Rina: Enhancing Ring-AllReduce with In-network Aggregation in Distributed Model Training

Zixuan Chen[†], Xuandong Liu[†], Minglin Li[†], Yinfan Hu[†], Hao Mei[†],
Huifeng Xing[†], Hao Wang[†], Wanxin Shi[†], Sen Liu[†], and Yang Xu[†◇*]

[†]School of Computer Science, Fudan University, Shanghai, China
[◇]Pengcheng Laboratory, Shenzhen, China

Abstract—Parameter Server (PS) and Ring-AllReduce (RAR) are two widely utilized synchronization architectures in multi-worker Deep Learning (DL), also referred to as Distributed Deep Learning (DDL). However, PS encounters challenges with the "incast" issue, while RAR struggles with problems caused by the long dependency chain. The emerging In-network Aggregation (INA) has been proposed to integrate with PS to mitigate its incast issue. However, such PS-based INA has poor incremental deployment abilities as it requires replacing all the switches to show significant performance improvement, which is not cost-effective. In this study, we present the incorporation of INA capabilities into RAR, called RAR with In-Network Aggregation (Rina), to tackle both the problems above. Rina features its agent-worker mechanism. When an INA-capable ToR switch is deployed, all workers in this rack run as one abstracted worker with the help of the agent, resulting in both excellent incremental deployment capabilities and better throughput. We conducted extensive testbed and simulation evaluations to substantiate the throughput advantages of Rina over existing DDL training synchronization structures. Compared with the state-of-the-art PS-based INA methods ATP, Rina can achieve more than 50% throughput with the same hardware cost.

Index Terms—Distributed Deep Learning, In-network Aggregation, Ring-AllReduce

I. INTRODUCTION

The field of Deep Learning (DL) has seen remarkable advances in recent years, driving transformative breakthroughs across various domains. The advent of Artificial General Intelligence (AGI) and large language generation models, such as the Generative Pre-trained Transformer (GPT) [1], have significantly advanced machine intelligence and Natural Language Processing (NLP) [2], [3]. Furthermore, models like Segment Anything (SA) [4] have enriched DL's capability in image segmentation.

As model complexity and dataset sizes continue to expand exponentially, Distributed Deep Learning (DDL) has emerged as a pivotal approach for efficient training. Data parallelism, a

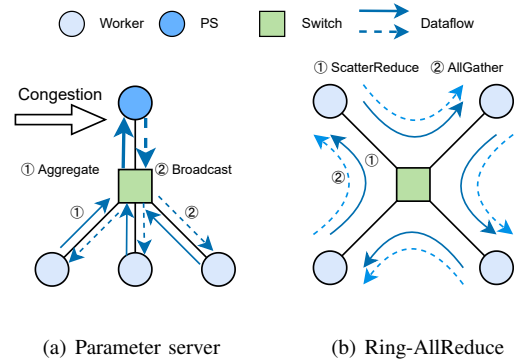(a) Parameter server   (b) Ring-AllReduce

Fig. 1. Popular synchronization architectures in DDL.

strategy for managing vast datasets, divides the dataset among distinct processing units for concurrent training. This technique enables DL models to accommodate substantially larger datasets, thereby significantly enhancing training efficiency.

As depicted in Figure 1, Parameter Server (PS) and Ring-AllReduce (RAR) are two prominent methods employed for parameter synchronization in data parallelism. PS operates by partitioning the dataset across multiple servers, thereby enabling parallel processing units to update these parameters independently. Conversely, RAR adopts a decentralized approach wherein each processing unit retains a full set of parameters and communicates with others in a ring-like logical topology (i.e., Figure 1(b)). The goal of both methods is to keep consistent parameter updates across all workers throughout the training phase while maintaining high throughput.

The popular PS architecture suffers from a critical bottleneck known as the "incast" issue [5]. This bottleneck emerges when numerous workers attempt simultaneous communication with a single PS, potentially leading to network congestion and consequently impeding the system's overall performance and scalability. The inherent limitations of PS have driven research towards alternative strategies that circumvent such challenges. In-Network Aggregation (INA) has been proposed as a solution to alleviate network congestion in PS. This approach, as incorporated into several works, including SwitchML [6], ATP [7], INAlloc [8], and PANAMA [9], has significantly improved network congestion and DDL efficiency. However, the incremental deployment capabilities of PS-based INA approaches remain weak. Notably, PS-based INA methods

necessitate the replacement of nearly all congestion-point switches in the topology for significant network throughput enhancements. For example, even if we replace 80% of the regular switches with INA switches, the network throughput will be only 50% (please refer to § III-B).

The RAR architecture is another method to alleviate network congestion in the PS. RAR avoids network congestion issues due to its unique communication structure - a unidirectional loop. This design ensures that as long as the workers involved in the synchronization process are topologically linked, there will be no communication bottlenecks. However, a significant issue in RAR is the lengthy dependency chain, where the length of the chain is directly proportional to the number of participating workers. This dependency chain problem introduces a throughput degradation into the system as the chain extends. What is more, the performance degradation or downtime of even a single worker can significantly impact the overall system performance. For details, please refer to § III-A.

A novel architecture should be proposed to fully leverage the capability of in-network computing switches to alleviate traffic bottlenecks, thereby enhancing DDL training throughput while offering improved incremental deployment capability. Our approach uses an agent-worker model to infuse **R**ing-AllReduce with **I**n-**N**etwork **A**ggregation (**Rina**) capabilities. The Rina design ingeniously amalgamates the INA switch's functionality into RAR's synchronization process, proposing a holistic architecture and workflow. Rina accommodates two types of workers - the abstracted worker (embodied by a Rina-enabled rack) and the autonomous worker (a regular RAR worker) - to ascertain both compatibility and peak throughput. Consequently, Rina achieves superior throughput while attaining a higher incremental deployment capability compared to PS-based INA methods, without being encumbered by RAR's long dependency chain issue. To the best of our knowledge, no existing work has yet incorporated INA capabilities into RAR. In this context, Rina emerges as a pioneer.

In summary, this study makes the following contributions:

1) We propose the Bandwidth-Occupation Model (BOM) to model existing PS-based INA methods. Through the BOM, we identify the lack of incremental deployment capabilities in PS-based INA methods. Concurrently, we analyze the problem of throughput degradation caused by the lengthy dependency chain in RAR.

2) We introduce Rina, the first initiative to infuse RAR with INA capability. Rina leverages the agent-worker model to bring about the novel design, which uses the INA switch and the agent to abstract the workers under one rack. This results in Rina possessing superior incremental deployment capabilities compared to PS-based INA approaches. Additionally, the comprehensive design of Rina addresses the long dependency chain issue found in RAR by compressing the long dependency chain under a rack into a single hop, thus enhancing the throughput of RAR-based DDL. We implement a prototype of Rina on a P4 programmable switch.

