Published in IEEE/ACM Transactions on Networking (Volume: 32, Issue: 5, October 2024)

Releasing the Power of In-Network Aggregation With Aggregator-Aware Routing Optimization

Shouxi Luo, Xiaoyu Yu, Ke Li, Huanlai Xing

***Abstract*—By offloading partial of the aggregation computation from the logical central *parameter servers* to network devices like programmable switches, In-Network Aggregation (INA) is a general, effective, and widely used approach to reduce network load thus alleviating the communication bottlenecks suffered by large-scale distributed training. Given the fact that INA would take effects if and only if associated traffic goes through the same in-network aggregator, the key to taking advantage of INA lies in routing control. However, existing proposals fall short in doing so and thus are far from optimal, since they select routes for INA-supported traffic without comprehensively considering the characteristics, limitations, and requirements of the network environment, aggregator hardware, and distributed training jobs.**

To fill the gap, in this paper, we systematically establish a mathematical model to formulate i) the up-down routing constraints of Clos datacenter networks, ii) the limitations raised by modern programmable switches' pipeline hardware structure, and iii) the various aggregator-aware routing optimization goals required by distributed training tasks under different parallelism strategies. Based on the model, we develop ARO, an Aggregator-aware Routing Optimization solution for INA-accelerated distributed training applications. To be efficient, ARO involves a suite of search space pruning designs, by using the model's characteristics, yielding tens of times improvement in the solving time with trivial performance loss. Extensive experiments show that ARO is able to find near-optimal results for large-scale routing optimization in tens of seconds, achieving 1.8$\sim$4.0$\times$ higher throughput than the state-of-the-art solution.

***Index Terms*—Distributed machine learning, in-network aggregation, routing optimization, programmable switches**

I. Introduction

Nowadays, machine learning (ML), especially deep learning, has demonstrated great capabilities and achieved great success in abundant fields like *machine vision* [1], *natural language processing* [2], *weather prediction* [3], *content generation* [4], and *game playing* [5]. With the development of ML, new advanced models are constantly proposed. Both the size of the model and the scale of the training dataset show explosive growth trends. In order to complete the model training in a reasonable time, distributed machine learning (DML), especially with the paradigm of data parallelism, has become an inevitable design. However, simply increasing the cluster scale to enhance the compute capacity often fails to achieve the corresponding performance improvements. During the data-parallel distributed training, to guarantee the convergence of the global model, training workers have to synchronize their locally trained gradients or updated model parameters periodically [6]. As confirmed by recent studies [7–10], with the training cluster's scale increases, the communication cost of model synchronization gradually becomes a prominent performance bottleneck for the entire training.

Regarding the implementation of model synchronization, the well-known communication architecture of Parameter Server (PS) is widely used [6]. In this architecture, participating nodes are logically divided into workers responsible for distributed model training, and *parameter servers* responsible for the serving and aggregation of model parameters and gradient. To synchronize models with PS, training workers will send the gradient (or updated model parameters) to one or more PSs and then fetch the updated results every one or more epochs of local training. In this process, network communication is usually the bottleneck of the entire distributed training [8, 11]. Thus, reducing the traffic triggered by model synchronization and resolving the bottleneck have become the keys to improving the performance of such DML systems.

Since the calculations involved in the aggregation of gradients and model parameters are usually simple operations like *summation* and *weighted average*, a general, effective, and widely used approach is to offload partial of them to network devices (e.g., programmable switches, middlebox, smart NICs) for in-network aggregation (INA) [7, 10–13]; thereby both the workload of PS and the volume of traffic in the network could be greatly reduced. Following this direction, various solutions, e.g., SwitchML [8], PANAMA [13], ATP [11], and Libra [14], have been proposed, demonstrating the benefits of INA for data-parallel distributed training. Then, a critical and fundamental question is raised: *In an INA-enabled cluster, how to fully release the power of these deployed aggregators to accelerate the aggregation of gradients for training workers, e.g., by maximizing their throughput of gradient uploading?*

Since the prerequisite requirement of performing INA is that the associated traffic passes through the same aggregation devices during the journey, the key to exerting the capability of aggregators deployed in a network lies in routing control. The concept of routing (or path) control (or optimization) here is more than selecting the next hops for a packet just based on its destination address. Compared with schemes (e.g., ATP [11]) that directly reuse the ECMP routing scheme for INA-supported traffic, a fine-grained yet application-aware path control is needed. Fortunately, existing readily deployable explicit routing techniques like OpenFlow [15], SRv6 [16], and

This work was supported in part by NSFC under Projects 62002300 and 62202392, in part by the Fundamental Research Funds for the Central Universities under Project 2682024ZTPY050, and in part by NSFSC under Project 2023NSFSC0459. *(Corresponding author: Shouxi Luo.)*

The authors are with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China, and also with the Engineering Research Center of Sustainable Urban Intelligent Transportation, Ministry of Education, Chengdu 611756, China (e-mail: sxluo@swjtu.edu.cn; yu2022@my.swjtu.edu.cn; keli@swjtu.edu.cn; hxx@swjtu.edu.cn).

XPath [17] have provided such abilities. As we will show in this paper, for INA-supported networks, a poor routing scheme would lead to insufficient use of the aggregation capability of network devices, which is more prominent in scenarios where only a part of the switches support INA.

The recent work of [18] has noted the importance of routing and proposed a solution named GRID. However, it limits each gradient data to be aggregated by in-network aggregators at most once before reaching its PS, thus being unable to make efficient use of available INA capabilities. Another recent work on this topic is AggTree [19], which takes the characteristics of both the network topology and link capacities into account for routing control. But, it builds upon a heuristic algorithm design that generally only finds local optimum, thus far from optimal as well, especially when links have homogeneous capacities. Besides, existing routing optimization schemes also do not explicitly formulate the possible impacts of the pipelined hardware structure of the aggregators [20], suffering from possible performance loss as Section II-B will show.

To fully exploit the power of INA with routing optimization, we identify i) the constraints raised by both the routing principle of datacenter networks and the hardware structure of modern multi-pipelined programmable switches, which are widely used as aggregators, and ii) the attributes that a practical aggregator-aware routing optimization scheme must have. Based on the findings, we accurately formulate the aggregator-aware routing optimization problem for Clos networks, the *de facto* networking architecture for production datacenters [21], as a math model and then design our routing optimization solution ARO. By using commercial off-the-shelf solvers like Gurobi [22], the math model of ARO can be solved efficiently. As ARO could give the optimal results for INA-aware routing optimization, it can be used as the baseline for future solutions.

Compared with existing solutions [11, 18, 19], ARO not only fully encodes the characteristics of the datacenter networks and the constraints of the pipeline structure of aggregator hardware, but also supports various distributed training scenarios. Specifically, by utilizing the topology characteristics of Clos networks, ARO ensures that the found routes always satisfy the well-known up-down routing principles [23, 24]; by formulating the relationship between pipelines explicitly, ARO is aware of the infeasibility of cross-pipeline aggregation, if unsupported [11, 25]; and by providing hyperparameters, ARO allows users to limit the search space size for faster model solving, respecting their requirements—Results show significant solving accelerations with trivial loss of throughput. Beyond supporting the case where a group of workers conduct data-parallel model training using a single PS, ARO could also optimize routes for the scenarios of multiple concurrent aggregation tasks respecting one or more training jobs. Extensive experiments imply that ARO can find near-optimal routing plans for aggregation tasks within tens of seconds using accelerations, even if the network involves 240 switches.

To summarize, our main contributions are four-fold.

- A thorough analysis that identifies the problems stemming from the pipeline structure of modern aggregator hardware along with the challenges for conducting aggregator-aware routing optimization for distributed training in modern datacenter networks (§II);
- ARO, a generic aggregator-aware routing optimization scheme—It not only accurately encodes the constraints of aggregator's pipelined hardware structure and the characteristics of Clos datacenter networks but also supports multi-PSs and multi-jobs distributed training scenarios (§III-A,§III-B,§III-D).
- A suite of acceleration designs that could greatly reduce the time cost of model solving with controllable throughput loss, by taking advantage of the characteristics of both the math model and the Clos network (§III-C).
- Extensive evaluations validating the effectiveness of ARO and demonstrating its significant performance improvements over state-of-the-art solutions (§IV).

In the rest of this paper, Section II first introduces the background and motivation, then, Section III illustrates the design details of ARO, and Section IV evaluates its performance. After discussing the related work in Section V, we finally conclude the article in Section VI.

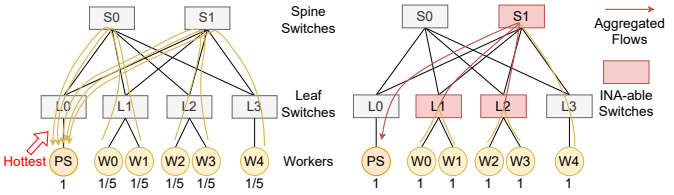
II. Background and Motivation

A. In-Network Aggregation

With the rapid growth of both the size of the training datasets and the number of model parameters, DML techniques, especially data-parallel distributed training, are widely employed nowadays to train models in a reasonable time. To guarantee the coverage of the trained model, workers would periodically synchronize their local training results like the newly generated gradients or updated model parameters. In practice, the well-known *parameter servers* (PS) architecture is widely employed for this purpose [6]. According to their roles, participant servers in PS-based distributed training are logically classified into two types, namely *training workers* and *parameter servers*, respectively. Taking the popular data-parallel distributed training as an example, during the training, workers would iteratively train its local replica of the model with the subset of training data it holds. Every several training epochs, they must send either the generated gradients or updated model parameters to one or more PSs for aggregation, and then fetch the results to drive the next round of training. During such a workflow, the PSs are prone to be the communication bottleneck. For the bottleneck effects faced by the delivery of the global aggregated results, techniques like IP multicasting is able to eliminate duplicated traffic [26–28] and the reverse of the routes planned for aggregation can be used for the delivery. Thus, in this paper, we mainly focus on relieving the efficiency of result upload and aggregation.

Since a PS would complete the aggregation only after it has obtained all the inputs from the workers and the data sent by each worker could be dense and with an identical size, such a process is dominated by the slowest uploading flow. Accordingly, we name *the smallest uploading rate among all workers* as their *throughput*, and aim to optimize the throughput(s) for training tasks in this paper.

