RAIL: ReArranging Inter-GPU Links for GPU-Centric Clusters

Haixin Nan, Jun Xu, Peirui Cao, Zhaochen Zhang, Yizhi Wang, Zhehao Lin, Yuhang Li, Chengyuan Huang, Wentao Fan, Xiaohu Xu, Zhongming Ji, Shengju Zhang, Dongxu Wang, Lingkun Meng, Rong Gu, *Member, IEEE*, Guihai Chen, *Fellow, IEEE*, and Chen Tian, *Senior Member, IEEE*

***Abstract*—In modern GPU-centric clusters, large-scale AI training relies on two distinct communication domains: a high-bandwidth intra-node domain using proprietary interconnects (e.g., NVLink), and a scale-out inter-node network domain (e.g., RDMA). We observe that the widely-used ring algorithm, often create a significant load imbalance across these domains. This leads to the counter-intuitive scenario where the expensive, high-bandwidth intra-node domain becomes a performance bottleneck, while the inter-node network remains underutilized. This inefficiency is further exacerbated by the disparity in bandwidth provisioning: inter-node network bandwidth is generally more cost-effective and accessible, whereas intra-node bandwidth is often proprietary and more costly to scale. To address this fundamental imbalance, we propose RAIL, aimed at resolving the intra-node bottleneck by strategically rearranging inter-GPU communication paths. This rebalancing ensures that traffic loads are appropriately matched with the distinct transmission capabilities of each domain, thereby maximizing overall communication performance. RAIL incorporates a Load Distributing Strategy (LDS) that can accurately partition physical nodes into logical nodes based on the a transmission capabilities of both domains, shifting excess traffic from the overloaded intra-node domain to the underutilized network domain. Additionally, the Intra-Rail Strategy (IRS) leverages topological characteristics to ensure optimal communication paths through the network domain between logical nodes. Our evaluation demonstrates that RAIL effectively mitigates congestion and achieves a 30.7% average increase in collective communication bus bandwidth compared to the widely-used NCCL solution.**

***Index Terms*—GPU-centric clusters, collective communication.**

I. INTRODUCTION

IN RECENT years, the emergence of large-scale model training has rapidly increased the demand for computational and memory resources in data centers. With the slowdown of Moore's Law, performing distributed training on scalable GPU clusters has become the main approach to support ultra-large models with trillions of parameters [1]. Collective communication plays a crucial role in distributed training, facilitating the transfer of model parameters and intermediate computation results among GPUs in clusters.

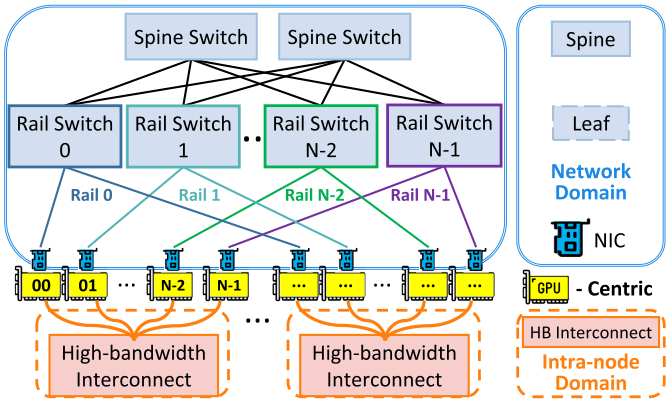


Fig. 1. In GPU-centric cluster, GPUs connect both the network domain, composed of NICs and switches, and the intra-node domain, formed by high-bandwidth interconnects.

However, many studies have reported that the communication time cost is significant [2], [3] and can account for more than 60% of the total training time [4], making it a major bottleneck in accelerating distributed training. For inter-GPU communication, GPUs within a node are interconnected through high-bandwidth, short-range links, such as NVLink [5] or InfinityFabric [6], forming what is referred to as the high-bandwidth domain [7] as shown in Figure 1. Across nodes, GPUs are linked via high-performance RDMA networks, forming the "network domain". These two domains connect GPUs to form GPU-centric clusters. During collective communication, selecting appropriate data transmission path for GPUs based on topology and transmission performance is critical for maximizing bus bandwidth [8], as bus bandwidth is inversely proportional to completion time. It can improve hardware utilization and reduce model training time cost.