3) We conduct extensive evaluations on NS3 and a real testbed using five real-world DL models and four datasets. These evaluations affirm the effectiveness and efficiency of Rina in enhancing the performance of common DDL training tasks. Compared to the state-of-the-art INA approaches like ATP, Rina delivers a 50% throughput advantage at an equivalent cost. Furthermore, in comparison with traditional RAR and PS, Rina can boost the throughput by up to 6x.

The remainder of this paper is organized as follows. § II reviews the background. § III presents the motivation and design concepts of Rina. Rina's design details are provided at § IV and implementation details are provided at § V. § VI shares the evaluation results, followed by the discussion in § VII. Related works are introduced at § VIII. We conclude the study in § IX.

II. BACKGROUND

A. Distributed Deep Learning

With the rapid growth of the size of datasets and models, DDL has gained lots of research interest and has become the primary method to improve the training efficiency and throughput to meet the demands both industrially and academically. There are two prominent parallelism schemes of DDL, data parallelism [10] and model parallelism [11], [12]. Data parallelism duplicates training models across all computing workers. In a single iteration, each computing worker processes different mini-batches of data to calculate the local gradient updates, which are exchanged with other workers later before updating the model parameters [13]. The synchronization in data parallelism is the main optimization objective of this study.

Model parallelism divides the model parameters into multiple workers to make it possible to train large models. Each worker holds a subset of model parameters or layers. In every iteration, the sampled mini-batch of datasets is copied to all workers, and different parts of the DL model are computed on different workers. Model parallelism is also an important area of study, which is orthogonal to the data parallelism emphasized in this paper [14].

B. Synchronization Architectures

1) Parameter Server Architecture: The PS architecture [15] is a straightforward method for parallel computing between multiple workers (Figure 1(a)). In this architecture, a PS node maintains and manages a global model. During each training iteration, each worker computes its local gradients based on its own mini-batch and communicates these results to the PS. The PS updates the global model and synchronizes it with each worker. Typically, there are two synchronization models: Bulk Synchronous Parallel (BSP) [16] and Asynchronous Parallelism (ASP) [17]. In BSP, workers must await a synchronization barrier before initiating the next iteration. Conversely, ASP removes this synchronous barrier. Generally, BSP tends to yield higher accuracy, while ASP significantly increases training throughput.
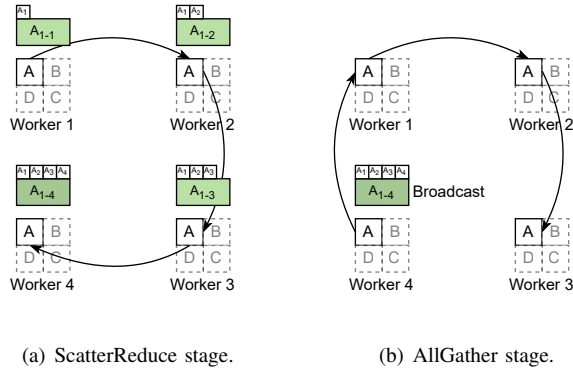
(a) ScatterReduce stage.　　　(b) AllGather stage.

Fig. 2. Details for Ring-AllReduce.

2) Ring-AllReduce Architecture: AllReduce (AR) is a decentralized architecture proposed to alleviate communication bottlenecks in the PS architecture (Figure 1(b)). AR treats all machines as workers, thus eliminating the need for PS. Ring AllReduce (RAR) stands out among AR algorithms because of its superior bandwidth performance. RAR splits communication phases into ScatterReduce and AllGather. In the ScatterReduce phase, each of the N workers divides their local gradients into N chunks. In every iteration, each worker transmits a chunk to its neighbor, receives one, and adds it to the corresponding chunk. The chunks transmitted and received in each iteration are different, with each worker forwarding the chunk received in the previous iteration. After $N - 1$ iterations, each worker possesses a globally updated chunk. For instance, in Figure 2(a), worker 1 forwards chunk A to worker 2 in the first iteration. Worker 2 adds it to its local chunk and passes it on to worker 3 in the subsequent iteration. After the ScatterReduce phase, worker 4 will possess a fully updated chunk A. During the AllGather phase, each worker transmits its complete chunk to the next worker and obtains one from the previous. Like in ScatterReduce, each worker forwards the chunk it received in the previous iteration. After $N - 1$ iterations, every worker has fully updated gradients. As illustrated in Figure 2(b), worker 4 sends its fully updated chunk A to worker 1, who then forwards it to the next worker (i.e., worker 2). Thus, at the end of the AllGather phase, each worker possesses a fully updated result for all chunks. Unlike PS, RAR operates solely in BSP mode. Although RAR achieves optimal bandwidth performance, it suffers from issues of extensive dependency chains and vulnerability to a single point of failure [18].

C. Bring INA into DDL

In recent years, advancements in programmable networks have driven a surge of research employing INA techniques to expedite DDL training. INA leverages the computational power within programmable switches to aggregate gradients from multiple nodes, reducing network traffic and accelerating DDL training. Notable works in this domain include SwitchML [6] and ATP [7], both of which aim to enhance overall training speed by offloading the gradient aggregation task to switches. In SwitchML, all gradients are aggregated

within the switches, thus training speed hinges significantly on the switches' aggregation capabilities. ATP adopts a best-effort strategy for gradient aggregation, where gradients not aggregated at the switch level are relayed to the PS for aggregation.

Nevertheless, these optimization efforts are tailored specifically for INA within the PS architecture. The RAR architecture, renowned for its efficient communication performance, is gaining increased attention. To our knowledge, no existing research explores INA utilization within the RAR architecture, presenting a distinct set of challenges and opportunities at the core of this study.

III. MOTIVATION AND CONCEPTS OF RINA

In this section, we first propose to analyze the issues with RAR methods, specifically **their long dependency chains**. Next, we present the Bandwidth-Occupation Model (BOM) for all existing PS-based INA methods to illustrate their problem: **the lack of incremental deployment capability**. Finally, we present the design concepts and architecture of Rina, briefly elaborating on its advantages over both the state-of-the-art PS-based INA methods and traditional RAR methods.

A. Long Dependency Chain Problem in Ring-AllReduce

Compared to PS-based INA, RAR does not have communication bottlenecks, which are determined by the communication mode of RAR. The following provides quick proof.

1) We can view a network as a connected undirected graph. Let $G = (V, E)$ be a connected undirected graph.
2) Transform G into a directed graph $D = (V, D)$ by replacing each $u, v \in E$ with two directed edges (u, v) and (v, u) in D.
3) By this transformation, for each vertex $v \in V$, the in-degree equals the out-degree.
4) According to the Eulerian path and circuit conditions [19] in directed graphs, since the in-degree equals the out-degree for all vertices in D, there exists an Eulerian circuit.
5) Hence, a path in D starts from any worker, visits every worker once, and finally ends at the starting worker. This guarantees the RAR's requirements for communication without bottleneck.