Fortunately, as demonstrated by recent studies [8, 11, 13], by upgrading partial or all of the legacy switches with new



(a) Without INA (b) With INA

Fig. 1: Examples showcase the benefits of INA. Without INA, workers would suffer from a low throughput of $\frac{1}{5}$ when uploading their locally trained results to the PS, where the link from L0 to the PS is the most congested under the given routes; while if L1, L2, and S1 support INA, the amount of triggered traffic could be reduced on the way to the PS and all workers would enjoy the uploading throughput of 1.

INA-supported devices that could selectively aggregate associated packets passing by, both the volume of the triggered traffic in the network and the workload of PS(s) could be greatly reduced. Such a design is generic and powerful. For instance, consider that 5 workers are training a deep neural network model in a data-parallel manner with the help of a single PS. These worker nodes along with the PS nodes are networked with 4 leaf switches and 2 spine switches which formulate a typical *leaf-spine* topology. All links have the capacity of 1 unit. Figure 1a shows a possible state of routes for the traffic from workers to the PS when ECMP is used. Because all flows squeeze into the link from L0 to the PS, this link becomes the hottest one across the network, yielding a bottleneck sending rate (e.g., throughput) of $\frac{1}{5}$ unit among all workers, under the bandwidth-sharing principle of per-flow fairness. Distinguished from Figure 1b, when L1, L2, and S1 are INA-enabled, by offloading part of the aggregation computation from the PS to them, the amount of data transmitted in the network can be reduced and all workers can upload gradients at the rate of 1 unit.

B. Multi-Pipelined INA Hardware

Regarding the implementation of INA, the most widely used design today is to use the commercially available data-plane programmable switches as the in-network aggregator. Several recent switch ASICs like Tofino [20] and Trio [12] have provided such abilities and works like SwitchML [8] and ATP [11] have shown successful case designs upon them.

As Figure 2 shows, according to the hardware design, to increase the number of ports while ensuring forwarding efficiency, the data plane of programmable switching ASICs like Tofino is usually divided into multiple pipelines, each of which is responsible for processing packets received at and sent to a portion of the ports [20]. Functionally, each pipeline is made up of two sub-parts in turn, namely, *ingress* and *egress*, respecting their logical locations in the workflow of packet processing. For each incoming packet, after being processed by the ingress pipeline, a traffic manager would bring it to the selected egress pipeline(s) for further processing; and finally, this packet might get dropped, forwarded to the subsequent device(s), or subjected to other operations [20]. To achieve high-performance line-rate packet processing, different pipelines in the ASIC, including both the ingress and egress, do not share memories by design [8, 11, 20].

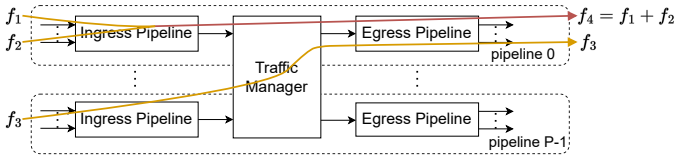


Fig. 2: High-performance programmable switch ASICs like Tofino [20] generally contain multiple hardware pipelines that have independent, unshareable memories. Cross-pipeline aggregation is not natively supported, since the stateful aggregation operation resides in the ingress pipeline by design [8, 11].

In practice, for the sake of performance and design convenience, current switch-based INA solutions like [8, 10, 11, 18, 29] generally run the aggregation logic in the ingress pipeline by design.[1] This makes the associated INA-supported flows that pass through the same switch but different ingress pipelines unable to be directly aggregated since *aggregation* is a stateful operation that generically uses the high-speed memories residing in the hardware pipeline. As the example in Figure 2 shows, for these three flows, f_1 and f_2 can be aggregated since they pass by the same ingress pipeline, generating the aggregated flow of f_4, while f_3 can not participate in the aggregation since it goes through a different ingress pipeline.

Technically, the *recirculation* feature of P4 hardware [20] makes the tasks of implementing cross-pipeline aggregation from scratch and upgrading existing aggregators like ATP [11] to support the goal possible. However, it not only requires non-trivial engineering efforts, making both the implementation of the INA function and the management of INA-supported traffic more complex, but also might cause performance degradation as the data plane has to process each involved packet multiple times [20, 31]. Accordingly, when aggregators are built upon pipelined hardware but without enabling cross-pipeline aggregation, the awareness of the pipeline structure would benefit the routing optimization for the efficient use of INA capacities.

C. Why Existing Routing Schemes Fall Short

Despite abundant INA proposals have been proposed [8, 11], they mainly focus on the problem of how to implement INA functions efficiently and make them ready-deployable, without looking into the impacts of routing schemes on the performance of INA. Indeed, the prerequisite requirement of performing INA is that the associated traffic goes through the same aggregation-enabled network device. As we will show, the unawareness of either aggregators or their pipeline hardware structures in routing control would prevent them from fully releasing INA abilities, resulting in poor performance.

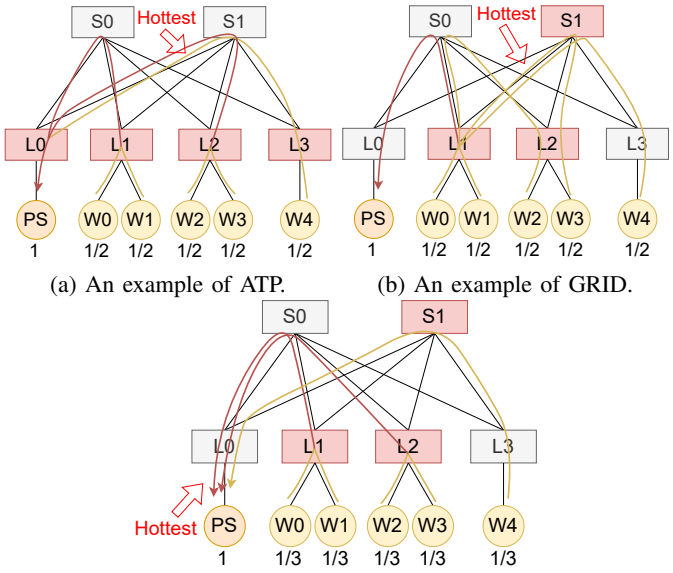
Take the advanced ATP as an example [11]. It directly reuses the default ECMP scheme to route INA-supported traffic and only deploys the aggregation functions on ToR switches.

[1] In some scenarios, network measurement functions (e.g., heavy-hitter detection) might take up most memory of the egress pipeline [30], making it impossible to run INA at egress as well.

Figure 3a shows a possible route status for the uploading flows stemming from workers W0, W1, W2, W3, and W4 to the PS under ECMP. In this instance, despite that all the 4 ToR (Leaf) switches support INA, the aggregated flows from L2 and L3 compete for the bandwidth of the link from S1 to L0, resulting in a bottleneck sending rate of $\frac{1}{2}$. Revisit the example shown in Figure 1b, with just three programmable switches, a well-planned routing could enable all workers to enjoy the sending rate of 1, significantly higher than that ATP obtains. Such a result demonstrates the necessity of supporting non-leaf INA deployments and INA-aware route optimization for the acceleration of gradient aggregation, as a better performance could be obtained even with fewer INA-enabled switches.

GRID [18] and AggTree [19] are two typical publicly reported works on aggregator-aware routing control. The core of GRID is i) formulating the problem of selecting an aggregator or the PS to aggregate the traffic from each worker as a *mixed integer linear programming* (MILP) model, such that workers could have the maximum throughput, then ii) solving it with an algorithm based on *randomized rounding*. However, since the model overlooks many important facts about the underlay datacenter networks, the routes given by GRID are far from optimal and might violate the principles of routing schemes in modern datacenter networks. More specifically, the model of GRID does not encode the feature of datacenter networks, leading to unreasonable routes from workers to the PS(s) with high probabilities. Take the case shown in Figure 3b as an example. GRID may choose L1 as the aggregator for all workers. To achieve this in the given *leaf-spine* topology, the paths from workers W2, W3, and W4 to the PS would go up to a spine first, then down to L1, and then up to the spine S0, and finally down to the PS. Such a route violates the well-known up-down datacenter routing principles [23, 24]. More fundamentally, GRID limits that INA-support traffic should be aggregated by in-network aggregators at most once before reaching the PS, thus being unable to make efficient usage of the aggregation capabilities of available programmable switches. For example, as Figure 1b shows, compared with the result of GRID shown in Figure 3b, by allowing the traffic stemming from workers W0, W1, W2, and W3 to go through multiple aggregators, we could obtain the bottleneck sending rate of 1, rather than $\frac{1}{2}$. Last but not least, the model of GRID overlooks the pipeline structure of the aggregator hardware we have explained in Section II-B [20]. Hence, the routes it plans might not contribute to INA, if cross-pipeline aggregation is not supported by the aggregator.

Different from GRID, AggTree heuristically determines routes for workers one by one [19]. If there are multiple possible paths, it chooses the one with the most available bandwidth to the PS. Here, the maximum available bandwidth a worker would obtain on a link with the original capacity of b, is estimated by $\frac{b}{w}$. w denotes the number of workers that have selected this link to send traffic. For each INA-enabled switch, AggTree also defines an aggregation rate to represent its aggregation capacity and limit the total rates of flows passing through it. Such a heuristic design only guarantees the local optimum of the generated routes, thus are far from optimal. As a result, AggTree is more suitable for the case where there are significant differences in the capacities of links, and the PS's access link has sufficient bandwidth. *Once links have homogeneous capacities or the weight of each path is always dominated by the $\frac{b}{w}$ value of the PS's access link, according to [19], AggTree would degrade into conducting routing controls quite similar to ECMP.* Besides, AggTree also does not explicitly capture the impact of aggregators' pipeline hardware structures, thus suffering from additional performance loss when cross-pipeline aggregation is not supported by the underlying. Because of these drawbacks, it is unable to fully release the power of aggregators. As the example in Figure 3c shows, due to the heuristic routing optimization algorithm designs, workers might only achieve the throughput of $\frac{1}{3}$ under the control of AggTree.



(a) An example of ATP. (b) An example of GRID.

(c) An example of AggTree.

