InArt: In-Network Aggregation with Route Selection for Accelerating Distributed Training

Jiawei Liu
Yutong Zhai
Gongming Zhao
liujiawei@mail.ustc.edu.cn
zyt1996@mail.ustc.edu.cn
gmzhao@ustc.edu.cn
School of Computer Science and Technology, University of Science and Technology of China
Suzhou Institute for Advanced Research, University of Science and Technology of China

Hongli Xu
Jin Fang
xuhongli@ustc.edu.cn
fangjin98@mail.ustc.edu.cn
School of Computer Science and Technology, University of Science and Technology of China
Suzhou Institute for Advanced Research, University of Science and Technology of China

Zhen Zeng
Arizona State University
zzeng22@asu.edu

Ying Zhu
School of Computer Science and Technology, University of Science and Technology of China
isaaczhu@mail.ustc.edu.cn

ABSTRACT

Deep learning has brought about a revolutionary transformation in network applications, particularly in domains like e-commerce and online advertising. Distributed training (DT), as a critical means to expedite model training, has progressively emerged as a key foundational infrastructure for such applications. However, with the rapid advancement of hardware accelerators, the performance bottleneck in DT has shifted from computation to communication. In-network aggregation (INA) solutions have shown promise in alleviating the communication bottleneck. Regrettably, current INA solutions primarily focus on improving efficiency under the traditional parameter server (PS) architecture and do not fully address the communication bottleneck caused by limited PS ingress bandwidth. To bridge this gap, we propose InArt, the first work to introduce INA with routing selection in a multi-PS architecture. InArt employs a multi-PS architecture to split DT tasks among multiple PSs, and selects appropriate routing schemes to fully harness INA capabilities. To accommodate traffic dynamics, InArt adopts a two-phase approach: splitting the training model among multiple parameter servers and selecting routing paths for INA. We propose Lagrange multiplier and randomized rounding algorithms for these phases, respectively. We implement InArt and evaluate its performance through experiments on physical platforms (Tofino switches) and Mininet emulation (P4 Software Switches). Experimental results show that InArt can reduce communication time by 48% ∼ 57% compared with state-of-the-art solutions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '24, May 13–17, 2024, Singapore, Singapore
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0171-9/24/05…$15.00
https://doi.org/10.1145/3589334.3645394

CCS CONCEPTS

• **Networks** → **In-network processing**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Web Infrastructure, Distributed Training, In-Network Aggregation, Route Selection, Programmable Switches

ACM Reference Format:
Jiawei Liu, Yutong Zhai, Gongming Zhao, Hongli Xu, Jin Fang, Zhen Zeng, and Ying Zhu. 2024. *InArt*: In-Network Aggregation with Route Selection for Accelerating Distributed Training. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* 11 pages. https://doi.org/10.1145/3589334.3645394

1 INTRODUCTION

Over the past decade, deep learning (DL) has become an essential component of many web applications [13, 14, 33, 43]. It plays a crucial role in domains such as e-commerce [13], social media [14], and online advertising [43], enabling the development of personalized recommendation systems, content analysis, and targeted advertising. The success of DL solutions lies in their sophisticated models, which contain numerous parameters and are trained on substantial amounts of data [34, 37]. However, training such models is time-consuming and computationally demanding. For instance, training a BERT model with 110 million parameters on a single server takes over 1.5 months [18]. To address this bottleneck and expedite model training, the adoption of DT is widespread in web infrastructure [34, 37, 42]. By harnessing the power of DT, web applications can efficiently process large datasets and leverage advanced DL models to deliver high-quality predictions and decision-making.

Following the typical PS architecture [26, 27], a DT system usually consists of a PS and multiple workers that perform many rounds of iterative training. In each iteration, workers compute a local gradient and send it to the PS for aggregation. The above two processes are called *gradient computation* and *gradient aggregation*,

respectively [29]. On the one hand, the rapid development of hardware accelerators (*e.g.*, GPU and FPGA) can significantly improve computing speed. On the other hand, considering that DL models employed in web applications often possess a substantial number of parameters, (*e.g.*, BERT [18] with about 110 million parameters, widely used in online advertising), the DT will introduce several gigabytes of data transfers. As a result, the performance bottleneck of DT has *shifted from computation to communication* [29, 32, 37]. For example, for training the DT job of BERT on 10Gbps links, more than half of the DT time is spent on communication [37].

To alleviate the communication bottleneck in DT, previous works usually focus on gradient compression [10, 11, 15] or communication scheduling [24, 28, 41]. However, gradient compression will inevitably lead to training accuracy degradation, and communication scheduling does not reduce the traffic volume and may still encounter the communication bottleneck on links or/and PSs. With the advent of programmable network hardware (*e.g.*, programmable switches [5] and smartNICs [6]), *in-network aggregation (INA)* [16, 29, 37] holds great promise in solving the communication bottleneck. Specifically, some gradients can be aggregated by programmable devices inside the network. In this way, the traffic volume sent to the PS will be reduced, thereby alleviating the inbound bandwidth bottleneck of the PS (see details in §2.1).

Accelerating DT with INA is complicated, and only a few works [29, 37] have made preliminary exploration in this field. SwitchML [37] aggregates gradients on top-of-rack (ToR) P4-programmable switches of workers to minimize communication overheads at a single-rack scale. ATP [29] proposes a protocol based on P4-programmable switches to support INA for multi-tenant learning across racks. GRID [20] addresses the selection of appropriate gradient aggregation points for each worker in a DT cluster. However, these works primarily focus on improving INA effectiveness under the traditional PS architecture. In reality, under this architecture, as training tasks scale up, the PS's ingress bandwidth may not meet the demands of synchronizing parameters, especially in DT clusters with numerous worker nodes. The INA scheme alone cannot fully resolve the communication bottleneck at the PS due to the transmission of a significant volume of gradient updates.

To tackle this challenge, we introduce a multi-PS architecture [40, 44] within the INA scheme. Notably, the multi-PS architecture and INA are two complementary and mutually beneficial approaches for alleviating communication bottlenecks in DT. By employing a multi-PS architecture to split DT tasks among multiple PSs and selecting appropriate routing schemes to fully harness the INA capabilities, we can effectively alleviate communication bottlenecks and enhance the efficiency of DT (as demonstrated in §5). Therefore, *there is an urgent need for INA solutions with route selection in multi-PS architectures*. However, it is a non-trivial mission to achieve the goal. Firstly, we need to decide which PS each gradient should be sent to, *i.e.*, *the routing destination is uncertain*. Secondly, aggregating gradients on programmable switches will change the traffic size in forwarding, *i.e.*, *the routing traffic volume is variable*. Thirdly, performing DT will face multi-dimensional *resource constraints*, such as switch/PS processing capacity and link bandwidth. To address these challenges, this paper proposes InArt and the main contributions are as follows:

- We design InArt, the first-of-its-kind INA work with routing selection in a multi-PS architecture for accelerating DT.
- Due to traffic dynamics, we take a two-phase approach: splitting the training model among multiple PSs and selecting the routing paths for INA. For the first phase, we propose a Lagrange multiplier-based algorithm (L-InArt). For the second phase, we design a randomized rounding-based algorithm (R-InArt).
- We implement InArt on both the hardware testbed (with Tofino switches) and software emulation (with bmv2 P4 switches). Experimental results show that InArt achieves superior performance compared with state-of-the-art solutions.

