PARING: Joint Task Placement and Routing for Distributed Training With In-Network Aggregation

Yuhang Qiu, Gongming Zhao, *Member, IEEE*, Hongli Xu, *Member, IEEE*, He Huang, *Senior Member, IEEE, Member, ACM*, and Chunming Qiao, *Fellow, IEEE*

***Abstract*— With the increase in both the model size and dataset size of distributed training (DT) tasks, communication between the workers and parameter servers (PSs) in a cluster has become a bottleneck. In-network aggregation (INA) enabled by programmable switches has been proposed as a promising solution to alleviate the communication bottleneck. However, existing works focused on in-network aggregation implementation based on simple DT placement and fixed routing policies, which may lead to a large communication overhead and inefficient use of resources (*e.g.*, storage, computing power and bandwidth). In this paper, we propose PARING, the first-of-its-kind INA approach that jointly optimizes DT task placement and routing in order to reduce traffic volume and minimize communication time. We formulate the problem as a nonlinear multi-objective mixed-integer programming problem, and prove its NP-Hardness. Based on the concept of Steiner trees, an algorithm with bounded approximation factors is proposed for this problem. Large-scale simulations show that our algorithm can reduce communication time by up to 81.0% and traffic volume by up to 19.1% compared to the state-of-the-art algorithms.**

***Index Terms*— In-network aggregation, distributed training, task placement, gradient routing.**

I. Introduction

As MACHINE learning applications (*e.g.*, natural language processing [1] and recommender systems [2]) become increasingly complex, distributed training (DT) has been brought into the spotlight [3], [4], [5]. In order to perform DT in commercial clusters with numerous servers, the parameter server (PS) architecture [6], [7] is commonly adopted, whereby the servers can be assigned as either workers or PS(s). During each training iteration, workers compute gradients independently, and send the results to PS(s) for aggregation. After that, the aggregated gradients will be returned back to the workers for the next iteration. Due to the growth in model size and dataset size of DT tasks, the amount of transferred traffic during aggregation has ballooned, *e.g.*, up to 145 GB for a DeepLight task with only 32 workers during each iteration [8]. As a result, communication among the workers and PS(s) takes up to 50%-70% of the training time [8] and becomes a performance bottleneck for DT [9], [10].

To address the communication bottleneck, in-network aggregation (INA) has been proposed as a promising solution [8], [11], [12], [13]. During the typical DT workflow, the gradients are generated from the workers, forwarded through some switches, and fully aggregated on the PS(s). With INA, programmable switches (*e.g.*, P4-based [14], FPGA-based [15]) can participate in the gradient aggregation, and pass on the intermediate results until they reach the PS(s) (see Sec. II-A for more details). This reduces the communication overhead of the DT task and also relieves some processing pressure on the PS(s), thereby helping increase the gradient sending rate of the workers and accelerate the training process.

A few existing works focused on the efficient realization of INA on programmable switches while assuming fixed placement and routing strategies of DT tasks [12], [16], [17]. For example, SwitchML [16] performed gradient aggregation over ToR (Top of Rack) switches for communication optimization within a single rack. ATP [12] enabled in-network aggregation in multi-rack and multi-tasking scenarios for datacenters. In fact, different placement and routing strategies will greatly affect the training efficiency and network resource utilization in a cluster [11], [18]. On the one hand, inappropriate task placement leads to longer transmission paths from workers to PS(s), exacerbating the communication bottleneck. On the other hand, given the fixed location of available programmable switches in the network, improper routing policies cannot take full advantage of the available INA resources (see details in Sec. II-A), thus contributing little to relieving the communication pressure. Therefore, to further relieve communication bottleneck, *it is essential to find an appropriate DT task placement and routing policy with INA.*

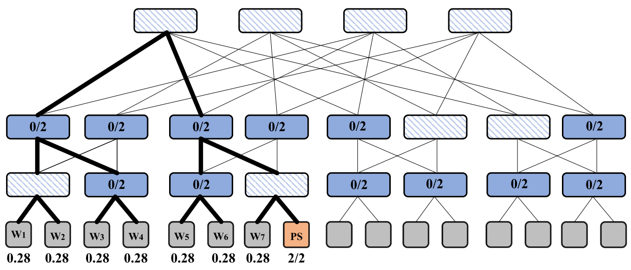
Manuscript received 24 February 2023; revised 4 October 2023, 7 February 2024, and 2 May 2024; accepted 12 June 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Park. This work was supported in part by the National Science Foundation of China (NSFC) under Grant 62372426, Grant 62132019, and Grant 62102392; in part by the National Science Foundation of Jiangsu Province under Grant BK20210121; in part by the Fundamental Research Funds for the Central Universities; and in part by the Youth Innovation Promotion Association of the Chinese Academy of Science under Grant 2023481. *(Corresponding author: Gongming Zhao.)*

Yuhang Qiu, Gongming Zhao, and Hongli Xu are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou, Jiangsu 215123, China (e-mail: qiu_yuhang@mail.ustc.edu.cn; gmzhao@ustc.edu.cn; xuhongli@ustc.edu.cn).

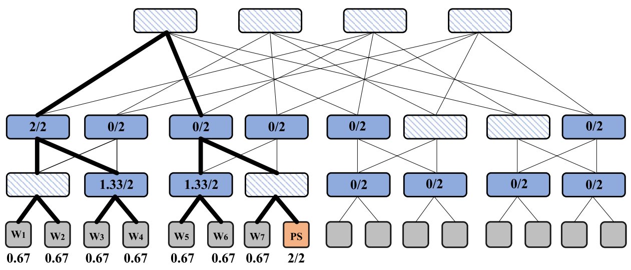
He Huang is with the School of Computer Science and Technology, Soochow University, Suzhou, Jiangsu 215123, China (e-mail: huangh@suda.edu.cn).

Chunming Qiao is with the Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 16260 USA (e-mail: drqiao01@gmail.com).

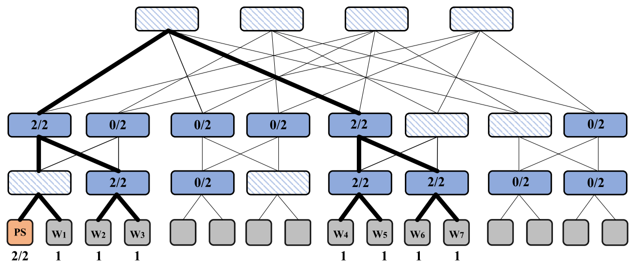
Digital Object Identifier 10.1109/TNET.2024.3414853



(a) DeepSys



(b) DeepSys+ATP



(c) PARING

Method	Communication Time	Traffic Volume	Switch Utilization Rate
DeepSys	28 s	34 GB	0%
DeepSys+ATP	12 s	18 GB	19.4%
Ours	8 s	15 GB	41.7%

(d) Performance Comparison

Fig. 1. A fat-tree cluster with a scale of 4 contains twenty switches (including switches from the edge, aggregation, and core layers, ranging from the closest to the farthest to the servers) and sixteen idle servers. The eight switches with diagonal lines are unavailable for aggregation, while the other twelve FPGA-based switches are available. These switches as well as the server assigned as PS come with an aggregation capacity of 2 Gbps. The gradient size is 1 GB, and the maximum gradient sending rate on workers is 1 Gpbs. The DT task for placement requires one PS and seven workers. Bold lines represent the routing of gradients. Let the values *load/capacity* near PSs and switches represent the ratio of usage and capacity, and the values near workers denote their actual gradient sending rate. The units are omitted in the figures for simplicity. Plot (a) shows the network workload for DeepSys with a traffic volume of 34 GB and an overall communication time of 28 seconds. Plot (b) shows the network workload of DeepSys+ATP, whose traffic volume is 18 GB and communication time is 12 seconds. Plot (c) shows the placement solution of PARING, which can reduce the traffic volume to 15 GB and the communication time to 8 seconds. The table for performance comparison is in plot (d).

While some existing works focused on scheduling of DT tasks [18], [19] or jointly optimizing task placement and routing [20], [21] in order to address communication bottlenecks, joint DT task placement and routing with INA faces a few new challenges of its own. In fact, apart from deciding which servers to serve as workers and PS(s) (which become the sources and destinations for routing), the *aggregation locations* (*e.g.*, programmable switches) need to be chosen for in-network aggregation. Furthermore, we need to consider the following two factors. First, *the routing traffic volume is variable.* Aggregation on a programmable switch changes the size of the traffic, leading to uncertainty in the routing traffic volume. Second, *the aggregation capacity is limited.* More specifically, bandwidth, memory, and computing resources on servers as well as programmable switches are limited, resulting in constrained line-rate aggregation. In short, we are faced with a joint optimization problem with constraints which is expected to be quite challenging to solve.

To overcome these challenges, we propose joint task Placement And Routing for distributed training with In-Network aGgregation (PARING) in this paper. Specifically, given a DT task and the network topology of a cluster along with the available programmable switches, we try to give a placement and routing solution that includes the assignment of workers and PS on the servers and an aggregation path for data from all workers to the PS. The main contributions in this work are as follows:

1) We design PARING, the first-of-its-kind work on task placement and routing with in-network aggregation for accelerating distributed training.
2) We formulate PARING as a nonlinear multi-objective mixed-integer programming problem and give a solution based on the Steiner tree and minimum-cost flow. We show the problem complexity and prove PARING can achieve the approximation factor of $\frac{3}{2}$ for gradient sending rate, and the approximation factor of $O(\log n)$ for system traffic volume, where n is the number of workers in a DT task.
3) We conduct large-scale simulations using real-world topologies and datasets to show that the proposed algorithm can achieve superior performance compared with other solutions. For example, PARING reduces 19.1% of the traffic volume and 81.0% of the communication time compared to the state-of-the-art algorithms.

The rest of this paper is organized as follows. Section II demonstrates a motivating example and overview of PARING. Section III introduces the preliminaries of our work, including system model and problem formulation. Section IV presents our proposed algorithm along with performance analysis. The simulation results are presented in Section V. Section VI presents related works. We conclude this paper in Section VII.

II. Motivation

In this section, we give an example to illustrate the pros and cons of some state-of-the-art solutions, which motivates our study.

A. A Motivating Example

Consider a fat-tree with a scale of 4 shown in Fig. 1(a), which contains sixteen idle servers and twenty switches (including switches from the edge, aggregation, and core layers, ranging from the closest to the farthest to the servers). Twelve out of twenty switches are FPGA-based and available for aggregation. A task requires one PS and seven workers for distributed training. We set the available aggregation capacity of the PS as well as FPGA-based switches to 2 Gbps. The gradient size is 1 GB, and the maximum gradient sending rate on workers is 1 Gbps. The switch memory is considered to be sufficient.

We first consider a classic task placement scheme called DeepSys [1]. Workers are preferentially placed on the same rack and gradients are sent along the shortest path towards the PS to reduce communication overhead, as shown in Fig. 1(a). In this scenario, workers W_1-W_7 and PS are assigned to the servers in the first and the second pod. The gradient data are scheduled through the first core switch, as well as the first switch in both pods' aggregation layer, resulting in the total traffic volume is $6GB*4+4GB*2+2GB=34GB$. Due to the limited aggregation capability of the PS, the system can achieve a maximum gradient sending rate of $\frac{2Gbps}{7}=0.28Gbps$. Therefore, the overall communication time should be $\frac{1GB}{0.28Gbps}=28$ seconds.

