Accepted to appear in IEEE/CIC International Conference on Communications in China, 2025 ©IEEE

Pushing the Performance Boundary of In-Network AllReduce With Joint Topology and Routing Optimization

Shouxi Luo[1,2], Xiaoyu Yu[1], Ke Li[1], Huanlai Xing[1]
[1]School of Computing and Artificial Intelligence, Southwest Jiaotong University
[2]Tangshan Institute, Southwest Jiaotong University

***Abstract*—Parameter server (PS) based AllReduce is widely employed by distributed machine learning (DML) applications for model synchronization. As recent studies show, the execution of synchronization can be greatly accelerated by letting powerful switches (a.k.a., aggregators) inside the network pre-aggregate the associated traffic during the journey, yielding the solution of In-Network (accelerated) AllReduce. We i) find that the key to pushing the performance boundary of in-network AllReduce relies on controlling the system's topology and routing jointly since associated flows must go through the same aggregator(s) for in-network acceleration, thus ii) propose the solution of ATRO, i.e., Aggregator-aware Topology and Routing Optimization, to achieve this optimality. At the core, ATRO i) envisions a realistic scenario where commercially available optical switching devices are employed to build a topology-reconfigurable cluster and ii) jointly manages its topology and routing with a novel theory-empowered algorithm to maximize the throughput of in-network accelerated AllReduce. As confirmed by evaluation studies, ATRO achieves excellent performances: compared to the state-of-the-art scheme that i) only optimizes the routing, ii) jointly optimizes the routing and task placements, and iii) jointly optimizes the routing and aggregator placements, it could accelerate the AllReduce's execution (in terms of the achieved throughput) by up to $10.4\times$, $22.1\times$, and $14.0\times$, respectively.**

***Index Terms*—In-network acceleration, routing optimization, topology reconfiguration.**

I. Introduction

Nowadays, the collective operation of parameter server (PS) based AllReduce has been widely employed by data-parallel distributed machine learning (DML) jobs to achieve efficient synchronization of the local results during the training [1], [2]. To guarantee and accelerate the convergence of the distributed training, workers in practice are designed to iterate to the next round of training until its depending AllReduce operation(s) has been completed. Accordingly, accelerating the execution of such AllReduce operation is crucial for optimizing the efficiency of distributed training [1]–[6].

For the acceleration of PS-based AllReduce, recent advances in high-performance data-plane programmable network devices [3], [4], [7] have provided a promising and genetic design, by making efficient usage of the capacities of advanced network devices, yielding the solution of in-network (accelerated) AllReduce [1], [2]. More specifically, in the PS-based implementation, the entire workflow of AllReduce is logically split into a reduction process followed by a broadcast: i) the workers that hold the input data first push their data to the parameter server(s) for reduction (a.k.a, aggregation); ii) once the results are generated, the parameter server(s) broadcast/deliver them back to involved workers. By using the processing and cache capacities of powerful network devices (e.g., P4 switches [3], [4]), it is possible to let some switches act as aggregators to pre-aggregate reduction flows when they go by, i.e., conducting in-network aggregation (INA) [4]. As a result, the reduction procedure can be accelerated since both the triggered traffic in the network and the workload of the PS can be greatly reduced. Following the reversed routing paths for delivering results and using these powerful switches to conduct multicast at the network layer [8], [9], the triggered traffic and workloads for the broadcast can be relieved as well.

In practice, it is likely that only partial switches support in-network acceleration (aggregation and multicast) due to the limited budget [3]–[6]. As pointed out by [4], in these scenarios, the key to releasing the power of in-network acceleration relies on routing optimization, since associated flows must go through the same aggregator to conduct in-network acceleration. In this paper, we notice that only controlling the routing is insufficient to reach the performance boundary of in-network aggregators. This is because the optimization space that routing optimization can explore, is fundamentally limited by the feasible paths among workers, aggregators, and the PS(s), provided by the network. To address this, the recent proposals of PARING [5] and SPAR [6] have explored the designs of jointly optimizing the placements of workers/PS, and the aggregators, along with the routing of reduction traffic, respectively. Unfortunately, their optimizations are not fundamental, since the entire network is still in fixed forms like *Leaf-Spine* and *Fat-Tree* [4], limiting the space that routing optimization can explore, and mismatching with the requirements of in-network AllReduce workloads.