2 MOTIVATION

2.1 A Motivation Example

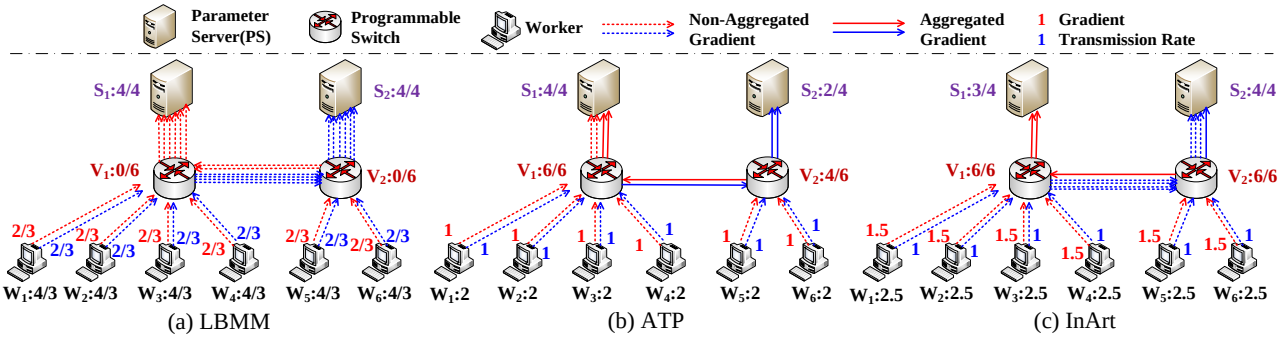
This section illustrates the pros and cons of state-of-the-art solutions through an example, which motivates our study. As shown in Fig. 1, a DT system using a multi-PS architecture consists of two PSs (*i.e.*, S_1 and S_2), six workers (*i.e.*, W_1-W_6) and two programmable switches (*i.e.*, V_1 and V_2). For simplicity, the switch processing capacity, PS processing capacity and link bandwidth are set to 6, 4 and 6, respectively, and the unit is omitted. In the example, we first need to split the model, and each PS maintains a certain partition of the model, *i.e.*, *sub-model*. Then, we need to decide at what rate the gradients should be sent to the corresponding PS under various resource constraints. In this paper, we consider the typical *synchronous* scheme, *i.e.*, PSs will aggregate the gradient after receiving gradients from all required workers [42]. Usually, a faster gradient sending rate means a shorter communication time. Thus, our objective is to maximize the gradient sending rate of workers.

Let's first consider the Load Balance Min-Min (LBMM) algorithm [35], a classical algorithm used in multi-PS architecture without INA. LBMM selects the link with the lightest load for load balancing routing. As shown in Fig. 1(a), each worker sends half of the gradient to S_1 for aggregation and the other half to S_2 for aggregation. In addition, since the total processing capacity of the two PSs is 8 and there are six workers, the maximum gradient sending rate of each worker should be $4/3$ (*i.e.*, $2/3$ to S_1 and $2/3$ to S_2). Otherwise, PSs will be overloaded.

We then consider a recent work on INA, called ATP [37]. Specifically, ATP performs best-effort aggregation on the ToR switch for each worker under the corresponding rack, and the results are shown in Fig. 1(b). Since ATP does not involve the splitting of the model, here we assume the model is split equally, with each worker sending half of the gradient to S_1 and the other half to S_2. In this case, gradient fragments (*i.e.*, portions of gradients) [21, 29] from W_1, W_2 to S_1 are aggregated on switch V_1, fragments from W_3, W_4 to S_1 are directly routed to S_1 without INA, fragments from W_1, W_2, W_3, W_4 to S_2 are aggregated on switch V_1, fragments from W_5, W_6 to S_1 and $S2$ are aggregated on switch V_2. Therefore, the maximum gradient sending rate of ATP is 2.

2.2 Our Intuition

A question immediately following the above discussion is *can we do better by combining the merits of LBMM and ATP?* In Fig. 1(c), we demonstrate that by selecting an appropriate partitioning scheme among multiple PSs and implementing an optimal routing scheme



(a) LBMM (b) ATP (c) InArt

Figure 1: A DT system consists of two PSs (*i.e.*, S_1 and S_2), six workers (*i.e.*, W_1-W_6) and two programmable switches (*i.e.*, V_1 and V_2). The switch processing capacity, PS processing capacity and link bandwidth are set to 6, 4 and 6, respectively. We use colors to distinguish gradients sent to different PSs. Values near PSs and switches represent their loads, while those near workers denote maximum gradient sending rates. The left plot shows the gradient communication scheme using the LBMM method [35], and the maximum sending rate is $4/3$. The middle plot shows the gradient communication scheme by ATP [29] with a maximum sending rate of 2. The right plot shows that our proposed InArt can achieve a maximum sending rate of 2.5.

for INA, we can achieve a maximum gradient sending rate of 2.5. This rate is 87.5% faster than LBMM and 25% faster than ATP. In this case, gradient fragments from W_1, W_2, W_3, W_4 to S_1 are aggregated on switch V_1, fragments from W_5 and W_6 to S_1 are aggregated on switch V_2, gradient fragments from W_1, W_2, W_3 to S_2 are directly routed to S_2 without INA and fragments from W_4, W_5, W_6 to S_2 are aggregated on S_2. With our findings, *this paper aims to accelerate distributed training by designing an efficient route selection in a multi-PS architecture for INA, to maximize the gradient sending rate.*

3 PROBLEM DEFINITION

3.1 System Model

A typical multi-PS architecture mainly contains two components, worker set $W = \{w_1, ..., w_{|W|}\}$ and PS set $S = \{s_1, ..., s_{|S|}\}$, where $|W|$ and $|S|$ are the numbers of workers and PSs, respectively. According to the above definition, we model the DT cluster as $G = (W, S, V, E)$, where $V = \{v_1, ..., v_{|V|}\}$ is the set of programming switches (*e.g.*, Intel Tofino switches [5]), and $E = \{e_1, ..., e_{|E|}\}$ represents the communication links among these switches, workers and PSs. During the process of gradient aggregation, we regard gradient fragments with the same source (*i.e.*,worker) and aggregation location (*i.e.*, switch or PS) as a *flow* for simplicity. Let $P_{s,w}$ denote a set of feasible routing paths from worker w to PS s. Similarly, let $P_{v,w}$ and $P_{s,v}$ denote the feasible routing path set from switch v to worker w and from PS s to switch v, respectively. Moreover, we use $P = \{P_{s,w} \cup P_{v,w} \cup P_{s,v} \mid \forall w \in W, v \in V, s \in S\}$ to denote the all feasible routing path in the cluster G. For each switch $v \in V$, we use $C(v)$ to denote the total processing capacity, and $c(v)$ to denote the processing capacity used by background traffic. Moreover, for each PS $s \in S$, let $B(s)$ represent the total ingress bandwidth, and $b(s)$ represent the amount of ingress bandwidth already occupied. Similarly, let $B(e)$ and $b(e)$ denote the total bandwidth and the bandwidth used by background traffic for each link e, respectively. For simplicity, we focus on accelerating the training time of a single DT job. It is worth noting that our proposed scheme can be readily extended to accommodate scenarios involving multiple DT jobs (see Appendix B.1 for details). To ensure system stability, we adopt a synchronous approach [17] for model updating, where the PSs aggregate gradient fragments after receiving them from all required workers. In this context, a faster gradient sending rate f generally leads to a shorter training time. For ease of reference, Table 1 summarizes the key notations.

Notations	Semantics
W, w	The worker set, a worker in W
V, v	The programmable switch set, a switch in V
S, s	The PS set, a PS in S
E, e	The link set, a link in E
P, p	The routing path set, a path in P
$C(v)$, $c(v)$	Total, used processing capacity of v
$B(s)$, $b(s)$	Total, used bandwidth of s
$B(e)$, $b(e)$	Total, used ingress bandwidth of v
x_s	Proportion of model that s is responsible
$y^{s,w}$	Are gradient fragments from w to s aggregated by s
$y_v^{s,w}$	Are gradient fragments from w to s aggregated by v
$q_p^{s,w}$	Are gradient fragments from w to s routed on p
$q_p^{v,w}$	Are gradient fragments from w to v routed on p
$q_p^{s,v}$	Are gradient fragments from v to s routed on p
z_v^s	Are aggregated gradient fragments on v required sent to s
f	Gradient sending rate of workers

Table 1: Key Notations

3.2 Problem Definition of InArt

The key idea of InArt is to make the following three decisions.

- The proportion of the model aggregation that each PS is responsible for. Let variable x_s represent the proportion of the model that the PS s is responsible for aggregation.
- The location where each gradient is aggregated. Let variable $y^{s,w} \in \{0, 1\}$ denote whether gradient fragments from worker w to PS s are directly aggregated on the PS s or not. We use variable $y_v^{s,w} \in \{0, 1\}$ to represent whether fragments from worker w to PS s are aggregated by programmable switch v or not.
- The routing path for each gradient. Let binary variables $q_p^{s,w}$, $q_p^{v,w}$ and $q_p^{s,v}$ denote whether gradient fragments from worker w to PS s, from worker w to switch v and from switch v to PS s will be routed on path p or not, respectively.

We further consider the following six constraints when performing INA with route selection.