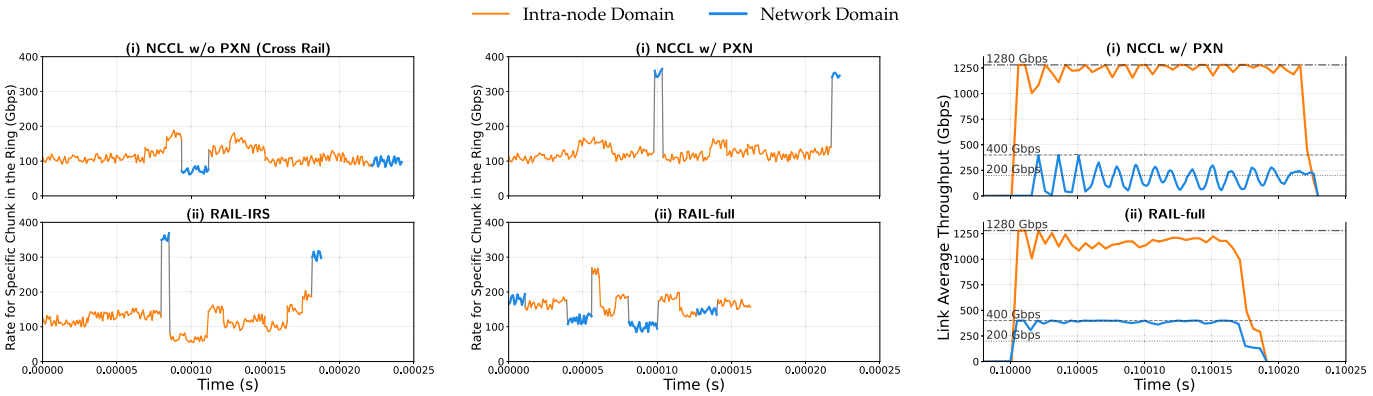
With the plateau of the scaling law for large-scale AI models [9] and considering the associated costs, startups are using GPUs with more modest interconnect bandwidths to build clusters, while maintaining improvements in model performance [10]. Additionally, according to recent studies,

Received 12 March 2025; revised 7 September 2025 and 14 December 2025; accepted 22 January 2026; approved by IEEE TRANSACTIONS ON NETWORKING Editor J. P. Jue. Date of publication 28 January 2026; date of current version 3 February 2026. This work was supported in part by the National Key Research and Development Program of China under Grant 2024YFB2906700; in part by NSFC under Grant 62325205, Grant U25B2035, and Grant 62502194; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20243053; and in part by Nanjing University–China Mobile Communications Group Company Ltd. *(Corresponding author: Peirui Cao.)*

Haixin Nan, Peirui Cao, Zhaochen Zhang, Yizhi Wang, Zhehao Lin, Yuhang Li, Chengyuan Huang, Rong Gu, Guihai Chen, and Chen Tian are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: caopeirui@nju.edu.cn).

Jun Xu, Wentao Fan, Xiaohu Xu, Zhongming Ji, Shengju Zhang, Dongxu Wang, and Lingkun Meng are with China Mobile (Suzhou) Software Technology Company Ltd., Suzhou 215000, China.

Digital Object Identifier 10.1109/TON.2026.3658194



(a) Cross-rail communication severely degrades the transmission rate in the network domain.

(b) RAIL balances the load disparity between the two domains and avoids cross-rail, accelerating chunk transmission within the ring.

(c) RAIL balances the load disparity between the two domains, maximizing the utilization of link bandwidth in both domains.

Fig. 2. The traversal of a chunk through the ring guided by RAIL's strategies, the impact on the links under collective communication across GPUs, and a performance comparison between cross-rail, baseline, RAIL-IRS and RAIL-full, answering Q2 in section VI. We use a rail-optimized cluster based on NVIDIA H800-NVLink and Mellanox CX-7 (16 nodes, all 8 leaf switches in a single subtree) and ALLGATHER as the workload, where the hardware configurations and bandwidths of both domains are the same as those in [10]. The smaller number of GPUs and the use of ALLGATHER, which only involves communication, allow us to clearly and intuitively observe changes in communication performance. The collective communication data size is 64MB.

the intra-node domain often bears excessive load, yet its actual transmission rate only reaches 80% of its nominal value [10], [11], [12], potentially leading to congestion and further performance degradation [11]. The aforementioned issue with the intra-node domain makes it a bottleneck in collective communication paths (see Figure 2b(i)). In contrast, the network domain is often provisioned with resources to scale the cluster and is designed with various congestion management algorithms to prevent bottlenecks [13], [14], [15]. In practical distributed model training, the network domain remains underutilized with low loads [16].

Existing algorithms [11], [17], [18], [19] focus on independently selecting paths for inter-GPU communication, either within the network domain or the intra-node domain, neglecting the gap in load and transmission capacity between the two domains. To address the aforementioned issues in the state-of-the-art rail-optimized GPU-centric clusters [20], the following requirements must be met:

Matching load and transmission capability: How to schedule the ***intra-node*** communication paths to achieve precise matching between the load and the transmission capacity of both domains? Ideally, the load in both domains should be proportional to their respective transmission capacities (i.e., bandwidth). This ensures that each link in the path takes equal time to transmit data chunks, avoiding bottlenecks.

Eliminating cross-rail traffic: How to select ***inter-node*** paths that align with the characteristics of the rail-optimized topology? Cross-rail traffic severely degrades performance (see Figure 2a), so it is crucial to eliminate cross-rail paths by considering the topology's characteristics when optimizing communication paths.

Scalability in heterogeneous clusters: How to adapt to clusters with varying domain transmission capacities and different cluster sizes? In clusters where transmission capabilities are heterogeneous across domains, how can the communication paths be adjusted? Moreover, can the efficiency of path planning be maintained as the cluster scales?