Although RAR has been proven to be free from bandwidth bottleneck, it still suffers from the issue of long dependency chains [20]. The most significant problem caused by long dependency chains is that throughput performance becomes affected by the increasing number of nodes. Take Figure 3 as an example. The figure shows the workflow of an RAR pipeline. The same part of the gradients shares the same color. Each node sends its computed gradient G_i to the next worker. However, according to the implementation of the latest MPI libraries such as NCCL [21] and OpenMPI [22], each round of communication has a barrier to global synchronization. Interference caused by load fluctuation, interrupts, garbage collection, or background tasks during worker 3 processing G_6 will defer the global completion time of this step, indicating
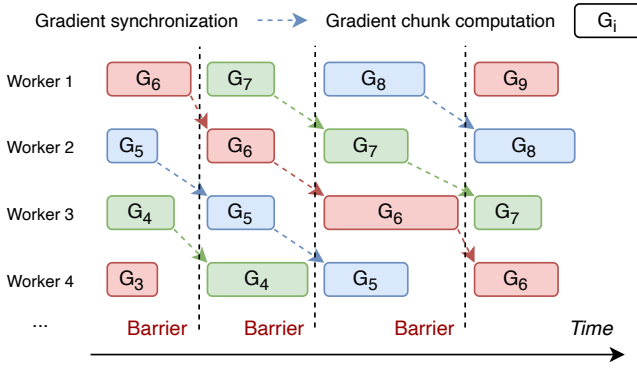
Fig. 3. Barriers will block the whole RAR synchronization process.

that single-point failure will directly slow down the entire training process. This is the fundamental problem for the long dependency chain.

Suppose that the whole cluster has N workers. The system overhead of sending gradient (including network protocol, memory movement, et al.) chunk i is $O(G_i)$, while the computation and communication time is $C(G_i)$. For the i-th round synchronization, the time consumption of the worker n will be $O(G_u) + C(G_u), u = (i + n)\%N$. Since the existence of barriers, the estimated time consumption of the ScatterReduce phase will be $\sum_{u=1}^{N} Max(O(G_u) + C(G_u))$.

In practical scenarios, $O(G_u)$ can be considered a fixed overhead independent of N. Typically, the distribution of $C(G_u)$ is proportional to N in a linear fashion. We assume that $C(G_u)$ follows a normal distribution, where its mean is proportional to N and the variance is a fixed value, expressed as $C(G_u) \sim \mathcal{N}\left(k \cdot \frac{1}{N}, \sigma^2\right)$. Here, k is a constant. The standard deviation σ_u is also assumed to be a constant σ. For a random variable X that follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$, the expected maximum value M_n (taken from n independent and identically distributed samples) can be approximated as $\mathbb{E}[M_n] \approx \mu + \sigma\sqrt{2\ln n}$. Thus, the time consumption T of RAR during the ScatterReduce phase can be expressed as:

$$T = N \cdot O(G_u) + \sum_{u=1}^{N} Max(C(G_u)) \tag{1}$$
$$\approx N \cdot O(G_u) + k + N \cdot \sigma\sqrt{2\ln N}$$

From Equation 1, it can be seen that the value of T increases with the size of N, which demonstrates that the synchronization completion time of RAR increases as the number of nodes increases. It is also noteworthy that an increase in σ will also lead to an increase in T, which means that if the nodes' performance is unstable, the synchronization completion time of RAR will be further prolonged. This phenomenon is common [15], [23], [24], thus reducing the long dependency chain of RAR is highly necessary.

As a widely used AllReduce method in the industry, Hybrid AllReduce (H-AR) [25] has been proposed to address the issue of long dependency chains through a multistep AllReduce process. H-AR first performs a ScatterReduce within the rack,

then an AllReduce between ToR switches, and finally an AllGather within the ToR. This approach can mitigate the long dependency chain problem and achieve better performance than RAR.

However, the control logic of H-AR is far more complex than that of RAR. If in-network computing is required to accelerate performance for H-AR, it would necessitate complex modifications to the switches. On the other hand, the in-network computing logic introduced by Rina itself is not complicated. Furthermore, Rina's utilization of INA switches can achieve even higher throughput compared to H-AR. A detailed comparison is provided in § VI-B.

B. Modeling PS-based INA with BOM

A major advantage of the PS-based INA approaches in improving the throughput of DDL training tasks is its ability to reduce network traffic [9]. In this section, we quantify this through the BOM model and discuss their weakness in incremental implementation.

Assumptions: The DDL training cluster uses the BSP synchronization algorithm. All nodes need to send the gradients of their local models to the PS synchronously, followed by a broadcast generated by the PS. The INA switch can fully aggregate incoming traffic (as proven to be feasible in INAlloc [8] under the single-job scenario). If the corresponding PS-based INA method does not require a PS server, the farthest INA switch is treated as the PS. The entire topology is homogeneous, with a link bandwidth of B_0. There is no multipath scenario in the topology, that is, there is exactly one path from all nodes to the PS.

Lemma 1: *For a topology only containing regular switches and W workers, the worker throughput is $1/W$.*

As shown in the sub-topology T_1 in Figure 4, this topology does not include any INA switches. Assuming that the throughput of its outbound switch 2 is B_1, the worker throughput in this topology is $B_1/4$. The proof is as follows.

Assume a complex topology T. From the topology T, we select the PS node that works as the root to build a Directed Acyclic Graph (DAG) tree, which can be used to represent network traffic during the gradient aggregation phase.

Given a DAG tree $G = (V, E)$, where V is the set of vertices (or nodes) and E is the set of directed edges. G is a subtree of T. The root node is denoted as r and L is the set of leaf nodes. The output rate of a node v, denoted as $OR(v)$, is defined as the number of outgoing edges from v.

We aim to prove that $\forall l \in L$, $OR(l)$ is determined by the output rate of the root node r, $OR(r)$.

Proof: Consider a tree topology where the outbound rate of the root node is $OR(r)$. We observe that the outbound rate of the root must be shared amongst all its child nodes (and their child nodes, and so on). This is because all nodes' data must be transmitted via the root.

Now, suppose we have a working node v that is d levels deep from the root (with the root being at level 0). All parent nodes of v (i.e., all nodes on the path from the root to v) must distribute their bandwidth amongst their child nodes.
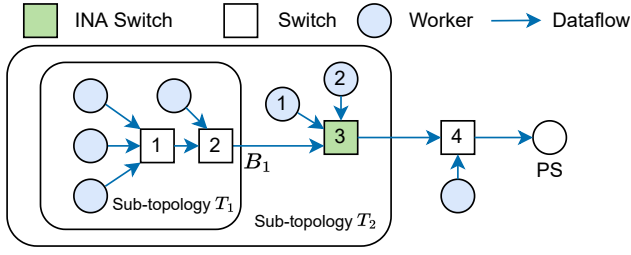
Fig. 4. A sample of traffic on the switch under PS architecture.

(a) Fat-tree (k=4).　　(b) Dragonfly (a=4, g=9, and h=2).

Fig. 5. Lack of incremental deployment capability for PS-based INA.

Therefore, the output rate $OR(v)$ of node v is a fraction of the bandwidth of all its parent nodes.

In particular, the root node r must distribute its bandwidth $OR(r)$ amongst all W working nodes. Therefore, the output rate of each working node is at least $OR(r)/W$. Given that the sum of the output rates of all working nodes cannot exceed $OR(r)$, it follows that the output rate of each working node must be $OR(r)/W$.