- *Model partition constraints:* We split the model into several sub-models and each PS is responsible for a sub-model. This means that each sub-model must have a corresponding PS for aggregation, represented as the equation $\sum_{s\in S} x_s = 1$.
- *INA constraints:* Considering the limited number and processing capacity of programmable switches in the cluster, similar to [29, 37], we assume that each gradient fragment will be aggregated in-network once at most to balance the problem complexity and the network performance, which is $y_v^{s,w} \le z_v^s, \forall s, w, v$.
- *Routing constraints:* Each gradient fragment must be routed from a worker to a PS for global aggregation through a feasible path. Specifically, if gradient fragments from worker w to PS s are directly aggregated by PS without INA, we have $\sum_{p\in P_{s,w}} q_p^{s,w} = y^{s,w}, \forall s, w$. If fragments from worker w to PS s are aggregated on the programmable switch v, we have $\sum_{p\in P_{v,w}} q_p^{v,w} = y_v^{s,w}, \forall s, w, v$, and $\sum_{p\in P_{s,v}} q_p^{s,v} = y_v^{s,w}, \forall s, w, v$.
- *Switch capacity constraints:* Each programmable switch can only aggregate gradient fragments at a limited rate due to switch processing capacity limitations. Therefore, we have $\sum_{s\in S} f\cdot x_s \cdot \sum_{w\in W} y_v^{s,w} + c(v) \le C(v), \forall v$.
- *Link capacity constraints:* For each link e, its traffic load should not exceed its bandwidth capacity $C(e)$. Thus, we have $\sum_{s\in S} f \cdot x_s \cdot \sum_{v\in V} \sum_{p\in P: e\in p} \left((q_p^{s,v} + \sum_{w\in W}(q_p^{v,w} + q_p^{s,w})\right) + b(e) \le B(e), \forall e$.
- *PS capacity constraints:* For each PS s, the forwarding rate can't exceed its ingress bandwidth $B(s)$. Let binary variable z_v^s indicate whether aggregated gradient fragments exist on switch v that need to be sent to PS s. Obviously, we have $y_v^{s,w} \le z_v^s, \forall s, w, v$. Note that two types of gradient fragments are routed to the PS for global aggregation: fragments forwarded directly by workers without network aggregation (*i.e.*, $y^{s,w} = 1$), and fragments aggregated by programmable switches (*i.e.*, $z_v^s = 1$). Accordingly, we have $f \cdot x_s \cdot \left(\sum_{w\in W} y^{s,w} + \sum_{v\in V} z_v^s\right) + b(s) \le B(s), \forall s$.

Furthermore, our objective is to maximize the gradient sending rate of workers f. Formally, we define the problem as Eq. (1).

$$
S.t.\begin{cases}
\max f & \\
\sum_{s\in S} x_s = 1, & \\
y^{s,w} + \sum_{v\in V} y_v^{s,w} = 1, & \forall s, w \\
\sum_{p\in P_{s,w}} q_p^{s,w} = y^{s,w}, & \forall s, w \\
\sum_{p\in P_{v,w}} q_p^{v,w} = y_v^{s,w}, & \forall s, w, v \\
\sum_{p\in P_{s,v}} q_p^{s,v} = y_v^{s,w}, & \forall s, w, v \\
\sum_{s\in S} f\cdot x_s \cdot \sum_{w\in W} y_v^{s,w} + c(v) \le C(v), & \forall v \\
\sum\limits_{s\in S} f x_s \sum\limits_{v\in V} \sum\limits_{p\in P: e\in p} ((q_p^{s,v} + \sum\limits_{w\in W}(q_p^{v,w} + q_p^{s,w})) + b(e) \le B(e), & \forall e \\
y_v^{s,w} \le z_v^s, & \forall s, w, v \\
f \cdot x_s \cdot (\sum_{w\in W} y^{s,w} + \sum_{v\in V} z_v^s) + b(s) \le B(s), & \forall s \\
x_s \in [0,1], & \forall s \\
y^{s,w}, y_v^{s,w} \in \{0,1\}, & \forall s, w, v \\
z_v^s \in \{0,1\}, & \forall s, v \\
q_p^{s,w}, q_p^{v,w}, q_p^{s,v} \in \{0,1\}, & \forall s, w, v, p \\
f \ge 0 &
\end{cases} \tag{1}
$$

The first equality in Eq. (1) represents the model partition constraints. The subsequent set of equalities denotes the INA constraints. Following that, the third to fifth sets of equalities describe the routing constraints. The sixth set of inequalities represents the switch capacity constraints. The seventh set of inequalities represents the link capacity constraints. Finally, the last two inequalities denote the PS capacity constraints.

However, it is difficult to directly solve the problem in Eq. (1). Note that the left side of the sixth set of inequalities in Eq. (1) contains the product of two continuous variables x_s, f and a binary variable $y_v^{s,w}$. In other words, InArt is a typically nonlinear mixed-integer programming (NMIP) problem, which is NP-hard [31]. Designing an algorithm for InArt is far from trivial and is in urgent need.

4 ALGORITHM DESIGN

4.1 Algorithm Workflow

In a cluster, where multiple DT jobs or applications are running simultaneously, the network traffic can undergo significant changes. Thus, in order to adapt to traffic uncertainty/dynamics, we should update routing paths and INA policy frequently. However, modifying the model scale on the PSs during training is not feasible due to consistency concerns.

To address this challenge, we propose a two-phase approach to solve the InArt problem. In the first phase, conducted at longer intervals such as several hours or a day, we divide the model among multiple PSs without considering route selection (§4.2). This simplifies InArt into a nonlinear programming problem, which we solve using the Lagrange multiplier method. In the second phase, triggered by events such as network congestion, we focus on the selection of routing paths for INA (§4.3). We maintain a fixed model partition ratio and transform InArt into an integer programming problem. To efficiently handle this, we design a randomized rounding-based algorithm for INA with route selection.

4.2 Algorithm Design for Splitting the Model

In the first phase, we mainly split the model among multiple PSs and determine the sub-model that each PS is responsible for, *i.e.*, get the value of variables $x_s \in [0,1]$. The procedure for this task is outlined in Algorithm 1. Specifically, at first, we focus on the switch capacity constraints and the PS capacity constraints in Eq. (1). Then, we relax the variables $y^{s,w}$, $y_v^{s,w}$ and z_v^s from integer to fractional. The problem in Eq. (1) converts to nonlinear programming as Eq. (2). Note that variables $y^{s,w}$, $y_v^{s,w}$ and z_v^s are integral in Eq. (1), but fractional in Eq. (2). Since Eq. (2) is a nonlinear programming, we design a generalized Lagrange multiplier (LM) based algorithm [30], called L-InArt, to get the value of variables x_s.

$$
S.t.\begin{cases}
\max f & \\
\sum_{s\in S} x_s = 1, & \\
y^{s,w} + \sum_{v\in V} y_v^{s,w} = 1, & \forall s, w \\
\sum\limits_{s\in S} f\cdot x_s \cdot \sum_{w\in W} y_v^{s,w} \le C(v) - c(v), & \forall v \\
y_v^{s,w} \le z_v^s, & \forall s, w, v \\
f \cdot x_s \cdot (\sum_{w\in W} y^{s,w} + \sum_{v\in V} z_v^s) \le B(s) - b(s), & \forall s \\
x_s \in [0,1], & \forall s \\
y^{s,w}, y_v^{s,w} \in [0,1], & \forall s, w, v \\
z_v^s \in [0,1], & \forall s, v \\
f \ge 0 &
\end{cases} \tag{2}
$$

Let symbol X represents all variables in Eq. (2), *i.e.*, $X = \{x_s, y^{s,w}, y_v^{s,w}, z_v^s\}$. We consider the Lagrange function $\mathcal{L}(X)$ of Eq. (2) as follows:

$$\begin{aligned}\mathcal{L}(X) = &\sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{w\in\mathcal{W}}\lambda_{s,w}h_{s,w}(X) + \alpha w(X) + \sum\nolimits_{v\in\mathcal{V}}\rho_v p_v(X)\\ &+ \sum\nolimits_{s\in\mathcal{S}}\theta_s q_s(X) + \sum\nolimits_{s\in\mathcal{S}}\sum\nolimits_{w\in\mathcal{W}}\sum\nolimits_{v\in\mathcal{V}}\sigma_{v,s,w}r_{v,s,w}(X)\\ &- f - \tau_s x_s - \beta_{s,w}y_{s,w} - \delta_{s,v,w}y_{s,v,w} - \zeta_{s,v}z_{s,v} + \eta_s(x_s - 1)\\ &+ \mu_{s,w}(y_{s,w} - 1) + \omega_{s,v,w}(y_{s,v,w} - 1) + \gamma_{s,v}(z_{s,v} - 1) \qquad (3)\end{aligned}$$