We next consider a state-of-the-art method with in-network aggregation, named ATP [2], which performs best-effort aggregation, as shown in Fig. 1(b). Each worker tries to aggregate on the first programmable switch it encounters on its shortest path to the PS, if any (*i.e.*, W_1,W_2 aggregate on the aggregation layer switch; W_3-W_6 aggregate on edge layer switches; W_7 aggregates on the PS). If the aggregation capacity of a programmable switch is exhausted, it will forward the redundant gradients directly to the PS. Considering that ATP does not involve the assignment of workers and PS(s), we use the same placement scheme as DeepSys here. Since the PS needs to aggregate the redundant gradients, it remains to be the bottleneck of DT task acceleration if the gradient sending rate is too high. Under this circumstance, when the gradient sending rate is 0.67 Gbps, the aggregation layer switch and the PS become fully loaded, with the former aggregating gradients from W_1,W_2 as well as intermediate results from W_3,W_4, while the latter further aggregates it with gradients from W_7 and intermediate results from W_5,W_6. As a result, the traffic volume for DeepSys+ATP is $2GB+5GB+2GB+9GB=18GB$, and the overall communication time is $\frac{1GB}{0.67Gbps}=12$ seconds.

B. Our Intuition

From the above example, we observe that both solutions aim at reducing the communication overhead of the system. DeepSys shortens the distance between workers and PS(s) through an optimized placement of DT tasks. ATP adopts best-effort in-network aggregation to decrease traffic volume by reducing the number of gradients on common paths. A simple combination of ATP+Deepsys reduces the communication time and traffic volume of the DT task effectively compared to Deepsys. A question immediately following the above discussion is that *How to further optimize the communication time and traffic volume taking advantage of both DT task placement as well as routing with INA?*

As seen from Fig. 1(b), gradients from W_1,W_2 are far from the aggregation location (*i.e.*, the aggregation layer switch), which results in an increased traffic volume in the system. Therefore, we need to jointly optimize the assignment of workers and PS(s) as well as in-network aggregation routing. As shown in Fig. 1(c), W_4-W_7 are placed in the third pod, and their gradients are aggregated within the pod at minimal communication cost. The PS and W_1-W_3 are placed in the first pod, using two more switches to reach the maximum gradient sending rate of 1 Gbps. The global switch utilization rate is 41.7%, over twice the rate of DeepSys+ATP. The communication time is $\frac{1GB}{1Gbps}=8$ seconds, which reduces by 71.4% and 33.3% compared to DeepSys and DeepSys+ATP, respectively. Moreover, the system traffic volume is $2GB+2GB+2GB+3GB+2GB+4GB=15GB$, which optimizes by 55.9% and 16.7%, respectively. See Fig. 1(d) for the detailed performance comparison. Motivated by this, we design a DT task placement and routing with in-network aggregation framework, called PARING. Note that, this example shows the performance gains over a simple topology. Further simulation results show that PARING can optimize up to 60.0% and 46.1% of system traffic volume, compared with DeepSys and DeepSys+ATP, respectively.

III. Preliminaries

A. System Model

We first present the proposed DT task placement model with in-network aggregation (INA) in a cluster. A DT task requires several workers for training and a parameter server (PS) to aggregate the gradients from workers. A cluster mainly consists of many servers and a set of switches. To place the DT task, we need to assign appropriate servers as the PS and workers. In addition, an aggregation path needs to be scheduled for the gradient data from each worker to the PS via switches or other workers. After gradient aggregation, the distribution process remains unchanged as it would be without INA. The gradients are multicast to each worker through switches, programmable or non-programmable.

1) We use $G=(V,E)$ to denote the topology of the cluster, where V denotes the vertex set and E denotes the edge set.
2) Vertex set $V=V_h+V_s$ consists of the set of servers (hosts) V_h and the set of switches V_s. Each vertex $v\in V$, no matter a server or a switch, has an aggregation capacity B_v. For example, if $v_1\in V_s$ denotes a traditional switch, we have $B_{v_1}=0$. When server $v_2\in V_h$ is assigned as the PS, it has an aggregation capacity of B_{v_2}. Note that it is allowed to aggregate part of the gradient and forward the remainder at the same time on a server or a programmable switch.
3) We do not consider link bandwidth constraint on edge $e\in E$. As in datacenter networks, the topology remains stable, and high-bandwidth links are commonly utilized, assuming sufficient link bandwidth is reasonable [22]. We leave the consideration of link bandwidth constraints as future work.
4) The DT task needs n workers and one PS for training. We assume that the size of the gradient produced by each worker in each iteration remains constant, denoted as b. For each worker, the upper bound of the gradient sending rate is R.

In this paper, we primarily focus on FPGA-based switches, where the bandwidth variable B_v is constrained by the FPGA chips [11]. As for P4-based switches, which are subject to different constraints, we will discuss these in Sec. IV-C.5.

B. Model Conversion

Our goal is to select servers acting as workers and PS, and lay out the aggregation and forward route for gradients from workers with maximum gradient sending rate and minimum communication cost. It is difficult to describe aggregation paths using the model in Sec. III-A. This is because when an edge is selected as part of the aggregation route, we need to further record whether the gradient data is aggregated or forwarded.

To solve this problem, we design a logical topology model. The main idea is that since the traditional switches in the physical topology are not capable of aggregating and the gradient is simply forwarded on some programmable switches, we try to remove these vertices in the logical topology for a more intuitive representation of aggregation routes. Specifically, we expect a logical topological model $G' = (V', E')$. The vertex set $V' = V'_h + V'_s$ contains the set of all idle servers V'_h as well as the set of programmable switches V'_s that may be the aggregation location. Edge $e(v_1, v_2) \in E'$ indicates that the next aggregation location of the gradient from vertex v_1 is possibly v_2, or the opposite. Also, edge e comes with a cost c_e which represents the smallest number of hops from v_1 to v_2. In this way, we can represent aggregation routes as a tree, where all leaf vertices are workers, the root vertex is the PS, and each edge indicates a gradient involved in an in-network aggregation. The detailed description of model conversion is given in Alg. 1.

Algorithm 1 Model Conversion

Input: An undirected graph $G(V, E)$ with non-negative costs on its vertices and edges.

Output: An undirected graph $G'(V', E')$ containing virtual edges with costs.

```
1: Initialization: $V' = V$
2: Remove occupied servers from $V'$.
3: Remove switches unavailable for aggregation from $V'$.
4: Find the minimum edge cost between any two vertices $v_1, v_2 \in V'$ as $c_{v_1,v_2}$ in the graph G with Floyd's algorithm.
5: for any two vertices $v_1, v_2 \in V'$ do
6:     Add undirected edge $e(v_1, v_2)$ with cost $c(v_1, v_2)$ to $E'$.
```

C. Problem Formulation

In this subsection, we formulate the problem of distributed training task placement and routing with in-network aggregation (PARING). We place workers as well as the PS in the cluster, and select programmable switches or servers to build an aggregation tree in the logical topology model obtained in Sec. III-B. The maximum gradient sending rate is expected to achieve the best DT task acceleration [8]. In addition, we want to reduce the traffic volume of the DT task in order to reduce the influence on the cluster.

When aggregating the gradients, the uniform gradient sending rate is denoted by r. We use binary $x_v \in \{0, 1\}$ to denote whether the vertex $v \in V'$ is selected as worker, the PS or programmable switch participating in the DT task or not, and binary $x_e \in \{0, 1\}$ with $e_{v_1,v_2} \in E'$ to represent whether

TABLE I
KEY NOTATIONS

Parameters	Description
V	the set of servers and switches in the physical topology
V_h	the set of servers in the physical topology
V_s	the set of switches in the physical topology
B_v	the aggregation capacity for $v \in V$
E	the set of links in the physical topology
V'	the set of servers and switches in the logical topology
V'_h	the set of servers in the logical topology
V'_s	the set of switches in the logical topology
E'	the set of links in the logical topology
c_e	the number of hops for edge $e(v_1, v_2) \in E'$ from v_1 to v_2
x_v	whether $v \in V'$ is selected as worker, the PS or programmable switches in the DT task or not
x_v^p	whether $v \in V'$ is selected as the PS or not
x_v^w	whether $v \in V'$ is selected as worker or not
x_v^s	whether $v \in V'$ is selected as programmable switch in the DT task or not
x_e	whether the gradient chooses aggregation path $e \in E'$ or not
$\delta(P)$	the set of edges with exactly one end-point in P
$f(P)$	whether there are terminals (*i.e.*, servers assigned as workers or the PS) in both P and $V' - P$ or not
n	the number of workers of the DT task
R	the upper bound of gradient sending rate
b	the size of gradient generated from each worker

gradient from v_1 is sent to v_2 for aggregation, or not. For a set $P \subset V'$, $\delta(P)$ denotes the set of edges with exactly one end-point in P, and $f(P) \in \{0, 1\}$ represents whether there are terminals (*i.e.*, servers assigned as workers or the PS) in both P and $V' - P$. There are four following constraints in PARING:

1) Connectivity constraint. All terminals need to be connected to each other, so that each worker has an aggregation path to the PS. More specifically, for any vertex set P containing partial terminals, there should be edges connecting vertices that are not in P (*i.e.*, edges with one and only one endpoint in P). That is

$$\sum_{e \in \delta(P)} x_e \geq f(P), \quad \forall P \subset V'.$$

2) Capacity constraint. Servers assigned as workers or PS as well as programmable switches can aggregate gradients with a limited processing rate, which can be formulated as

$$\sum_{e \in \delta(\{v\})} x_e \leq \max(\lfloor \frac{B_v}{r} \rfloor \cdot x_v, 1), \forall v \in V'_h,$$

and

$$\sum_{e \in \delta(\{v\})} x_e \leq \lfloor \frac{B_v}{r} \rfloor \cdot x_v + 1, \quad \forall v \in V'_s.$$

It is worth noting that if a server is not assigned as a worker or PS, it cannot participate in aggregation (*i.e.*, its aggregation capacity is 0), even if it is idle.

3) Server assignment constraint. One PS and n workers need to be assigned to different servers, while one server

cannot be assigned to both worker and PS. We have

$$\sum_{v \in V'_h} x^p_v = 1, \sum_{v \in V'_h} x^w_v = n, x^p_v + x^w_v \leq 1, \quad \forall v \in V'_h.$$

4) Sending rate constraint. The uniform gradient sending rate on servers and programmable switches cannot exceed R, the upper bound of gradient sending rate, that is $0 < r \leq R$.