Fig. 3: Examples showcase the drawbacks of existing schemes, where Figures 3a, 3b, and 3c show a possible route state under the control of ATP [11], GRID [18], and AggTree [19], respectively. In Figure 3a, despite that all leaf switches are INA-enabled, workers only achieve a throughput of $\frac{1}{2}$, because of ATP's random and INA-agnostic routing designs. In Figure 3b, all workers might choose L1 as their aggregator under the control of GRID; to enforce this in *leaf-spine* datacenters, paths from workers to the PS would violate the up-down routing principle [23, 24]. In Figure 3c, AggTree selects spine switches for flows from leaf switches in a load-balancing manner, unable to fully leverage the capability of aggregators.

In contrast to these schemes, by routing INA-supported traffic following the paths shown in Figure 1b, which is exactly the results generated by ARO, training workers would obtain the maximum and optimal throughput of 1.

D. Our Work And Design Targets

Based on the above analysis, in this paper, we design an aggregator-aware routing optimization scheme that satisfies the requirements of the up-down principle to fully release the power of deployed aggregators in Clos networks. Currently,

various techniques including SRv6 [16], OpenFlow [15], and XPath [17] have provided ready-deployable ways to setup explicit and flexible routing paths for traffic in modern datacenter networks. Hence, we mainly focus on designing algorithms to obtain optimized routing paths for workers. However, it is quite challenging to achieve this goal, since the proposed solution must possess the following attributes.

- **Expressiveness**. First of all, to make efficient usage of the capacities of deployed aggregators while satisfying the up-down routing principles, besides supporting multiple-stage aggregation, the proposed algorithm should be expressive enough to accurately characterize i) the structure of datacenter networks, ii) the effects of aggregation on the volume of flows, and iii) the constraint of the pipeline structure of aggregator hardware if it involves.
- **Flexibility**. Second, in production, the model parameters of a training job might get sharded among multiple PSs; and moreover, as a shared infrastructure, there might be multiple training jobs using these aggressors at the same time [32, 33]. Thus, the proposed algorithm should be flexible to support multi-PSs and multi-jobs aggregator-aware routing optimization scenarios.
- **Scalability**. Last but not least, in practice, there might be multiple training jobs using the shared infrastructures and the scales of both the distributed training and datacenter networks might be very large. Hence, to be practical, the proposed algorithm should be able to obtain optimized routing results within a reasonable time cost.

III. Aggregator-Aware Routing Optimization

Now, we describe the design of ARO in detail. Motivated by the fact that modern datacenters widely employ *Clos* network topology designs [21], the current version of ARO is specialized in conducting aggregator-aware routing optimization for Clos datacenter networks.

ARO relies on a logical network controller to collect the needed states of the network, including i) the involved links and their bandwidth, ii) the subset of programmable switches that support INA, along with their pipeline structures and port mapping relationships. Then, for distributed training jobs, the controller formulates the problem of finding the routes to maximize the gradient upload throughput of workers as a *Mixed Integer Quadratic Programming* (MIQP), which not only precisely encodes the state of the network and the requirement of the triggered communication task, but also enforces the up-down routing principles on the Clos networks (§III-A, §III-B). The core notations involved in this model are summarized in Table I. By solving the model with acceleration designs, ARO is able to find and construct optimized routes for aggregation tasks within a reasonable time (§III-C, §III-D). Finally, the network controller carries them out for efficient INA, using any of the explicit routing control techniques supported by the underlying network [15–17].

A. Network Model

Unfolding the topology for up-down routing principles. In this paper, we consider the cluster following the switch-centric design, where the routing operations are carried out by switches[2] and both the worker and PS servers lie at the edge and connect to the network with network interface cards. To observe the up-down routing principle and avoid loops, for a given Clos network, ARO unfolds it to form a new directed graph for route planning. Also, to reduce the size of the graph, it only takes the directed shortest paths from workers to the corresponding PSs as the candidates. As an example, Figure 4 shows the result of i) unfolding the *leaf-spine* network, a typical folded Clos topology, used in Figures 1 and 3, and ii) extracting the possible paths for the flows from the workers to their shared PS. Here, each leaf switch, saying L0 for instance, has been unfolded into two nodes, i.e., $L0^{in}$ and $L0^{out}$. Note that, we only keep the shortest path in the graph. Thus, in case there are workers under the same leaf switch with the PS, e.g., L0 in this case, they are directly connected to $L0^{out}$ rather than $L0^{in}$ in the unfolded network.

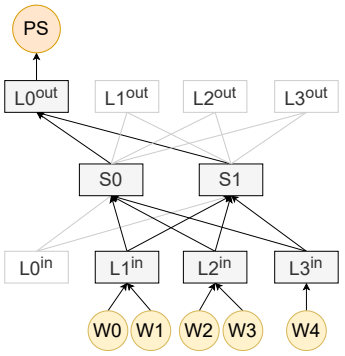


Fig. 4: The unfolded directed graph of the *leaf-spine* network (a well-known folded Clos topology [21]) shown in Figures 1 and 3. To simplify the graph for routing planning, only the directed shortest paths from workers to the corresponding PSs are selected as candidates.

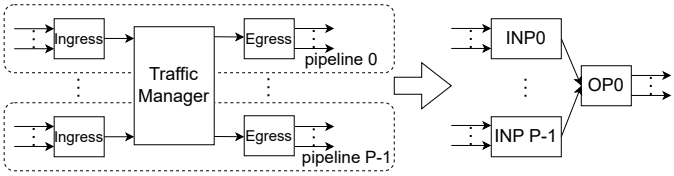


Fig. 5: By decoupling each INA device into P INPs and one abstracted OP, we can capture the limitations raised by the pipeline structure of INA hardware.

Decoupling INA devices to capture pipeline structures. Given that the aggregation operations are generally implemented at the ingress pipeline by design, then, as Figure 5 shows, ARO decouples each INA-supported programmable switch involving P pipelines into P Input Pipelines (INPs) together with one abstracted Output Pipeline (OP), to encode the limitation raised by the pipeline hardware structure. Indeed, even for aggregators that do not employ pipeline-based hardware designs (e.g., Trio [12]), we can just treat them as involving a single flat pipeline (i.e., $P = 1$). Therefore, besides

[2]This paper focuses on planning the forwarding paths for workers and thus treats switches and routers as exchangeable.

workers and PSs, there are other three types of forwarding nodes in the network, i.e., INP, OP, and INA-agnostic common switch (CS), respectively. Their relationships are as follows: i) an INP would receive packets from other CSs, OPs, and workers; it then tries to aggregate multiple related flows into a single one, and forward the flow to the corresponding OP of the same programmable switch, ii) an OP would only receive packets from the INPs of the same programmable switch, and then send them to the next hop(s), which might be INP, CS, or PS, iii) a CS would receive packets from workers and other CSs and OPs, and then forward them to the next hop(s), which might be INP, PS, or CS.

Encoding network states with constants. According to the above definition, in an unfolded directed graph, there are 5 types of nodes in total, namely worker (w), PS (ps), CS (cs), INP (inp), and OP (op), respectively. Given a pair of nodes, saying u and v for instance, we use the binary constant a_u^v to represent whether there is a directed link from u to v; and if so, we use b_u^v to represent its available bandwidth. Note that, according to the definition, in the unfolded and decoupled directed graph, there are several facts: i) a reasonable routing always starts from a worker to the task's PS, without encountering any loops; ii) the next hop of a worker must be either INP or CS, and the pre-hop of a PS must be either CS or OP; and iii) the next hop of an INP must be the corresponding OP, and the pre-hop(s) of an OP must be the corresponding INP(s). Accordingly, there are nine possible types for the value of link parameter (u, v), i.e., (w, cs), (w, inp), (cs, cs), (cs, inp), (cs, ps), (inp, op), (op, inp), (op, cs), and (op, ps). Besides, we also use K_u^v and K_u to denote the set of aggregation tasks that might go through link (u, v) and node u, respectively.

B. Problem Formulation

In this paper, we define the case, in which a group of data-parallel training workers send aggregatable data to the same PS for result aggregation, as an aggregation task. Given that the completion of the aggregation is dominated by the slowest sending rate among workers (referred to as *throughput*), without loss of generality, enforcing all workers to send data at this rate would not defer the completion. Indeed, as aggregator devices like Tofino switches [20] generally have limited cache sizes, workers sending the related data at the same rate (i.e., synchronously) would maximize the effect of INA.

Without loss of generality, for an aggregation task i, we use $N[i]_\kappa$, where $\kappa \in \{w, ps, cs, inp, op\}$, to denote the set of nodes with the type of κ that this task involves, and define variable $r[i]$ to denote its throughput. Obviously, for each aggregation task, it has only one PS in $N[i]_{ps}$ and we refer to it as $D[i]$. In practice, a PS could support multiple training jobs; and similarly, it is possible for a server (worker) to run multiple training jobs at the same time. Thus, different aggregation tasks are not required to have entirely distinct workers and PSs. The above formulation of aggregation tasks is flexible enough to support various training scenarios.

Objectives. According to the number of aggregation tasks and training jobs in the cluster, there are three typical training scenarios for the problem of INA-aware routing optimization.