Greek variables in Eq. (3) represent the Lagrange multiplier corresponding to the constraints in Eq. (2). For example, the variable α corresponds to the first set of constraints of x_s in Eq. (2). Meanwhile, these variables should be non-negative. In addition, the functions $h_{s,w}(X)$, $w(X)$, $p_v(X)$, $q_s(X)$, and $r_{v,s,w}(X)$ denote the first set to the fifth set of constraints in Eq. (2), respectively. The definition of these functions is as follows:

$$\begin{cases} h_{s,w}(X) = y^{s,w} + \sum_{v\in V} y_v^{s,w} - 1, & \forall s, w\\ w(X) = 1 - \sum_{s\in\mathcal{S}} x_s, & \\ p_v(X) = \sum_{s\in S} f\cdot x_s \cdot \sum_{w\in W} y_v^{s,w} - (C(v) - c(v)) & \forall v\\ q_s(X) = f\cdot x_s\cdot(\sum\limits_{w\in W} y^{s,w} + \sum\limits_{v\in V} z_v^s) - (B(s) - b(s)), & \forall s\\ r_{v,s,w}(X) = y_v^{s,w} - z_v^s, & \forall v, s, w \end{cases} \qquad (4)$$

To determine the extreme point, we utilize the Karush-Kuhn-Tucker (KKT) conditions [22, 30] and find the partial derivative, which yields a set of equations for x_s. Solving these equations through Gaussian elimination [23] provides us with the values of x_s. For a more comprehensive understanding, the reader can refer to [30, 39]. Once we have obtained the calculated values of x_s, we split the DT model among multiple servers accordingly. Notably, a detailed description of the model splitting process can be found in Appendix B.2 due to space limitations.

Algorithm 1: L-InArt: LM-Based Algorithm for InArt

Step 1: Relaxing the InArt problem
Focus on the switch and PS capacity constraints in Eq. (1)
Relax $y^{s,w}$, $y_v^{s,w}$, and z_v^s from integer to fractional
Construct a nonlinear programming in Eq. (2)
Step 2: Deriving the extreme point of x_s
Give the Lagrange function $\mathcal{L}(X)$ in Eq. (3)
Obtain an equation set for x_s by take partial derivative to the Lagrange function $\mathcal{L}(X)$ of Eq. (3) and Eq. (4)
Solve these equations and split the model among multiple PSs based on the value of x_s

4.3 Algorithm Design for INA and Routing

The second phase of InArt gives the INA and routing schemes. Since the value of x_s is solved in the first phase, we introduce the result into Eq. (1), and simplify InArt into an integer linear programming problem, which is challenging to solve in a polynomial time. Accordingly, in this section, we propose a randomized rounding-based algorithm for the second phase, called R-InArt. The R-InArt algorithm is formally described in Algorithm 2. In the first step of R-InArt, we construct linear programming as relaxation of Eq. (1). Specifically, InArt assumes that each gradient fragment will be routed on a feasible path and aggregated on at most one switch. By relaxing these assumptions, each gradient fragment is splittable, can be routed through several feasible paths and aggregated by multiple switches. We formulate the linear programming LP-InArt as follows:

Algorithm 2: R-InArt: RR-Based Algorithm for InArt

Step 1: Solving the relaxed problem of Eq. (1)
Construct the linear programming LP-InArt in Eq. (5)
Derive the optimal solution $\widetilde{y}^{s,w}$, $\widetilde{y}_v^{s,w}$, $\widetilde{z}^{s,v}$, $\widetilde{q}_p^{s,w}$, $\widetilde{q}_p^{v,w}$, $\widetilde{q}_p^{s,v}$
Step 2: Selecting Routing Path
Obtain an integer solution $\widehat{y}^{s,w}$ and $\widehat{y}_v^{s,w}$ by RR
for *each PS* $s \in S$ **do**
  for *each worker* $w \in W$ **do**
    if $\widehat{y}^{s,w} == 1$ **then**
      Obtain an integral solution $\widehat{q}_p^{s,w}$ by RR
      for *each path* $p \in P_{s,w}$ **do**
        if $\widehat{q}_p^{s,w} == 1$ **then**
          for *each switch* v *along path* p **do**
            Install a flow entry on switch v
    for *each switch* $v \in V$ **do**
      if $\widehat{y}_v^{s,w} == 1$ **then**
        Set the value of z_v^s to 1
        Install a INA rule on switch v for the flow from worker w to PS s
        Obtain an integral solution $\widehat{q}_p^{v,w}$ by RR
        for *each path* $p \in P_{v,w}$ **do**
          if $\widehat{q}_p^{v,w} == 1$ **then**
            for *each* v *along path* p **do**
              Install a flow entry on v
  for *each switch* $v \in V$ **do**
    Obtain an integral solution $\widehat{q}_p^{s,v}$ by RR
    for *each path* $p \in P_{s,v}$ **do**
      if $\widehat{q}_p^{s,v} == 1$ **then**
        for *each switch* v *along path* p **do**
          Install a flow entry on switch v

$$S.t.\begin{cases} \max f & \\ y^{s,w} + \sum\limits_{v\in V} y_v^{s,w} = 1, & \forall s, w\\ \sum\limits_{p\in P_{s,w}} q_p^{s,w} = y^{s,w}, & \forall s, w\\ \sum\limits_{p\in P_{v,w}} q_p^{v,w} = y_v^{s,w}, & \forall s, w, v\\ \sum\limits_{p\in P_{s,v}} q_p^{s,v} = y_v^{s,w}, & \forall s, w, v\\ \sum\limits_{s\in S} f\cdot x_s \cdot \sum\limits_{w\in W} y_v^{s,w} + c(v) \le C(v), & \forall v\\ \sum\limits_{s\in S} f x_s \sum\limits_{v\in V}\sum\limits_{p\in P: e\in p}((q_p^{s,v} + \sum\limits_{w\in W}(q_p^{v,w} + q_p^{s,w})) + b(e) \le B(e), & \forall e\\ y_v^{s,w} \le z_v^s, & \forall s, w, v\\ f\cdot x_s\cdot(\sum\limits_{w\in W} y^{s,w} + \sum\limits_{v\in V} z_v^s) + b(s) \le B(s), & \forall s\\ y^{s,w}, y_v^{s,w} \in [0,1], & \forall s, w, v\\ z_v^s \in [0,1], & \forall s, v\\ q_p^{s,w}, q_p^{v,w}, q_p^{s,v} \in [0,1], & \forall s, w, v, p\\ f \ge 0 & \end{cases} \qquad (5)$$

Note that variables $y^{s,w}$, $y_v^{s,w}$, z_v^s, $q_p^{s,w}$, $q_p^{v,w}$, and $q_p^{s,v}$ are integer in Eq. (1), but fractional in Eq. (5). Eq. (5) is a linear programming

problem, we can use a linear programming solver (*e.g.*, Cplex [2]) to solve it in polynomial time. Assume that the optimal solution for Eq. (5) is denoted as $\{\widetilde{y}^{s,w}, \widetilde{y}_v^{s,w}, \widetilde{z}^{s,v}, \widetilde{q}_p^{s,w}, \widetilde{q}_p^{v,w}, \widetilde{q}_p^{s,v}\}$, and the optimal result is denoted as $\widetilde{f}$. Since Eq. (5) is a relaxation of Eq. (1), $\widetilde{f}$ is the upper-bound for Eq. (1).

In the second step of R-InArt, we give the INA scheme and routing path. At first, using the randomized rounding (RR) method [45], we derive the integral solution $\{\widehat{y}^{s,w}, \widehat{y}_v^{s,w}\}$, for $\forall s \in S$, $\forall w \in W$, and $v \in V$. Specifically, if $\widehat{y}^{s,w} = 1$, it means that gradient fragments from worker w to PS s will not be aggregated by any switches, but will be aggregated on PS s. If $\widehat{y}_v^{s,w} = 1$, it means that the fragments from worker w to PS s will be aggregated on switch v. Moreover, we will set the value of z_v^s as 1. Next, we give the routing path q of each gradient fragment from worker w to PS s. Then we derive the integral solution by RR, denoted as $\{\widehat{q}_p^{s,w}, \widehat{q}_p^{v,w}, \widehat{q}_p^{s,v}\}$. Note that each gradient fragment will be aggregated in at most one switch for INA, and be assigned one feasible path for routing by InArt. Owing to space limitations, we have included specific RR details in Appendix B.3.