Our approach: RAIL. In this work, we propose RAIL, which stands for ReArranging Inter-GPU Links. We firstly analyze the load and bandwidth characteristics of the two domains in GPU-centric clusters. Based on this analysis, we rearrange the inter-GPU links, forming rings that connect all GPUs involved in the collective communication. To the best of our knowledge, RAIL is the first work that optimizes inter-GPU links by balancing the intra-node domain and the network domain, while leveraging the characteristics of the rail-optimized topology. Our contributions are summarized as follows:

· *Contribution 1: Revealing collective communication bottleneck.* We rethink the load characteristics induced by the ring algorithm in GPU-centric clusters, the bandwidth gap between the two domains, and the topological characteristics of the network domain. Through simulation modeling and analysis, we reveal that, in contrast to its intended role, the so-called high-bandwidth domain within nodes often becomes the bottleneck of collective communication bandwidth.

· *Contribution 2: Load distributing and intra-rail strategy.* We propose RAIL, consisting of Load Distributing Strategy (LDS) and Intra-Rail Strategy (IRS). For LDS, we transform the problem of matching load and transmission capacity between the two domains into a problem of partitioning logical nodes. This trade-off between the two domains prevents either domain from becoming the bottleneck in the ring. For IRS, leveraging the characteristics of the rail-optimized topology, we rearrange the inter-GPU links between logical nodes to ensure that all network domain traffic remains intra-rail, avoiding the severe performance degradation caused by cross-rail communication. RAIL perfectly satisfies the three requirements mentioned earlier and operates with constant time complexity.

· *Contribution 3: Extensive evaluation and analysis.* We extend the network system simulator ns-3 [21] and prototype RAIL based on it, conducting an extensive evaluation. Under mainstream cluster configurations, compared to the baseline of

NCCL with PXN [20], RAIL improves the bus bandwidth of ALLREDUCE, ALLGATHER, and REDUCESCATTER collective communication primitives by an average of 30.7%. In clusters with significant bandwidth differences between domains, RAIL increases the bus bandwidth by up to 96.4%. Additionally, RAIL significantly improves network domain bandwidth utilization while maintaining the cumulative average utilization below 70%. The bandwidth utilization of the intra-node domain also decreases from over 90% to an average of below 80%, avoiding congestion and preventing performance degradation.

II. BACKGROUND

A. GPU-Centric Cluster

As the size of model scales rapidly [22], [23], [24], exceeding the computational and memory limits of a single node, distributed training has become essential. To manage GPU communication challenges [25], [26] in distributed training, GPU-centric clusters, as shown in Figure 1, with heterogeneous topologies have been developed, featuring efficient intra-node interconnects and rail-optimized networks, referred to as the high-bandwidth domain and the network domain [7]. For consistency, the high-bandwidth domain will be referred to as the intra-node domain in the remainder of this paper.

Intra-node domain. Intra-node communication between GPUs is critical for maintaining efficient parallel processing and minimizing bottlenecks during distributed training [12]. For efficient intra-node connectivity, low-latency, short-range proprietary interconnect technologies have been developed, forming what we referred to as the intra-node domain. This domain, also referred to as the GPU scale-up domain, utilizes switch-based or P2P-based fully connected topologies, such as those seen in technologies like NVSwitch [27], NVLink [5], HCCS [28], and Infinity Fabric [6], provides interconnect bandwidths of several hundred GB/s between GPUs within a node, thereby accelerating the training process.

There remain several suboptimal intra-node interconnect options [29], [30], [31], [32]. For example, in the NVIDIA V100-NVLink system [33], eight GPUs are fully connected through NVSwitch and NVLink, offering a nominal unidirectional bandwidth of 150 GB/s between any two GPUs. These alternatives, while offering lower bandwidths, are more cost-effective and are often used to control costs, such as using lower-bandwidth interconnects or even relying on consumer-grade GPUs with PCIe connectivity.

Network domain. Inter-node communication is essential for scaling out computational resources and accelerating the training process. While traditional data centers often use generic network designs like Clos fabrics, the topology for large-scale AI training is typically co-designed with the distributed algorithms it supports. A foundational concept in this domain is the ***rail-optimized topology***.

The defining characteristic of this architecture, illustrated in Figure 1, is the logical grouping of GPUs into "**rails**". A rail consists of all GPUs that share the same local index across different physical nodes (e.g., the set of all "GPU 0s" from every node forms rail 0). Critically, all GPUs belonging to the same rail are physically wired to a single, dedicated leaf switch, with each GPU utilizing its own Network Interface Card (NIC) for this connection. This structure ensures that GPUs on the same rail have a direct communication path.

This design offers significant inter-node communication performance advantages by fundamentally optimizing network traffic patterns. The primary benefit is the containment of intra-rail communication: any traffic between GPUs on the same rail is resolved within their dedicated leaf switch. This design drastically minimizes the volume of traffic that needs to traverse to upper-tier spine switches, thereby reducing latency, contention, and the likelihood of routing hash collisions.