Distinguished from these existing schemes [4]–[6], in this paper, we push the performance boundary of in-network AllReduce with the novel design of Aggregator-aware Topology and Routing Optimization (ATRO). Rather than relying on a fixed cluster topology, ATRO i) envisions a realistic scenario where commercially available optical switching devices like *optical patch panel* [10], [11] are employed to build a topology-reconfigurable cluster for AI training, and then ii) jointly controlling the cluster's topology and routing to maximize the throughput of aggregator-accelerated AllReduce operations. The core of ATRO's scheduling algorithm is to precisely formulate

This work was supported in part by NSFSC under Grant 2025ZNSFSC0489, in part by the Hebei Natural Science Foundation under Grant F2025525008, and in part by the Fundamental Research Funds for the Central Universities under Grant 2682024ZTPY050. *(Corresponding author: Shouxi Luo.)*

the joint topology and routing optimization problem as a novel yet easy-to-solve *Integer Quadratic Programming (IQP)* model and solve it to obtain the optimal joint scheduling scheme. Performance studies confirm the significant performance improvements of ATRO over existing solutions. For instance, compared to the baselines of i) organizing the cluster using *Leaf-Spine* or *Fat-Tree*, then ii) only optimizing the routing with the optimal design of ARO [4], jointly optimizing the routing along with task placements with the approximation algorithm of PARING [5], and jointly optimizing the routing together with the aggregator placement with the greedy scheme of SPAR [6], by optimizing the network topology and routing jointly, ATRO could increase the throughput of in-network AllReduce up to $17.4\times$, $24.8\times$, and $10.6\times$, respectively.

In short, the main contributions of this paper are three-fold.

- A thorough analysis of the importance of joint topology and routing optimization for accelerating in-network AllReduce, which motivates the design of ATRO (§II-B).
- ATRO, a topology-reconfigurable cluster architecture along with a novel IQP-based solution that could maximize the throughput of PS-based AllReduce by jointly conducting aggregator-aware optimal optimization of network topology and routing (§III).
- Performance evaluations confirming the significant performance improvements of ATRO over state-of-the-art (SOTA) schemes (§IV).

Next, we first overview the background and motivation in Section II, then describe our design of ATRO in Section III. After presenting the results of the performance evaluation in Section IV, we finally conclude the paper in Section V.

II. Background and Motivation

A. In-Network AllReduce

The collective operation of AllReduce is widely employed by distributed applications like data-parallel model training to synchronize the local results. The efficiency of completing AllReduce operations is pivotal to the application's performance—Because many distributed applications could move to the next step of processing only until the launched AllReduce has been completed by design [3], [4], [12], [13].

To achieve efficient performance, AllReduce has abundant implementations having various traffic patterns, ranging from *direct peer-to-peer* [14], *rings* [15], *trees* [12], and PS-based *star* [4]. Among them, the PS-based solution is widely employed in the context of data center networks [3]–[6], where involved host nodes are classified into two types, namely *workers* and *parameter servers* (PS), according to their roles. Using PS-based implementation, the execution of an AllReduce operation is decoupled into two stages, i.e., i) workers firstly *reduce* all their input data to the PS via the network, then ii) the PS logically *broadcast*s the results back to all workers. When the data is large, workers and the PS can split the data into chunks and conduct the corresponding *reduction* and *broadcast* in a pipelined fashion [12].

As confirmed by abundant recent studies [1]–[4], [13], the emerging technique of in-network aggregation (INA) has provided a generic and powerful solution to accelerate the execution of PS-based AllReduce. The key idea is to let the capable network devices (e.g., switches) pre-aggregate (i.e., reduce) the data sent by workers before it reaches the PS, such that the traffic inside the network triggered by reduction can be greatly reduced along the journey, and meanwhile, the workload of the PS is also eased [3]. The recent advances in data-plane programmable switches [3], [4], [7], [16] have made the above vision readily deployable. Regarding the broadcast of the results, existing IP multicast techniques have provided bandwidth-efficient solutions [8], [9]. Empowered by the above designs, the PS-based AllReduce's performance can be greatly improved, yielding an efficient in-network (accelerated) AllReduce solution for distributed applications.