5 EVALUATION

5.1 Performance Metrics and Benchmarks

5.1.1 Performance Metrics. We adopt the following eight performance metrics to evaluate the improvement of our proposed InArt for DT: (1) the gradient sending rate of workers; (2) the training throughput; (3) the per-iteration time; (4) the communication time; (5) the training speed; (6) the accuracy over training time; (7) the network throughput; (8) the ingress traffic amount of PSs.

During a testbed run, we use iftop [3] to monitor the egress bandwidth of each worker as *the gradient sending rate*. We measure the number of processed samples (*e.g.*, images) per second as *the training throughput*. In addition, we record the time between two consecutive iterations as *the per-iteration time*. In each iteration, we measure the duration from a worker sending gradient fragments to receiving the updated model as *the communication time* of one iteration. Furthermore, we record the number of iterations over a period of time as *the training speed* and record *the accuracy of each iteration*. During an emulation run, we measure *the gradient sending rate* and *the communication time*. In each iteration of the emulation experiment, we calculate the traffic volume of gradient fragment transferred by all the links as *the network throughput*. In addition, we measure the total traffic volume of gradient fragments from the workers and programmable switches to PSs per iteration, as *the ingress traffic amount of PSs*.

5.1.2 Benchmarks. We choose three benchmarks for performance comparison. The first benchmark splits the model in the same proportion among multiple PSs (*e.g.*, a DT architecture contains four PSs, each maintaining 25% of the total model), and then performs R-InArt for gradient route selection. The second benchmark is the Load Balance Min-Min scheduling (LBMM) algorithm [35]. LBMM is an efficient routing algorithm that does not consider INA in the cluster. For gradient fragments from workers to PSs, LBMM chooses the routing path with the most negligible impact on routing load balancing. The third benchmark, called ATP [37], is a state-of-the-art INA method. In ATP, gradient fragments are aggregated on ToR programmable switches in the cluster. Then the aggregated traffic will be routed to the PSs from the ToR switches with the least link load. Note that ATP does not involve model splitting, and for fair comparison, we assume that the model is split equally across PSs in the following evaluations.

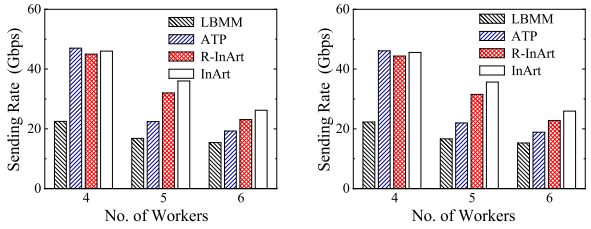
5.2 Testbed Evaluation

5.2.1 Testbed Settings. We use eight servers running Ubuntu 18.04 (Linux kernel version 5.4) and two Wedge100BF-32x programmable switches with Intel Tofino chip [5] to build the testbed. The topology of the testbed is the same as that of the example (Fig. 1) in §2.1. Specifically, all servers have a 22-core Intel Xeon 6152 processor, 128GB RAM, and an NVIDIA GeForce RTX 3090. Each server is equipped with a Mellanox ConnectX-6 dual-port 100Gbps NIC. Besides, all the servers are connected with programmable switches via 100Gbps links.

In terms of implementation details, similar to [29], we run PyTorch on each worker to carry out DT jobs. To implement INA on the switch, we write the P4 program in P4-16 with Tofino Native Architecture (TNA) [7]. More specifically, we pre-calculate our solution's model splitting and routing scheme with Pyomo [8] and install the corresponding entries to the programmable switches using the Barefoot Runtime Interface (BRI). We train two popular models on the Cifar-10 dataset [25]: ResNet50 [12] with a size of 97MB and VGG19 [38] with a size of 548MB. Besides, the batch size is set as 32 for all training jobs. We run each testbed 10 times and calculate the average value as the results.

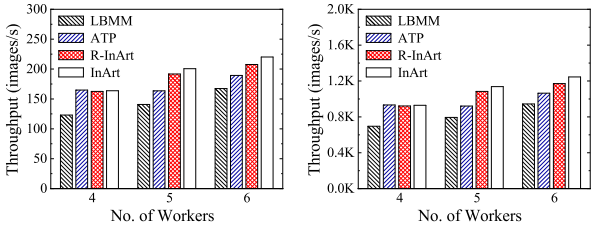
5.2.2 Testbed Results. We run three sets of experiments to evaluate the performance of InArt and benchmarks. In the first set of experiments, we observe the gradient sending rate of workers and the training throughput, as shown in Figs. 2-3. It is evident that InArt can achieve the best performance among all solutions. Fig. 2 shows that as the number of workers increases, InArt always obtains the highest gradient sending rate. For example, given 6 workers in VGG19, the gradient sending rate of InArt, R-InArt, ATP and LBMM are 26.2Gbps, 23.1Gbps, 19.25Gbps and 15.4Gbps, respectively. It means that InArt can increase the gradient sending rates by 13.4%, 36.5% and 72%, compared with R-InArt, ATP and LBMM, respectively. In Fig. 3, InArt consistently achieves the highest training throughput with increasing numbers of workers. Specifically, with 6 workers in VGG19, InArt achieves a throughput of 223 images/s. Comparatively, R-InArt, ATP, and LBMM achieve throughputs of 206 images/s, 188 images/s, and 162 images/s, respectively. In other words, InArt can improve the training throughput by 8.3%, 18.6% and 37.7% compared with R-InArt, ATP and LBMM, respectively. The reason is that InArt selects a proper routing path under the INA framework and designs an appropriate model splitting scheme to maximize the gradient sending rate of workers.

The second set of experiments measures the total time and the communication time of one iteration. Fig. 4 shows the per-iteration time with different numbers of workers. Note that per-iteration time consists of the local training time and the communication time. Our method doesn't optimize the local training time but can co-exist with solutions decreasing local training time if needed. We observe that the per-iteration time increases as the number of workers increases, while InArt always obtains the least per-iteration



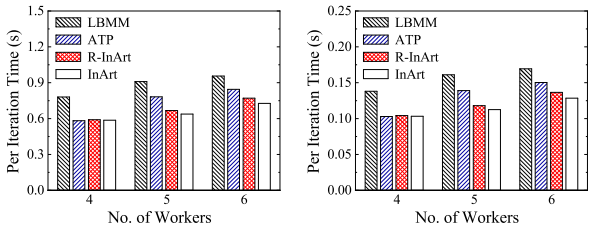
(a) VGG19 (b) ResNet-50

Figure 2: Gradient Sending Rate vs. No. of Workers



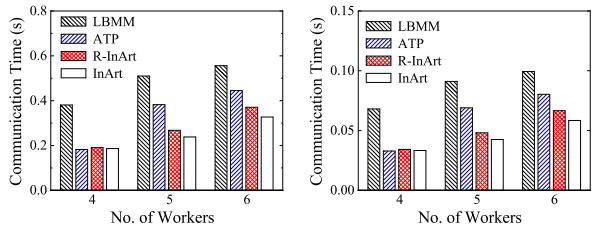
(a) VGG19 (b) ResNet-50

Figure 3: Training Throughput vs. No. of Workers



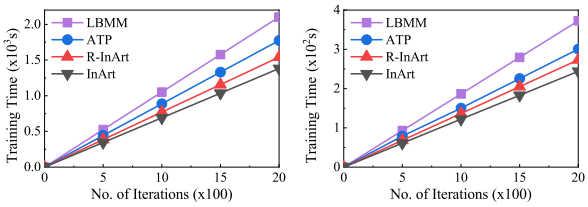
(a) VGG19 (b) ResNet-50

Figure 4: Per Iteration Time vs. No. of Workers



(a) VGG19 (b) ResNet-50

Figure 5: Communication Time vs. No. of Workers



(a) VGG19 (b) ResNet-50

Figure 6: Training Time vs. No. of Iterations

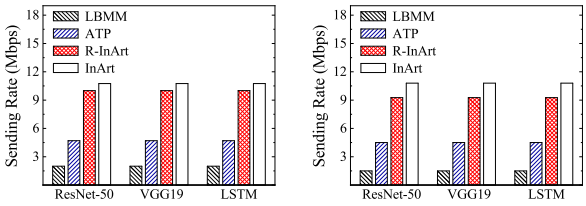
time. For example, when the number of workers is 6 in VGG19, the per-iteration times of LBMM, ATP, R-InArt and InArt are 0.97s, 0.85s, 0.78s and 0.69s, respectively. That means, InArt reduces the per-iteration time by 29%, 19% and 12% compared with LBMM, ATP and R-InArt, respectively. As shown in Fig. 5, InArt always has the shortest communication time in each iteration. Given 6 workers in VGG19, the communication time of LBMM, ATP, R-InArt and InArt are 0.56s, 0.45s, 0.38s and 0.31s, respectively. InArt decreases the communication time by 45%, 32% and 19%, compared with LBMM, ATP and R-InArt, respectively. The reason is that InArt has the highest gradient sending rate of workers (as described in Fig. 2), thereby reducing the communication time.