In PARING, we have two objectives. First, we should maximize the uniform gradient sending rate to achieve the best DT task acceleration, which can be formulated as

$$\max O_1 = r.$$

While achieving the maximum gradient sending rate, the second objective, *i.e.*, minimizing the system traffic volume, should be pursued. This objective can be formulated as

$$\min O_2 = \sum_{e \in E'} x_e \cdot c_e \cdot b.$$

With these constraints and objectives, the problem can be formulated as follows:

$$\begin{aligned}
&\max O_1 = r \\
&\min O_2 = \sum_{e \in E'} x_e \cdot c_e \cdot b \\
&S.t. \begin{cases}
\sum_{e \in \delta(P)} x_e \geq f(P), & \forall P \subset V' \\
\sum_{e \in \delta(\{v\})} x_e \leq \max(\lfloor \frac{B_v}{r} \rfloor \cdot x_v, 1), & \forall v \in V'_h \\
\sum_{e \in \delta(\{v\})} x_e \leq \lfloor \frac{B_v}{r} \rfloor \cdot x_v + 1, & \forall v \in V'_s \\
\sum_{v \in V_h} x^p_v = 1 & \\
\sum_{v \in V'_h} x^w_v = n & \\
x^p_v + x^w_v \leq 1, & \forall v \in V'_h \\
0 < r \leq R & \\
x^p_v \in \{0,1\}, & \forall v \in V'_h \\
x^w_v \in \{0,1\}, & \forall v \in V'_h \\
x^s_v \in \{0,1\}, & \forall v \in V'_s \\
x_e \in \{0,1\}, & \forall e \in E'
\end{cases}
\end{aligned} \tag{1}$$

The first set of inequalities ensures the connectivity constraint. The second to third sets of inequalities correspond to the capacity constraint. The fourth to sixth sets of in/equalities mean the server assignment constraint. The seventh inequality represents the sending rate constraint. Our first optimization goal is to maximize the gradient sending rate, and the second objective is to minimize the system traffic volume.

Theorem 1: PARING is one of the most difficult problems in NP-hard class.

Proof: As generally known, the Steiner tree problem (STP) in graphs is a classical NP-hard problem since the corresponding decision problem (i.e., determining whether a Steiner tree having at most a given total cost exists for a given graph) is one of Karp's 21 NP-complete problems. It can be observed that, when given a specific gradient sending rate and ignoring the capacity constraint, the special case of PARING aligns precisely with STP, where the Steiner tree is the aggregation tree, with workers being leaves and PS being the root. Consequently, PARING is at least NP-hard. □

Algorithm 2 PARING

Input: An undirected network $G(V, E)$ with vertex aggregation capacity B_v and edge communication cost c_e, a terminal set $T \subset V$, the number of workers n and an upper bound of gradient sending rate R.

Output: A maximum feasible gradient sending rate r and an aggregation tree $H(W, F)$.

```
1:  Step 1: Selecting PS
2:  Select vertices $v_i$ with the largest capacity $B_{v_i}$ to join the PS candidate set $T'$.
3:  for $v \in T'$ do
4:      Select $n$ vertices $v_i \in T - \{v\}, i = 1 \dots n$ with smallest edge cost $l_{v,v_i}$.
5:      Let $ave_v$ be the average edge cost of the selected $n$ edges.
6:  Let the PS be vertex $v_p \in T'$ with minimum $ave_{v_p}$.
7:  Step 2: Obtaining Degree Constraint
8:  Let set $V_1 = V_s + \{v_p\}$, and add $n$ vertices $v_i \in V_h - \{v_p\}$ to $V_1$ with the largest capacity $B_{v_i}$.
9:  $V_2 = V_1$.
10: $r = \min\{R, \sum_{v \in V_1} B_v / |V_1 - \{v_p\}|\}$.
11: while $\sum_{v \in V_1} \lfloor \frac{B_v}{r} \rfloor < |V_1 - \{v_p\}|$ do
12:     $V_1 = V_2$.
13:     $r = \max\{\frac{B_v}{\lfloor \frac{B_v}{r} + 1 \rfloor}, \forall v \in V_1\}$.
14:     for all $v \in V_1, B_v < 2r$ do
15:         $V_1 = V_1 - \{v\}$
16: for all $v \in V$ do
17:     if $v \in V_s$ then
18:         $d(v) = \lfloor \frac{B_v}{r} \rfloor + 1$.
19:     else
20:         $d(v) = \max\{1, \lfloor \frac{B_v}{r} \rfloor\}$.
21: Step 3: Constructing Aggregation Tree
22: Find a minimum-cost degree-constraint Steiner tree $H$ rooted at PS using Alg. 3
23: $r = \min\{\frac{B_{v_s}}{d(v_s)-1}, \frac{B_{v_h}}{d(v_h)}$ for $v_s \in W \cap V_s, v_h \in W \cap V_h\}$.
24: Return $r$, $H(W, F)$.
```

IV. Algorithm Description

In this section, we first present an approximate algorithm, called PARING, which aims at joint optimization of DT task placement and INA routing in Alg. 2. Then, we prove that there is a bounded approximation factor to our proposed algorithm.

A. The PARING Algorithm

1) Algorithm Overview: The goal of Alg. 2 is to place the DT task and schedule INA routing with first maximum gradient sending rate, and then minimum traffic volume. To achieve this, we construct an aggregation tree, with its root being the PS, other vertices being workers and programmable switches,

Algorithm 3 Find a Steiner Tree

Input: An undirected network $G(V,E)$ with vertex degree constraint $d(v)$ and edge cost c_e, a terminal set $T \subset V$, the number of partial terminals n and a vertex v_p as PS.

Output: A minimum-cost degree-constraint Steiner tree $H(W,F)$ rooted at PS.

```
1:  Set $W = T, F = \emptyset$.
2:  For each edge $e_{a,b}$ with edge cost $c_{e_{a,b}} > 0$, add a virtual
    vertex $v_{a,b}$ with cost $c_{e_{a,b}}$. Replace edge $e_{a,b}$ with $e_{a,v_{a,b}}$
    and $e_{v_{a,b},b}$.
3:  repeat
4:    Let $m$ be the number of workers already connected to
      $v_p$. Set $q = n - m$.
5:    Set $I(V',E') = G(V,E)$.
6:    Set $d'(v) = 0$ as current degree of vertex $v$.
7:    repeat
8:      for every vertex $v \in V'$ that $d(v) \geq 2$, consider $v$ as
        the center of a spider do
9:        for $i = 1$ to $\min\{\lfloor \frac{3d(v)}{2} - d'(v) \rfloor, d(v), n-m\}$
          do
10:         Find a minimum-cost spider centered at $v$ in $I$
            with $i$ new workers as its feet and connected
            with the component containing PS using Alg.4.
11:     Among all the spiders produced in line 8, choose one
        of minimum ratio-cost, defined as the ratio of the cost
        of all the vertices in the spider to the number of feet
        in it.
12:     Merge the spider to $H(W,F)$. Update $m$.
13:     for all $v \in V'$ do
14:       Update $d'(v)$ as current degree of vertex $v$.
15:   until $q \leq 6$ or $n - m < \frac{7q}{8}$
16:   if $q > 6$ then
17:     for all $v \in V'$ do
18:       if $d'(v) \geq \frac{6}{5}d(v)$ then
19:         $V' = V' - \{v\}$,$V = V - \{v\}$.
20: until $m = n$
21: Undo line 2.
22: Return $H(W,F)$.
```

and edges indicating the routing paths of the gradients. PARING takes a three-step approach to construct the aggregation tree. First, we select a server as the PS, which serves as the root of the aggregation tree. In the second step, with the PS being selected, we get a view of the available aggregation capacity in the network, and obtain the degree constraint for vertices in the aggregation tree according to the capacity constraint and the number of workers. Finally, given the tree root as well as the degree constraint, we construct a minimum-cost degree-constraint Steiner tree as the aggregation tree, which indicates the DT task placement and INA routing policy.

2) Description of Step 1: The first step will choose one server as PS. We select servers with the largest capacity as candidates for PS (Line 2). After collecting the set T' of PS candidates, we calculate the average distance from $v \in T'$ to the nearest n servers in $T - \{v\}$ (Lines 3-5). Then we choose the vertex with the minimum average distance as the PS.

Algorithm 4 Find Spiders

Input: An undirected graph $H(V,E)$ with vertex cost c_v, a vertex v as the center and a number of feet j.

Output: A minimum-cost spider connecting PS and centered at v with j feet that are terminals.

```
1: Let $I(V',E')$ be a directed graph with $V' = V$.
2: $\forall e(v_1,v_2) \in E$, add edge $e(v_1,v_2)$ to $E'$ with cost $c_{v_1}$
   and add edge $e(v_2,v_1)$ to $E'$ with cost $c_{v_2}$.
3: $\forall e(v,v_1) \in E'$, set $c_{e(v,v_1)} = \frac{c_v}{j}$.
4: Attach a new sink vertex $t_v$ to $V'$. $\forall v' \in T$, add edge
   $e(v',t_v)$ to $E'$ with cost 0.
5: $\forall v' \in V' - \{v,t_v\}$, set the capacity bound of $v'$ to be 1.
   Set the capacity bound of $v$ and $t_v$ to be $j$.
6: if there is a path from $v$ to $v_p$ then
7:   Find a minimum-cost flow of value $j$ from $v$ to $t_v$.
8: else
9:   Find a minimum-cost flow of value $j-1$ from $v$ to $t_v$
     and flow of value 1 from $v$ to vertices in the connected
     component containing PS.
```

3) Description of Step 2: The second step of Alg. 2 is to obtain the degree constraint for vertices in V. Intuitively, we expect the throughput of each vertex for aggregation to not exceed its capacity. In other words, the degree constraint (*i.e.*, the number of gradients a vertex can aggregate) is determined by the aggregation capacity and gradient sending rate. We initialize the gradient sending rate r by an overestimated assumption $\min\{R, \sum_{v \in V_1} B_v/|V_1 - \{v_p\}|\}$ (Line 10), at which all vertices are making full use of their aggregation capacity. For ease of description, V_1 is the set of vertices participating in aggregation (*i.e.*, workers, the PS and the programmable switches), and the number of gradients generated in the network is $|V_1 - \{v_p\}|$. Since complete gradients are required for aggregation, we run a loop to update r with a more accurate value. Specifically, in each iteration we fix the gradient sending rate by $r = \max\{\frac{B_v}{\lfloor \frac{B_v}{r} + 1 \rfloor}, v \in V_1\}$ (Line 13), so that vertex v is capable of aggregating $\lfloor \frac{B_v}{r} \rfloor$ gradients. Here, vertices unable to aggregate two incoming gradients are ignored (Lines 14-15). When no more adjustment will be made in V_1 (*i.e.*, when $\sum_{v \in V_1} \lfloor \frac{B_v}{r} \rfloor \geq |V_1 - \{v_p\}|$), r is a reasonable assumption of gradient sending rate. Based on the gradient sending rate, we calculate the degree constraints for programmable switches and servers separately (Lines 17-20) to make sure that an aggregation route exists from workers to PS.

4) Description of Step 3: The third step of Alg. 2 is to construct an aggregation tree using a minimum-cost degree-constraint Steiner tree algorithm in Alg. 3. Since the constructed Steiner tree by Alg. 3 may violate the degree constraint (see Theorem 2 for more details), we update the gradient sending rate (Line 23), which is dominated by the vertex breaking the degree constraint most.