B. Importance of Joint Topology and Routing Optimization

To conduct INA, associated reduction flows sent by workers must go through the same aggregator (i.e., INA-support switch) before reaching the PS. In many cases, not all switches support INA [4] [6]. Accordingly, managing associated reduction flows to i) go through the same aggregators and ii) simultaneously reduce the introduced network congestion occurring on links, is the key to achieving optimal in-network AllReduce by making efficient usage of available aggregators [4].

Given an in-network AllReduce task, it is obvious that the room for optimization is generally dominated by two coupled factors, i.e., i) how the elements including the PS, workers, and aggregators are networked, and ii) how the reduction traffic is routed over the network. The recent work of ARO [4] has explored the design of conducting near-optimal (if not optimal) routing optimization to fully release the power of deployed aggregators to accelerate in-network AllReduce over Clos networks. Despite being efficient, the benefits of ARO are still limited by the network topology since a routing plan can be generated if and only if available paths exist between workers, aggregators, and the PS. Beyond routing optimization, the work of PARING [5] also jointly controls the placement of tasks; alternatively, the work of SPAR [6] explores the design of jointly controlling the placement of aggregators rather than workers and the PS along with routing. Although jointly controlling the placement of tasks and/or aggregators along with the routing paths does bring benefits to make efficient usage of aggregators. These solutions are still not fundamental, as the entire network is still in fixed forms like *Leaf-Spine* and *Fat-Tree*, limiting the space for routing optimization. Accordingly, the fundamental way to push the performance boundary of in-network AllReduce relies on conducting aggregator-aware joint optimization of the network topology and routing.

C. Reconfigurability of Network Topology

Nowadays, abundant techniques like *optical patch panel* and 3D MEMS [10], [11] have made the interconnection relationship between devices in the network/cluster reconfigurable at runtime. These techniques have various reconfiguration delays, ranging from several microseconds to minutes; in general, the higher configurable delay it suffers from, the larger number

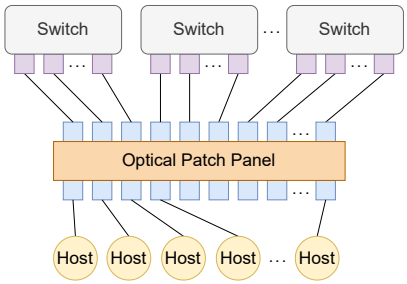

Fig. 1: A topology-reconfigurable cluster architecture for the in-network acceleration of PS-based AllReduce.

of ports a device built upon that technique could support [10]. For example, the current commercially available optical patch panel device provides the runtime reconfigurability for 1008 ports, with a reconfiguration delay of a couple of minutes; while the 3D-MEMS device supports the reconfiguration of 384 ports with a delay of about 10ms [10].

In practice, it generally takes hours, days, or even weeks for distributed machine learning applications to complete the training of deep models. Once a training job is deployed, the patterns of its triggered AllReduce workload are determined. Accordingly, the operator only needs to reconfigure the cluster topology before launching the training. Compared to the entire training duration, minutes of reconfiguration delays are trivial. Thus, commercially available solutions like optical path panels satisfy the demand for reconfigurability.

III. Aggregator-aware Topology and Routing Optimization

To support the reconfigurability of the cluster topology at runtime, as Figure 1 shows, we consider the case where all hosts and switches (including aggregators) are connected to a reconfigurable optical switching device (e.g., optical patch panels, 3D MEMS [10], [11]). The cluster's logical topology can be tuned on demand by configuring the optical device. Based on the above cluster architecture, by considering PS-based distributed training as a concrete example (§III-A), we now describe how to formulate the joint topology and routing optimization problem as an IQP (§III-B) and solve it using off-the-shelf commercial available solvers (§III-C). Our design can also be extended to distributed training jobs built upon tree-AllReduce.