1) **Single-Job-Single-Task (SJST) routing optimization**: As the simplest case, the objective of planning routes to accelerate the completion of a single aggregation task can be straightforwardly formulated as maximizing its throughput (i.e., the bottleneck sending rate of all workers), using Equation 1.

$$\text{Maximize } r[i] \tag{1}$$

2) **Single-Job-Multiple-Task (SJMT) routing optimization**: In practice, to relieve the bottleneck effect of PS, the model might be sharded among a group of PSs, and these data-parallel training workers could send different parts of their gradients or model parameters to different PSs for better performance. Then, a training job can involve multiple aggregation tasks, and we name this an aggregation job. By letting T_s be the set of aggregation tasks for the training job s, the objective of accelerating the aggregation for training job s by planning routes for involved workers and balancing the workloads among multiple PSs can be formulated as Equation (2). Here, the values of $r[i]$s also give a guideline for the ratio of how the workload should be distributed among these PSs to make efficient use of them.

$$\text{Maximize } \sum_{i \in T_s} r[i] \tag{2}$$

3) **Multiple-Job-Multiple-Task (MJMT) routing optimization**: Furthermore, if there are multiple training jobs in the cluster, to achieve per-job fairness, the target of routing optimization would be maximizing the minimum total throughput for all aggregation jobs, as specified by Equation (3). Here, S refers to the set of all active aggregation jobs, and β_s is a weight representing the model size of job s; μ is a small constant ($0 < \mu \ll 1$) helping ARO making work-conserving bandwidth allocations.

$$\text{Maximize } \min_{\forall s \in S} \beta_s \sum_{i \in T_s} r[i] + \mu \sum_{\forall s \in S} \beta_s \sum_{i \in T_s} r[i] \tag{3}$$

Routing control variables. Due to INA, for aggregation task i, its data transfer rate on a link must be a (non-negative) multiple of $r[i]$, where this multiple denotes the number of flows on this link. Therefore, for a link whose $a_u^v = 1$ and an aggregation task $i \in K_u^v$, we define a non-negative integer variable $y[i]_u^v \in \mathbb{N}^0$ to represent the number of aggregated flows belonging to this task on the link. Indeed, $\{y[i]_u^v : \forall (u, v)\}$ implicitly denote the routing states for this aggregation task i. In the following Section III-D, we will show the details of how ARO generates forwarding paths for workers with an efficient algorithm.

$$y[i]_u^v \in \mathbb{N}^0, \ (u, v, i) \in \{(u, v, i) : \exists a_u^v = 1 \wedge i \in K_u^v\} \tag{4}$$

Constraints. The constraints of our model are as follows.

1) **Constraints of the sending rates of workers**: The worker u of task i must send its data to the connected switch v, which may be a CS or an INP:

TABLE I: Table of notations

Notation	Description
a_u^v	a 0-1 constant indicating whether there is a directed link from node u to node v in the unfolded graph
b_u^v	the available bandwidth of the directed link from node u to node v: i.e., (u, v)
$N[i]_\kappa$	the set of nodes with the type of κ that might be involved by the aggregation task i, where $\kappa \in \{w, ps, inp, op, cs\}$
$D[i]$	the target PS for the aggregation task i, where $N[i]_{ps} = \{D[i]\}$
T_s	the set of aggregation tasks for the training job s
β_s	a weight representing the model size of job s
S	the set of aggregation jobs
K_u	the set of aggregation tasks that might go through the node u
K_u^v	the set of aggregation tasks that might go through the directed link (u, v)
$r[i]$	a variable indicating the throughput, i.e., the bottleneck sending rate of workers, for the aggregation task i
$x[i]_u$	a 0-1 variable, indicating whether the traffic belonging to the aggregation task i goes through the INP u,
$y[i]_u^v$	an integer variable, indicating the ratio of $r[i]$, i.e., the number of the aggregation task i's aggregated flows, on the link (u, v)

$$\sum_{v:\exists a_u^v=1 \wedge i \in K_u^v} y[i]_u^v = 1, \quad u \in N[i]_w, \forall i \tag{5}$$

2) **Constraints of the aggregation on traffic loads**: For task i that may go through INP u, since INP can aggregate multiple associated flows into one, the outgoing sending rate of INP u for task i is either $r[i]$ or 0, depending on whether other workers, OPs, or CSs send data of task i to this INP or not. Hence, for each INP u and task i, we use binary variable $x[i]_u$ to represent whether there is a flow for the task passing through this INP.

$$x[i]_u \in \{0, 1\}, \quad u \in N[i]_{inp}, \forall i \tag{6}$$

$$\frac{1}{M} \sum_{u:\exists a_u^v=1 \wedge i \in K_u^v} y[i]_u^v \le x[i]_v \le \sum_{u:\exists a_u^v=1 \wedge i \in K_u^v} y[i]_u^v, \quad v \in N[i]_{inp}, \forall i \tag{7}$$

$$x[i]_u = \sum_{v:\exists a_u^v=1 \wedge i \in K_u^v} y[i]_u^v, \quad u \in N[i]_{inp}, \forall i \tag{8}$$

Here, M is a constant value larger than the number of available workers in the cluster. Using it, constraints (7) could enforce the value of $x[i]_v$ ($v \in N[i]_{inp}$) to be either 1 or 0, depending on whether there is traffic of task i passing by, or not. Then, the constraint (8) specifies that the outgoing traffic load of task i after the aggregation of INP u should equal to $x[i]_u$, i.e., either 1 or 0. Thus, the impacts of INA on the traffic load have been encoded.

3) **Constraints of the forwarding rates of OPs and CSs**: For both OP and CS nodes, since they do not conduct aggregation, the total receiving rate for an aggression task must be equal to its total sending rate on each node.
By using u and v to denote the possible upstream and downstream nodes for this OP or CS z, respectively, we would have the following constraints.

$$\sum_{u:\exists a_u^z=1 \wedge i \in K_u^z} y[i]_u^z = \sum_{u:\exists a_z^v=1 \wedge i \in K_z^v} y[i]_z^v, \quad z \in N[i]_{op} \cup N[i]_{cs}, \forall i \tag{9}$$

4) **Constraints of link capacities**: Last but not least, for each link (u, v), the total transfer rates of all tasks passing through it should not exceed its capacity b_u^v, i.e., constraints (10).[3] Regarding the "virtual" links from INPs to their OP that we add for decoupling, there is no need to consider their capacity limits. Thus, we have $v \notin \cup_i N[i]_{op}$.

$$\sum_{i \in K_u^v} y[i]_u^v r[i] \le b_u^v, \quad (u, v) \in \{(u, v) : \exists a_u^v = 1 \wedge K_u^v \neq \emptyset \wedge v \notin \cup_i N[i]_{op}\} \tag{10}$$

Among the above three scenarios, SJMT is a specific instance of MJMT and SJST is a specific instance of SJMT in turn; thus, MJMT is the most generic.

C. Efficient Model Solving

So far, we have formally formulated the optimization problem of planning the routes to accelerate the completion of aggregation tasks respecting various scenarios including SJST, SJMT, and MJMT as *Mixed Integer Quadratic Programming* (MIQP). In this model, there are three types of variables for each task i, i.e., $r[i] \ge 0$, $y[i]_u^v \in \{0, 1\}$, and $x[i]_u \in \{0, 1\}$. For these models, a straightforward solution is directly employing commercial off-the-shelf solvers like Gurobi [22] to solve. However, with the increase in the network scale and the number of tasks, the time cost would grow fast, becoming unacceptable (up to hours and days and even more in our tests). To deal with this issue, we provide three heuristic methods, namely, *staged routing optimization* (SRO), *aggregator pre-pruning* (APP), and *quantized sending rates* (QSR), to accelerate the solving, at the possible expense of tiny performance loss. By default, ARO enables all these optimizations jointly. These acceleration schemes are generic and applicable for the aggregator-aware routing optimization of various Clos networks. In the following, we use the *leaf-spine* networks as concrete examples to explain how they work in detail.

Staged routing optimization (SRO). Given an aggregation task, there might be a group of aggregators, from which,

[3] In some cases, the aggregation of data and the multicast of results might execute in pipeline thus both types of traffic coexist. To support this, we can let the multicast traffic go the converse routes of the aggregation using the same sending rate, and upgrade the left-hand side of the constraints (10) from $\sum_{i \in K_u^v} y[i]_u^v r[i]$ to $\sum_{i \in K_u^v} y[i]_u^v r[i] + \sum_{j \in K_v^u} y[j]_v^u r[j]$.

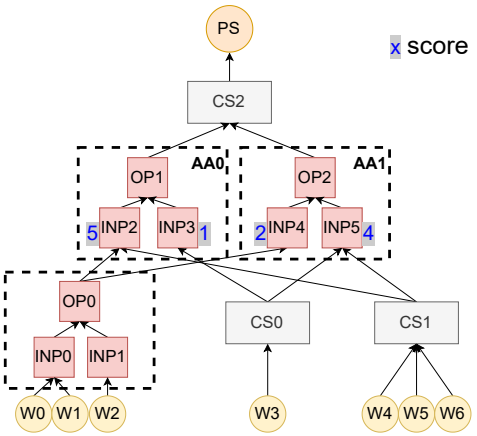


Fig. 6: An example showcases how ARO selects an appropriate candidate aggregator (referred to as AA for short) for aggregation based on the variance of their INPs' *scores*, shown as shaded numbers alongside INPs. Here, INA-supported programmable switches are framed in black dashed boxes. The scores of INP2, INP3, INP4, and INP5 are 5, 1, 2, and 4, respectively, leading to the variances of $(5-\frac{5+1}{2})^2+(1-\frac{5+1}{2})^2=8$ and $(2-\frac{2+4}{2})^2+(4-\frac{2+4}{2})^2=2$ for AA0 and AA1, respectively. Hence, ARO would select AA1 rather than AA0 as the spine aggregator when only one is allowed.

ARO could select one or multiple to conduct INA-accelerated data delivery. We call these aggregators forming an *alternative aggregator* (or AAs for short) group, as they are functionally interchangeable. Following the definition, the smallest AA group can contain just one aggregator. In Clos networks, it is straightforward to split INA-supported switches into AA groups based on their locations. Consider the *leaf-spine* network as an example. In this type of network, ARO needs to determine a spine switch for each cross-leaf flow. If there are multiple INA-supported spine switches in the unfolded graph, they act as a group of AAs. Observably, when there are a large number of aggregators in this AA group, according to the design of ARO, it would try to enforce the flows of an aggregation task to use a part rather than all these AAs to maximize the benefits of INA. Motivated by this, ARO splits the routing optimization into two stages. In the first stage, for each aggregation task, it tries to "estimate" INPs that might not be used. ARO achieves this by 1) just removing all INA-agnostic spine switches along with the involved paths from the graph, then 2) solving the math model in Section III-B to check the $x[i]_u$ values for each INP—"zeros" means no use. In the second stage, ARO excludes these aggregators along with the involved paths from the unfolded graph model to find the finally optimized routes for all tasks. Besides, we can get the minimum value of $r[i]$ in the first stage as the lower bound of the $r[i]$ in the second stage. Since the core of both these two stages is to solve a simplified model, ARO could still obtain a smaller total time cost. For more complex Clos networks like *fat-tree*, there might be multiple groups of AAs for each aggregation task and this optimization is applicable as well.