Hence, we conclude that for any topology containing only regular switches, the output rate of each working node is determined by the output rate of the root node, i.e., $OR(r)/W$, where W represents the number of working nodes. This completes the proof.

Lemma 2: *The INA switch and its children can be viewed as one worker.*

INA switches can aggregate the received gradients and output dataflow without redundant positional gradients. In this way, to the parent node of this INA switch (no matter the other INA switch or PS), its behavior is manifested as an independent worker.

Lemma 3: *For an INA switch, the actual throughput depends on the worst-performing child.*

Take Figure 4 as an example. For the outbound INA switch 3 in topology T_2, even though it has sufficient INA capabilities to allow workers 1 and 2 to function at 100% throughput, it is still limited by the slowest child node (topology T_1). This implies that the actual outbound bandwidth of the topology T_2 is $B_0/4$, which is also the actual global throughput.

At this point, given the topology, nodes, and placement of INA switches, we can calculate the actual throughput of all the workers in this cluster during the synchronization phase, based on BOM. Take Figure 4 as an example. For switch 4, as its child nodes consist of an INA switch (treated as one worker) and a worker, the limitation on the throughput imposed by switch 4 is $B_0/2$. For switch 3, since it is an INA switch, workers 1, 2, and switch 2 can all send gradients to it at 100% throughput. This implies that $B_1 = B_0$. For sub-topology T_1, based on Lemma 1, the actual throughput limit is $B_0/4$. The actual throughput of all workers globally is taken as the minimum value, which is $B_0/4$.

C. Incremental Deployment is Challenging for PS-based INA

The incremental deployment capability of PS-based INA methods is poor. Using BOM, we evaluate the changes in DDL training throughput in a specific topology scenario,
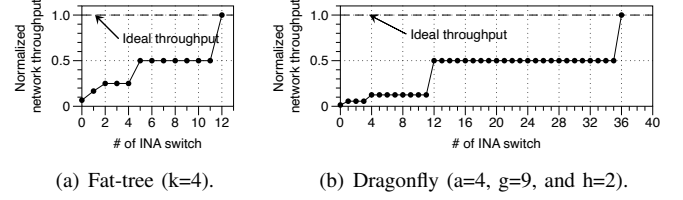
starting from "all switches are regular switches" to replacing all switches in the topology with INA switches.

The training task we selected is ResNet50 [26], using the CIFAR-10 [27] dataset. We have chosen two topologies, namely the standard Fat-tree [28] (k=4) and standard Dragonfly [29] (a=4, g=9, and h=2) topologies, which are commonly used in data centers. The corresponding results are shown in Figure 5. As can be seen, to achieve significant throughput improvements, PS-based INA methods need to replace regular switches in the entire network with high-performance programmable switches as much as possible. If only a part of the switches is replaced, the effect of INA cannot be well-utilized.

Incremental deployment capability can bring numerous benefits, both economically and in terms of data center robustness. Economically, supporting incremental deployment allows service providers (ISPs) to achieve explicit performance improvements with each equipment upgrade. Considering the robustness of data centers, when some switches degrade to traditional switches due to sudden errors that prevent them from providing in-network computing capabilities, poor support for incremental deployment in the synchronization architecture can lead to severe performance degradation. Therefore, designing a DDL synchronization architecture that is responsive to incremental deployment is one of the important design principles of Rina.

D. Rina Design Concepts and Challenges

Before introducing details of Rina, we summarize the existing PS-based INA methods from a higher perspective. The existing PS-based INA involves the INA device wrapping all its attached devices into a unified external device. From the viewpoint of other devices after the INA device, the INA and its workers are combined as a larger-scale worker.

If we aspire to incorporate the same INA capability into RAR and resolve the issues of long dependency chains, we should adopt a method similar to PS-based INA. **This involves considering the INA switch and its attached nodes as a whole, that is, as an abstracted worker.** The design of Rina follows this concept, and here are the other challenges.

1) The workflow of RAR is significantly more complex than PS, making the integration of INA into RAR a challenging task. How should the architecture and workflow of Rina be designed?
2) The introduction of INA switches will lead to new design issues, such as congestion control and reliability assurance. How does Rina address these issues?
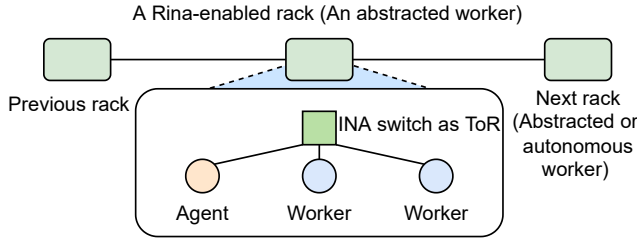
Fig. 6. Agent-worker model for Rina.

3) The INA method based on PS lacks incremental deployment capability. How can Rina achieve incremental deployment capability? How should switches be incrementally deployed?

In the following section, we will provide a detailed introduction to the design of Rina addressing these challenges.

IV. DESIGN DETAILS OF RINA

Rina is designed to incorporate the INA switch into DDL training tasks using the RAR synchronous architecture as a basis. Rina not only retains the benefits of the RAR architecture, including the absence of communication bottlenecks, but it also effectively mitigates the issue of the long dependency chain. Essentially, Rina upholds the RAR workflow pattern, which includes the ScatterReduce and AllGather phases, and optimizes these stages to leverage the capabilities of INA devices. Moreover, Rina introduces a new workflow based on the agent-worker model, discussed in § IV-A. This model allows INA devices to manage all workers within each rack, as detailed in § IV-B. We implement a lightweight congestion control protocol to meet the unique requirements of Rina and its utilization of INA capabilities. Furthermore, when compared to PS-based INA, Rina provides superior incremental deployment capabilities. We discuss in detail in § IV-D how Rina achieves incremental deployment capability and how incremental deployment is carried out.

A. Agent-worker Model

Figure 6 presents a comprehensive illustration of the agent-worker model. **Rina adopts the rack as its operative unit (i.e., an abstracted worker), superseding each rack's ToR switch with an INA switch to enable the INA capability, which is called "group".** When viewed from an individual rack's perspective, once its ToR switch is supplanted by an INA switch, the worker of the lowest rank within that rack assumes the responsibility of managing the rack's INA switch and its workers. This low-rank worker is termed the "agent". In addition to performing computational tasks, the agent also performs several additional functions, such as initiating Rina, assigning tasks to the INA switch, and provocation of synchronizations for other workers.

B. Rina's Architecture, Workflow, and Dataflow

As illustrated in Figure 7, Rina is an architecture facilitating the harmonious operation of regular and INA switches. If a rack with an INA-enabled ToR switch exceeds two nodes,

Rina can be deployed, designating such a rack as an abstracted worker and Rina-enabled rack. Rather than assigning tasks individually, Rina adopts a group-based approach, considering Rina-enabled racks as an abstracted worker. Conversely, for Rina-disabled racks, each worker is regarded as an autonomous worker or an autonomous group.