A. System and Workload

Consider the widespread use case and implementation of AllReduce, where a group of workers conduct data-parallel training locally and send gradients to one PS via an INA-support network for reduction (a.k.a., aggregation). As the completion of the reduction is dominated by the slowest sending rate among all workers, same with [4], we enforce all workers to send data at the same rate, denoted by the non-negative integer variable r, bringing benefits to INA [3]. Once the aggregated results are generated, the PS can reliably disseminate the data back to workers using IP multicast [8] over the reversed routing paths at the same rate. When the data is large, both the reduction and the dissemination processes can work in pipeline [12].

$$r \in \mathbb{Z}_{\geq 0} \tag{1}$$

In such systems, there are four types of nodes, namely, worker (w), PS (ps), INA-agnostic common switch (cs), and INA-supported switch, a.k.a. aggregator (agg). We denote them by N_κ, where $\kappa \in \{w, ps, cs, agg\}$, respectively. For convenience, we also use $N_s = N_{cs} \cup N_{agg}$ to represent all switches including cs and agg, $N_n = N_w \cup N_{ps}$ to represent all workers and the PS, and $N = N_s \cup N_n$ to represent all nodes. As the topology between these nodes is re-configurable, we use a binary variable x_u^v to indicate whether there is a connection between node u and another node v (i.e., 1) or not (i.e., 0). Because the sending rates of all workers are the same, the data transmission rate on each link must be a non-negative multiple of r, where this multiple denotes the number of flows on this link. Following [4], we define a non-negative integer variable y_u^v to represent the number of flows for reduction traffic from u to v. Accordingly, all these variables $\{x_u^v : (u,v) \in \mathcal{P}\}$ denote how all the nodes are networked (i.e., the network topology); and all these variables $\{y_u^v : (u,v) \in \mathcal{P}\}$ denote how the INA-supported traffic gets routed over that network.

$$x_u^v \in \{0,1\}, \quad \forall (u,v) \in \mathcal{P} \tag{2}$$

$$y_v^u \in \mathbb{Z}_{\geq 0}, \quad \forall (u,v) \in \mathcal{P} \tag{3}$$

Here, $\mathcal{P}$ denotes the feasible set of node pairs that might have links as Eq. (4) specifies. Given that nodes like workers and the PS generally have only one link connecting to the network via either a switch or an aggregator, there would be no direct connections between workers and the PS. Thus, node pairs belonging to $\{(u,v) \in N_n \times N_n\}$ are excluded from $\mathcal{P}$.

$$\mathcal{P} := \{(u,v) \in N \times N : u \neq v\} \setminus \{(u,v) \in N_n \times N_n\} \tag{4}$$

B. Objective and Constraints

To accelerate the completion of the AllReduce operation, we maximize its throughput:

$$\text{Maximize } r \tag{5}$$

To ensure that there exists a network topology along with a routing scheme that achieves the optimal throughput, there are various types of constraints that our variables r, $\{x_v^u : (u,v) \in \mathcal{P}\}$ and $\{y_v^u : (u,v) \in \mathcal{P}\}$ must satisfy with. Now, we explain these constraints in detail.

- **Limited numbers of ports**: In practice, the number of links connected with each switch/aggregator cannot exceed the number of ports it owns. We use η_u to represent the number of available ports of node u and assume that the inter-node connections are bidirectional. Then, we would have constraints (6) and (7).

$$x_v^u = x_u^v, \quad \forall (u,v) \in \mathcal{P} \tag{6}$$

$$\sum_{v \in N \setminus \{u\}} x_u^v \leq \eta_u, \quad \forall u \in N \tag{7}$$

It's worth noting that, as both worker and PS nodes generally have only one port, their $\eta_u = 1$.

- **Networked workers and PS**: To be networked, each worker and the PS has exactly one connection to either a switch or an aggregator, i.e.,

$$\sum_{u \in N_s} x_u^v = 1, \quad \forall v \in N_n \tag{8}$$

- **Effects of in-network aggregators**: Since an aggregator can reduce multiple associated reduction flows to a single one, the sending rate of aggregator u is either r or 0, depending on whether other nodes in the network send data to it or not. Such a requirement can be jointly formulated with constraints (9) and (10), where M is a constant value larger than the number of workers. Using it, constraints (10) can enforce the value of $\sum_{v \in \mathcal{P}^1(u)} y_u^v$ to be either 1 or 0, respecting the value of $\sum_{v' \in \mathcal{P}^2(u)} y_{v'}^u$.

$$\sum_{v \in \mathcal{P}^1(u)} y_u^v \leq 1, \quad \forall u \in N_{agg} \tag{9}$$

$$\sum_{v \in \mathcal{P}^1(u)} y_u^v \leq \sum_{v' \in \mathcal{P}^2(u)} y_{v'}^u \leq M \sum_{v \in \mathcal{P}^1(u)} y_u^v, \forall u \in N_{agg} \tag{10}$$

