasets do not affect the magnitude of the gradients, we only show the sending rates of IncV3-C100. Since InGo-64 performs the same INA routing as InGo, the sending rate of the InGo and InGo-64 is very close. Thus, we just compare InGo, ATP, and GRID. Notably, InGo consistently maintains the highest sending rate as the number of workers increases. Taking training with 10 workers as an example, InGo achieves a sending rate of 3.74Gbps, while GRID and ATP have sending rates of 1.39Gbps and 1.61Gbps, respectively. This demonstrates that InGo significantly outperforms both GRID and ATP. It increases the sending rates by 169.5% and 133.3%, respectively. The key factor behind InGo's superior performance lies in its accurate modeling of link bandwidth limitations, considering real-world scenarios with varying background traffic in clusters.



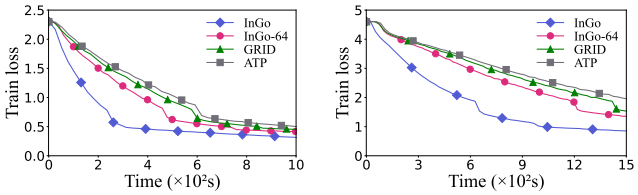
(a) Alex-C10 (b) Alex-C100

(c) IncV3-C10 (d) IncV3-C100

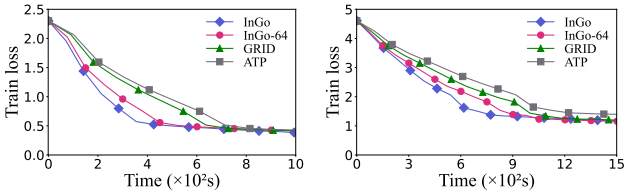
Fig. 4: Test Accuracy vs. Time

Comparision on ATEs: This set of evaluations gives ATEs of different schemes, as depicted in Fig. 3. Similar to the comparision on sending rate, we only show the ATEs of IncV3-C100 and just compare InGo with ATP and GRID. InGo demonstrates its ability to maximize the number of elements aggregated per second as the number of workers increases. For instance, with 10 workers, InGo aggregates 152.61×10^6 elements per second, surpassing ATP and GRID, which only aggregate 50.32×10^6 and 72.67×10^6 elements per second, respectively. This clearly illustrates InGo's superiority, as it can aggregate 203.22% and 109.99% more elements per second compared to ATP and GRID, respectively. The remarkable performance of InGo in ATEs is attributed to its effective utilization of the aggregation capability of programmable switches through INA routing.

Comparison on Test Accuracy and Train Loss: We conduct distributed training using 10 workers and record the test accuracy and train loss, as shown in Figs. 4-5. Notably, InGo exhibits the fastest convergence rate and the most rapid decrease in the loss function. We set the target test accuracy to 76% on the CIFAR-10 dataset and 54% on the CIFAR-100 dataset according to [17]. When training Inception-V3 on CIFAR-100, InGo achieves 54% test accuracy in 617.7s, while InGo-64, GRID, and ATP required 826.2s, 908.7s, and 1120.4s, respectively. That means InGo reduces the completion time by 25.2%, 32.0% ,and 44.9% compared to InGo-64, GRID and ATP respectively. Furthermore, InGo consistently maintains the lowest loss function value throughout the training process. This favorable outcome is attributed to InGo's ability to adjust the batch size while optimizing the INA