Furthermore, the architecture can be enhanced to handle cross-rail traffic more efficiently. For instance, NVIDIA's clusters leverage a technology known as PXN [20] (a combination of PCIe and NVLink) to create a bypass for traffic between GPUs on different rails but located within the same physical node, albeit at the cost of consuming valuable intra-node interconnect bandwidth. This allows certain cross-rail data exchanges to occur without ascending to the spine layer. Indeed, this PXN-based optimization is widely employed by NVIDIA to accelerate key collective operations such as ALLREDUCE and ALLTOALL [20].

Given these benefits, the rail-optimized topology and its derivatives have become a de facto standard for hyperscale AI training infrastructure, with widespread adoption by industry leaders such as NVIDIA [20], ByteDance [34], Tencent [35], and Alibaba [36].

B. Collective Communication

Collective communication is essential for distributed model training, enabling efficient data exchange across GPUs. Libraries like NCCL [17] implement these primitives using various algorithms, such as the double binary tree and ring algorithms.

Collective communication in distributed training. In distributed training, the choice of collective communication primitives depends on the parallelization strategy. Megatron [37] employs a PTD strategy: P for **p**ipeline parallelism; T for **t**ensor parallelism, where attention layers or MLPs are split across GPUs within a tensor group; and D for **d**ata parallelism, where each GPU in a DP group holds an identical model copy, and input data is split and distributed across GPUs. During the forward and backward passes of TP and DP, ALLREDUCE accumulates partial outputs and gradients across GPUs. Megatron also uses sequence parallelism to partition the LayerNorm and Dropout layers, employing ALLGATHER and REDUCESCATTER.

In DeepSpeed's ZeRO [38], ZeRO-1 and ZeRO-2 distribute optimizer states and gradients across GPUs. During the backward pass, REDUCESCATTER accumulates gradients for each GPU's portion of the optimizer state, followed by ALLGATHER to aggregate the updated weights. In ZeRO-3, weights, along with optimizer states and gradients, are also distributed. ALLGATHER collects the full set of weights during the forward pass, and again during the backward pass for gradient computation. REDUCESCATTER then retains only the relevant gradients per GPU. Unlike ZeRO-1 and ZeRO-2,

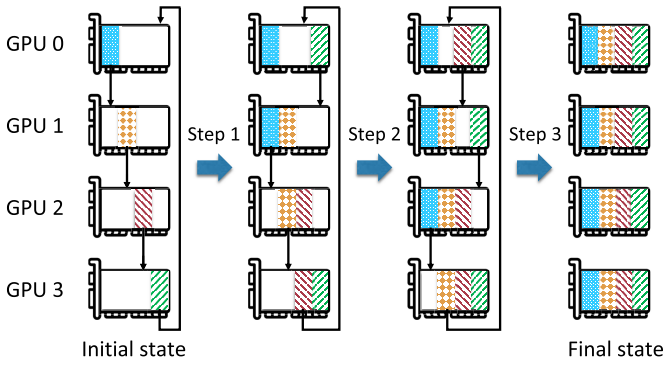


Fig. 3. Collective communication primitive AllGather under ring algorithm for four GPUs.

ZeRO-3 avoids an additional AllGather for weight updates, as each GPU maintains only a subset of the weights.

Ring algorithm. The ring algorithm is the primary subject of this paper. In this algorithm, all GPUs are logically connected in a ring, with each GPU receiving data from its predecessor and simultaneously sending data to its successor, as shown in Figure 3.

As the bandwidth-optimal algorithm [39], ring algorithm maximizes link bandwidth utilization and reduces the likelihood of direct link contention. Since data flows through communication path constructed by the ring algorithm, the overall bandwidth of the ring is determined by the bandwidth of the bottleneck link. Therefore, in the ring algorithm, it is crucial to minimize bandwidth disparities between links by carefully selecting paths to form the ring.

III. Motivation

When using the ring algorithm, the load and cost imbalances between the intra-node domain and the network domain, combined with the characteristics of cluster—rails and PXN, may reduce link efficiency and slow down collective communication, posing a challenge for distributed training.

A. The Intra-Node Domain Is High-Cost and Burdened With Excessive Load

In the intra-node domain, fully connected links between GPUs are typically provided by two primary architectures: the switch-based approach, exemplified by NVIDIA's NVSwitch [27], which uses a central, high-radix fabric to interconnect all GPUs; and the P2P-based approach, such as AMD's Infinity Fabric [6], which forms a dense mesh of direct, point-to-point links between the GPUs. However, because each ring traverses all GPUs within a node, as shown in Figure 4, this significantly burdens the inter-GPU links in intra-node domain. With large volumes of communication (where the ring algorithm excels and bandwidth becomes the key factor in communication completion time) and multiple connections, the accumulated ingress/egress bandwidth of the NVSwitch can experience a noticeable degradation due to internal resource contention and flow control mechanisms [11], demonstrating how excessive load turns the intra-node domain into a potential performance bottleneck. The profiling results of the $\alpha - \beta$ cost model for NVIDIA DGX-2 (interconnected with V100) in [11] also show that the β of its NVLink is 8 us/MB, corresponding to 125 GBps, which is only about 80% of the nominal 150 GBps.