Here, the functions of $\mathcal{P}^1(u)$ and $\mathcal{P}^2(v)$ are defined as Eqs. (11) and (12), respectively.

$$\mathcal{P}^1(u) := \{v : \exists (u, v) \in \mathcal{P}\} \tag{11}$$

$$\mathcal{P}^2(v) := \{u : \exists (u, v) \in \mathcal{P}\} \tag{12}$$

- **Behaviors of common switches**: Since a common switch cannot perform aggregation, the total receiving rate of the reduction traffic, must be equal to its total sending rate.

$$\sum_{v \in \mathcal{P}^2(u)} y_v^u = \sum_{v' \in \mathcal{P}^1(u)} y_u^{v'}, \quad \forall u \in N_{cs} \tag{13}$$

- **Limited link capacities**: For each possible link (u, v), the transfer rate passing through it should not exceed its capacity. Here, we consider the case that the reduction and the dissemination involved in the AllReduce are conducted in pipeline. Thus, we take the bidirectional traffic on each link into account, obtaining the constraint of (14).

$$(y_u^v + y_v^u) r \leq b_u^v, \quad \forall (u, v) \in \mathcal{P} \tag{14}$$

- **Relationships between connection and transfer rate**: Let binary variable a_u^v donate whether node u sends data to node v for reduction (i.e., 1) or not (i.e., 0). Then, we have the following two constraints.

$$a_u^v \in \{0, 1\}, \quad \forall (u, v) \in \mathcal{P} \tag{15}$$

$$a_u^v \leq y_u^v \leq M a_u^v, \quad \forall (u, v) \in \mathcal{P} \tag{16}$$

Obviously, there must be a link from u to v, if u is sending reduction traffic to v, i.e., constraint (17).

$$a_u^v \leq x_u^v, \quad \forall (u, v) \in \mathcal{P} \tag{17}$$

- **Loop-freedom of routes**: Given that an aggregator could aggregate multiple flows into one, as the example in Figure 2 shows, the above constraints do not guarantee that the data sent from workers would finally reach the PS, because routing loops may occur. Thus, we further introduce the concept of hierarchy—By limiting the upper bound of the length of routing paths from workers to the PS as $L \leq |N_s|$, a tunable configuration parameter. More specifically, we organize switches and aggregators in a layered structure and use an integer variable l_u to represent the layer of node u. Notably, this l_u does not indicate the number of distances/hops to the PS, as cross-layer connections are allowed.

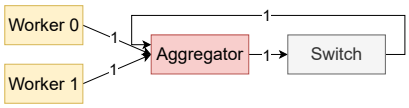

Fig. 2: A possible loop due to INA which aggregates multiple flows into one.

$$l_u \in \{1, \cdots, L\}, \quad \forall u \in N_s \tag{18}$$

For simplification, we limit the l_u of the switch or aggregator u directly sending reduction traffic to the PS v (i.e., $a_u^v = 1$) to 1, yielding the following constraint.

$$l_u \leq 1 + L(1 - a_u^v), \quad \forall u \in N_s, v \in N_{ps} \tag{19}$$

Also, if a network device u is directly sending reduction traffic to another device v (i.e., $a_u^v = 1$), they must have the relationship of $l_u \geq l_v + 1$, which can be formulated by constraint (20).

$$l_v + 1 \leq l_u + L(1 - a_u^v), \quad \forall (u, v) \in \mathcal{S} \tag{20}$$

Here, $\mathcal{S}$ defined by Eq. (21), represents the feasible set of network device pairs.

$$\mathcal{S} := \{(u, v) \in N_s \times N_s : u \neq v\} \tag{21}$$

C. Joint Optimization

Putting all the constraints and the objective together, we have formally formulated the problem of joint network topology and routing optimization for in-network AllReduce as an IQP. Motivated by the solving acceleration design of *quantized sending rates (QSR)* proposed by ARO [4], we also use a parameter to restrict that the optimized rate is selected from a pre-defined collection of integers for fast solving.