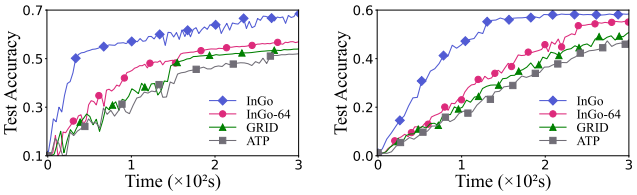


(a) Alex-C10 (b) Alex-C100

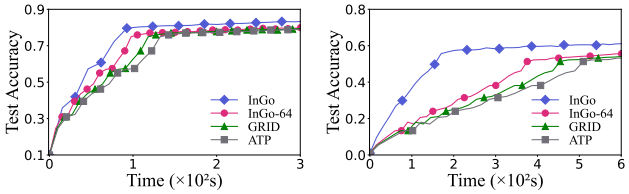


(c) IncV3-C10 (d) IncV3-C100

Fig. 5: Train Loss vs. Time



(a) Alex-C10 (b) Alex-C100



(c) IncV3-C10 (d) IncV3-C100

Fig. 8: Test Accuracy vs. Time

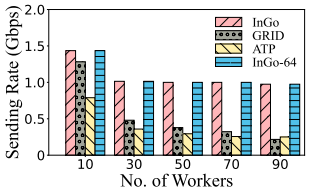


Fig. 6: Sending Rate of IncV3-C100 vs. No. of Workers

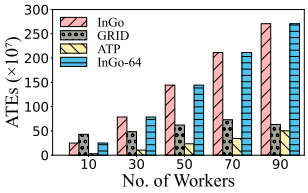


Fig. 7: ATEs of IncV3-C100 vs. No. of Workers

routing, effectively accelerating model training and achieving faster convergence.

C. Simulation Evaluation

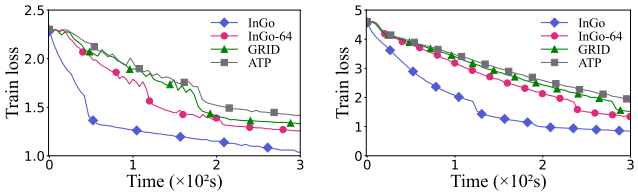
Simulation Settings: We perform simulations on a physical server equipped with an Intel Core i9-10900 processor and an NVIDIA GeForce RTX 3090 GPU. During our simulations, we encounter limitations in using Mininet to replace switches with BMv2 for simulating the behavior of programmable switches. The performance of BMv2 is proved insufficient to support large-scale simulation experiments. Specifically, when the number of workers reach tens of levels, the bandwidth of BMv2 is limited to just a few Mbps, and exhibits a high packet loss rate. This observation aligns with the findings reported in [8]. In light of these limitations, we make the decision to conduct tests by simulating the aggregation behavior of gradients within a single server. Moreover, we utilize the linear equation solver PuLP for computing the routing results and use the PyTorch framework for model training.

We conduct simulation experiments on a fat-tree topology, a commonly used network architecture in clusters. We give the performance comparisons of the sending rate and ATEs by varying the number of workers and measure the test accuracy and train loss with 50 workers as the default setting. To simulate link bandwidth heterogeneity, we set the link bandwidth to random values between 3Gbps and 10Gbps. Additionally, we set the switch processing capacity to 3.2Tbps and the PS processing capacity to 100Gbps. To expedite the algorithm execution, we enforce the restriction that workers cannot select switches from other pods for aggregation during the algorithm execution. By enforcing this restriction, we can progressively speed up the algorithm execution while having only a minimal impact on the final solution. To avoid randomness, we repeat the worker position selection 10 times and execute R-InGo 10 times for each selection to decide the INA routing. Finally, we average the results and obtain the final results.

Comparison on Sending Rate: As depicted in Fig. 6, the results demonstrate that regardless of the number of workers involved, InGo consistently achieves the highest sending rate. For instance, when 50 workers are conducting in-network aggregation routing, InGo achieves a remarkable sending rate of 1.00Gbps, whereas GRID and ATP only reach 0.38Gbps and 0.29Gbps, respectively. This highlights InGo's ability to accelerate the worker's sending rate by 240.4% and 164.5% compared to GRID and ATP, respectively. By incorporating link effects during the modeling process, InGo optimizes bandwidth utilization by avoiding congestion.

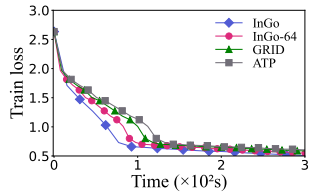
Comparision on ATEs: As depicted in Fig. 7, InGo consistently outperforms in aggregating the maximum number of elements per second in most cases. For instance, with 50 workers, InGo can aggregate 144.3×10^7 elements per second, while ATP and GRID can only aggregate 23.5×10^7 and 62.0×10^7 elements per second, respectively. InGo achieves 514.0% and 132.7% higher ATEs than GRID and ATP, respectively. This is attributed to InGo's effective utilization of switch resources through INA routing. Notably, when the number of workers is 10, InGo may not achieve the highest ATEs due to the processing capacity of the PS does not become a bottleneck, leading to partial aggregation being performed by the PS.