Finally, we run two DT jobs with 6 workers to evaluate the performance of training time. It can be observed from Fig. 6 that InArt takes the least time to complete the DT job. For example, it takes the 1380s for InArt to complete 2000 iterations of the VGG19 training job, while the numbers are 1541s, 1775s and 2105s when we use R-InArt, ATP and LBMM, respectively. In addition, we assess the accuracy of each benchmark over training time, and due to space limitations, we summarize the results, with detailed findings available in Appendix A.1. InArt can reach the target accuracy 1.2×, 1.38× and 1.84× faster than R-InArt, ATP and LBMM, respectively.

5.3 Emulation Evaluation

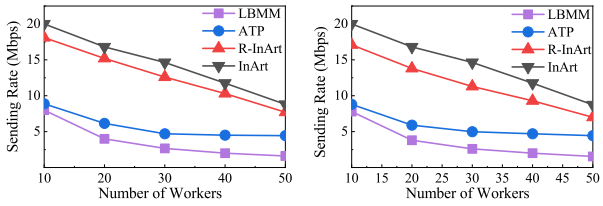
5.3.1 Emulation Settings. We implement a middle-scale emulation with the classical fat-tree topology [19], which is commonly adopted in clusters. We use the Mininet tool [4] to implement the fat-tree topology, which consists of 9 core switches, 18 aggregation switches, 18 ToR switches, and 54 servers. By default, we randomly selected 4 servers as PSs and the remaining servers as workers. Since we cannot support P4 hardware switches of a certain scale, we obtain results using bmv2 [1] software switches. Unfortunately, bmv2 software switches are not designed for line-rate packet processing [36]. Therefore, we cannot inject Gbps traffic into bmv2 switches for our evaluations, and shrink the experimental setup by a factor of 1000. Furthermore, These evaluations are performed under two common network scenarios. The first is a homogeneous scenario, in which the capacity of each link is 20Mbps. The second is a heterogeneous scenario, and the link capacity is randomly generated between 10Mbps and 30Mbps. We set the processing capacity of PSs and aggregation capacity of bmv2 switches as 20Mbps and 9Mbps, respectively. To implement the INA and routing, we pre-program the INA logic of bmv2 switches and pre-install the flow table by P4 language. The emulation tests three different models, LSTM, VGG19, and ResNet-50. Specifically, the LSTM model is commonly used for time series prediction, and VGG19 and ResNet-50 are widely used for image classification. We set the gradient size in one iteration of LSTM, VGG19 and ResNet-50 by a factor of 1000 to 1627KB, 548KB, and 97KB, respectively [37]. To emulate the synchronous gradient communications, we deploy tcpreplay [9] tools on each worker to send packets at the same rate. We run each emulation 10 times and calculate the average value as the result.

5.3.2 Emulation Results. We run three sets of experiments for performance evaluations. The first set of experiments focused on comparing the gradient sending rate of workers. On the one hand, we observe the sending rate by varying the number of workers, as shown in Figs. 7-8. From the left plot of Fig. 7, when training LSTM jobs, the sending rate of workers is 10.75Mbps and 4.7Mbps by InArt



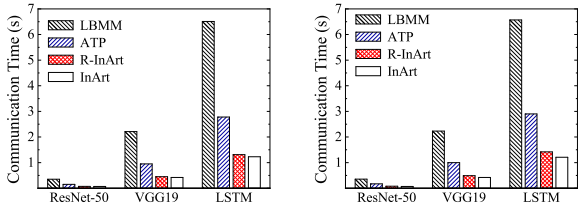
(a) Homogeneous Network (b) Heterogeneous Network

Figure 7: Gradient Sending Rate in Different Models



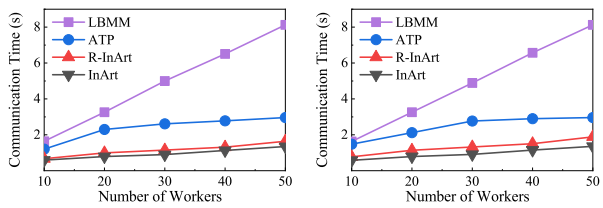
(a) Homogeneous Network (b) Heterogeneous Network

Figure 8: Gradient Sending Rate vs. No. of Workers



(a) Homogeneous Network (b) Heterogeneous Network

Figure 9: Gradient Communication Time in Different Models



(a) Homogeneous Network (b) Heterogeneous Network

Figure 10: Gradient Communication Time vs. No. of Workers

and ATP, respectively. In Fig. 8, as the number of workers increases, the gradient sending rate will gradually decrease since more workers can use more network resources in the cluster. Inspiringly, our solution can achieve a faster-sending rate than other benchmarks. From the right plot of Fig. 8, when there are 50 workers in the network, the sending rate is 8.73Mbps and 4.44Mbps by InArt and ATP, respectively. It means our solution improved the sending rate by 97% compared with ATP. On the other hand, we examine the sending rate by varying the number of PSs. Due to space limitations, we provide a summary of the results here, and detailed findings can be found in Appendix A.2. InArt can achieve 2.37× improvements in sending rate compared to ATP. This is because InArt sensibly distributes DT tasks among PSs and strategically selects routing schemes to utilize the capabilities of INA effectively.

The second set of experiments observes the communication time of one iteration. In Fig. 9, we first observe the gradient communication time of three models. As the gradient size increases, the communication time will become longer. The communication time by InArt is much slower than that of ATP and LBMM. From the left plot in Fig. 9, in the homogeneous scenario, the gradient communication time of Resnet-50 is 0.068s, 0.152s, and 0.35s by InArt, ATP, and LBMM, respectively. Similarly, our solution performs better than other benchmarks in the heterogeneous scenario. In Fig. 10, we observe the impact of the number of workers on communication time. As the number of workers increases, the communication time accordingly increases. However, the increasing rate of InArt is much slower than that of ATP and LBMM. For example, in the heterogeneous scenario, compared to ATP, considering scenarios with 10, 20, 30, 40, and 50 workers, InArt reduces the communication time by 48%, 54%, 57%, 53% and 49%, respectively. That is because a faster sending rate can effectively reduce communication time.

Our third set of experiments measures the network throughput and the ingress traffic amount of PSs per iteration. Due to space constraints, we present a summary of the results, while more detailed results can be found in Appendix A.2. InArt significantly improves the network throughput compared to state-of-the-art INA works, achieving approximately 1.6× higher throughput. Additionally, our approach reduces the load on PSs by 53%. These improvements are attributed to InArt's utilization of a combined INA scheme that incorporates submodel partitioning and route selection.

6 CONCLUSION AND FUTURE WORK

In this paper, we design and implement InArt, the first-of-its-kind work on INA with route selection in a multi-PS architecture, to accelerate distributed training. InArt utilizes a multi-PS architecture to distribute DT tasks among multiple PSs and effectively selects routing schemes to fully leverage the capabilities of INA. Due to traffic dynamics, InArt takes a two-phase approach: splitting the training model among multiple PSs and selecting the routing paths for INA. Two algorithms have been designed for these two phases, respectively. Experiment results show that InArt can achieve a superior gradient sending rate and less communication time than the state-of-the-art solutions. In the future, we intend to explore the application of INA with route selection in asynchronous distributed training with a multi-PS architecture.

ACKNOWLEDGMENTS

The corresponding authors of this paper are Gongming Zhao and Hongli Xu. This research received partial support from several funding sources, including the National Science Foundation of China (NSFC) under Grants 62372426, 62132019, and 62102392; the Open Research Projects of Zhejiang Lab under Grant 2022QA0AB04; the National Science Foundation of Jiangsu Province under Grant BK20210121; the Fundamental Research Funds for the Central Universities; and the Youth Innovation Promotion Association of the Chinese Academy of Science (2023481).