We find that for in-network AllReduce jobs with a modest scale, we can directly obtain the corresponding IQP model's optimal results efficiently (e.g., within a few seconds or minutes) by using off-the-shelf high-performance commercial available solvers like Gurobi [17]. Then, the obtained $\{x_u^v : (u, v) \in \mathcal{P}\}$ indicates the structure of the network. Notably, if there is a node pair (u, v) whose $x_u^v = 1$ but $y_u^v + y_v^u = 0$, it means this link is not used by the AllReduce; thus, we can directly delete the connection between them to simplify the topology. Finally, just like the design of ARO [4], based on the obtained $\{y_u^v : (u, v) \in \mathcal{P}\}$, we can generate the routing paths to accelerate the in-network AllReduce task by making full usage of the capacities of aggregators.

IV. Performance Evaluation

Now, we evaluate the performance of ATRO by using SOTA schemes, including the aggregator-aware routing optimization scheme of ARO [4], the joint routing and task placement optimization scheme of PARING [5], and the joint routing and aggregator placement optimization scheme of SPAR [6], as baselines. Results show that ATRO achieves excellent performance. By directly conducting the joint optimal topology and routing optimizations for in-network AllReduce workloads, ATRO achieves about $4.0 \sim 17.4\times$, $5.8 \sim 24.8\times$, and $5.6 \sim 10.6\times$ higher throughput, than the designs of i) organizing the cluster following the well-known architectures like *Leaf-Spine* and *Fat-Tree* then ii) optimizing the workloads using ARO, PARING, and SPAR, respectively.

A. Methodology

1) Workloads and baselines: We consider the case in which n workers, together with one PS, launch an in-network AllReduce for model synchronization. To the best of our knowledge, ATRO is the first proposal that optimizes the topology and routing for in-network AllReduce jointly. As the baselines, i) the cluster is built upon the typical network architecture of either *Leaf-Spine* or *Fat-Tree* (variant), and ii) a suite of SOTA schemes ARO [4], PARING [5], and SPAR [6] are employed to conduct optimization on the fixed topology, and the AllReduce task involves 90 workers by default. In both *Leaf-Spine* and *Fat-Tree*, there are 20 switches each with 24 ports, and by default 4 of them support INA. All links have the same bidirectional capacity of 100Gbps. To be fair, ATRO manages the same amounts of switches and aggregators for joint optimization and limits the maximum allowed layer (i.e., L) to 5. The detailed baseline topology settings are as follows.

- *Leaf-Spine*. There are 10 spines and 10 leaves; each leaf supports 14 hosts, yielding 140 in total.
- *Fat-Tree (variant)*. 20 switches form a 4-ary architecture, with 4 *cores*, 8 *aggregations*, and 8 *edges*. Each edge supports 12 hosts, yielding 96 in total and the oversubscription is $6:1$.

The three baseline algorithms employ the following settings.

- ARO [4]. Workers and the PS are embedded into hosts randomly with no overlapping. Then, their routing paths are optimized with ARO [4]. For efficient model solving, we enable the solving acceleration of *quantized sending rates (QSR)* for ARO using $N = 100$ [4].
- PARING [5]. The placements of workers and the PS, and their routing paths are jointly optimized with the approximation algorithm of PARING [5]. PARING abstracts the network among workers, aggregators, and the PS out as an overlay to determine the high-level overlay routing, without explicitly controlling how the traffic is routed between these elements inside the underlay network. In tests, we assume the shortest path first routing principle is used, and if there are multiple shortest paths, PARING selects one randomly. When constructing the degree-constraint Steiner tree upon the overlay network for PARING, we assume that the PS has sufficient degrees and aggregators have sufficient capacities.
- SPAR [6]. Workers and the PS are randomly and exclusively embedded into hosts. Then, their routing paths and the placements of the aggregators are jointly optimized with the greedy algorithm of SPAR [6]. Respecting SPAR's design, only top-of-rack (i.e., leaf or edge in our settings) switches would be selected to support INA [6]. When using SPAR [6], like the setting used for PARING, aggregators are assumed to have sufficient capacities, and the shortest path first routing principle is used for inter-element traffic.