Aggregator pre-pruning (APP). Given that the number of AAs affects the size of the search space for routing optimization, thus impacting the solving efficiency greatly, a generic yet efficient design is to directly limit the scale of each AA group. Take the case of the *leaf-spine* network as an example again. Supposing that there are n programmable spine switches supporting INA, ARO could only use $\lfloor \alpha n \rfloor$ of them as the candidate AAs for each aggregation task. A straightforward design is to employ random selection. However, such a design suffers from the problem of imbalanced aggregator usages with a high probability, leading to significant performance loss. To avoid this issue, ARO selects these $\lfloor \alpha n \rfloor$ AAs respecting the variances of their INPs' maximum possible traffic loads and in round robin.

Basically, when there is a significant difference in the maximum possible traffic loads (referred to as *scores*) among INPs in an AA, it will increase the bandwidth pressure on INPs with higher scores. Therefore, ARO prefers to keep smaller differences in scores, represented by variance, among INPs under the same AA. Figure 6 shows an example, where ARO prefers to select only one spine switch, out of AA0 and AA1, as the aggregator for the aggregation task. To compute the variance of the INPs' scores for AA0 and AA1, ARO needs to compute the scores of INP2, INP3, INP4, and INP5 first. For INP2, as it might receive data from both OP0 and CS1, its score is the number of flows that OP0 and CS1 may send to it. Since the flows from W0 and W1 are aggregated at INP0, the number of flows that OP0 may send is 2 rather than 3. As a result, the score of INP2 is 2+3=5. Similarly, the scores of INP3, INP4, and INP5 are 1, 2, and 4, respectively. Accordingly, for AA0, its mean score is $\frac{1+5}{2}=3$, yielding the variance of $(5-3)^2+(1-3)^2=8$; and for AA1, its mean score is $\frac{2+4}{2}=3$, yielding the variance of $(2-3)^2+(4-3)^2=2$. Thus, AA1 would be selected.

In the case that there are multiple aggregation tasks i.e., SJMT or MJMT, to reduce the overlapping use of AAs between these tasks, ARO allocate AAs to each task in a round-robin manner. ARO maintains a list with the initial state involving all AAs to generate the $\lfloor \alpha n \rfloor$ candidates for aggregation tasks one by one. At each time, ARO pops out the AA with the least variance from the list for the current task, until $\lfloor \alpha n \rfloor$ AAs have been selected. If the list becomes empty or all left AAs in the list have been selected for the current task, but the allocation has not been over yet, ARO re-initializes the list to continue the generation. Also, it is worth noting that, for other Clos networks rather than *leaf-spine*, to prevent the model from being unsolvable, ARO will add constraints when generating the candidate set. For instance, in fat-tree networks, if all switches in a pod support INA, at each level, at least one switch should be chosen for this pod.

Quantized sending rates (QSR). By default, the variable of the task i's throughput $r[i]$ is defined as a non-negative real number, making the model an MIQP. We experimentally observe that by enforcing the value of $r[i]$ to be chosen only from a limited set of integers, i.e., *using quantized sending rates instead of continuous values*, we can transform the original model into a pure *Integer Quadratic Programming* (IQP). Such a design could achieve a significant improvement

in the solving efficiency, with only trivial performance loss. Generally, there is a trade-off between the level of quantization and the loss of achieved bottleneck sending rate. Thus, to make this tunable, we assume that $r[i] \in \{0, 1, \cdots, N\}$ and update the right-hand value of constraint (10) from b_u^v to $\lceil b_u^v N/\bar{b} \rceil$, where N controls the level of quantization and $\bar{b}$ is the upper bound of all the original links, computed via Equation (11). Then, for each computed integer $r[i]$, it is easy to get the actual throughput (i.e., sending rate) using $r[i]\bar{b}/N$. In this paper, ARO uses 100 as the default value of N.

$$\bar{b} = \max_{(u,v):\exists a_u^v = 1 \wedge v \notin \cup_i N[i]_{op}} b_u^v \tag{11}$$

D. Route Generation

Now, we describe the details of how ARO generates the optimized routes for each aggregation task i based on the solved values of $\{y[i]_u^v : \forall (u,v)\}$.

According to the definition, $y[i]_u^v$ stands for the number of aggregated flows belonging to aggregation task i passing link (u,v). Hence, for each worker s of aggregation task i (i.e., $\forall s \in N[i]_w$), ARO needs to find a path from this worker to its PS $D[i]$ in the unfolded (directed) Clos network, saying l for instance, while ensuring that the $y[i]_u^v$ value of each involved link (u,v) is non-zero, i.e., $y[i]_u^v > 0$ for all $(u,v) = (l[0], l[1]), (l[1], l[2]), \cdots, (l[n-2], l[n-1])$, where $n = len(l)$, denoting the number of nodes in path l. Indeed, thanks to the staged topology structure of Clos networks and shortest-path-based routing designs, there is no loop in the unfolded directed graph. Thus, the well-known *depth-first search* (DFS) algorithm yields an efficient solution to generate paths. Recall that an INP node will aggregate the multiple related flows passing by into a single one. Accordingly, if ARO finds an available path from a worker s to INP u, and there is already a path from u to the PS, the flow stemming from s must reuse this existing subpath starting from u. Thus, for each INP, ARO only needs to maintain one determined path from it to the PS. However, such a characteristic is held neither by CS nor OP, since they might distribute multiple flows to multiple next hops for load balancing.

Putting all the above observations together, Algorithm 1 specifies the details of how ARO generates paths, following the workflow of DFS. For each worker, INP, and PS node, if a path from u to the PS has been founded, ARO would record it in $R[u]$ and reuse it to accelerate the process when necessary (Line 14). To start the search, ARO first initialize the path from the PS $D[i]$ to itself as $[D[i],]$ (Line 1) and sets the value of all other $R[u]$ to nil (Lines 2-4). Then, ARO iteratively finds paths for all workers (Lines 5-22). Here, l is used to record the path to generate. Each time, ARO first checks whether it has reached a node (i.e., the current location u) already having an established path to the PS (Line 8). If so, it directly extends the found path l with $R[u]$ (Line 14); otherwise, it appends u to the tails and continues to find and move to a new node v that has a non-zero $y[i]_u^v$ from its current location u (Lines 9-12). Finally, the path to the PS would be generated. Once the path is estimated, for each INP in the path, saying $l[j]$ for instance, ARO would record the subpath from $l[j]$ to the PS with $R[l[j]]$ (Line 19), which might be used when processing other following workers (Line 14).

In practice, after getting the routes of all workers for each aggregation task, ARO can establish the paths with existing techniques like OpenFlow [15], XPath [17], and SRv6 [16].

Algorithm 1 Generate Routes for Aggregation Task i

Input: $i, \{y[i]_u^v\}, N[i]_w, D[i], N[i]_{inp}$
Output: $\{(s, R[s]) : s \in N[i]_w\}$
1: $R[D[i]] \leftarrow [D[i],]$ ▷ The path from PS D[i] to itself
2: **for** $u \in \{u : \exists y[i]_u^v > 0\}$ **do**
3: $R[u] \leftarrow nil$
4: **end for**
5: **for** $s \in N[i]_w$ **do** ▷ Generate path for each worker
6: $l \leftarrow [\,]$ ▷ Record the route to $D[i]$
7: $u \leftarrow s$ ▷ Current node
8: **while** $R[u] = nil$ **do**
9: l.append(u) ▷ Expand l to the current node u
10: $v \leftarrow \min\{v : y[i]_u^v > 0\}$ ▷ Select next node
11: $y[k]_u^v \leftarrow y[k]_u^v - 1$
12: $u \leftarrow v$ ▷ Move to the next node
13: **end while**
14: l.extend($R[u]$) ▷ u is either INP or PS
15: $R[s] \leftarrow l$ ▷ The generated path from s to $D[i]$
16: $n \leftarrow len(l)$
17: **for** $j \leftarrow 1, 2, \cdots, n-1$ **do**
18: **if** $l[j] \in N_{inp}$ and $R[l[j]] \neq nil$ **then**
19: $R[l[j]] \leftarrow l[j:n]$ ▷ Path from this INP to PS
20: **end if**
21: **end for**
22: **end for**
23: **return** $\{(s, R[s]) : s \in N[i]_w\}$ ▷ Generated paths

IV. Performance

Now, we evaluate the performance of ARO through extensive tests and use the state-of-the-art schemes GRID, ATP, and AggTree as baselines. Extensive results imply that:

1) ARO is effective and flexible. Compared with the state-of-the-art scheme, i.e., AggTree, it achieves about 2.2∼4.0×, 1.8∼2.5×, and 1.9∼2.3× higher throughput for SJST, SJMT, and MJMT scenarios, respectively.
2) ARO is very efficient. Even for scenarios involving hundreds of switches and workers, thanks to the novel acceleration designs, it obtains near-optimal optimized routes within a few minutes.

A. Methodology

Workloads and metrics. Following the settings used in several recent works [10, 18], in tests, we consider that distributed training jobs are executed by a leaf-spine cluster, in which 576 servers are networked with 24 leaf switches (with 24 servers under each leaf) and 24 spine switches. And all links have the same bidirectional capacities of 100Gbps. By default, consistent with the setting of [18], among these 48 switches, $\gamma = 20\%$ randomly selected switches are programmable thus

supporting INA, and each contains 4 hardware pipelines [20]. We mainly consider three types of training scenarios: *SJST*, *SJMT*, and *MJMT*, respectively. Thus, there might be one or multiple aggregation tasks in the cluster, depending on the test settings. By default, in the scenario of SJST, SJMT, and MJMT, we assume that the scale of each aggregation task is 200, 100, and 100 workers, respectively. And for the aggregation task i, we further assume that its PS $D[i]$ is located under the i-the leaf switch which always supports INA. Note that, in the case of SJMT and MJMT, as the training jobs we consider here follow data parallelism distributed training designs, if multiple aggregation tasks belong to the same job, they share the same set of worker servers, but distinct PS nodes. Finally, these exclusive worker servers of all the training jobs are distributed in the cluster uniformly at random.