Inspired by [23], Alg. 3 maintains a set W of vertices and a set F of edges to describe the Steiner tree. Initially W contains all the terminals and F is empty. We first subdivide each edge in G by introducing a new virtual vertex with a cost equal to the edge cost (Line 2), so that the cost of the Steiner tree can be represented by the total cost on vertices. Then, the algorithm connects vertices in the terminal set T iteratively. For convenience, we make the following definition.

Definition 1 [23]: A spider is a tree with at most one vertex whose degree is greater than two. A center of a spider is a vertex from which there are vertex-disjoint paths (called legs) to the leaves of the spider. Note that if a spider has at least three leaves, its center is unique. The leaves of the spider are also called the feet of the spider. A nontrivial spider is one with at least two feet.

Initially, all terminals are not connected to the PS, and the variable q (*i.e.*, the number of unconnected terminals) is n. Next we repeatedly execute a greedy algorithm to choose one or several spiders such that the addition of these spiders to the current tree H will reduce the number of unconnected terminals by a constant factor (*e.g.*, at least $\frac{1}{8}$ in each iteration) (Line 15). We run an inner loop to find these spiders. In each iteration of the inner loop, we first find a set of minimum-cost spiders with the degree constraint centered at each vertex using Alg. 4 (Lines 8-10), and then merge the spider of minimum ratio-cost (*i.e.*, the ratio of the total vertices cost in the spider to the number of feet) to the tree H. After connecting at least $\frac{q}{8}$ terminals, we leave out the vertex v with the current degree satisfying $d'(v) \geq \frac{6}{5}d(v)$ (Line 18), so that they will no longer be part of any spider in the next iteration. When there are no more than 6 unconnected terminals, it is no need to remove the vertices breaking the degree constraint. Finally, we transform the graph with costs on vertices back to one with edge costs, so that the result H would be the aggregation tree.

The procedure of finding a minimum-cost spider centered at v with j feet is described in Alg. 4. First, Alg. 4 converts the vertex cost of the undirected graph to the edge cost of a directed graph. Then we reassign all the edges leaving the center at a cost of $\frac{c_v}{j}$, so that the total cost of j edges leaving v is c_v. Next, we set up a virtual sink vertex to connect all terminals. Finally, we obtain the spider by finding a minimum-cost flow of value j from v to the sink vertex (Lines 5-9).

B. Performance Analysis

Theorem 2: The gradient sending rate of the solution in Alg. 2 denoted by r_1 is at least $\frac{2r_0}{3}$, where r_0 is the optimal gradient sending rate for the problem of PARING.

Proof: By Lines 16-20, Alg. 2 calculates a degree constraint for each vertex in G. We first prove that the obtained gradient sending rate r_2 is optimal without considering routing from workers to the PS. Assume that there is a better solution r' than r_2 (*i.e.*, $r' > r_2$). By the capacity constraint, we have

$$\sum_{v\in V_1} \lfloor \frac{B_v}{r'} \rfloor \geq |V_1 - \{v_p\}|. \tag{2}$$

Here V_1 the set of vertices participating in aggregation, including workers, the PS and switches $v \in V_s$ satisfying $B_v \geq 2r'$. Because r_2 is obtained by Line 13 and $r' > r_2$, we have

$$\lfloor \frac{B_v}{r'} \rfloor \leq \lfloor \frac{B_v}{r_2} \rfloor,\ \forall v \in V_1 \text{ and } \lfloor \frac{B_v}{r'} \rfloor < \lfloor \frac{B_v}{r_2} \rfloor,\ \exists v \in V_1. \tag{3}$$

However, by Line 13 we can conclude that there exist a previous value r'' satisfying

$$r'' \geq r' \text{ and } \lfloor \frac{B_v}{r''} \rfloor = \lfloor \frac{B_v}{r'} \rfloor, \quad \forall v \in V_1. \tag{4}$$

Therefore, we have

$$\sum_{v\in V_1} \lfloor \frac{B_v}{r''} \rfloor \geq |V_1 - \{v_p\}|. \tag{5}$$

In other words, a better solution r'' obtainable by Step 2 exists compared to r_2. Therefore a contradiction arises and r_2 is the optimal gradient sending rate without considering routing from workers to the PS.

In fact, an aggregation tree without breaking the degree constraint after Step 2 does not necessarily exist. If we relax the degree constraint, gradient sending rate will be decreased. That is,

$$r_0 \leq r_2. \tag{6}$$

From Line 9 in Alg. 3, each newly select spider will not make the degree of its center v greater than $\frac{3d(v)}{2}$. That is, the gradient sending rate r_1 by Alg. 2 will be not less than $\frac{2r_2}{3}$ due to degree constraint violation at this vertex, and we have

$$r_1 \geq \frac{2r_2}{3} \geq \frac{2r_0}{3}. \tag{7}$$

□

Theorem 3: There are $O(\log n)$ iterations of Line 3 in Alg. 3, where n is the number of workers.

Proof: Observing Line 15, each iteration of the algorithm reduces the number of terminals by a constant factor $\frac{1}{8}$ until it is less than 6. Therefore, the number of iterations will not exceed $O(\log_{\frac{7}{8}}\frac{6}{n}) + 1 = O(\log n)$. □

Theorem 4: Let C denote the cost of the subgraph added subsequently in Alg. 3 (Line 12), and k denote the number of terminals the subgraph merge. Then we have

$$k \geq \frac{C \cdot q}{24C_{opt}},$$

where C_{opt} is the minimum cost of any Steiner tree with degree restrictions.

Proof: Let τ be an optimal Steiner tree with cost C_{opt}. At the beginning of iteration i in Line. 7, the set of unconnected terminals is denoted by T_i, and the set of deleted vertices (*i.e.*, vertices in $V - V'$) is denoted by D_i. We construct a tree τ_i from τ by connecting vertices in T_i as well as v_p to a virtual vertex with zero cost, and removing necessary edges from τ to make it acyclic. Then, from τ_i we remove all the vertices in D_i and associated edges. By the construction rules above, the degree of vertex u in τ_i, denoted as $d_{\tau_i}(u)$, is at most $d(u)$. Furthermore, u is deleted by our algorithm only if its degree, denoted as $\bar{d}(u)$, exceeds $\frac{6d(u)}{5}$ due to the spiders merged in a given iteration i of Line 3. Thus we have

$$\bar{d}(u) \geq \frac{6d(u)}{5} \text{ and } d_{\tau_i}(u) \leq d(u), \quad \forall u \in D_i. \tag{8}$$

In each iteration, the total degrees of all the deleted vertices is at most q. Combining these observations, we get

$$\sum_{u\in D_i} d_{\tau_i}(u) \leq \frac{5}{6}\sum_{u\in D_i} \bar{d}(u) \leq \frac{5q}{6}. \tag{9}$$

Thus, the total number of edges deleted from τ is also at most $\frac{5q}{6}$, which means at least $|T_i| - \frac{5q}{6}$ terminals are in the newly merged subgraph. Since $|T_i| \geq \frac{7q}{8}$, at least $\frac{7q}{8} - \frac{5q}{6} = \frac{q}{24}$ terminals are merged.

We split the subgraphs merged in each iteration of Alg. 3 (Line 3) into nontrivial spiders, each centered at different vertices $v_1, \ldots, v_t$, with the number of feet being $l_1, \ldots, l_t$, and the cost of the spiders being $S_1, \ldots, S_t$. Then, the ratio cost of the spider centered at v_j in this loop is at most $\frac{S_j}{l_j}$. Since we choose the spider of minimum ratio-cost in Line 10, for each spider we have

$$\frac{S_j}{l_j} \geq \frac{C}{k}, j = 1, 2, \ldots, t. \tag{10}$$

Summing over all the constructed spiders in an iteration, it follows

$$\sum_{j=1}^{t} S_j \geq \frac{C}{k} \sum_{j=1}^{t} l_j. \tag{11}$$

Obviously, the number of feet of the spiders is not less than the newly connected terminals. That is,

$$\sum_{j=1}^{t} l_j \geq \frac{q}{24}. \tag{12}$$

We also note that

$$\sum_{j=1}^{t} S_j \leq C_{opt}. \tag{13}$$

Combining Eqs. (11)-(13), we have

$$k \geq \frac{C \cdot q}{24 C_{opt}}. \tag{14}$$

□

Theorem 5: The total cost of added vertices to the solution in each iteration of Alg. 3 (Line 3) is at most $O(C_{opt})$.

Proof: First we complete the proof with regard to the incremental cost in the last iteration. At the beginning of the last iteration, the number of unconnected terminals is at most 6. Given our algorithm reduces to that of [24] for node-weighted Steiner tree at this moment, the cost of the added vertices is at most $O(C_{opt} \log 6) = O(C_{opt})$.

For a previous iteration i, let the center of f chosen spiders (Line 11) in this iteration be $v_1, \ldots, v_f$ in the chosen order. Let ϕ_j denote the number of terminals that need to connect after choosing spider centered at v_j. For instance, $\phi_0 = q$ (*i.e.*, the number of remaining terminals at the beginning of this iteration in H), $\phi_{f-1} > \frac{7q}{8}$ and $\phi_f \leq \frac{7q}{8}$. Let the number of terminals merged by the spider centered at vertex v_j be m_j. Then we have

$$\phi_j = \phi_{j-1} - (m_j - 1). \tag{15}$$

Let C_j denote the incremental cost when spider centered at v_j is chosen. By Theorem 4, we have

$$m_j \geq \frac{C_j \cdot q}{24 C_{opt}} \geq \frac{C_j \cdot \phi_{j-1}}{24 C_{opt}}. \tag{16}$$

Substituting Eq. (16) into Eq. (15) and simplifying using $m_j \geq 2$ yields

$$\phi_j \leq \phi_{j-1}(1 - \frac{C_j}{48 C_{opt}}). \tag{17}$$

Simplifying Eq. (17), we obtain

$$\phi_{f-1} \leq \phi_0 \prod_{j=1}^{f-1} (1 - \frac{C_j}{48 C_{opt}}). \tag{18}$$

Taking natural logarithms on both sides and simplifying using the approximation $\ln(1 + x) \leq x$, we obtain

$$48 C_{opt} \ln \frac{\phi_0}{\phi_{f-1}} \geq \sum_{j=1}^{f-1} C_j. \tag{19}$$

Note that $\phi_0 = q$ and $\phi_{f-1} > \frac{7q}{8}$, so we have

$$\sum_{j=1}^{f-1} C_j \leq 48 C_{opt} \ln \frac{8}{7} = O(C_{opt}). \tag{20}$$

To complete the proof, we bound the cost of the spider centered at v_f, the last vertex chosen in this iteration. Using Theorem 4 and noting that $m_f \leq q$ we have

$$C_f \leq 6 C_{opt}. \tag{21}$$

Using Eqs. (20) and (21), the total cost of added vertices in this iteration is

$$\sum_{j=1}^{f} C_j = O(C_{opt}). \tag{22}$$

□

Theorem 6: The total cost of added vertices to the solution in Alg. 3 is at most $O(C_{opt} \log n)$.

Proof: We have proved that the number of iterations in Alg. 3 is $O(\log n)$ in Theorem 3 and the total cost of added vertices in each iteration of Line 3 is at most $O(C_{opt})$ in Theorem 5. Obviously, we can conclude that the total cost of added vertices in the entire algorithm is at most $O(C_{opt} \log n)$. □

Theorem 7: The traffic volume for an iteration on the obtained aggregation tree is at most $O(C_{opt} \log n)$.