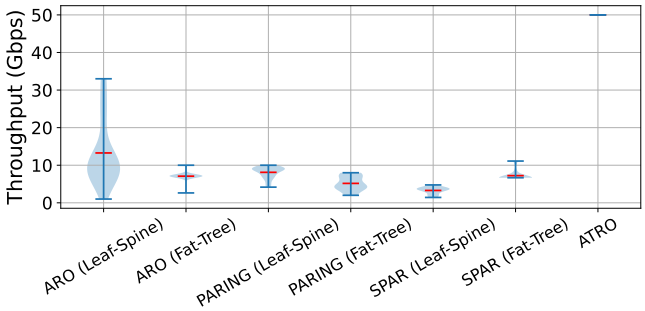

Fig. 3: ATRO achieves the best in-network AllReduce throughput than all SOTA baselines.

2) Simulators and metrics: We implement a simulator with Python 3 following that of ARO [4] to assess the performance of ATRO and baselines. In tests, we mainly use the throughput achieved by the optimized in-network AllReduce as the metric. To reduce the noise of random factors in the experiment settings on the results, we repeat each group of experiments 30 times (except ATRO, which runs only once as no random factors are involved) and report the average values. All experiments are conducted on a desktop PC equipped with an Intel i5-12400 CPU and two 16G memory cards.

B. Results

1) Case studies: Figure 3 shows the details of the in-network AllReduce throughput achieved by ATRO and the baselines, under the default settings, where the red lines in the displayed violins indicate the mean values. Compared to the design of using the fixed network topology and then optimizing the in-network AllReduce workload with ARO, PARING, and SPAR, ATRO obtains the throughput of 50Gbps, yielding improvements of about $3.77\times$ /$9.67\times$, $7.06\times$ /$15.2\times$ and $6.17\times$ /$6.73\times$ in *Leaf-Spine*/*Fat-Tree* topology, respectively.

2) Impacts of the scale of AllReduce: When the number of aggregators is fixed at 4, Figure 4 shows the throughput achieved by different schemes as the number of workers increases from 60 to 90. Obviously, ATRO achieved the maximum throughput. When the number of workers did not exceed 80, ATRO could achieve a full bandwidth throughput of 100Gbps. However, as the number of workers continued to increase, ATRO's throughput decreased by half to 50Gbps. This is because when the number of workers is small, ATRO

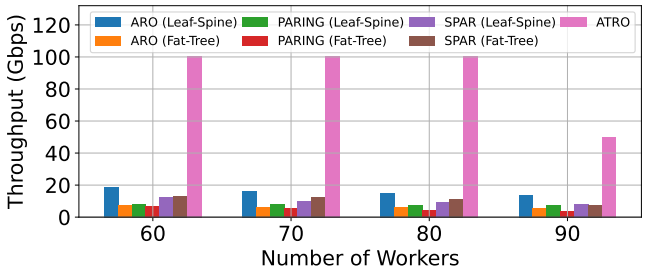

Fig. 4: Impacts of the number of involved workers on the achieved throughput of in-network AllReduce.

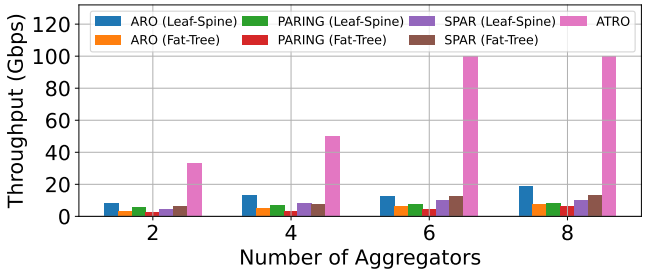

Fig. 5: Impacts of the number of switches supporting INA on the achieved throughput of in-network AllReduce.

can fully utilize the aggregators to ensure that there is more than one aggregated flow on each link. For other schemes, although ARO has achieved maximum optimization of routing, it is limited by the fixed topology; although PARING optimizes routing and task placement jointly, it does not consider the impact of bandwidth and only uses the number of hops as the communication cost between two nodes, resulting in poor performance; SPAR optimizes routing and aggregator placement together, but uses a greedy method to determine the placement of aggregators, making it far from optimal. Besides, compared to *Fat-Tree*, ARO and PARING perform better in *Leaf-Spine*, because *Leaf-Spine* can provide higher throughput limits for workers with lower oversubscription. However, SPAR does not have a clear trend in both topologies, mainly because it uses aggregator placement to alleviate the upward communication pressure of ToR switches in *Fat-Tree*.