Regarding the metrics, we mainly use the achieved bottleneck sending rate (i.e., a job's throughput) along with the time cost of route computation to assess the performances of routing optimization algorithms. Note that, in the case of SJMT, where multiple concurrent aggregation tasks T_s belong to the same training job s, their achieved bottleneck sending rate is computed by $\sum_{i \in T_s} r[i]$; and in the case of MJMT, there is a group of training jobs S—their achieved throughput is mainly evaluated by the values: $\min_{s \in S} \beta_s \sum_{i \in T_s} r[i] + \mu \sum_{\forall s \in S} \beta_s \sum_{i \in T_s} r[i]$, where β_s=1 for $s \in S$ and μ=0.001.

Baselines and simulators. Regarding the baselines, we mainly compare ARO with ATP [11] (referred to as RANDOM), GRID [18] and AggTree [19] using the following settings.

- RANDOM. In ATP [11], all leaf switches support INA, and flows are routed randomly, using ECMP. However, in the scenarios we consider in this paper, not all leaf switches support INA, and some spine switches might support INA as well. Thus, we upgrade ATP to select a random programmable spine switch to generate routes and refer this method to as RANDOM hereafter.
- GRID. As for GRID [18], we set its switch processing capacity to a maximum of 3200, and then solve its relaxed LP model with Gurobi. To ensure that the routing paths generated by GRID do not violate the rule of up-down routing [24], for each worker, we let GRID's model only consider its directly connected programmable leaf switches, all programmable spine switches, and the programmable leaf switch connected to the PS, as candidate aggregators. By solving this augmented model, we determine the aggregator for each worker. Given that the model of GRID does not take CS into consideration, we then use a new MILP to compute the practical and optimal routes for workers to maximize their throughput, without considering the pipeline structure. Finally, we incorporate the impacts of pipeline into this route and obtain the actual throughput for aggregation tasks.
- AggTree. Regarding AggTree [19], in our tests, all links have the same capacity; thus, each path's weight is always dominated by that of the last hop (i.e., the PS's access link), and AggTree behaves just like ECMP. Since INA-supported leaf switches might aggregate the flows sent by workers, slightly distinguished from the algorithms specified in [19], we directly select spine switches for (possible aggregated) flows, rather than raw workers in a load-balancing manner to drive experiments.
- Our proposed ARO and its variants. To investigate the benefits and costs of the proposed solving acceleration designs specified in §III-C, in some instances, besides the raw version of ARO (i.e., ARO-RAW), we also employ the accelerated versions of ARO, including ARO-ACC, ARO-SRO, ARO-APP, and ARO-QSR as baselines, which have all or parts of proposed acceleration optimization designs enabled, as specified in Table II. By default, we use $N = 100$ for QSR, $\alpha = 0.6$ for APP, and compare the performances of ARO and ARO-ACC against these of RANDOM, GRID, and AggTree.

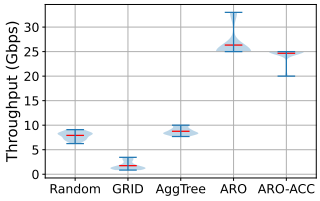
TABLE II: ARO and its variants used in tests

#	SRO	APP	QSR
ARO	✗	✗	✓
ARO-ACC	✓	✓	✓
ARO-SRO	✓	✗	✓
ARO-APP	✗	✓	✓
ARO-RAW	✗	✗	✗
ARO-QSR	✗	✗	✓

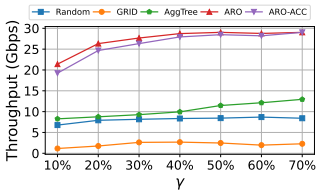
We implement a simulator with Python 3 that could optimize routes for aggression tasks with ARO along with its variants and baselines. For the solving of LP, MILP, MIQP, and IQP models, it directly employs the commercial off-the-shelf Gurobi solver [22]. All experiments are conducted on a desktop PC equipped with an Intel i5-12400 CPU and two 16G memory cards. To reduce the impact of random factors in the setting on the experiment results, we repeat each group of experiments 30 times and report the average values.

B. Effectiveness

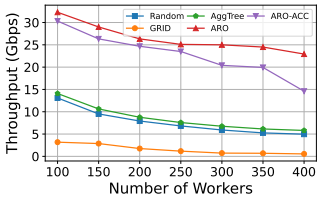
SJST case studies. We first look into the case where there is only one training job containing one single aggregation task in the cluster. Figure 7a shows the details of the throughput achieved by different routing optimization schemes when the aggregation task involves 200 workers, where the red lines in the displayed violins indicate the mean values. Obviously, compared with RANDOM, GRID, and AggTree, whose average throughput is 7.92Gbps, 1.75Gbps, and 8.76Gbps, respectively, the result of ARO is 26.33Gbps, yielding about 3.3×, 15×, and 3× improvements. We also find that GRID generally achieves the worst throughput. There are two main reasons, rooted in GRID's design. Firstly, GRID limits flows to be aggregated at most once, missing opportunities for multiple-stage aggregation. As a contrast, RANDOM, AggTree, and our proposed ARO would not; under their route planning, flows might be aggregated 2 or 3 times when passing the tested *leaf-spine* network. Secondly, GRID tends to use a small number of aggregators, while RANDOM and ARO would distribute flows across all usable aggregators. Results also show that ARO-ACC achieves excellent performance as well—Compared with ARO, its throughput loss is within 6.4%.

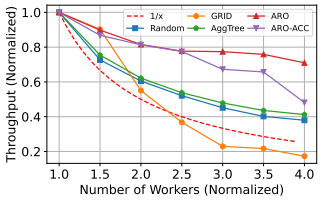


(a) Throughput when $\gamma = 20\%$.



(b) Impacts of aggregator amount.





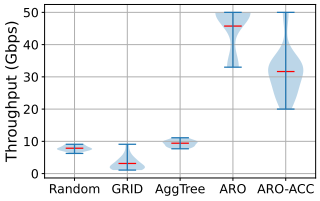
(c) Impacts of the aggregation task's scale.

Fig. 7: [SJST] When there is only one training job containing one aggregation task in the cluster, compared with RANDOM, GRID, and AggTree, ARO could achieve up to $4\times$ higher throughput. Also, the gaps between ARO and ARO-ACC are trivial, within 7%. By default, the aggregation task involves 200 workers and there are 9 INA-supported programmable switches (a.k.a., aggregators) in the network.

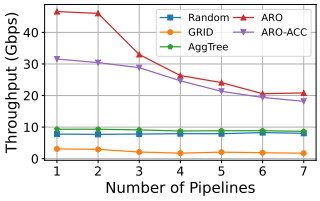
Impacts of aggregator amount. To study the impacts of aggregator amount on the achieved throughput of different schemes, we fix the aggregation task's scale to 200 workers and increase the proportion of INA-enabled switches from 10% to 70%. Figure 7b shows the results. As the number of aggregators increases, the throughput achieved by RANDOM, AggTree, and PRID gradually increases overall, while that achieved by GRID has little change. The reason for this has been mentioned above: GRID will only select a small number of aggregators, so when the number of aggregators reaches a certain value, its impact on GRID is small. We also observe that, as the number of aggregators increases, the performance loss of ARO-ACC relative to ARO is decreasing. Because as the number of aggregators increases, the increase in throughput by aggregators will decrease.

Impacts of aggregation task scale. Figure 7c shows the change in the average throughput achieved by different schemes with the aggregation task's scale (i.e., the number of involved workers) growing from 100 to 400. As expected, for all schemes, the throughput decreases. Ideally, without INA, increasing the scale $g\times$ would make the throughput decrease from 1 to $\frac{1}{g}$. Such a phenomenon is observed in the results of GRID, implying that its routing schemes are far from optimal. Distinguished from this, thanks to the outstanding routing optimization designs, the degradation encountered by ARO is smaller than $\frac{1}{g}$, exhibiting its high efficiency.

Impacts of the number of pipelines (P). Then, we study the impact of the number of hardware pipelines for each aggregator (i.e., P) on the achieved throughput for all schemes. As Figure 8a shows, if each aggregator is made up of a single flat pipeline, i.e., related flows arriving at the same programmable switch would always get aggregated, both



(a) Throughput when $P = 1$.

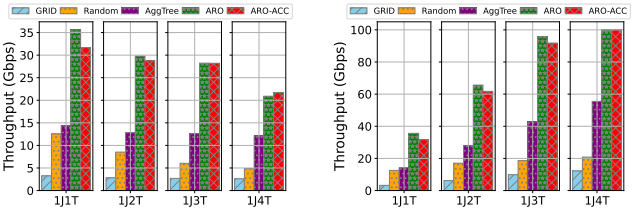


(b) Impacts of P.

Fig. 8: [SJST] ARO always achieves much higher throughput than all other schemes, even under the case where each aggregator is built upon a flat pipeline. With the number of pipelines increasing, the room for routing optimization decreases, leading to tapered throughput for all schemes.

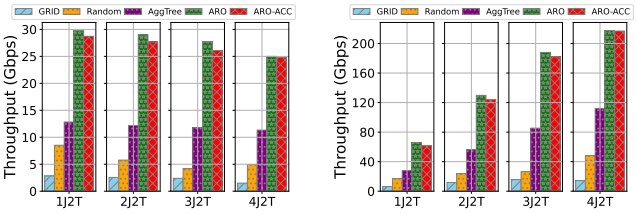
GRID and AggTree are able to achieve higher throughput than the case where each aggregator contains 4 pipelines (i.e., Figure 7a). Even so, they still underperform ARO a lot. Such results not only confirm the excellent effectiveness of ARO on aggregator-aware routing optimization, but also imply that ARO could work very well for the case where the aggregators do not have the pipelined hardware constraints (e.g., [12, 34]). From Figure 8b, we also observe that, with the number of pipelines increasing, the achieved throughput would decrease gracefully for most schemes. This is reasonable. Basically, the increase in the pipelines leads to a decrease in the possibility of in-network aggregation for pipeline-agnostic schemes like GRID and AggTree, and a narrower room for the routing optimization of pipeline-aware schemes like ARO and ARO-ACC. However, RANDOM is an exception, which is reasonable—under this scenario, the bottleneck sending rate among workers is determined by the number of workers under the same common leaf switches, irrelevant to aggregators.