Proof: The traffic volume of the obtained aggregation tree is actually the cost of Steiner tree by Step 3. We divide the total cost of the Steiner tree into the cost of edges connecting PS and the others.

First we consider the cost of edges unconnected with the PS. For each spider selected in our solution, we remove the edges unassociated with v_p and connect the spider center to the PS v_p' in the optimal solution instead. By sorting these spiders in ascending order of ratio-cost, we find a spider selection sequence that the constructed Steiner tree is identical to ours, apart from edges associated with PS. Since the degrees of the vertices remain unchanged and spiders are merged in ratio-cost order, Theorems 3-5 still hold. From Theorem 5, for any legitimately chosen spider, we guarantee that the cost of each merged subgraph in an iteration is $O(C_{opt})$. It means that the total cost of our solution will not exceed $O(C_{opt} \log n)$ without considering the cost of the edges connecting PS.

Next we calculate the cost of edges connecting PS. In the worst case, the PS in our solution connects as many costly vertices as possible, which has a total cost of $\lfloor \frac{\max\{B_v, v\in V_h\}}{r_0} \rfloor \cdot d$, where d is the distance between the two farthest vertices, and r_0 is the optimal gradient sending rate. The worst-case cost is independent of n and is much larger than the actual one, because the first spider connected to PS has the best ratio-cost and tries to connect more cost-efficient vertices.

Combining both costs above, we can get the total cost of the Steiner tree is at most $O(C_{opt} \log n) + \lfloor \frac{\max\{B_v, v\in V_h\}}{r_0} \rfloor \cdot d = O(C_{opt} \log n)$. The total cost of the Steiner tree represents the traffic volume of gradient aggregation. The path for gradient distribution is a multicast tree, which, in the worst case, incurs a cost equivalent to that of the Steiner tree. Therefore, the traffic volume for an iteration is $O(2C_{opt} \log n) = O(C_{opt} \log n)$. □

Theorem 8: The time complexity of Alg. 2 is $O(|V|^4 \log n)$, where $|V|$ is the number of vertices.

Proof: The running time of Alg. 2 is dominated by of Alg. 3 in Step 3. Alg. 3, in turn, is dominated by the inner-loop (Line 7) nested within the outer-loop (Line 3). The inner-loop calls Alg. 4 for each vertex, which finds a minimum-cost flow with a running time of $O(|V|^2)$ [25] since initially G is a fully connected graph. This gives an overall running time of $O(|V|^4)$ for the outer-loop in Alg. 3 while connecting at least $\frac{1}{8}$ of the unconnected terminals. The outer-loop runs $O(\log n)$ iterations, implying an overall running time of $O(|V|^4 \log n)$. □

C. Discussion

This subsection discusses potential enhancements to our proposed mechanism.

1) Handling Multiple Tasks: PARING supports the placement of multiple INA tasks in datacenter networks, where there are typically numerous tasks including both INA and non-INA tasks. When a new INA task arrives, we update the network parameters, including available switch aggregation capacities and available servers, and then run the PARING algorithm. When an INA task ends, the relevant resources are released. For other ongoing INA tasks, we can choose to continue operating with the original placement scheme, or optimize the INA traffic without changing the placement of workers and PS(s), which can be achieved by modifying Alg. 3 to treat the position of workers as terminals.

2) Limited Switch Memory Scenarios: To address the issue of insufficient memory, we partition the aggregated gradient data and perform aggregation in multiple INA subtasks. The size of each subtask depends on the minimum available memory of the switches. If there is no switch memory available for INA in the network, the aggregation of that particular subtask will be entirely handled by the PS (as well as workers). It is worth noting that limited memory often occurs with P4-based switches, which typically have a 10 MB-level memory capacity [14]. On the other hand, FPGA-based switches offer GB-level memory capacity, which is sufficient for most scenarios.

3) Extending to the Scenario With Multiple PSs: In general, our algorithm takes full advantage of the available programmable switches in the cluster, and is able to achieve decent gradient sending rates and system traffic volume when using a single PS. When computing resources are scarce in the cluster, multiple PSs are helpful in order to achieve a higher gradient sending rate and lower system traffic volume. In fact, our system can work in the multi-PS scenario. First, we greedily assign servers for multiple PSs referring to Step 1 in Alg. 2. Then, for each PS, we use Alg. 3 to obtain the solution of the subtask, and update the aggregation capacity B_v for vertices in the cluster after each run of Alg. 3. In this way, we are able to obtain the solution for the multi-PS scenario by combining the previously acquired results.

4) Dealing With Fee-for-Service Programmable Switches: In Sec. III, we did not take the fee of programmable switch usage into account. This is because our intention of PARING was to leverage available switch resources in data center networks to accelerate DT tasks. We believe that with the promotion of INA and its application in cloud scenarios, switch resources may become scarce, and fee-for-service programmable switches are highly possible. In this situation, our goal becomes finding the aggregation tree that minimizes both communication volume and switch costs, *i.e.*, the Steiner Tree with the minimum cost on both edges and vertices, which can be solved by Alg. 3.

5) INA Using P4-Based Switches: Unlike FPGA-based switches, P4-based switches execute aggregation operations within their processing pipelines. This enables P4-based switches to aggregate gradients from all ports at the highest possible port throughput. Additionally, P4-based switches typically possess more constrained memory resources, which often leads to limitations on the size of gradients for a given (sub)task. To facilitate the application of PARING in environments with P4-based switches, we may set B_v to infinity, thereby allowing a switch to aggregate an unlimited number of gradients. Moreover, it is necessary to apply the strategy outlined in Sec. IV-C.2 due to the limited switch memory.

6) INA With Communication Scheduler: Some previous works [26], [27] accelerated DT tasks by scheduling the fraction of gradients for all-reduces, enabling overlap with computation. Specifically, gradients are generated layer by layer on workers, and later-generated gradients are prioritized, since the corresponding parameters are utilized earlier in the subsequent iteration. To combine PARING with such implementation, we may employ these communication schedulers to determine the order and priorities of the gradient fractions sent by each worker.

V. Performance Evaluation

In this section, we compare PARING with state-of-the-art solutions. We first present the workflow of employing PARING for INA in Sec. V-A. Then we introduce the performance metrics and benchmarks in Sec. V-B. Finally, the simulation results are presented in Sec. V-C, to illustrate the performance of PARING.

A. Workflow of PARING System

The PARING algorithm can integrate with existing INA implementation. The workflow for the DT task using PARING mainly consists of 5 steps.

1) Obtain the aggregation capacities of the switches and servers.

2) Utilize the PARING algorithm to solve the task placement and INA routing scheme.
3) Modify the Reconfigurable Match Tables (RMT) [28] in the switches to categorize incoming packets into locally aggregated packets and forwarded packets. Additionally, employ existing INA implementation solutions and use P4 or FPGA programs to implement the aggregation functionality on the switches.
4) During the aggregation phase of DT tasks, control the synchronized arrival of gradients at the switches.
5) The PS collects all gradients and performs the final aggregation.

B. Performance Metrics and Benchmarks

1) Performance Metrics: We adopt the following three performance metrics to evaluate the efficiency of our algorithm for distributed training. (1) the gradient sending rate; (2) the total traffic volume per iteration; (3) the communication time per iteration.

We first measure the uniform gradient sending rate on the workers. Also, in each round, we calculate the size of gradient transferred by all the links as the total traffic volume per iteration. Finally, we record the time of gradient update each iteration as the communication time.

2) Benchmarks: We compare PARING with five benchmarks.

The first benchmark is DeepSys [18], a classic DT task placement scheme. Specifically, workers are preferentially placed on the same rack. Rather than resorting to programmable switches for gradient aggregation, we send all gradients directly to the PS via the shortest path.

The second one is DeepSys+SOAR. SOAR [29] is a state-of-the-art method for in-network aggregation, which designs an exact algorithm, and gives a solution that minimizes the total traffic volume in a given broadcast tree, using a restricted number of programmable switches participating in aggregation. In order to adapt SOAR to a generalized network topology, we place workers and the PS and construct the aggregation tree according to DeepSys.

The third benchmark is DeepSys+ATP. ATP [12] is an efficient best-effort in-network aggregation policy. Specifically, when aggregating part of the arriving gradients is allowed, the programmable switch will only aggregate the gradients within its capabilities and forward the remainder, if it receives too many gradients. Furthermore, the gradient sent by each worker is constrained in ATP to be able to aggregate only at the first encountered programmable switch (*e.g.*, edge switches in a fat-tree topology) or PS. Like SOAR, we resort to DeepSys for aggregation trees.

The forth benchmark is MARINA [30]. MARINA is a multitenancy and redundancy-aware INA task placement scheme. It first queries the switch resources and finds the aggregation tree by constructing a vertex-weighted Steiner tree. Then, it reshapes the aggregation tree to balance the load of switches. MARINA does not optimize for traffic volume and may not operate well in networks with regular switches.

The last benchmark is PANAMA [11], an INA framework for DT tasks on shared clusters. PANAMA distributes INA traffic across multiple trees, with each switch along the path capable of aggregation. The INA traffic is distributed in a round-robin manner across the aggregation trees and utilizes a TCP-like congestion control strategy.

C. Simulation Evaluation

1) Simulation Settings: We conduct simulation experiments to compare PARING with three benchmarks in the classical fat-tree topology [31], which is commonly adopted in clusters. The fat-tree topology contains a three-tier switch architecture and a one-tier server architecture. Our simulation uses a fat-tree topology with a scale of 6, which consists of 45 switches (*i.e.*, 9 core switches, 18 aggregation switches and 18 edge switches) and 54 servers. All servers are homogeneous, and can be assigned as workers or PS.

For simplicity, we set the aggregation capacity of programmable switches and the PS to 10 Gbps to perform close-to-reality simulation. As for the size of the gradient data sent in each round, we refer to the LSTM model [8] and set it to 1627 MB, the same for each worker. Since the programmable switches are capable of line rate processing [8], [11], we ignore the latency introduced by INA as in [11].

The simulations are performed under two scenarios. The first scenario allows partial aggregation, *i.e.*, programmable switches can only aggregate part of the arriving gradients and forward the remainder to the next hop. This hypothetical scenario tests the performance of DT task placement with fine-grained INA routing, which may lead to difficult management during multitask scheduling while achieving efficient use of resources (*e.g.*, table entries and computing power) on programmable switches. The second scenario forbids partial aggregation (*i.e.*, programmable switches can either aggregate all of the received gradients or forward them all), and performs a relatively coarse-grained INA routing, which is simple to manage but unable to make full use of INA resources. To perform PARING when forbidding partial aggregation, we first find a solution ignoring this constraint, and then make adjustments according to the obtained INA routing. In the PANAMA framework, there is no PS, and the switches aggregate all incoming traffic. Therefore, we only simulated its performance when programmable switches are prohibited from aggregating part of the arriving gradients and forward the remainder.