Contrary to conventional understanding, we find that the so-called high-bandwidth domain used for intra-node GPU communication often becomes the bandwidth bottleneck in the ring algorithm due to the reasons mentioned above, slowing down the bus bandwidth and completion time of collective communication (see Figure 2b). This is also the deeper insight why we refer to it as the intra-node domain.

On the other hand, with the plateau of the scaling law for large-scale AI models [9], simply configuring better GPUs for the training cluster is no longer the most cost-effective approach. The recent DeepSeek V3 [10], for instance, uses only 2048 NVIDIA H800 GPUs [32], yet it performs excellently across a variety of benchmark tests while effectively controlling costs. The report mentions that the interconnect bandwidth of NVLink (i.e., the intra-node domain) is only 160 GBps, which is just 80% of the nominal 200 GBps of H800, and is far lower than the nominal 450 GBps of the H100 [40].

B. The Network Domain Is Readily Available But Suffers From Under-Utilization

In the network domain, each GPU has a dedicated network interface card (NIC) connected to a leaf switch. In NCCL's default communication ring construction, to fully utilize NIC resources, each NIC is typically traversed to attempt building a ring with each as the starting point. The construction is fundamentally topology-agnostic; it connects GPUs sequentially based on their communication rank, creating a single, logical communication loop that passes through each GPU exactly once. Since each ring only utilizes a NIC for one outgoing and one incoming transmission per node, the load on each NIC is usually kept low, as shown in Figure 4.

Several vendors offer high-bandwidth, low-latency RDMA-capable NICs (RNICs), such as Mellanox ConnectX-7 [41], BlueField-3, and Broadcom's P1400G, all of which support nominal rates of 400Gbps per port. Similarly, in the case of DeepSeek V3 [10], the interconnect bandwidth of IB (i.e., the network domain) in their cluster is 50 GBps. Intel's E810 [42] supports up to 200Gbps across dual ports. These vendors build open ecosystem and competition for RNICs, making high-bandwidth RNICs more accessible to users.

As the demand for computation and memory resources in distributed training grows, increasing scalability by adding network dimensions has become a trend, which further exacerbates the issue of low bandwidth utilization in the network domain [16].

C. The Characteristics of Topology Also Matter

To avoid intra-node congestion caused by PCIe bandwidth contention, GPUs typically use the NICs connected under the same PCIe switch. If the network topology's rail characteristics are not considered, as shown in Figure 4(a), GPU01 and GPU08 communicate through their respective NICs via the network domain. However, GPU01 and GPU08 are on different rails, causing their NICs to connect to two different

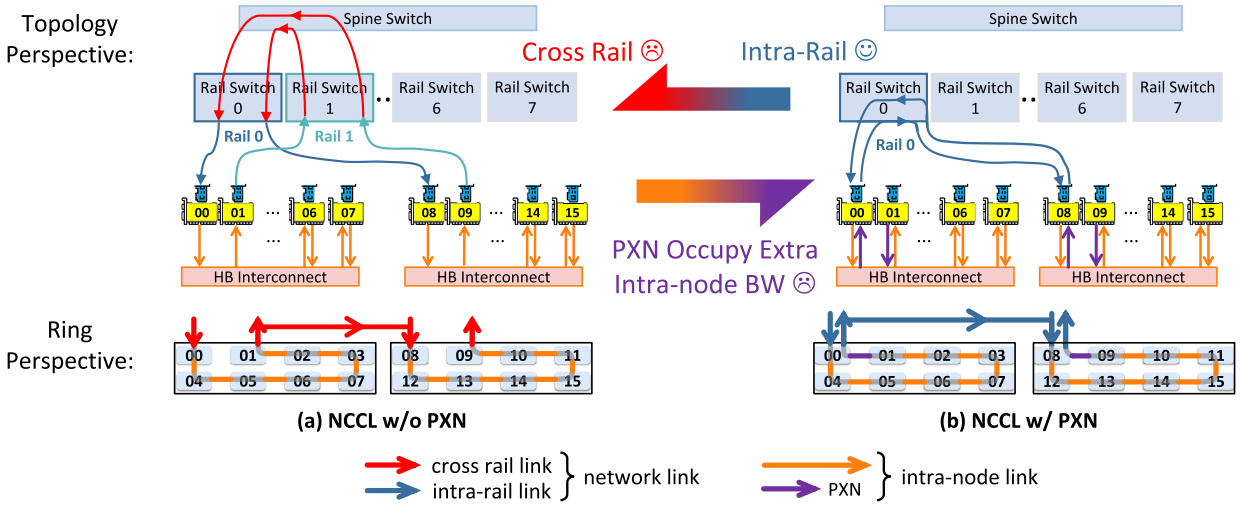
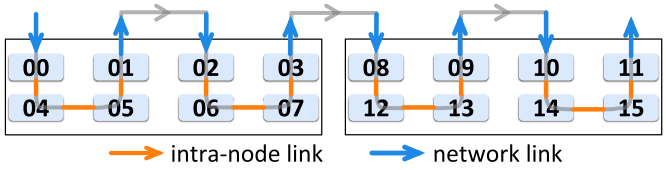