3) Impacts of the number of aggregators: We then change the number of aggregators with 90 workers in an AllReduce task and obtain Figure 5. As expected, despite all schemes obtaining higher throughput when the number of aggregators increases, the improvements achieved by ATRO are the most significant. As Figure 5 shows, when there are only 4 aggregators, ATRO only achieves the throughput of about 33Gbps, a third of the full link capacity; but once the number of aggregators reaches 6, the full throughput of 100Gbps can be obtained.

4) Efficiency of ATRO: Despite ATRO relying on solving IQP models for the joint scheduling, we find it very efficient for modest scale in-network AllReduce tasks—In our tests, the time cost ranges from 2 to 330 seconds respecting the scale of the instance. Such a delay is acceptable since we only need to solve the model once before launching a long-running training.

V. Conclusion and Future Work

In this paper, we show that joint topology and routing optimization are the keys to pushing the performance boundary of PS-based in-network AllReduce, and propose the proposal of ATRO. The core of ATRO relies on formulating the joint optimization problem as an IQP using a suite of novel techniques, and solving it to obtain the optimal scheduling scheme accordingly. Extensive performance studies indicate that ATRO outperforms all SOTA schemes significantly, achieving improvements up to 24.8×.

Currently, ATRO focuses on the optimization of a single AllReduce task and directly solves the math model with Gurobi [17]. Once the network scale is large, it is impossible to obtain optimal results in a reasonable time. In future work, we plan to extend ATRO to support the optimization of multiple in-network AllReduce tasks and design efficient and effective heuristic algorithms, e.g., based on decomposition techniques.

References

[1] D. De Sensi, E. Costa Molero *et al.*, "Canary: Congestion-aware in-network allreduce using dynamic trees," *Future Generation Computer Systems*, vol. 152, pp. 70–82, 2024.
[2] H. Song, "In-network allreduce optimization with virtual aggregation trees," in *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing*, ser. NAIC '24. ACM, 2024, pp. 54–60.
[3] C. Lao, Y. Le *et al.*, "ATP: In-network aggregation for multi-tenant learning," in *Proceedings of the 18th NSDI*, Apr. 2021, pp. 741–761.
[4] S. Luo, X. Yu *et al.*, "Releasing the power of in-network aggregation with aggregator-aware routing optimization," *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 4488–4502, 2024.
[5] Y. Qiu, G. Zhao *et al.*, "Paring: Joint task placement and routing for distributed training with in-network aggregation," *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, pp. 4317–4332, 2024.
[6] L. Luo, S. Yang *et al.*, "Maximizing aggregation throughput for distributed training with constrained in-network computing," in *Proceedings of IEEE ICC*, 2023, pp. 3652–3657.
[7] M. Yang, A. Baban *et al.*, "Using trio: juniper networks' programmable chipset - for emerging in-network applications," in *Proceedings of the ACM SIGCOMM 2022 Conference*. ACM, 2022, pp. 633–648.
[8] 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.
[9] 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.
[10] W. Wang, M. Khazraee *et al.*, "TopoOpt: Co-optimizing network topology and parallelization strategy for distributed training jobs," in *Proceedings of the 20th NSDI*, Apr. 2023, pp. 739–767.
[11] K.-T. Foerster and S. Schmid, "Survey of reconfigurable data center networks: Enablers, algorithms, complexity," *SIGACT News*, vol. 50, no. 2, pp. 62–79, jul 2019.
[12] S. Luo, R. Wang, and H. Xing, "Efficient inter-datacenter allreduce with multiple trees," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 5, pp. 4793–4806, 2024.
[13] 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.
[14] S. Luo, P. Fan *et al.*, "Efficient parameter synchronization for peer-to-peer distributed learning with selective multicast," *IEEE Transactions on Services Computing*, vol. 18, no. 1, pp. 156–168, 2025.
[15] D. D. Sensi, T. Bonato *et al.*, "Swing: Short-cutting rings for higher bandwidth allreduce," in *Proceedings of the 21st NSDI*, Apr. 2024, pp. 1445–1462.
[16] D. De Sensi, S. Di Girolamo *et al.*, "Flare: flexible in-network allreduce," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. ACM, 2021.
[17] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com