1) Model and Dataset Partitioning: Data-parallel DDL training usually necessitates dataset partitioning to attain parallelism, with the addition of model splitting by RAR to guarantee synchronization throughput. Consequently, Rina calls for meticulous consideration in the partitioning of data and models. In both PS and RAR architectures, when all workers share equivalent computational capacity, an equal dataset segment is allocated to each worker to approximate global computation time. In contrast, with workers possessing heterogeneous computational capabilities, strategies such as batch size tuning [30], [31] are employed to synchronize computation time.

Regarding model partitioning during synchronization, the conventional RAR strategy divides the model evenly across workers to ensure near-equal synchronization time per step (refer to § II-B). However, Rina necessitates model partitioning in line with the number of groups. This stipulation arises primarily because each rack enabled by Rina, represented as an independent worker by its respective agent and the INA switch, must be counted on.

At the onset of training, both the dataset and model partitions are disseminated to ensure uniform initial parameter states across all workers. We designate a global control node, which will be the agent of the 0th group or 0th autonomous worker. As training commences, this node randomizes all parameters of the target model and transmits the partition data of both the dataset and model to all groups (Figure 7-(⓪)). To expedite the distribution of control messages within a Rina-enabled rack, Rina utilizes multicast. Once this preparatory phase is complete, all workers embark on their training tasks.

2) Synchronization Process: Before each round of DDL training, synchronization is necessitated for all workers to synchronize training results based on each node's respective dataset. This periodic process aligns well with near-equal computation times across nodes, enabling all workers to commence synchronization approximately simultaneously. As in the RAR system, upon completion of computation, each node promptly transmits its model partition's gradient to the subsequent worker while concurrently receiving the gradient from the prior worker and conducting local aggregation. This attribute is retained in Rina. For autonomous workers, their actions mirror those in RAR. Meanwhile, within Rina-enabled racks managed by the agent, the agent handles task delegation.

Similar to RAR, Rina is divided into two phases: ScatterReduce and AllGather. During the ScatterReduce phase, all nodes undergo a synchronization round, ensuring each node acquires the final computation result for a model gradient segment. This gradient portion is then broadcasted to all nodes in the AllGather phase. Participation of the INA switch in aggregation is necessary for ScatterReduce, while AllGather
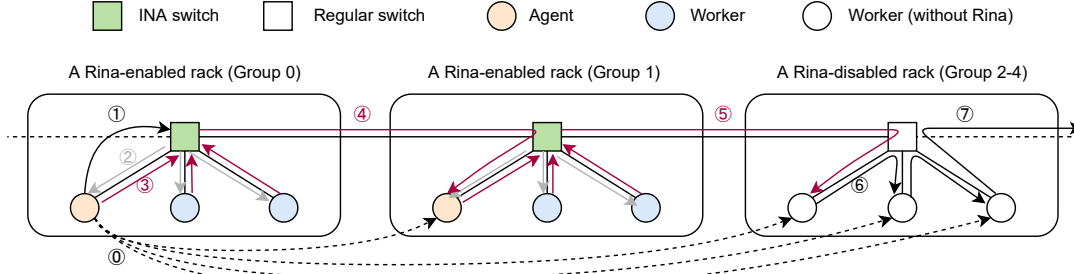
Fig. 7. Architecture and dataflow of Rina's ScatterReduce phase.

necessitates switch support in multicast.

In a cluster comprising N nodes, these two phases would necessitate $2(N-1)$ steps in the RAR system. In contrast, in Rina with G groups, each phase demands $2G-1$ steps. Given that a group may include several workers, in a cluster where each rack hosts eight computing nodes, G would equate to $N/8$. Thus, the synchronization steps demanded by Rina are notably fewer than those required in RAR, which contributes to a higher throughput.

3) ScatterReduce Phase: Given the inability of prevalent P4 programmable switches such as Tofino-1 [32] to autonomously generate packets, and considering synchronization requirements, the synchronization signals of workers are uniformly triggered by the agent for each group.

Referring to Figure 7, upon the agent's completion of its current computation round, it relays aggregation task data to the switch (①). This information comprises the model range and size destined for the next group, the reserved memory space in the switch, and the ID of the node currently engaged in the aggregation. On receipt of this data, the switch converts this packet into a data pull message (②), which is multicast to all workers (including the agent) under this rack. Triggered by the pull message, all nodes initiate the transmission of corresponding gradients to the ToR switch (③). These parameters are then aggregated at the ToR switch and dispatched to the agent of the subsequent group (④ and ⑤). When the next group's agent receives the aggregated results from (④ and ⑤), it combines them with the corresponding local gradients. These gradient portions are then used for synchronization in the ensuing stage.

For autonomous workers, the synchronization mechanism of Rina necessitates only minor adjustments to RAR. As illustrated in Figure 7, when the subsequent hop of an abstracted worker is an autonomous worker, this part of communication devolves to a standard RAR operation (⑥). Conversely, while transmitting data from an autonomous worker to the abstracted worker, the workflow stays the same as the synchronization between abstracted workers. This guarantees minimal alterations to existing RAR within Rina and supports compatibility with regular RAR.

4) AllGather Phase: When a specific partition of the gradients has been globally aggregated (i.e., after traversal through G groups), it promptly enters the AllGather phase. During this stage, the gradients are broadcasted to all workers via
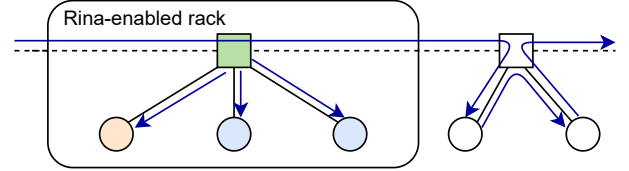

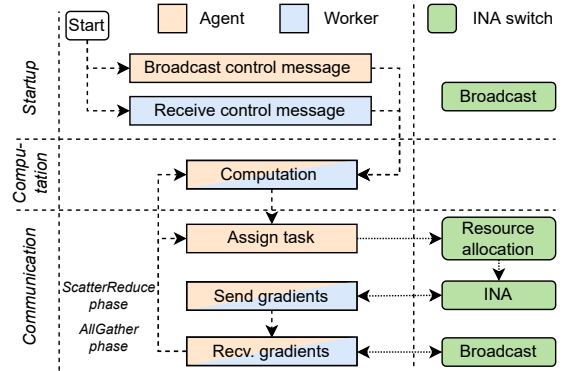Fig. 8. Dataflow of Rina's AllGather phase (Legends are the same as Fig. 7).

Fig. 9. The workflow of Rina.

a ring propagation path. In Rina, we harness the capabilities of INA switches to facilitate multicast, thereby enhancing the AllGather phase's performance.