REFERENCES

[1] [n. d.]. Behavioral model version 2 (bmv2). https://github.com/p4lang/behavioral-model Accessed: June. 14, 2023.

[2] [n. d.]. IBM ILOG CPLEX Optimization Studio. https://nl.mathworks.com/products/connections/product_detail/Ibm-ilog-cplex.html.

[3] [n. d.]. iftop. http://www.ex-parrot.com/~pdw/iftop/ Accessed: June. 14, 2023.

[4] [n. d.]. An instant virtual network on your laptop. http://Mininet.org Accessed: June. 14, 2023.

[5] [n. d.]. Intel Tofino. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html Accessed: June. 14, 2023.

[6] [n. d.]. Netronome Agilio SmartNIC. https://www.netronome.com/products/agilio-cx.

[7] [n. d.]. Open Tofino. https://github.com/barefootnetworks/Open-Tofino. Accessed: June. 14, 2023.

[8] [n. d.]. Pyomo. https://github.com/Pyomo/pyomo.

[9] [n. d.]. Tcpreplay - Pcap editing and replaying utilities. https://tcpreplay.appneta.com. Accessed: June. 14, 2023.

[10] Lusine Abrahamyan, Yiming Chen, Giannis Bekoulis, and Nikos Deligiannis. 2021. Learned gradient compression for distributed deep learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[11] Saurabh Agarwal, Hongyi Wang, Shivaram Venkataraman, and Dimitris Papailiopoulos. 2021. On the utility of gradient compression in distributed training systems. *arXiv preprint arXiv:2103.00543* (2021).

[12] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review* 38, 4 (2008), 63–74.

[13] Rand Jawad Kadhim Almahmood and Adem Tekerek. 2022. Issues and Solutions in Deep Learning-Enabled Recommendation Systems within the E-Commerce Field. *Applied Sciences* 12, 21 (2022), 11256.

[14] Saravanan Chandrasekaran, Aditya Kumar Singh Pundir, T Bheema Lingaiah, et al. 2022. Deep learning approaches for cyberbullying detection and classification on social media. *Computational Intelligence and Neuroscience* 2022 (2022).

[15] Chia-Yu Chen, Jiamin Ni, Songtao Lu, Xiaodong Cui, Pin-Yu Chen, Xiao Sun, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Wei Zhang, et al. 2020. Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training. *Advances in Neural Information Processing Systems* 33 (2020), 13551–13563.

[16] Ge Chen, Gaoxiong Zeng, and Li Chen. 2021. P4COM: In-Network Computation with Programmable Switches. *arXiv preprint arXiv:2107.13694* (2021).

[17] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[19] Raj P Dhanya and VS Anitha. 2023. Implementation and Performance Evaluation of Load Balanced Routing in SDN based Fat Tree Data Center. In *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*. IEEE, 1–6.

[20] Jin Fang, Gongming Zhao, Hongli Xu, Changbo Wu, and Zhuolong Yu. 2023. GRID: Gradient routing with in-network aggregation for distributed training. *IEEE/ACM Transactions on Networking* (2023).

[21] Jin Fang, Gongming Zhao, Hongli Xu, Zhuolong Yu, Bingchen Shen, and Liguang Xie. 2023. GOAT: Gradient Scheduling with Collaborative In-Network Aggregation for Distributed Training. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS57198.2023.10188783

[22] Geoff Gordon and Ryan Tibshirani. 2012. Karush-kuhn-tucker conditions. *Optimization* 10, 725/36 (2012), 725.

[23] Joseph F Grcar. 2011. Mathematicians of Gaussian elimination. *Notices of the AMS* 58, 6 (2011), 782–792.

[24] Sayed Hadi Hashemi, Sangeetha Abdu Jyothi, and Roy Campbell. 2019. Tictac: Accelerating distributed deep learning with communication scheduling. *Proceedings of Machine Learning and Systems* 1 (2019), 418–430.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[26] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. 2018. Flexps: Flexible parallelism control in parameter server architecture. *Proceedings of the VLDB Endowment* 11, 5 (2018), 566–579.

[27] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. 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)*. 463–479.

[28] Minkoo Kang, Gyeongsik Yang, Yeonho Yoo, and Chuck Yoo. 2020. TensorExpress: In-network communication scheduling for distributed deep learning. In *2020 IEEE 13th international conference on cloud computing (CLOUD)*. IEEE, 25–27.

[29] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning.. In *NSDI*. 741–761.

[30] Mengmou Li. 2018. Generalized Lagrange multiplier method and KKT conditions with an application to distributed optimization. *IEEE Transactions on Circuits and Systems II: Express Briefs* 66, 2 (2018), 252–256.

[31] Jesus Lopez-Perez. 2021. Elasticities on a Mixed Integer Programming Model for Revenue Optimization. In *XX SIGEF Congress-Harnessing Complexity through Fuzzy Logic*. Springer, 153–177.

[32] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2018. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In *Proceedings of the ACM Symposium on Cloud Computing*. 41–54.

[33] Yun Ma, Dongwei Xiang, Shuyu Zheng, Deyu Tian, and Xuanzhe Liu. 2019. Moving deep learning into web browser: How far can we go?. In *The World Wide Web Conference*. 1234–1244.

[34] Luo Mai, Guo Li, Marcel Wagenländer, Konstantinos Fertakis, Andrei-Octavian Brabete, and Peter Pietzuch. 2020. {KungFu}: Making Training in Distributed Machine Learning Adaptive. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 937–954.

[35] Shyam Singh Rajput and Virendra Singh Kushwah. 2016. A genetic based improved load balanced min-min task scheduling algorithm for load balancing in cloud computing. In *2016 8th international conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 677–681.

[36] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 150–156.

[37] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan Ports, and Peter Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 785–808.

[38] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[39] Ankur Sinha, Tharo Soun, and Kalyanmoy Deb. 2019. Using Karush-Kuhn-Tucker proximity measure for solving bilevel optimization problems. *Swarm and evolutionary computation* 44 (2019), 496–510.

[40] Suraiya Tairin, Haiying Shen, and Zeyu Zhang. 2023. Embracing Uncertainty for Equity in Resource Allocation in ML Training. In *Proceedings of the 52nd International Conference on Parallel Processing*. 423–432.

[41] Ching-Yuan Tsai, Ching-Chi Lin, Pangfeng Liu, and Jan-Jan Wu. 2018. Communication scheduling optimization for distributed deep learning systems. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 739–746.

[42] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. 2020. A survey on distributed machine learning. *ACM Computing Surveys (CSUR)* 53, 2 (2020), 1–33.

[43] Yanwu Yang and Panyu Zhai. 2022. Click-through rate prediction in online advertising: A literature review. *Information Processing & Management* 59, 2 (2022), 102853.

[44] Xiao Zeng, Ming Yan, and Mi Zhang. 2021. Mercury: Efficient on-device distributed dnn training via stochastic importance sampling. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. 29–41.

[45] Gongming Zhao, Jiawei Liu, Yutong Zhai, Hongli Xu, and Huang He. 2023. Alleviating the Impact of Abnormal Events Through Multi-Constrained VM Placement. *IEEE Transactions on Parallel and Distributed Systems* 34, 5 (2023), 1508–1523. https://doi.org/10.1109/TPDS.2023.3248681

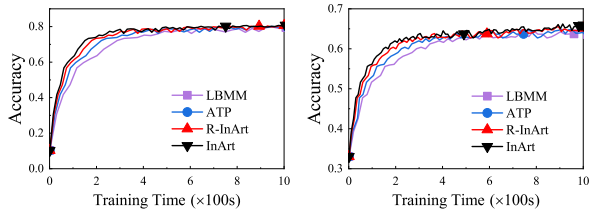
[46] Ruiting Zhou, Jinlong Pang, Qin Zhang, Chuan Wu, Lei Jiao, Yi Zhong, and Zongpeng Li. 2022. Online Scheduling Algorithm for Heterogeneous Distributed Machine Learning Jobs. *IEEE Transactions on Cloud Computing* (2022).

A ADDITIONAL EVALUATION DETAILS

A.1 Additional Testbed Evaluation

The experimental setup in this section is the same as the one described in §5.2.

A.1.1 Accuracy. We conduct two DT jobs with 6 workers to evaluate the performance of Accuracy over training time, as depicted in Fig. 11. It is evident that InArt achieves the specified accuracy in the shortest amount of time. For example, when the model is VGG19, InArt achieves an accuracy of 0.7214 in 151s, while R-InArt, ATP, and LBMM require 181s, 208s, and 278s, respectively. This illustrates that InArt reaches the target accuracy 1.2×, 1.38× and 1.84× faster than R-InArt, ATP, and LBMM, respectively. The results demonstrate that proper gradient routing with INA significantly speeds up distributed model training.