SJMT & MJMT. Now, we evaluate the performance of each scheme in the cases of SJMT and MJMT. For SJMT, we consider that there is a single training job involving 1, 2, 3, and 4 aggregation tasks (e.g.,1J4T), which share the same set of workers. As for MJMT, we test scenarios with 1, 2, 3, and 4 jobs each having 2 aggregation tasks (e.g.,4J2T). For RANDOM, GRID, and AggTree, we calculate routes for each task separately, and finally overlay the routes of all tasks and calculate the throughput. Figures 9a and 9b show the minimum and total throughput among all tasks, respectively. Results indicate that, by considering multiple tasks at the same time, ARO achieves more than twice the improvement compared to other schemes. It is also obvious that, without expectation, the minimum throughput decreases, and the total throughput increases as the number of tasks increases. In the scenario of 1J4T, the minimum throughput achieved by ARO-ACC is more than that achieved by ARO in Figure 9a, which seems abnormal. This is reasonable. According to the current objective design of SJMT (see Eq. 2), ARO optimizes the total throughput of aggregation tasks belonging to the same job without looking into their minimum throughput. Thus, the relationship between the minimum throughput of aggregation tasks achieved by ARO and ARO-ACC is uncertain. As shown in Figure 9b, both ARO and ARO-ACC achieve the upper



(a) Minimum Throughput. (b) Total Throughput.

Fig. 9: [SJMT] In the case of a single job involving multiple aggregation tasks, compared with baselines, ARO achieves more than $2\times$ higher performance in terms of both the minimum and total aggregation task throughput.



(a) Minimum Throughput. (b) Total Throughput.

Fig. 10: [MJMT] As expected and similar to the case of SJMT, ARO significantly outperforms baselines in the scenario that multiple training jobs trigger multiple aggregation tasks.

limit of the total throughput of 100Gbps for all tasks in the scenario of 1J4T. In the case of MJMT, a similar regularity is observed, as Figures 10a and 10b show.

C. Efficiency

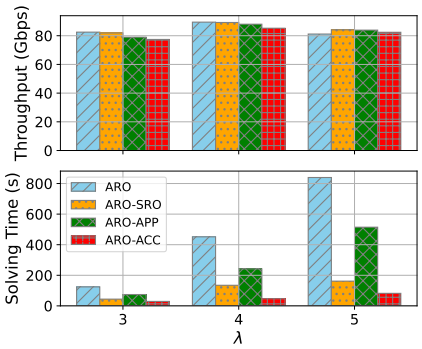


Fig. 11: [SRO&APP] ARO-ACC, which makes joint usage of both SRO and APP, shows the best acceleration effect, with trivial and controllable throughput loss.

Effects of SRO & APP. To study the effects of both *staged routing optimization* (SRO) and *aggregator pre-pruning* (APP), we consider the case where the cluster involves 48λ switches (50%/50% for leaf and spine switches, respectively), with 20% supporting INA and there are 5 concurrent training jobs, each launching an aggregation task involving 100λ workers. To ensure that Gurobi would terminate the solving in a given time, in tests, we set its time limit parameter to

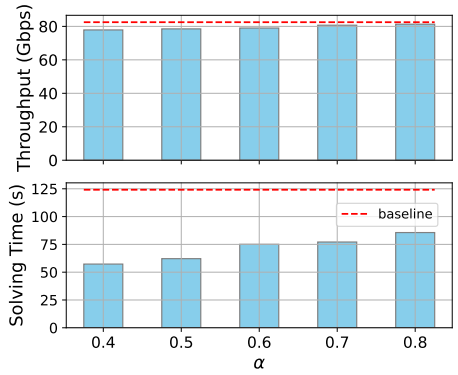
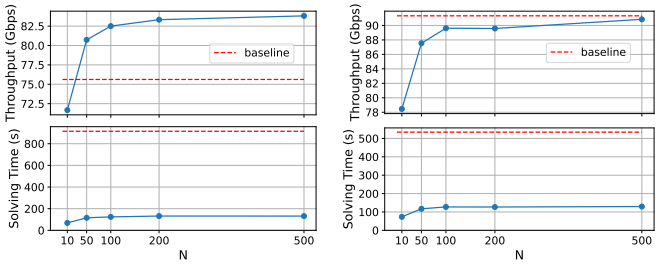


Fig. 12: [APP] The impacts of α on the effects of *aggregator pre-pruning* show that i) the achieved throughput of ARO is not very sensitive to the setting of α, and ii) $\alpha = 0.6$ is good enough for the acceleration of model solving in many cases.



(a) All 30 scenarios. (b) 13 non-timeout scenarios.

Fig. 13: [QSR] For the acceleration design of *qunantized sending rate*, in general, a larger N leads to a less-yet-decaying loss of achieved throughput with the cost of a longer-yet-decaying solving time; we argue that $N = 100$ would be good enough in practice since it would accelerate the model solving about $4\times$, with less than 2% throughput loss.

20 minutes, i.e., TimeLimit=1200 [22]. Figure 11 shows the performances of ARO and its variants when $\lambda = 3, 4, 5$, respectively. We find that i) the design of both SRO and APP could accelerate the model solving of ARO markedly, with only tiny throughput losses, and ii) making a joint usage of them, i.e., ARO-ACC, yields the best performance. Compared to ARO, ARO-ACC could reduce the solving time by about 90% within less than 10% throughput loss. Especially, in the case of λ=5, the performance of ARO-SRO, ARO-APP, and ARO-ACC can exceed that of ARO, because ARO may fail to obtain the optimal result within the limited time. In addition, we also test the impact of α on the effects of APP. As Figure 12 shows, with the value α growing, the number of aggregators available for each task increases, leading to slightly longer solving time and higher throughput. It can be observed that when $\alpha = 0.4$, compared to ARO, the amount of degraded throughput for ARO-ACC is only about 5%. Hence, in our tests, just letting $\alpha = 0.6$ would bring excessive spine aggregators for the optimization of routing.

Effects of QSR. Last but not least, we investigate the impacts of N, the level of quantization, on the effects of quantized

sending rate (QSR). It should be mentioned that we set the upper time limit of the model solving to 20 minutes, and ARO-RAW is likely to encounter timeout events because its MIQP model is hard to solve. As we can see in Figure 13a, the throughput of ARO-RAW (i.e., baseline) is far from optimal—because Gurobi has encountered timeout events in 17 out of all the 30 tested scenarios in solve the ARO-RAW's models; even for the fastest scenario, it still takes about 242s for Gurobi to solve the MIQP model of ARO-RAW. Instead, Gurobi is able to find the optimal results for the IQP model of ARO-QSR much faster. For example, when $N = 100$, for the 30 testing scenarios, the maximum and minimum model-solving time of ARO-QSR is 256s and 61s, respectively. Moreover, Figure 7b shows the results when these 17 timeout scenarios are excluded. As expected, with N increasing, the time cost of model solving also grows gracefully with a decaying throughput loss. Once $N > 100$, the amount of lost throughput is negligible, only less than 2% in the tests, with the advantage of reducing the solving time by about 85%.

In summary, the above performance studies have confirmed the excellent performance of ARO. On one hand, it outperforms other schemes in terms of throughput significantly—compared with RANDOM, GRID, and AggTree, it achieves about 2.8∼4.1×, 8.2∼41.6×, and 1.8∼4.0× performance improvements, respectively. On the other, with the acceleration design of SRO, APP, and QSR, its model-solving time can be shortened by 90%, with less than 10% loss of throughput.

V. Related Work

A. INA Solutions

Currently, there are various INA solutions having distinct designs and implementations. For example, SwitchML [8] implements the aggregator at data-plane programmable switches (e.g., Tofino [20]) and uses it to completely replace the role of PS. In contrast, ATP [11] employs switch-based aggregators to conduct INA in a best-effort manner, aiming at reducing the load of PS rather than replacing it. Different from them, Libra [14] observes that during the training of sparse models, there are "hot" parameters having a relatively higher update frequency than others; motivated by this, it designs a sampling mechanism to select such hot parameters out and only aggregates them at programmable switches, thus reducing the demand of the limited data-plane resources [14]. Likewise, for sparse data, OmniReduce [9] designs a transport scheme to only transmit non-zero data blocks, and further, ASK [35] proposes a solution to support key-value stream aggregation. Instead of relying on commercial off-the-shelf data-plane programmable switches, NetReduce [34] and PANAMA [13] explore the idea of using FPGA to implement aggregation and combine them with switches to achieve INA; SHARP [36] and Trio [12] design new switching chips and hardware for INA.

Given that INA devices like programmable switches generally have scarce memories, INAlloc [29] establishes a switch memory management layer along with a friendly schedule interface to support dynamic and consistent memory reallocation for concurrent aggregation tasks. Furthermore, DSA [37] implements priority-based preemptive aggregator allocation to improve the utilization of switch memory and accelerate the completion of jobs. A2TP [10] considers the impacts of limited switch memory on the congestion controls and designs an aggregator-aware in-network aggregation transmission protocol for better aggregation efficiencies. And Kim *et al* [38] envision the possibility of empowering data-plane programmable switches with remote memories.

As INA is able to reduce the traffic volume in the network, SOAR [39] discusses how to deploy a limited number of programmable switches in a determined INA task to minimize the network load. Similarly, SMC [40] aims at alleviating network congestion with aggregator placement.

As a supplementary, in this paper, we focus on designing routing optimization schemes to release the power of deployed aggregators for Clos networks fully. GRID [18] and AggTree [19] are related works targeting this goal. However, as discussed in Section II-C, they are far from optimal. Our proposed ARO is effective and generic. It is not limited to any specific aggregator designs and is thus able to work with all these existing INA solutions jointly.

B. Explicit Routing Control

To implement the routes optimized by ARO in practice, the cluster must support explicit routing controls. Fortunately, there are abundant alternative solutions. For instance, many modern switches have supported OpenFlow [15], which allows the path of each flow to be precisely configured from a logical central controller. Alternatively, emerging switches and routers have wide support of the Segment Routing (SR) techniques like SRv6 (Segment Routing over IPv6) [16]; with SR, the sender of a flow can encapsulate the desired path's waypoint information in the packet header for path controls. Besides, there are many other specialized proposals. For example, XPath [17] computes and allocates optimized IDs to all possible end-to-end paths, then installs their compressed forwarding information into the switch to achieve ID-based forwarding. By taking advantage of the up-down routing of Clos networks, VL2 achieves fine-grained path control using IP-in-IP encapsulation [41].