As depicted in Figure 8, when an agent obtains a gradient shard with the same ID as the one it initially synchronized with, the AllGather process for that gradient commences. This agent then forwards the gradient shard to the next group's agent. If the subsequent group is an autonomous worker, it will locally store the gradient shard and further pass it on to the next worker, which is the same as RAR. However, if the next group is an abstract worker, the corresponding INA switch will multicast this gradient to all its workers and the subsequent group. This design allows for faster broadcasting of the aggregated gradients to all workers compared to RAR. The complete workflow, which encompasses task initiation, computation, and the synchronization process, can be referred to in Figure 9.

C. Congestion Control and Reliability

1) Congestion Control: The implementation of effective congestion control can mitigate network congestion and control the memory bottleneck of INA switches. Rina designs

distinct congestion control mechanisms for ScatterReduce and AllGather. Given that the INA switch and the node maintain a one-hop distance in Rina, the congestion control can remain straightforward. Nonetheless, it necessitates cross-rack congestion prevention measures to avert substantial packet loss.

As illustrated in Figure 7, congestion control initiates when (②) prompts workers in the Rina-enabled rack to deliver gradient shards to the INA switch. During the ScatterReduce phase, workers transmit at full speed [1], paralleling DCQCN's congestion control [34]. Once the aggregate message reaches the following rack's agent, it returns an ACK. At this stage, congestion control employs an Additive Increase/Multiplicative Decrease (AIMD) approach, ceasing window expansion upon reaching "full speed".

During the AllGather phase, as other gradient shards' ScatterReduce phase might not have concluded, step (④) in Figure 7 monopolizes the agent's downstream bandwidth. Consequently, the sending rate at this juncture starts from zero. The ACK is issued either by the autonomous worker or the agent of the abstracted worker.

2) Reliability: Despite recent methods' inability to manage worker errors, the PS-based INA method's central management node offers a significant advantage. It allows for active node exclusion upon error detection, enhancing overall reliability. Such a feature is lacking in the RAR approach due to its Peer-to-Peer (P2P) architecture. The P2P nature of RAR results in difficulties in making node replacement upon error detection challenging. However, the architecture of Rina is designed to effectively manage errors. We classify node errors into two categories: agent errors and worker errors. Upon the occurrence of an agent error, the other workers in the corresponding rack can default to regular RAR, ensuring uninterrupted training. If a worker error occurs and the worker is part of a Rina-enabled rack, its corresponding agent can promptly detect and exclude the faulty node from subsequent aggregation processes. If the worker is autonomous, other workers will automatically bypass the node.

D. Incremental Deployment

Rina's agent-worker model can integrate all nodes under an in-network computing switch into a single worker. This means that replacing a conventional ToR switch connecting N nodes with an INA switch can reduce the length of the RAR dependency chain by N. Therefore, in the initial deployment phase, we should prioritize replacing normal switches with the most connected workers with INA switches.

As deployment, we consider replacing other normal switches in the topology that are not ToR switches. Refer to § III-B, constructing a minimum spanning tree rooted at a specific worker node that connects all worker nodes. Then, by treating an INA switch and all its downstream worker nodes as a single worker, we can similarly replace the conventional switch with the most downstream nodes with an INA

switch. Through gradual replacement of standard switches with those developed using our method, we can achieve effective incremental deployment capabilities. Each instance of switch replacement leads to a noticeable enhancement in the throughput of DDL tasks. A detailed evaluation of incremental deployment capabilities is provided in § VI-C.

It is worth noting that Rina does not impose requirements on the type of network topology. Different network topologies can lead to varying degrees of performance improvement when deploying Rina, but most will still experience performance improvements. Fundamentally, Rina's essence lies in improving performance by reducing the dependency chain length in RAR. When we replace a ToR switch in the topology, we can reduce the dependency chain length corresponding to that rack. Rina also does not have a hierarchical concept, as each node is considered equal in the context of RAR. We are currently considering designing a hierarchical RAR (similar to H-AR), which is one of the future work.

V. IMPLEMENTATION

We implement the Rina prototype on both P4 programmable switches [32] and workers. The deployment on the switches emphasizes INA capabilities, whereas the implementation on worker nodes principally aims to enhance the processing performance of small packets.

1) P4 Programmable Switch: Given the absence of floating-point computation capabilities in P4 switches, Rina adopts a method analogous to ATP [7], where all floating-point numbers are multiplied by an integer and then converted to integers at the worker nodes. This conversion enables the P4 switch to transform floating-point addition into a simpler integer addition operation. Moreover, considering hash-based memory allocation algorithm may cause collisions [7], Rina allocates contiguous memory space directly for INA tasks. To augment the INA throughput, we utilize the P4 switch's recirculate feature to extend the length of each INA packet.

2) Worker: We implement a Rina prototype as middleware, seamlessly integrated into PyTorch. By utilizing UDP, we develop a customized transport protocol that leverages Mellanox (NVIDIA) Raw Ethernet Programming [35] to expedite the processing of user-space data packets. To enhance packet processing performance, we also employ TCP segmentation offload (TSO) [36].

VI. EVALUATION

In this section, we evaluate the advantages of Rina compared to commonly used DDL training synchronization architectures, through simulation experiments and testbed experiments. The evaluation includes throughput, incremental deployment capabilities, and robustness.

A. Evaluation Setup

1) Simulator: We use the popular NS3 [37] simulator as our simulation tool. We developed real worker nodes and PS in NS3 and implemented the logic of PS and RAR synchronization architectures. For switches, we use the switch component

---

[1]"Full speed" here denotes the speed aligned with the INA's capability. For instance, for the Tofino-1 P4 switch with 100G ports, due to hardware restrictions, the INA switch typically achieves a speed of around 20Gbps [33].

(a) Fat-tree (k=4) with 16 workers.

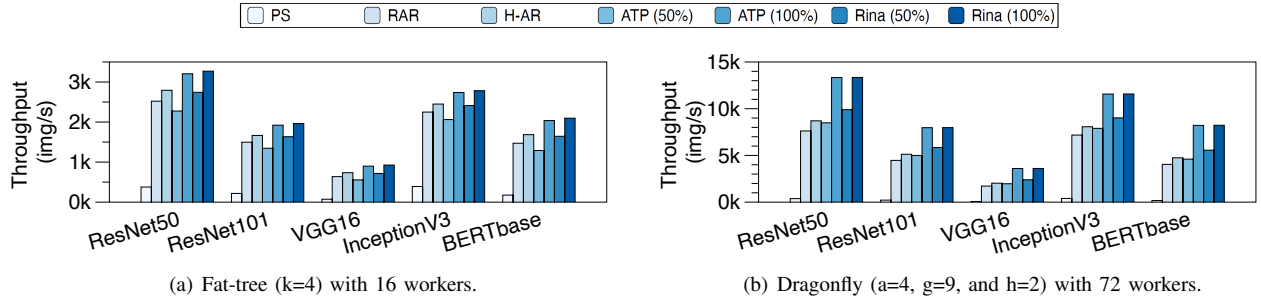(b) Dragonfly (a=4, g=9, and h=2) with 72 workers.