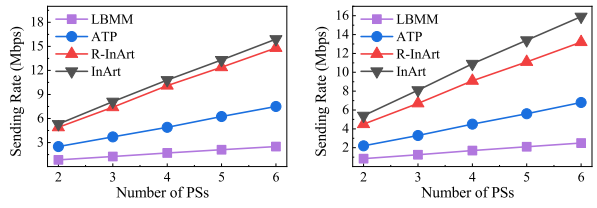
(a) VGG19 (b) ResNet-50

Figure 11: Accuracy over Training Time

A.2 Additional Emulation Evaluation

The experimental setup in this section is the same as the one described in §5.3.

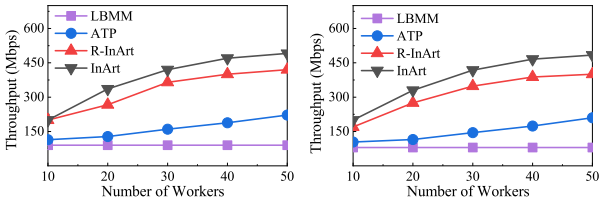
A.2.1 Gradient Sending Rate when No. of PS varies. In Fig. 12, we select 48 servers as workers and vary the number of PSs from 2 to 6 to observe the gradient sending rate during LSTM training. The results show high efficiency of our proposed algorithm, especially in the heterogeneous scenarios. Specifically, the sending rate gradually increases as the number of PSs increases. On average, in the heterogeneous network, InArt achieves a sending rate of 10.68Mbps. In comparison, the sending rates of R-InArt, ATP, and LBMM are 8.85Mbps, 4.5Mbps, and 1.68Mbps, respectively. This means that InArt can achieve 1.2×, 2.37×, and 6.34× improvements in sending rate compared to R-InArt, ATP, and LBMM, respectively, in heterogeneous networks.



(a) Homogeneous Network (b) Heterogeneous Network

Figure 12: Gradient Sending Rate vs. No. of PSs

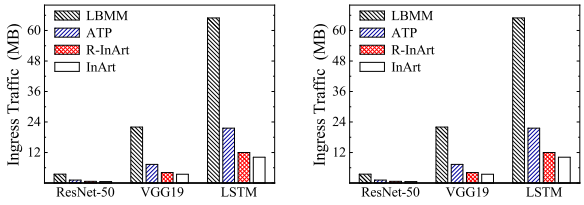
A.2.2 Network Throughput. As shown in Fig. 13, the network throughput gradually increases with the increasing number of workers, while our solution has the highest throughput. For example, when there are 30 workers in the left plot of Fig. 13, InArt, ATP, and LBMM achieves a network throughput of 420, 160, and 90 Mbps, respectively. This suggests that our INA and dynamic routing approach utilizes network resources more efficiently. Compared with ATP, InArt improves the network throughput by about 1.6×.



(a) Homogeneous Network (b) Heterogeneous Network

Figure 13: Network Throughput vs. No. of Workers

A.2.3 Ingress Traffic amount of PS. In Fig. 14, we indicate the ingress traffic amount of PSs. As expected that the ingress traffic amount of PSs using LBMM is much higher than that of other benchmarks. This is because the LBMM does not consider INA, and all the gradient fragments will be aggregated on PSs. Note that InArt significantly reduces the processing load on PSs. For example, when training VGG19, the ingress traffic amount of PSs is 3.44MB, 7.33MB, and 22MB by InArt, ATP, and LBMM, respectively. Compared with ATP, our method reduces the load on PSs by 53%.



(a) Homogeneous Network (b) Heterogeneous Network

Figure 14: Ingress Traffic amount of PSs in Different Models.

B DISCUSSION

B.1 Multi-DT Job Scenarios

This paper primarily focuses on minimizing the training time of a single DT job. Notably, our proposed method can be readily extended to handle multi-DT jobs, leveraging the inherent independence among different DT tasks. In particular, when processing a specific DT task, the others can be treated as background traffic [29, 46]. By sequentially processing each DT job, starting from any one of them, InArt can be effectively extended to scenarios involving multiple DT jobs.

B.2 Model Splitting Details

Model splitting in InArt refers to splitting the results (*i.e.*, gradient) of local training into multiple gradient fragments. This process consists of two steps. At first, we calculate the optimal splitting fraction (*i.e.*, the proportion of the model) for each PS based on the network topology and resource load by L-InArt. Then, we will split the gradient into multiple gradient fragments based on the splitting fraction, and encapsulate these gradient fragments into multiple data flows that are sent to different PSs.

Specifically, each worker independently trains the model locally during DT. Specifically, each worker independently trains the model locally. In each iteration t, each worker trains the model and generates a gradient by calculating $g = \nabla L(w_t)$, where ∇ represents the vector differential operator and L represents the loss function of the model w. Subsequently, workers transmit these gradients to the PSs for global aggregation. PSs solely engage in gradient aggregation (*i.e.*, calculate the average of a set of floating-point vectors), but do not involve complex forward and backward operations. Based on the above characteristics, we can treat it as data flows since each gradient is an array of floating-point numbers. Then, we can split the data flow into multiple sub-flows based on the result of L-InArt, and send each sub-flow to the corresponding PS for aggregation.

For example, when training VGG16 in a DT with two PS, the worker generates a gradient with 31.25M floating-point numbers (elements) per iteration. We may split the gradient into two aligned fragments according to L-InArt (*e.g.*, the first 16.25M elements to PS_1 and the last 15M elements to PS_2) . Subsequently, each worker encapsulates these gradient fragments into packets and transmits them to the corresponding PS. Upon receiving the gradients, each PS performs the gradient aggregation operations and returns the aggregated results back to the respective workers.

When a worker receives gradient fragments from multiple PSs, it needs to recombine these fragments in the order of the previous splitting scheme (*i.e.*, sub-model combination), for local updating. Notably, the operation of combining sub-models in InArt is easy to implement and incurs negligible time overhead.

B.3 RR Details of R-InArt

In the following, we take the rounding process of INA as an example to illustrate the specific RR details of the R-InArt algorithm. Specifically, there are two switches for INA, and the optimal solution $\widehat{y}^{s,w}$ and $\{\widehat{y}_v^{s,w}\}$ of a worker w equals to 0.1 and $\{0.4, 0.5\}$, respectively. Then the interval $[0, 1]$ is splitted into three parts: $(0, 0.4]$, $(0.4, 0.9]$, and $(0.9, 1]$. We generate a random value between 0 to 1, and choose at most one switch for INA depending on this value. If the value is less than 0.4, R-InArt will choose the first switch as the aggregation switch for the gradient fragment from worker w to PS s. Otherwise, if the value is larger than 0.4 and less than 0.9, then the controller will choose the second switch as the aggregation switch for this gradient fragment. Meanwhile, if the value is larger than 0.9, the gradient fragment will be aggregated on parameter servers but not aggregated in the cluster.

B.4 Accuracy Loss Caused by INA

Indeed, INA may result in a decrease in accuracy, but it is within an acceptable range [29, 37]. Specifically, due to the imperative of maintaining line-rate processing in programmable switches, the operations involve straightforward integer arithmetic and logic operations. Neither floating-point nor integer division operations are feasible under this condition. As a result, mainstream INA schemes [29, 37], including InArt, choose to convert floating-point approximations to integer processing and division approximations to shift operations, which will introduce some loss in training accuracy. Fortunately, [37] has undergone extensive testing on a diverse range of models, which indicates that this accuracy loss is acceptable. For example, the accuracy losses of the three models used in our experiments, LSTM, VGG19 and RestNet50, are 1.54%, 1.60% and 1.05%, respectively.

It is important to highlight that our benchmark LBMM does not utilize INA, therefore ensuring no loss of accuracy (see details in Appendix A.1). However, considering the additional advantages provided by INA, it becomes evident that related INA solutions like InArt and ATP outperform LBMM significantly. As a result, we believe that the accuracy loss introduced by programmable switches is acceptable. In summary, since the primary contribution of this paper lies in presenting the problem of INA with route selection in multi-PS architectures and designing two algorithms for solving this problem, and the P4 implementation of InArt is similar to that of existing works, relevant evaluation has been omitted.