2) Performance Comparison With Fine-Grained Routing: Here we run three sets of experiments for performance evaluation when programmable switches can aggregate part of the gradients and forward the remainder.

The first set of simulations illustrates the effect of DT task scale (*i.e.*, the number of workers). In this set of simulations, there are eight available programmable switches in the cluster. The upper bound of the gradient sending rate R is 5 Gbps, and the results are shown in Figs.2-4. In Fig. 2, we can learn that the total traffic volume of four solutions increases when the number of workers grows. PARING always acquires a lower traffic volume than the other three benchmarks. For example, given 35 workers in the cluster, our algorithm reduces the traffic volume by 60.0%, 19.1%, 51.6% and 46.1% compared with DeepSys, DeepSys+SOAR, DeepSys+ATP, and MARINA, respectively. Since PARING will take full advantage of programmable switches, it is natural to achieve a lower traffic volume, even though the available programmable switches are far from each other. From Fig. 3 we notice

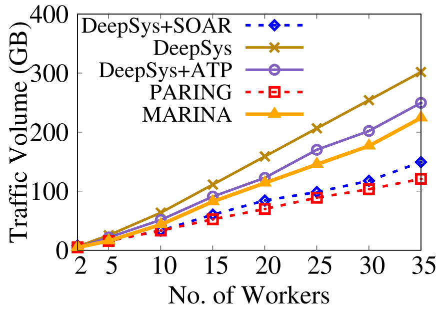


Fig. 2. Traffic volume vs. no. of workers.

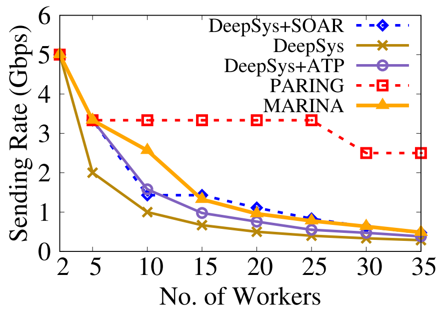


Fig. 3. Sending rate vs. no. of workers.

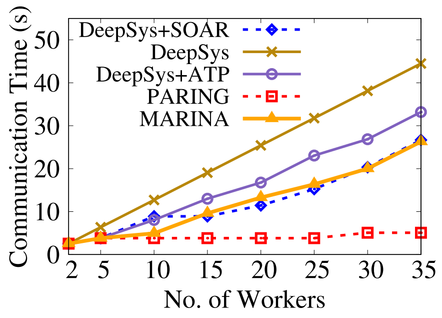


Fig. 4. Communication time vs. no. of workers.

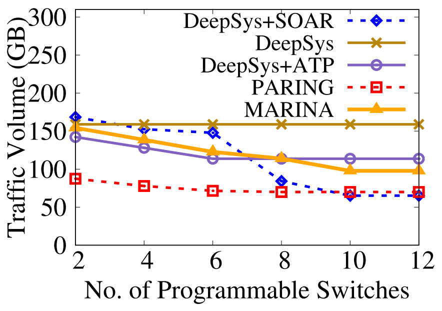


Fig. 5. Traffic volume vs. no. of programmable switches.

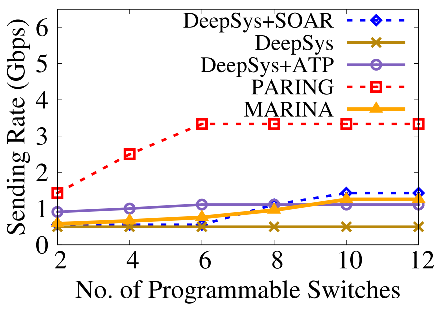


Fig. 6. Sending rate vs. no. of programmable switches.

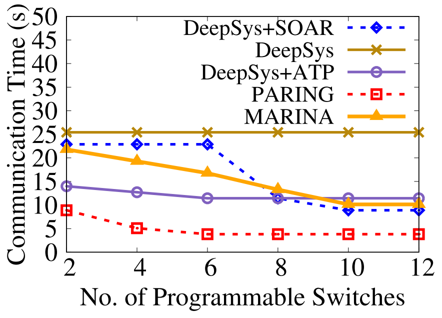


Fig. 7. Communication time vs. no. of programmable switches.

that as the number of workers grows, all solutions show a decrease in the gradient sending rate. This is because when there are too many workers, the sum of the magnitude of the gradients exceeds the aggregation capacity of the cluster, and a slowdown is required to avoid congestion. Also, PARING always outperforms other benchmarks in gradient sending rate. As a result, in Fig. 4, given 35 workers in the cluster, our algorithm reduces 88.6%, 81.0%, 85.0%, and 80.7% of the communication time compared to DeepSys, DeepSys+SOAR, DeepSys+ATP, and MARINA respectively. This is because PARING is able to distribute the aggregation widely across the programmable switches in the cluster.

The second group of simulations studies the effect of aggregation resources (*i.e.*, the number of available programmable switches) in the cluster. In this set of simulations, the number of workers n is 20 and the upper bound of the gradient sending rate R is 5 Gbps. To demonstrate different cluster configurations, we set the first six available programmable switches to be edge switches. To demonstrate the effect of having different aggregation resource configurations in the cluster, we change the number of available programmable switches by setting the first six of them to be edge switches, and the remaining to be aggregation switches in the fat-tree topology. The results are shown in Figs. 5-7. By Fig. 5, given increasing number of available programmable switches in the cluster, since more traffic is aggregated in advance, except for DeepSys where the total amount of traffic is constant, the metrics for the other three solutions gradually decrease. For instance, given two available programmable switches in the cluster, the total traffic volume for PARING, DeepSys+SOAR, DeepSys+ATP, DeepSys and MARINA is 87.3 GB, 168.4 GB, 142.3 GB, 158.9 GB, 154.4 GB, respectively. This implies that PARING outperforms the three benchmarks by 48.2%, 38.7%, 45.1%, and 43.5%, respectively. This is attributed to the joint optimization of the assignment of workers and the PS as well as routing for in-network aggregation. Given twelve available programmable switches in the cluster, the total traffic volume for PARING, DeepSys+SOAR, DeepSys+ATP, DeepSys and MARINA is 69.9 GB, 65.1 GB, 71.6 GB, 158.9 GB, 97.7 GB, respectively. PARING only increases the traffic by 7.4% compared to DeepSys+SOAR, but it is still 28.5% optimized over MARINA and 2.4% over DeepSys+ATP. This is because less improvement in placement optimization will achieve when there are enough aggregation resources. In Figs. 6-7,

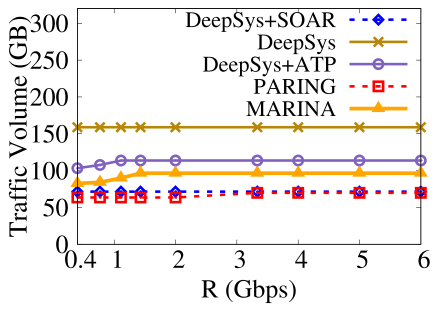


Fig. 8. Traffic volume vs. R.

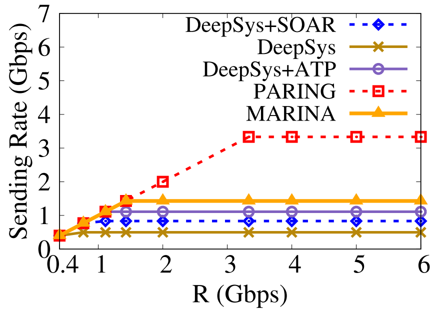


Fig. 9. Sending rate vs. R.

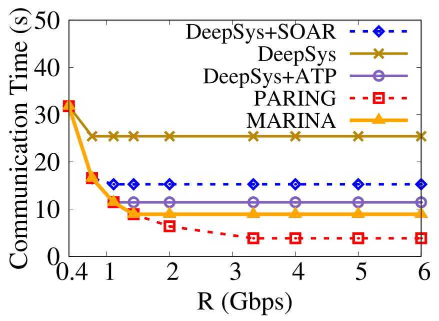


Fig. 10. Communication time vs. R.

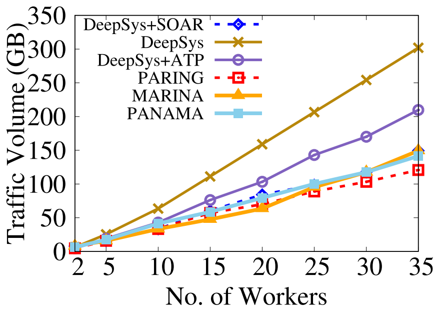


Fig. 11. Traffic volume vs. no. of workers.

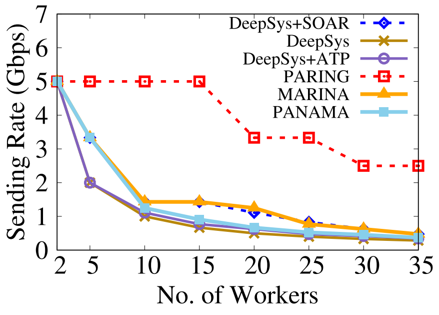


Fig. 12. Sending rate vs. no. of workers.

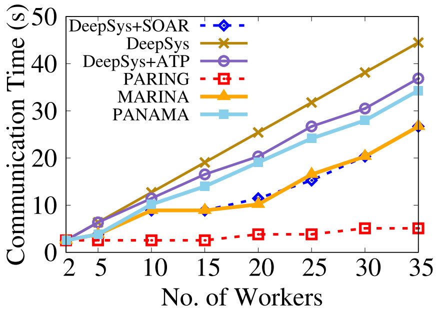


Fig. 13. Communication time vs. no. of workers.

given increasing number of programmable switches in the cluster, since the additional aggregating resources make it possible to perform line-rate aggregation at a higher rate, the communication time of the three in-network aggregation solutions decreases. The gradient sending rate of PARING improves over DeepSys+SOAR, DeepSys+ATP, DeepSys and MARINA by 157.1%, 57.2%, 185.7%, and 145.0% given two available programmable switches; 133.3%, 200.0%, 566.7%, and 165.7% given twelve available programmable switches, respectively.

The third set of simulations shows the influence of R (*i.e.*, the upper bound of gradient sending rate) on the performance. In this set of simulations, we place a DT task with 20 workers in a cluster with 8 available programmable switches. The results are shown in Figs. 8-10. From Fig. 8-10, we can conclude that the influence on traffic volume is minimal. PARING reduces traffic volume by 11.1%, 38.5%, 60.0% and 23.1% compared to DeepSys+SOAR, DeepSys+ATP, DeepSys and MARINA respectively when R is 0.4 Gbps; 2.2%, 38.5%, 56.0% and 27.6% when R is 6 Gbps. Figs. 9-10 show that, with the increase of R, the gradient sending rates for four solutions first increase and then level out, resulting in a decrease and later constant communication time. PARING always shows better performance compared to the three benchmarks, especially when the R is large. For example, when R is 6 Gbps, PARING reduces communication time 75.0%, 66.7%, 85.0%, and 57.1% over DeepSys+SOAR, DeepSys+ATP, DeepSys and MARINA respectively. This is because PARING shows more potential in the acceleration of DT tasks when the upper bound of the gradient sending rate is no longer an obstacle.