Fig. 10. Evaluation on throughput through large-scale simulation.

to simulate regular switches and use nodes to implement the simulation logic of INA switches. In the simulation, we additionally evaluate the incremental deployment capabilities of various methods in popular data center topologies. These topologies include standard Fat-tree [28] (k=4) and standard Dragonfly [29] (a=4, g=9, and h=2).

2) Testbed Configuration: We evaluate Rina using an 8-node cluster. The nodes are separated into 2 racks, for each rack has 4 nodes. They are interconnected through two Intel Tofino-1 P4 programmable switches (with 32x100Gbps ports) as the ToR switch. Each node has one AMD Epyc 7643 CPU (48 cores, 96 threads), 128GB RAM, and one Mellanox ConnectX-6 Ethernet Network Adapter with 2 100Gb ports. Additionally, each node has one NVIDIA RTX3090 GPU. The NVIDIA driver version is 460.91.03, and the CUDA [38] version is 11.2. The operating system is Ubuntu 20.04.2 with kernel version 5.15.0-75-generic. We use these 2 switches and 8 workers to build a spine-leaf simple topology for the evaluation of Rina.

3) Workload: In the evaluation, we conduct several experiments to evaluate the performance of 5 DL models and 4 datasets. The workloads include training ResNet50 [26] and VGG16 [39] models on the CIFAR10 [27], InceptionV3 [40] on the CIFAR100 [27], ResNet101 [26] on the ImageNet1K [41], and the BERTbase [42] model on the SQUAD1.1 [43]. The task for the BERTbase model is the fine-tuning task on the SQUAD1.1 dataset. The batch sizes for all image classification models are set as 64, while BERT is 12. In all results, the throughput unit for BERTbase is one question-answer pair per 10 seconds. All other hyper-parameters stay as default. These workloads are chosen to enable a thorough understanding of the strengths and limitations of Rina.

4) Baseline and Metric: We compare Rina with regular PS, RAR, H-AR [25], and PS-based INA (ATP [7]). The PS-based approaches use co-located PS for better performance. H-AR is a widely used method in the industry, achieving improved parallel performance than RAR. The two INA methods ATP and Rina ensure that the INA switches used have no memory bottlenecks and have similar aggregation throughput. We chose ATP here because it inherently offers higher throughput compared to SwitchML [6], and it can also support more complex topologies [7]. We evaluate their gains in throughput by gradually replacing the regular switches in the corresponding topology with INA switches.
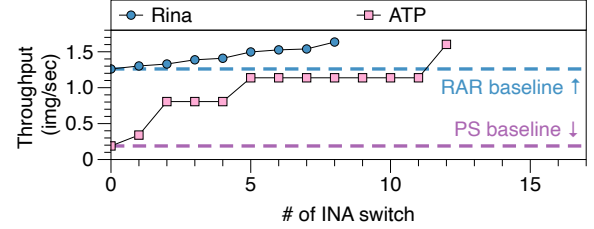
(a) Fat-tree (k=4).
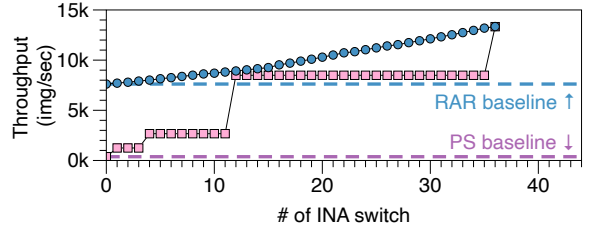
(b) Dragonfly (a=4, g=9, and h=2).

Fig. 11. Evaluation on incremental capability.

B. Evaluation on Throughput

We assess Rina's throughput using five distinct DL models across both Fat-tree and Dragonfly topologies. Initially, we establish PS and RAR as baselines and juxtapose ATP and Rina at 50% and 100% INA switch replacement rates. In the Fat-tree topology, 50% replacement entails using 6 INA switches in ATP and 4 in Rina, while for Dragonfly, it implies employing 18 INA switches in both methods. As depicted in Figure 10, Rina significantly exceeds the common baselines of PS and RAR. H-AR outperforms RAR in terms of performance, but Rina can achieve better throughput than H-AR by replacing only half of the network switches. Compared to ATP, Rina can significantly enhance throughput by replacing 50% of the network switches with INA switches. Furthermore, after replacing all switches with in-network computing switches, Rina performs comparably to ATP and even surpasses ATP in the ResNet50 model. This indicates that the integration of INA switches in Rina yields superior benefits and elevates overall performance. For ease of presentation, we express the throughput of the BERTbase model as Questions and Answers (QAs) every 5 seconds.

C. Evaluation on Incremental Capability

We further evaluate the incremental deployment capabilities of ATP and Rina using ResNet50, featuring a model
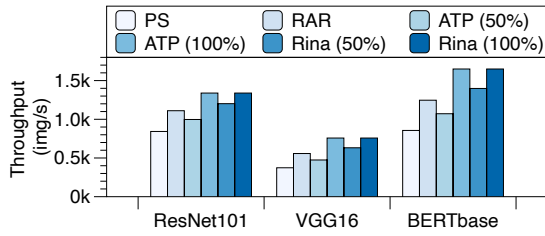
Fig. 12. Testbed verification with 8 workers.

with 98MB parameters. In both topologies, we progressively replace all switches with INA switches and measure their throughput. As illustrated in Figure 11, the throughput of Rina gradually increases as the count of INA switches rises. On the other hand, ATP, due to its lack of incremental deployment capabilities, only witnesses a throughput boost after a significant quantity of INA switch replacements. This indicates that with Rina, DDL training operators can experience performance enhancements proportional to their investment, thus offering substantial hardware cost-effectiveness.

D. Testbed Verification

We assess Rina's performance on a testbed, with the outcomes depicted in Figure 12. The results reveal that Rina mirrors the performance advantages observed in the simulation. Its performance remains closely matched with ATP when all switches are replaced by INA switches. However, when only a fraction of switches are replaced, Rina can attain a training throughput surpassing ATP. This suggests that Rina preserves comparable performance benefits in real-world situations and provides enhanced incremental deployment capabilities compared to PS, RAR, and other PS-based INA methodologies.

VII. DISCUSSION

1) Improving Robustness: Rina confers a centralized control mechanism, enhancing the robustness of the cluster. However, specific design details and comprehensive evaluations are still needed. We are currently conducting experiments to make Rina to provide increased reliability, expedited error recovery, and dynamic scalability, further strengthening the robustness of DDL training clusters. Moreover, Rina does not address the issue of node heterogeneity. The heterogeneity in node computational capabilities should be managed through other methods such as batch-size tuning [31].

2) Combined with Model Parallelism: Although this paper primarily discusses data parallelism and does not address model parallelism, it is worth noting that RAR is also frequently used in widely-used model parallel synchronization strategies. Consequently, if we attempt to introduce INA into model parallelism, Rina can be seamlessly integrated. We are continuing to investigate the potential challenges and solutions associated with incorporating Rina into model parallelism.