Comparison on Test accuracy and Train loss: We record the test accuracy and train loss, as depicted in Figs. 8-9. InGo achieves the fastest convergence rate and the most rapid decrease in the loss function. For instance, when training Inception-V3 on CIFAR-100, InGo reaches 54% test accuracy in 169.8s, while InGo-64, GRID, and ATP takes 495.7s, 599.8s, and 672.3s, respectively. Moreover, compared to InGo-64, GRID, and ATP, InGo reduces the completion time by

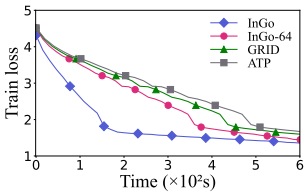


(a) Alex-C10

(b) Alex-C100



(c) IncV3-C10



(d) IncV3-C100

Fig. 9: Train Loss vs. Time

65.7%, 71.6%, and 74.7%, respectively. These significant performance gains can be attributed to InGo's intelligent INA routing and batch size adjustment strategies.

Based on the experimental results, we can draw the following conclusions. Firstly, as illustrated in Figs. 2 and 6, InGo achieves the highest sending rate, which represents that InGo can effectively reduce the communication time. Secondly, as shown in Figs. 3 and 7, InGo demonstrates a remarkable increase in ATEs because of the efficient utilization of bandwidth and switch resources. At last, InGo has the shortest completion time and the fastest loss function descent rate, as depicted in Figs. 4-5 and 8-9, which indicates that INA routing with batch size adjustment can effectively accelerate distributed model training.

VII. Conclusion

In this paper, we present InGo, a novel scheme that combines in-network aggregation routing and batch size adjustment to accelerate distributed training. We further analyze the model convergence bound and give the problem definition of in-network aggregation routing with batch size adjustment. In addition, we propose a randomized rounding based algorithm to solve the problem. Experimental results demonstrate that InGo significantly accelerates distributed model training.

Acknowledgement

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This work was supported in part by the National Science Foundation of China (NSFC) under Grants 62132019, 62372426 and 62102392, in part by the National Science Foundation of Jiangsu Province under Grant BK20210121, in part by the Hefei Municipal Natural Science Foundation under Grant 2022013, in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences under Grant 2023481, and in part by the Fundamental Research Funds for the Central Universities.

References

[1] 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.

[2] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] 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.

[5] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift, "Atp: In-network aggregation for multi-tenant learning." in *NSDI*, vol. 21, 2021, pp. 741–761.

[6] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3: Energy-efficient microservices on smartnic-accelerated servers." in *USENIX annual technical conference*, 2019, pp. 363–378.

[7] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 90–106.

[8] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "Grid: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Transactions on Networking*, 2023.

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, and K. Keutzer, "Firecaffe: near-linear acceleration of deep neural network training on compute clusters," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2592–2600.

[10] J. Ren, G. Yu, and G. Ding, "Accelerating dnn training in wireless federated edge learning systems," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 219–232, 2020.

[11] A. Devarakonda, M. Naumov, and M. Garland, "Adabatch: Adaptive batch sizes for training deep neural networks," 2018.

[12] L. Balles, J. Romero, and P. Hennig, "Coupling adaptive batch sizes with learning rates," *arXiv preprint arXiv:1612.05086*, 2016.

[13] H. Wang, Z. Qu, Q. Zhou, H. Zhang, B. Luo, W. Xu, S. Guo, and R. Li, "A comprehensive survey on training acceleration for large machine learning models in iot," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 939–963, 2021.

[14] J. Xu, S.-L. Huang, L. Song, and T. Lan, "Live gradient compensation for evading stragglers in distributed learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[15] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: closing the generalization gap in large batch training of neural networks," *Advances in neural information processing systems*, vol. 30, 2017.

[16] 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.

[17] Y. Xu, Y. Liao, H. Xu, Z. Ma, L. Wang, and J. Liu, "Adaptive control of local updating and model compression for efficient federated learning," *IEEE Transactions on Mobile Computing*, 2022.