3) Performance Comparison With Coarse-Grained Routing: Here we run three other sets of experiments when programmable switches either aggregate all of the gradients or forward them all.

In the fourth group of simulations, we focus on the effect of the DT task scale, as in the first group. The results are presented in Figs. 11-13. When the number of workers is not very large, *e.g.*, 15, this set of simulations shows a greater performance advantage of our algorithm than the first set, where the gradient sending rate of PARING, DeepSys+SOAR, DeepSys+ATP, DeepSys, MARINA and

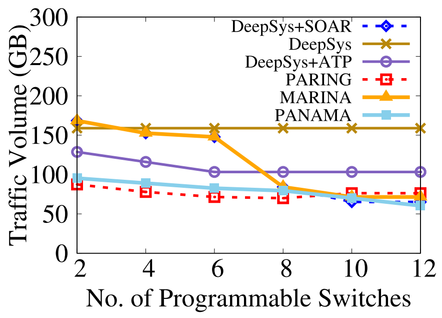


Fig. 14. Traffic volume vs. no. of programmable switches.

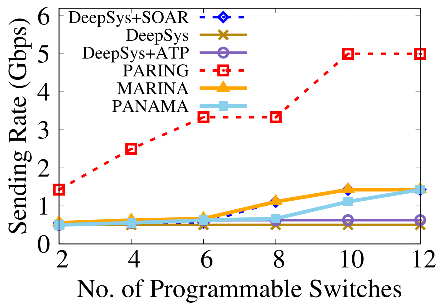


Fig. 15. Sending rate vs. no. of programmable switches.

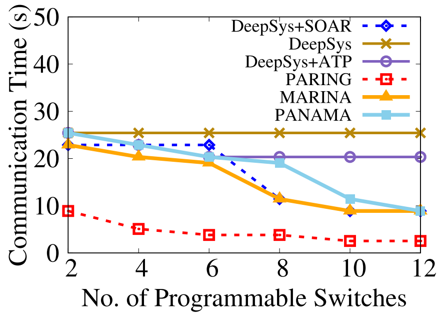


Fig. 16. Communication time vs. no. of programmable switches.

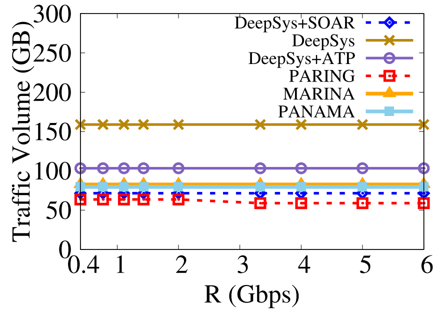


Fig. 17. Traffic volume vs. R.

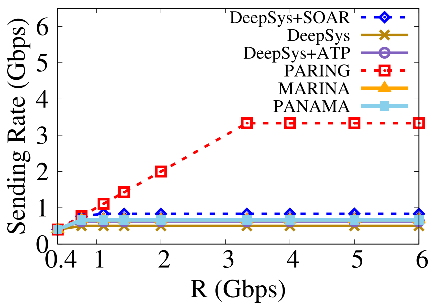


Fig. 18. Sending rate vs. R.

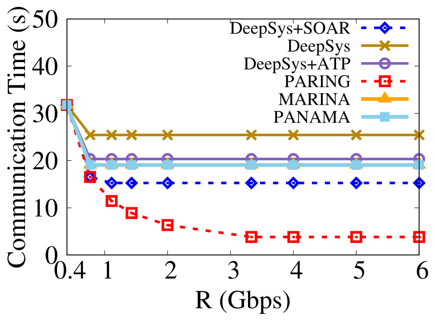


Fig. 19. Communication time vs. R.

PANAMA is 5.0 Gbps, 1.4 Gbps, 0.8 GB, 0.7 GB, 1.4 Gbps, and 0.9 Gbps, respectively, and the traffic volume is 57.2 GB, 60.4 GB, 76.3 GB, 111.2 GB, 47.7 GB, and 58.8 GB, respectively. This means that PARING has an improvement of 257.1% to 614.2% in the gradient sending rate compared to the other in-network aggregation solutions, while maintaining little difference in traffic volume. In addition, PARING reduces the traffic volume by 48.6% and improves the gradient sending rate by 614.3% relative to DeepSys. As a result, the optimization of communication time is significant in Fig. 13, with a 71.4%, 84.6%, 86.7%, 71.4%, and 81.8% improvement relative to DeepSys+SOAR, DeepSys+ATP, DeepSys, MARINA and PANAMA, respectively.

The fifth group of simulations shares the setting with the second group, with the difference being the coarse-grained routing. When more programmable switches are available, *e.g.*, 12, this set of simulations shows a performance advantage of PARING over the second set. At this time, the gradient sending rate of PARING, DeepSys+SOAR, DeepSys+ATP, DeepSys, MARINA and PANAMA is 5.0 Gbps, 1.4 Gbps, 0.6 Gbps, 0.5 Gbps, 1.4 Gbps, 1.4 Gbps respectively; the total traffic volume is 76.3 GB, 65.1 GB, 103.3GB, 158.9 GB, 71.5 GB, 60.4 GB respectively. PARING has a gradient sending rate improvement of 257.1% to 700.0% over the other INA benchmarks, while merely increasing the traffic volume up to 26.3%. Meanwhile, given twelve available programmable switches in the cluster, PARING reduces the traffic volume by 52.0% and improves the gradient sending rate by 900.0% relative to DeepSys. As can be observed in Fig. 16, PARING has 71.4%, 87.5%, 90.0%, 71.4%, and 71.4% communication time advantages over the five benchmarks. While, in Fig. 7, PARING has only 57.1%, 66.7%, 85.0%, and 62.4% communication time improvement relative to DeepSys+SOAR, DeepSys+ATP, DeepSys, and MARINA. This is because joint optimization of DT task placement and INA routing allows PARING to perform well even under coarse-grained routing.

The sixth group of simulations shares the setting with the third one, in order to study the influence of R. The results are in Figs. 17-19. Similar trends are shown to the third group, since PARING can still take more advantage of the upper bound of gradient sending rate. The influence on traffic volume is also minimal, which can be seen from Fig. 17. PARING reduces the traffic volume by 11.1%, 38.5%, 60.0%, 23.1% and 20.2% compared to DeepSys+SOAR, DeepSys+ATP, DeepSys, MARINA and PANAMA respectively when R is

0.4 Gbps; 17.8%, 43.1%, 63.0%, 28.8% and 26.0% when R is 6 Gbps. From Figs. 18-19 we can observe that PARING can take more advantage of the increasing upper bound of gradient sending rate, and achieve a reduction of communication time by 75.0%, 81.3%, 85.0%, 80.0% and 80.0% over DeepSys+SOAR, DeepSys+ATP, DeepSys, MARINA and PANAMA respectively.

From these simulation results, we can draw some conclusions. First, Figs. 2-4, 11-13 show that compared to the five benchmarks, PARING can always achieve the best gradient sending rate while guaranteeing the traffic volume. Second, from Figs. 5-7, 14-16, it can be seen that PARING is able to take full advantage of the available programmable switches in the cluster, providing a more significant traffic volume and communication time improvement, especially when aggregation resources are scarce. Finally, from Figs. 8-10, 17-19 we can conclude that PARING can show more potential in accelerating DT tasks when the upper bound of gradient sending rate is no longer an obstacle.

VI. Related Works

Distributed machine learning [32] can generally be divided into two phases: gradient computation and gradient update. For a typical PS architecture [33], after workers process the dispatched batch of data in the gradient computation phase, PS(s) collect the gradients from the workers, aggregate them and return the results for the next iteration in the gradient update phase. Plenty of works [8], [9], [10], [12] have shown that the bottleneck of distributed model training mainly lies in communication. For a VGG19 job [34], the gradient size reaches 548 MB, and 73% of the time is spent in communication at 10 Gbps Ethernet [8]. In this paper, we focus on addressing the communication bottleneck with a lot of related work on hardware optimization, model optimization, DT task placement and in-network aggregation.

Hardware Optimization. Network hardware optimization, especially expanding link bandwidth, can alleviate the communication bottleneck. Plenty of works [35], [36], [37], [38] showed the benefits of using optical links for better communication support in the datacenter. For example, SiP-ML [39] used silicon photonic links capable of providing multiple terabits-per-second of bandwidth, and designed corresponding task placement method to achieve acceleration by $1.3-9.1\times$. However, the drawback is that the optimization of the hardware is too costly [35].

Model Optimization. Some earlier works modified the model of DT tasks to achieve acceleration. For example, P3 [40] modified several layers in MXNet, while TicTac [41] modified TensorFlow and its PS implementation. Some other works focused on optimizing the data flow without altering the training model, such as the parameter structure [40], [42]. Furthermore, [43] and [44] dropped non-essential parameters during communication, and [45] enlightened a way to achieve efficient data transfer with gradient quantization. The work in [26] made it possible to automatically adjust the hyper-parameters of the model while training. Gradient compression [3], [10], [46], [47] is also an effective way of acceleration. OmniReduce [10] maximized effective bandwidth by exploiting the sparsity of gradient data and sending only non-zero data blocks. However, these works are difficult to generalize for other models and will lead to a loss of accuracy.

DT Task Placement. Task placement is also effective when optimizing distributed model training. Earlier works focused on the optimization of traffic scheduling for DT tasks. Efficient traffic schedulers [26], [41], [48], [49], as well as communication models [6], [50], [51], were designed to alleviate the bottleneck in communication. Geryon [49] designed a priority-based traffic scheduling model. BlueConnect [50] was proposed as an efficient communication library optimized for GPU-based platforms.

Some earlier works aimed at the overall optimization of task placement. For example, NEAT [21] constructed a task scheduling framework with a completion time predictor. DeepSys [18] presented a GPU cluster scheduler tailored for DDL jobs including a speed model to predict accurate training speed, and a memory model for high-quality resource utilization. The works above were based on the assumption of no correlation between tasks. When there is interdependency between tasks, [52] showed a way to optimize transmission time when placing tasks in the presence of task dependencies. Harmony [19] designed a deep learning-driven ML cluster scheduler that minimizes interference and maximizes performance. However, DT task placement actually does not reduce the amount of data transmission, so the communication bottleneck is not well addressed, especially on the PS side.

In-network Aggregation. As an emerging technology, in-network aggregation is based on the idea of offloading part of the gradient aggregation to network devices, which puts less pressure on servers. The traffic volume over the network is reduced, thus alleviating communication bottlenecks and accelerating distributed machine learning. In-network aggregation can be implemented on servers [9], [53], [54] or programmable switches [8], [12], [13], [55] in a cluster. PHub [9] co-designed the PS software and hardware to accelerate rack-level and hierarchical cross-rack parameter exchange. NetAgg [53] exploited a middlebox-like design, which connects network switches to dedicated servers with high-bandwidth links. Camdoop [54] designed a MapReduce-like system, and implemented in-network aggregation in the shuffle phase on CamCube servers.