Fig. 4. Introducing Two Perspectives for Visualizing GPU Communication Topologies. This figure serves as a visual guide to differentiate between physical hardware layout (Topology Perspective) and logical algorithm flow (Ring Perspective), using a 16-GPU ring across two nodes as an example. (Top) The Topology Perspective illustrates the physical arrangement of GPUs connected by both network links and intra-node links. (Bottom) The Ring Perspective abstracts the physical complexity into a simple, ordered sequence of communicating ranks. Understanding these two views is crucial for interpreting the RAIL strategies presented in Figs. 5 and 7, which are primarily shown from the ring perspective.

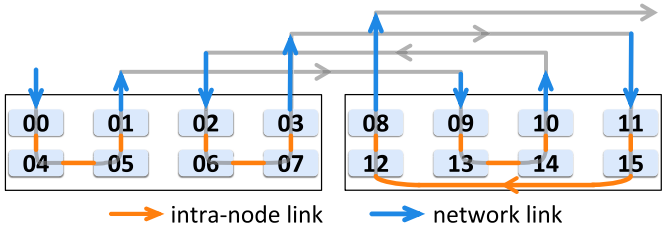
rail switches. As a result, their communication through the network domain cannot be directly achieved via a two-hop path using the rail switches; instead, it must be routed through the higher-level spine switch. This is known as **cross-rail** communication, whereas GPUs connected through the same rail switch are referred to as **intra-rail**. Due to the network over-subscription, increased collision probabilities from large-scale model training traffic, and more communication hops, cross-rail traffic experiences significant performance degradation (see Figure 2a). Therefore, in GPU-centric clusters, it is crucial to consider the network topology and ensure that network domain links remain intra-rail whenever possible.

To prevent cross-rail communication and enable the use of NICs on the same rail as the receiving GPU, NVIDIA introduced PXN technology in 2022 [20]. This approach routes data from the source GPU through the NVSwitch to a GPU within the same node and on the same rail as the destination GPU, before using the leaf switch for intra-rail communication, as shown in Figure 4(b). However, using NVSwitch for data relay consumes additional bandwidth resources in the intra-node domain, further increasing the load on it.

Therefore, it is crucial to account for the characteristics of the two-domain topology in the cluster when arranging inter-GPU links, considering both the rail and PXN features.



(a) An example of the inter-GPU rings formed by RAIL-LDS, which matches load and transmission capability between domains.



(b) An example of the inter-GPU rings formed by RAIL-full, which eliminates cross-rail traffic through RAIL-IRS on top of RAIL-LDS.

Fig. 5. The rings formed when $N = 8$ and $K = 2$, after sequentially applying RAIL-LDS and RAIL-IRS. The former ensures balanced load distribution, while the latter avoids cross-rail traffic.

IV. DESIGN OF RAIL

This section introduces RAIL, consisting of twofold design objectives:

- To reorganize the load distribution between the network and intra-node domains in GPU-centric clusters by partitioning logical nodes and trading off the load between the two domains based on their transmission capacity. The goal is to match the load and transmission capacity of the two domains by rearranging the paths, eliminating bottleneck links within the ring, and thereby improving collective communication bus bandwidth and link bandwidth utilization, corresponding to subsection III-A and subsection III-B.
- To leverage the characteristics of the GPU-centric cluster by ensuring intra-rail traffic within the network domain and forgoing PXN within the intra-node domain, thereby preventing the reduction of bus bandwidth and increase in latency caused by cross-rail traffic, while also avoiding additional load on the intra-node domain from PXN, corresponding to subsection III-C.

This section begins with a theoretical analysis of the bottlenecks in current intra-rail collective communication of ring algorithm, followed by an introduction to our **Load Distributing Strategy (LDS)** for the first goal, and the

TABLE I
CORE PROBLEMS ADDRESSED IN THIS WORK, CORRESPONDING SOLUTIONS, AND THEIR TERMINOLOGY

Problem	Solution	Terminology & Description
Mismatched load and transmission capacity between intra-node domain and inter-node network domain, aggravated by PXN. See Figure 4 and Figure 2b(i).	**Load Distributing Strategy (LDS):** A strategy that balances communication load across the two domains, by partitioning GPUs on a physical node into logical nodes, forcing ring traffic to traverse the inter-node network more frequently and evenly. Avoiding using PXN. See Figure 5a and Figure 2b(ii).	**Logical node:** A virtual group of GPUs within one physical node. The ring traverses all GPUs in a logical node before exiting to the inter-node network, acting as the core mechanism for the LDS to control traffic flow and enforce load balancing.
Cross-rail traffic: Suboptimal network traffic created when two GPUs, located on different nodes and connected to different rail switches, are placed adjacently in the ring. See Figure 4 and Figure 2a(i).	**Intra-Rail Strategy (IRS):** A strategy that reorders the sequence of logical nodes in the ring so that all network hops occur between NICs connected to the same rail switch. See Figure 5b and Figure 2a(ii).	**Rail:** A dedicated communication pathway formed by a set of network cards (NICs) across all physical nodes that share the same index (e.g., all NIC0s). All NICs within a same rail are connected to the same rail switch. See Figure 1.