VIII. RELATED WORK

1) Reduce the Communication Size in DDL: Communication compression approaches are proposed to reduce communication costs. Stochastic Rounding [44] randomly rounds the

parameters in a method that preserves the expected value of the parameters. QSGD [45] generalizes stochastic rounding to stochastic quantization and proposes multi-level gradient quantization schemes to further reduce transmission costs. Gradient sparsification can also reduce the communication size. A representative method of gradient sparsification [46] uses a static threshold to decide which gradients to send. LTP [5] utilizes the loss-tolerant transmission to reduce the communication time. Deep gradient compression [47] considers local gradient accumulation and guarantees the convergence of the training by accumulating momentum locally. These methods are orthogonal to Rina. However, all INA approaches require modifications to meet the needs of these methods.

2) Communication Synchronization: Synchronization models greatly affect the performance of DDL training. BSP [16] is a classical synchronous framework. Stale-synchronous parallel (SSP) [48] aims to alleviate the straggler problem of BSP without losing synchronization by allowing faster workers to do more updates without waiting for slower ones, but still guarantees a staleness-bounded barrier. Compared with the SSP, ASP [17] eliminates the synchronization. Each work transmits its gradients to the PS after it calculates the gradients. OSP [31] uses a 2-stage synchronization to reduce communication and speed up the training throughput. Local SGD [49] allows all workers to run a specific number of local updates independently before synchronization to guarantee good training accuracy. These approaches modified the synchronization architectures and are not applicable to INA capabilities.

3) INA Approaches: SwitchML [6] design a communication primitive to perform parts of the model aggregation within the network. ATP [7] explores the idea of partitioning aggregation functionality between switches and servers so as to seamlessly support multi-tenant scenarios. PANAMA [9] proposes a special transport layer protocol for load balance and congestion control. ASK [33] uses a key-value data structure for in-network aggregation to support compressed gradients aggregation. INAlloc [8] takes switch memory resources into consideration and designs a memory management mechanism to fully utilize memory resources in clustered switches. These methods are mostly discussed based on the PS scenario and have not attempted to integrate INA capabilities into RAR.

IX. CONCLUSION

In this study, we find that state-of-the-art PS-based INA approaches like ATP lack incremental deployment capabilities through mathematical modeling. This is detrimental to the construction and upgrade of the existing data center. Based on these issues, we propose Rina, which is known as the first to introduce INA capabilities into the RAR architecture. Rina not only provides excellent incremental deployment capabilities, but also greatly alleviates the problem of long dependency chain issues in RAR for throughput degradation. Through extensive testbed and simulation evaluations, we verify that Rina achieves better throughput and deployment cost-effectiveness than PS, RAR, and PS-based INA approaches under various topologies, models, and datasets.

REFERENCES

[1] OpenAI, "Gpt-4 technical report," 2023.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] F. Stahlberg, "Neural machine translation: A review," *Journal of Artificial Intelligence Research*, vol. 69, pp. 343–418, 2020.

[4] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," *arXiv preprint arXiv:2304.02643*, 2023.

[5] Z. Chen, L. Shi, X. Liu, X. Ai, S. Liu, and Y. Xu, "Boosting distributed machine learning training through loss-tolerant transmission protocol," *arXiv preprint arXiv:2305.04279*, 2023.

[6] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtárik, "Scaling distributed machine learning with {In-Network} aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 785–808.

[7] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. Swift, "{ATP}: In-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 741–761.

[8] B. Zhao, J. Dong, Z. Cao, W. Nie, C. Liu, and W. Wu, "Enabling switch memory management for distributed training with in-network aggregation," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications (to appear)*. IEEE, 2023.

[9] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 829–844, 2021.

[10] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *arXiv preprint arXiv:1811.03600*, 2018.

[11] B. Forrest, D. Roweth, N. Stroud, D. Wallace, and G. Wilson, "Implementing neural network models on parallel computers," *The Computer Journal*, vol. 30, no. 5, pp. 413–419, 1987.

[12] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang *et al.*, "Overlap communication with dependent computation via decomposition in large deep learning models," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2022, pp. 93–106.

[13] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th international conference on international conference on machine learning*, 2011, pp. 265–272.

[14] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 583–598.

[16] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[17] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," *Advances in neural information processing systems*, vol. 28, 2015.

[18] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf *et al.*, "Mlperf training benchmark," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 336–349, 2020.

[19] M. Bóna, *A walk through combinatorics: an introduction to enumeration and graph theory*. World Scientific, 2006.

[20] X. Wan, H. Zhang, H. Wang, S. Hu, J. Zhang, and K. Chen, "Rat-resilient allreduce tree for distributed machine learning," in *4th Asia-Pacific workshop on networking*, 2020, pp. 52–57.

[21] NVIDIA, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl, 2024.

[22] O. MPI, "Open mpi: Open source high performance computing," https://www.open-mpi.org/, 2024.

[23] A. Eisenman, K. K. Matam, S. Ingram, D. Mudigere, R. Krishnamoorthi, K. Nair, M. Smelyanskiy, and M. Annavaram, "Check-n-run: a checkpointing system for training deep learning recommendation models," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 929–943.

[24] E. Rojas, A. N. Kahira, E. Meneses, L. B. Gomez, and R. M. Badia, "A study of checkpointing in large scale training of deep neural networks," *arXiv preprint arXiv:2012.00825*, 2020.

[25] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," *arXiv preprint arXiv:1807.11205*, 2018.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[28] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[29] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *2008 International Symposium on Computer Architecture*. IEEE, 2008, pp. 77–88.

[30] C. Chen, W. Wang, and B. Li, "Round-robin synchronization: Mitigating communication bottlenecks in parameter servers," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 532–540.

[31] Z. Chen, L. Shi, X. Liu, J. Li, S. Liu, and Y. Xu, "Osp: Boosting distributed model training with 2-stage synchronization," *arXiv preprint arXiv:2306.16926*, 2023.

[32] Intel, "Intel® tofino™ programmable ethernet switch asic." [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[33] Y. He, W. Wu, Y. Le, M. Liu, and C. Lao, "A generic service to provide in-network aggregation for key-value streams," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 33–47.

[34] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.

[35] Nvidia, "Raw ethernet programming: Basic introduction - code example." [Online]. Available: https://enterprise-support.nvidia.com/s/article/raw-ethernet-programming–basic-introduction—code-example

[36] NVIDIA, "Tcp segmentation offload." [Online]. Available: https://docs.nvidia.com/networking/display/rdmacore50

[37] "ns-3 network simulator," https://www.nsnam.org/, 2011-2022.

[38] Nvidia, "Cuda, release: 11.2," 2020. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

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

[40] X. Xia, C. Xu, and B. Nan, "Inception-v3 for flower classification," in *2017 2nd international conference on image, vision and computing (ICIVC)*. IEEE, 2017, pp. 783–787.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

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

[43] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[44] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[45] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[46] N. Ström, "Scalable distributed dnn training using commodity gpu cloud computing," 2015.

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

[48] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," *Advances in neural information processing systems*, vol. 26, 2013.

[49] S. U. Stich, "Local sgd converges fast and communicates little," *arXiv preprint arXiv:1805.09767*, 2018.