However, server-based in-network aggregation has a high bandwidth requirement and low scalability. Recently, the rapid development of programmable switches (including P4-based [14], [56], FPGA-based [57]) have made it possible to perform in-network aggregation on programmable switches. The P4-based programmable switch is a welcome option. SwitchML [8] optimized in the rack-scale network, and offloaded part of the aggregation tasks to the top-of-rack (ToR) switches. SOAR [29] proposed an algorithm for programmable switch placement to minimize network utilization. GOAT [58] proposed an optimized gradient scheduling method with collaborative INA for asynchronously arriving gradients. FPGA-based programmable switches have shown more potential for computation and storage. SHArP [55] implemented in-network aggregation based on Mellanox's SwitchIB-2 ASIC. iSwitch [13] used real-world programmable switch NetFPGA boards to support in-network aggregation in reinforcement learning scenarios. PANAMA [11] designed an FPGA-based in-network aggregation accelerator and a corresponding routing and load-balancing protocol.

Unfortunately, none of the above works on in-network aggregation considers DT task placement, resulting in inefficient optimization in distributed model training.

Though the above four categories of works help to accelerate distributed training, they have their own limitations. Specifically, hardware optimization is too costly, while model optimization is only for specific models and will lead to a loss of accuracy. DT task placement does not efficiently reduce the traffic volume, and in-network aggregation is not able to make full use of the capacity of network devices due to ignoring task placement strategy. We have noticed a chance of joint optimization, and try to take advantage of both DT task placement and in-network aggregation in this paper.

VII. Conclusion

In this paper, we have proposed PARING, the first-of-its-kind work on jointly optimizing both DT task placement and routing with INA in order to accelerate distributed raining. We have formulated the PARING program as a nonlinear multi-objective mixed-integer programming problem and proved its NP-Hardness. An approximate algorithm based on the concept of Steiner trees has been proposed to solve this problem. Large-scale simulation results have shown that PARING can efficiently minimize traffic volume and reduce communication time.

References

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

[2] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 1–38, Jan. 2020.

[3] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017, *arXiv:1712.01887*.

[4] S. Li et al., "PyTorch distributed: Experiences on accelerating data parallel training," 2020, *arXiv:2006.15704*.

[5] J. Dean et al., "Large scale distributed deep networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2012, pp. 1223–1231.

[6] Y. Jiang, Y. Zhu, C. Lan, B. Yi, Y. Cui, and C. Guo, "A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters," in *Proc. 14th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2020, pp. 463–479.

[7] Y. Huang et al., "FlexPS: Flexible parallelism control in parameter server architecture," *Proc. VLDB Endowment*, vol. 11, no. 5, pp. 566–579, Jan. 2018.

[8] A. Sapio et al., "Scaling distributed machine learning with in-network aggregation," 2019, *arXiv:1903.06701*.

[9] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, "Parameter hub: A rack-scale parameter server for distributed deep neural network training," in *Proc. ACM Symp. Cloud Comput.*, Oct. 2018, pp. 41–54.

[10] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," in *Proc. ACM SIGCOMM Conf.*, Aug. 2021, pp. 676–691.

[11] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," in *Proc. Mach. Learn. Syst.*, vol. 3, 2021, pp. 829–844.

[12] C. Lao et al., "ATP: In-network aggregation for multi-tenant learning," in *Proc. 18th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2021, pp. 741–761.

[13] Y. Li, I. Liu, Y. Yuan, D. Chen, A. Schwing, and J. Huang, "Accelerating distributed reinforcement learning with in-switch computing," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2019, pp. 279–291.

[14] *Intel Tofino*. Accessed: Oct. 20, 2021. [Online]. Available: https://www.intel.cn/content/www/cn/zh/products/networkio/programmable-ethernet-switch/tofino-2-series.html

[15] *32-Port Programmable Switch*. Accessed: Jun. 13, 2022. [Online]. Available: https://newwavedv.com/products/appliances/32-port-programmable-switch/

[16] M. Li, D. G. Andersen, A. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, vol. 1, 2014, pp. 19–27.

[17] L. Zheng et al., "Alpa: Automating inter- and intra-operator parallelism for distributed deep learning," 2022, *arXiv:2201.12023*.

[18] Q. Li, J. Xu, and C. Cao, "Scheduling distributed deep learning jobs in heterogeneous cluster with placement awareness," in *Proc. 12th Asia–Pacific Symp. Internetware*, Nov. 2020, pp. 217–228.

[19] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2019, pp. 505–513.

[20] Y. Zhao, C. Tian, J. Fan, T. Guan, and C. Qiao, "RPC: Joint online reducer placement and coflow bandwidth scheduling for clusters," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 187–197.

[21] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling aware task placement in datacenters," in *Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2016, pp. 221–235.

[22] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "GRID: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Trans. Netw.*, vol. 31, no. 5, pp. 2267–2280, 2023.

[23] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, "Approximation algorithms for degree-constrained minimum-cost network-design problems," *Algorithmica*, vol. 31, no. 1, pp. 58–78, Sep. 2001.

[24] P. Klein and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," in *Proc. 3rd MPS Conf. Integer. Program. Combination Optim.*, 1993, pp. 323–332.

[25] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*. New York, NK, USA: Wiley, 1998.

[26] Y. Peng et al., "A generic communication scheduler for distributed DNN training acceleration," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 16–29.

[27] L. Zhang, S. Shi, X. Chu, W. Wang, B. Li, and C. Liu, "DeAR: Accelerating distributed deep learning with fine-grained all-reduce pipelining," in *Proc. IEEE 43rd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2023, pp. 142–153.

[28] P. Bosshart et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 99–110, Oct. 2013.

[29] R. Segal, C. Avin, and G. Scalosub, "SOAR: Minimizing network utilization with bounded in-network computing," in *Proc. 17th Int. Conf. Emerg. Netw. EXperiments Technol.*, Dec. 2021, pp. 16–29.

[30] S. Han, H. Lee, S. Han, H. Kim, and S. Pack, "Multi-tenancy- and redundancy-aware in-network aggregation using programmable switches," *IEEE Netw.*, vol. 37, no. 3, pp. 94–100, 2023.

[31] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

[32] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–33, 2020.

[33] M. Li et al., "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2014, pp. 583–598.

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

[35] L. Chen et al., "Enabling wide-spread communications on optical fabric with MegaSwitch," in *Proc. 14th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2017, pp. 577–593.

[36] H. Ballani et al., "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., Archit., Protocols Comput. Commun.*, Jul. 2020, pp. 782–797.

[37] W. M. Mellette et al., "RotorNet: A scalable, low-complexity, optical datacenter network," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 267–280.

[38] H. Liu et al., "Scheduling techniques for hybrid circuit/packet networks," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2015, pp. 1–13.

[39] M. Khani et al., "SiP-ML: High-bandwidth optical network interconnects for machine learning training," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 657–675.

[40] W. Wen et al., "TernGrad: Ternary gradients to reduce communication in distributed deep learning," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1508–1518.
[41] S. H. Hashemi, S. A. Jyothi, and R. Campbell, "TicTac: Accelerating distributed deep learning with communication scheduling," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 418–430.
[42] D. Alistarh, D. Grubic, J. Z. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1707–1718.
[43] Y. Zhao, J. Fan, L. Su, T. Song, S. Wang, and C. Qiao, "SNAP: A communication efficient distributed machine learning framework for edge computing," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 584–594.
[44] T. Chen, G. B. Giannakis, T. Sun, and W. Yin, "LAG: Lazily aggregated gradient for communication-efficient distributed learning," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5055–5065.
[45] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017.
[46] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via randomized quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 3, 2018, pp. 1710–1721.
[47] H. Lim, D. G. Andersen, and M. Kaminsky, "3LC: Lightweight and effective traffic compression for distributed machine learning," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 53–64.
[48] A. Jayarajan, J. Wei, G. Gibson, A. Fedorova, and G. Pekhimenko, "Priority-based parameter propagation for distributed DNN training," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 132–145.
[49] S. Wang, D. Li, and J. Geng, "Geryon: Accelerating distributed CNN training by network-level flow scheduling," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 1678–1687.
[50] M. Cho, U. Finkler, D. Kung, and H. Hunter, "BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 241–251.
[51] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ml," in *Proc. Mach. Learn. Syst.*, vol. 2, 2020, pp. 172–186.
[52] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 864–872.
[53] L. Mai et al., "NetAgg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol.*, 2014, pp. 249–262.
[54] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea, "Camdoop: Exploiting in-network aggregation for big data applications," in *Proc. 9th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2012, pp. 29–42.
[55] R. L. Graham et al., "Scalable hierarchical aggregation protocol (SHArP): A hardware architecture for efficient data reduction," in *Proc. 1st Int. Workshop Commun. Optimizations HPC (COMHPC)*, Nov. 2016, pp. 1–10.
[56] *NVIDIA Mellanox*. Accessed: Feb. 28, 2022. [Online]. Available: https://support.mellanox.com/s/productdetails/a2v1T000001JQCQQA4/sn4700
[57] *NetFPGA*. Accessed: Feb. 28, 2022. [Online]. Available: https://netfpga.org/
[58] J. Fang, H. Xu, G. Zhao, Z. Yu, B. Shen, and L. Xie, "Accelerating distributed training with collaborative in-network aggregation," *IEEE/ACM Trans. Netw.*, pp. 1–16, 2024, doi: 10.1109/TNET.2024.3387948.

Yuhang Qiu is currently pursuing the M.S. degree with the School of Computer Science and Technology, University of Science and Technology of China (USTC). His research interests include data center networks, software-defined networking, and distributed training.

Gongming Zhao (Member, IEEE) received the Ph.D. degree in computer software and theory from the University of Science and Technology of China in 2020. He is currently an Associate Professor with the University of Science and Technology of China. His current research interests include cloud computing, software-defined networking, data center networks, and networking for AI.

Hongli Xu (Member, IEEE) received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China (USTC), China, in 2002 and 2007, respectively. He is currently a Professor with the School of Computer Science and Technology, USTC. He has published more than 100 papers in famous journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, International Conference on Computer Communications (INFOCOM), and International Conference on Network Protocols (ICNP). He has held more than 30 patents. His research interests include software-defined networks, edge computing, and the Internet of Things. He received the Outstanding Youth Science Foundation of NSFC in 2018. He has won the best paper award and the best paper candidate in several famous conferences.

He Huang (Senior Member, IEEE) received the Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China (USTC), China, in 2011. From 2019 to 2020, he was a Visiting Research Scholar with Florida University, Gainesville, USA. He is currently a Professor with the School of Computer Science and Technology, Soochow University, China. He has authored more than 100 papers in related international conference proceedings and journals. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a member of the Association for Computing Machinery (ACM). He has served as the Technical Program Committee Member for several conferences, including IEEE INFOCOM, IEEE MASS, IEEE ICC, and IEEE Globecom. He received the Best Paper Awards from Bigcom 2016, IEEE MSN 2018, and Bigcom 2018.

Chunming Qiao (Fellow, IEEE) is currently a SUNY Distinguished Professor and the Chair of the Computer Science and Engineering Department, University at Buffalo. His current focus is on connected and autonomous vehicles and quantum networks. He was elected as an IEEE Fellow for his contributions to optical and wireless network architectures and protocols.