TABLE II
NOTATIONS USED IN BANDWIDTH ANALYSIS

Notation	Explanation
N	Number of GPUs within each physical node, typically 8.
B_l	Unidirectional bandwidth of the network domain link, e.g., 50GBps for Mellanox CX-7.
B_h	Unidirectional bandwidth of the intra-node domain link, e.g., 150GBps for NVIDIA V100.
b_l^r, b_h^r	Portion of the unidirectional bandwidth of the network domain link (i.e., B_l) or the intra-node domain link (i.e., B_h) allocated to ring r, where $b_l^r \leq B_l$ and $b_h^r \leq B_h$, as a single link may be shared by multiple rings.
b^r	The theoretical bandwidth of ring r, which is determined by the bottleneck bandwidth in the two domains, i.e., $b^r = \min(b_l^r, b_h^r)$.
K	The number of logical nodes partitioned within a physical node by the Load Distributing Strategy, where $K \leq \left\lfloor \frac{N}{2} \right\rfloor$ and K is a positive integer.
n_k	The number of GPUs in the k-th logical node, where $0 < k \leq K$ and k is an integer.
K_t	The *theoretically optimal* number of logical nodes in the Load Distributing Strategy, where the bandwidth allocated to a ring in both domains is equal, which may not be an integer.

Intra-Rail Strategy (IRS) for the second goal, both of which together form RAIL's inter-GPU links rearrangement strategy.

A. Bandwidth Bottleneck Analysis

We first introduce the current ring construction method used by the NCCL ring algorithm, and analyze under which conditions it may lead to bottlenecks. The used notations are illustrated in Table II.

In NCCL, GPUs in the intra-node domain are naively connected based on device IDs or theoretical bandwidth. Figure 4(a) provides an example of a possible ring, where the cross-rail communication between GPU01 and GPU08, as mentioned earlier, can degrade performance. If PXN is used to route GPU01's traffic via the intra-node link to GPU00, which is on the same rail as GPU08, the ring will require an additional 1/7 of the intra-node link, increasing the load by 14%.

Next, we analyze a more general scenario, where the number of GPUs within each node is denoted as N. Acting as intermediaries and cluster centers, each GPU is linked to both the network domain and the intra-node domain. The unidirectional bandwidth of the network domain link (typically an RNIC) is denoted by B_l, while the unidirectional bandwidth of the intra-node link (such as NVLink or other high-bandwidth P2P interconnections) is denoted by B_h. As shown in Figure 4, each ring uses ingress and egress bandwidth of the network domain link once (for example, for the left node in Figure 4(b), this applies to the ingress and egress bandwidth of GPU00). Therefore, at least N different rings are needed, each starting from a distinct NIC, to fully utilize all network domain links in a node with N NICs.

From a bandwidth perspective, for a single node, the placement of GPUs other than those designated as the node's entry and exit does not affect the utilization of the intra-node link bandwidth (which involves one ingress and one egress per GPU), nor does it impact the network domain link (since these GPUs do not use the network domain link in the current ring). This property allows our subsequent analysis and design to remain orthogonal to existing collective communication algorithms, as we are only concerned with the paths of the GPUs serving as the node (or logical node, as discussed later) entry and exit units. The ordering of the remaining GPUs can be determined by other collective communication algorithms.

For any ring r, we denote the unidirectional bandwidth allocated to it in the network domain link as b_l^r, and the unidirectional bandwidth allocated in the intra-node link as b_h^r. As mentioned earlier, N rings are constructed, each originating from a different NIC. Thus, each ring has exclusive use of the network domain bandwidth, giving $b_l^r = B_l$ in theory. In the fully connected intra-node domain, as shown in Figure 1, each ring utilizes every intra-node link once (for both ingress and egress). With N rings coexisting, the intra-node link bandwidth B_h is shared among the N rings, giving $b_h^r = \frac{B_h}{N}$ in theory.

The theoretical bandwidth of ring r, i.e. b^r, is constrained by the bottleneck link, given by

$$b^r = \min(b_l^r, b_h^r) = \min\left(B_l, \frac{B_h}{N}\right) \tag{1}$$

When B_l exceeds $\frac{B_h}{N}$, the intra-node domain becomes the bottleneck. The PXN, which increases the load in the intra-node domain, will further exacerbate the bottleneck bandwidth. This issue is not uncommon and is exacerbated by

the load and cost imbalances between the two domains, as discussed in section III. In practice, even without considering the congestion observed in NVSwitch under multiple connections [11], NVLink bandwidth only reaches 80% of its nominal value [10], [11], [12]. For the configurations in [10], the effective intra-node interconnection bandwidth of H800 is 160GBps, while Mellanox's ConnectX-7 NIC provides 400Gbps (50 GBps). In a machine with 8 GPUs, the ratio of $\frac{160}{8} < 50$ indicates that the intra-node domain becomes the bottleneck, which is also verified in our evaluation (see Figure 2b).