VI. Conclusions and Future Work

As is known, INA is a generic, effective, and widely used method to alleviate the communication bottlenecks in PS-based distributed ML systems. Obviously, the premise of performing INA is that associated flows pass through the same aggregator during the journey. Thus, the key to releasing the power of deployed aggregators lies in the optimization of routing. However, existing schemes are far from optimal since they ignore the detailed requirements and characteristics of modern datacenter networks. To fill the gap, we systematically analyze the constraints and design goals of providing practical routing optimization for INA-accelerated data-parallel distributed machine learning workloads in modern datacenter networks, and propose the novel solution of ARO. By formally formulating the routing optimization problem as a math model and accelerating the model solving with a suite of novel designs, ARO is able to find optimal or near-optimal solutions

for the routing optimization of INA-supported traffic in Clos networks within a reasonable time.

Recently, various studies [42, 43] have also shown the benefits of employing in-network aggregation for other types of distributed machine learning by using the computing power of edge clouds. In future work, we will investigate how to conduct INA-aware routing optimization for such scenarios.

REFERENCES

[1] K. He, X. Zhang *et al.*, "Deep residual learning for image recognition," in *Proceedings of CVPR*, 2016, pp. 770–778.

[2] J. Devlin, M.-W. Chang *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL*, vol. 1. ACL, Jun. 2019, pp. 4171–4186.

[3] H. Wu, H. Zhou *et al.*, "Interpretable weather forecasting for worldwide stations with a unified deep model," *Nature Machine Intelligence*, vol. 5, no. 6, pp. 602–611, Jun. 2023.

[4] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 428–436, Aug. 2020.

[5] S. F. Gudmundsson, P. Eisen *et al.*, "Human-like playtesting with deep learning," in *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2018, pp. 1–8.

[6] M. Li, D. G. Andersen *et al.*, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th OSDI*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 583–598.

[7] A. Feng, D. Dong *et al.*, "In-network aggregation for data center networks: A survey," *Computer Communications*, vol. 198, pp. 63–76, 2023.

[8] A. Sapio, M. Canini *et al.*, "Scaling distributed machine learning with In-Network aggregation," in *Proceedings of the 18th NSDI*. USENIX Association, Apr. 2021, pp. 785–808.

[9] J. Fei, C.-Y. Ho *et al.*, "Efficient Sparse Collective Communication and Its Application to Accelerate Distributed Deep Learning," in *Proceedings of the ACM SIGCOMM Conference*. ACM, 2021, pp. 676–691.

[10] Z. Li, J. Huang *et al.*, "A2TP: Aggregator-Aware In-Network Aggregation for Multi-Tenant Learning," in *Proceedings of the 18th EuroSys*. New York, NY, USA: ACM, 2023, pp. 639–653.

[11] C. Lao, Y. Le *et al.*, "ATP: In-network aggregation for multi-tenant learning," in *Proceedings of the 18th NSDI*. USENIX Association, Apr. 2021, pp. 741–761.

[12] M. Yang, A. Baban *et al.*, "Using Trio: Juniper Networks' Programmable Chipset - for Emerging in-Network Applications," in *Proceedings of the ACM SIGCOMM Conference*. New York, NY, USA: ACM, 2022, pp. 633–648.

[13] N. Gebara, P. Costa, and M. Ghobadi, "PANAMA: In-network aggregation for shared machine learning clusters," in *Proceedings of MLSys*, April 2021.

[14] H. Pan, P. Cui *et al.*, "Enabling fast and flexible distributed deep learning with programmable switches," *CoRR*, vol. abs/2205.05243v2, 2022.

[15] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward Adaptive and Scalable OpenFlow-SDN Flow Control: A Survey," *IEEE Access*, vol. 7, pp. 107 346–107 379, 2019.

[16] P. L. Ventre, S. Salsano *et al.*, "Segment routing: A comprehensive survey of research activities, standardization efforts, and implementation results," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 182–221, 2021.

[17] S. Hu, K. Chen *et al.*, "Explicit Path Control in Commodity Data Centers: Design and Applications," in *Proceedings of the 12th NSDI*. Oakland, CA: USENIX Association, May 2015, pp. 15–28.

[18] J. Fang, G. Zhao *et al.*, "Grid: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Transactions on Networking*, vol. 31, no. 5, pp. 2267–2280, 2023.

[19] J. Nie and W. Wu, "AggTree: A routing tree with in-network aggregation for distributed training," in *Proceedings of IEEE IPCCC*, 2023, pp. 116–122.

[20] Intel, "$P4_{16}$ Intel® Tofino™ Native Architecture – Public Version (Application Note)," 2021, https://github.com/barefootnetworks/Open-Tofino, Accessed on 2023-10-12.

[21] A. Singh, J. Ong *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," in *Proceedings of the ACM SIGCOMM Conference*. New York, NY, USA: ACM, 2015, pp. 183–197.

[22] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[23] V. Giotsas and S. Zhou, "Valley-free violation in Internet routing — Analysis based on BGP Community data," in *Proceedings of IEEE ICC*, 2012, pp. 1193–1197.

[24] S. Hu, Y. Zhu *et al.*, "Tagger: Practical pfc deadlock prevention in data center networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 889–902, 2019.

[25] "The implementation of atp," 2021, https://github.com/in-ATP/ATP, Accessed on 2023-10-12.

[26] S. Luo, H. Yu *et al.*, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1161–1175, 2020.

[27] S. Luo, H. Xing, and P. Fan, "Softwarized ip multicast in the cloud," *IEEE Network*, vol. 35, no. 6, pp. 233–239, 2021.

[28] S. Luo, H. Xing, and K. Li, "Near-optimal multicast tree construction in leaf-spine data center networks," *IEEE Systems Journal*, vol. 14, no. 2, pp. 2581–2584, 2020.

[29] B. Zhao, C. Liu *et al.*, "Enabling Switch Memory Management for Distributed Training with In-Network Aggregation," in *Proceedings of IEEE INFOCOM*, 2023, pp. 1–10.

[30] M. Chiesa and F. L. Verdi, "Network Monitoring on Multi-Pipe Switches," in *Proceedings of SIGMETRICS*. New York, NY, USA: ACM, 2023, pp. 49–50.

[31] "Can we recirculate packets to specific pipelines?" https://forum.p4.org/t/can-we-recirculate-packets-to-specific-pipelines/1189, Accessed on 2024-05-20.
[32] M. Jeon, S. Venkataraman *et al.*, "Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads," in *Proceedings of the USENIX ATC*. Renton, WA: USENIX Association, Jul. 2019, pp. 947–960.
[33] P. Zhou, X. He *et al.*, "Jpas: Job-progress-aware flow scheduling for deep learning clusters," *Journal of Network and Computer Applications*, vol. 158, p. 102590, 2020.
[34] S. Liu, Q. Wang *et al.*, "In-Network Aggregation with Transport Transparency for Distributed Training," in *Proceedings of the 28th ASPLOS*, vol. 3. New York, NY, USA: ACM, 2023, pp. 376–391.
[35] Y. He, W. Wu *et al.*, "A Generic Service to Provide In-Network Aggregation for Key-Value Streams," in *Proceedings of the 28th ASPLOS*, vol. 2. New York, NY, USA: ACM, 2023, pp. 33–47.
[36] R. L. Graham, L. Levi *et al.*, "Scalable hierarchical aggregation and reduction protocol (sharp)tm streaming-aggregation hardware design and evaluation," in *High Performance Computing: 35th International Conference, ISC High Performance 2020*. Berlin, Heidelberg: Springer-Verlag, June 22–25 2020, pp. 41–59.
[37] H. Wang, Y. Qin *et al.*, "Preemptive switch memory usage to accelerate training jobs with shared in-network aggregation," in *Proceedings of the 31st IEEE ICNP*, 2023, pp. 1–12.
[38] D. Kim, Y. Zhu *et al.*, "Generic external memory for switch data planes," in *Proceedings of the 17th HotNets*. ACM, 2018, pp. 1–7.
[39] R. Segal, C. Avin, and G. Scalosub, "SOAR: Minimizing Network Utilization with Bounded in-Network Computing," in *Proceedings of the 17th CoNEXT*. New York, NY, USA: ACM, 2021, pp. 16–29.
[40] R. Segal, C. Avin, and G. Scalosub, "Constrained In-network Computing with Low Congestion in Datacenter Networks," in *Proceedings of IEEE INFOCOM*, 2022, pp. 1639–1648.
[41] A. Greenberg, J. R. Hamilton *et al.*, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM Conference*. ACM, 2009, pp. 51–62.
[42] S. Luo, P. Fan *et al.*, "Eliminating communication bottlenecks in cross-device federated learning with in-network processing at the edge," in *Proceedings of IEEE ICC*, 2022, pp. 4601–4606.
[43] L. Luo, C. Zhang *et al.*, "Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network," *IEEE Transactions on Services Computing (Early Access)*, pp. 1–14, 2024.

Shouxi Luo (Member, IEEE) received the bachelor's degree in communication engineering and the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 2011 and 2016, respectively. He is currently an Associate Professor with Southwest Jiaotong University. His research interests include data center networks, software-defined networking, and networked systems.

Xiaoyu Yu received the bachelor's degree in software engineering from Southwest Jiaotong University, Chengdu, China, in 2022. Currently, he is pursuing the master's degree in computer science and technology at Southwest Jiaotong University. His research interests include distributed deep learning and networked systems.

Ke Li received the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 2012. She is currently a Lecturer with Southwest Jiaotong University. Her research interests include machine learning, distributed systems, and the Internet of Things.

Huanlai Xing (Member, IEEE) received the B. Eng. degree in communications engineering from Southwest Jiaotong University, Chengdu, China, in 2006, the M. Eng. degree in electromagnetic fields and wavelength technology from the Beijing University of Posts and Telecommunications, Beijing, China, in 2009, and the Ph.D. degree in computer science from the University of Nottingham, Nottingham, U.K., in 2013. Currently, he is an Associate Professor with Southwest Jiaotong University. His research interests include multi-access edge computing, time series mining, evolutionary computation, multi-objective optimization, etc.