B. Load Distributing Strategy

Based on the bandwidth bottleneck analysis, RAIL addresses the intra-node domain bottleneck through its Load Distributing Strategy.

Alleviating the load on the intra-node domain by avoiding the use of PXN. We re-examine the construction of the inter-GPU ring, with the first step being to avoid using PXN to reduce the load on the intra-node domain.

As shown in Figure 4, not using PXN allows each ring to reduce the bidirectional bandwidth usage of the intra-node link by one time per ring. It is evident that when the number of rings is equal to the number of GPUs/NICs per node, the usage frequency of intra-node and network links per ring corresponds to the number of rings dividing the bandwidth of a single intra-node and network link. Therefore, not using PXN reduces the number of rings that evenly distribute the intra-node link bandwidth by one. This means that the theoretical bandwidth allocated to a single ring on the intra-node link, b_h^r, is improved from $\frac{B_h}{N}$ to $\frac{B_h}{N-1}$, yielding an improvement of $\frac{N}{N-1}$. The comparison of evaluation results through Figure 2a(ii) and Figure 2b(i) demonstrates the throughput improvement of chunks in the intra-node domain without using PXN.

Trading off load across the two domains by partitioning physical nodes into multiple logical nodes to match with their transmission capacities.

Our load distributing strategy is based on the following observation: When PXN is not used, *in a node consisting of N GPUs and N RNICs, a ring always utilizes the network domain's ingress and egress unidirectional bandwidth exactly once, and it uses the intra-node link bidirectional bandwidth $N-1$ times.* A diagram for $N = 8$ is shown in Figure 4(a).

Thus, our approach is *to partition the physical nodes into K logical nodes*. Within each logical node, the GPUs are connected sequentially via the intra-node domain, while logical nodes are connected through the network domain. For $N = 8$ and $K = 2$, one possible way to partition the logical nodes is shown in Figure 5a, where both logical nodes contain 4 GPUs each. Let the number of GPUs in the k-th logical node be denoted as n_k, where $0 < k \leq K$ and k is an integer. Since each logical node must contain at least two GPUs, we have:

$$\sum_{k=1}^{K} n_k = N \quad \text{and} \quad n_k \geq 2 \quad \text{with} \quad K \leq \left\lfloor \frac{N}{2} \right\rfloor \tag{2}$$

At this point, we re-examine link usage after partitioning logical nodes using LDS. Taking Figure 5a as an example, based on our observation of link usage in the two domains for a node with N GPUs, a single ring utilizes the intra-node links $\sum_{k=1}^{K}(n_k - 1) = N - K$ times. This is equivalent to the bandwidth of a single intra-node link being evenly divided among $N - K$ rings. Consequently, the theoretical bandwidth per ring on the intra-node link is $b_h^r = \frac{B_h}{N-K}$. For the network domain, since the node is partitioned into K logical nodes, the usage of network links per ring increases by K times. Thus, the theoretical bandwidth per ring on the network link becomes $b_l^r = \frac{B_l}{K}$.

The theoretical bandwidth of ring r, i.e. b^r, is constrained by the bottleneck link, given by

$$b^r = \min(b_l^r, b_h^r) = \min\left(\frac{B_l}{K}, \frac{B_h}{N-K}\right) \tag{3}$$

To maximize b^r, the problem becomes one of finding the maximum value of the function $b^r(K)$, as illustrated in Figure 6. It is clear that, when $K \in \left(0, \left\lfloor \frac{N}{2} \right\rfloor\right]$, $\frac{B_l}{K}$ decreases as K increases, while $\frac{B_h}{N-K}$ increases as K increases. We define K_t as the value of K when the two equations are equal. At this point, we can derive:

$$b^r(K) = \begin{cases} \dfrac{B_h}{N-K} & \text{if } K \leq K_t \\ \dfrac{B_l}{K} & \text{if } K > K_t \end{cases}, \quad K_t = \frac{N \cdot B_l}{B_h + B_l} \tag{4}$$

Since K_t may not be an integer, RAIL selects the integer K from the two integers surrounding K_t that results in a

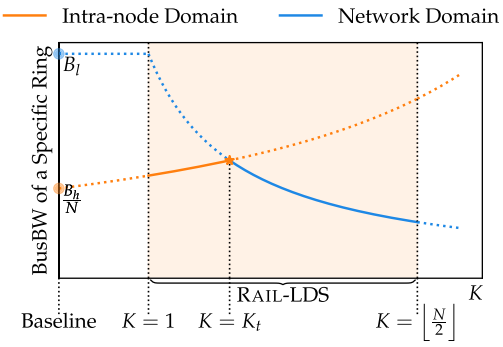


Fig. 6. The illustration of $b^r(K)$, showing the theoretical bandwidth per ring as a function of the number of partitioned logical nodes K when the link bandwidth of both domains are fixed. The range of K is $(0, \left\lfloor \frac{N}{2} \right\rfloor]$, where $K = 0$ represents the NCCL with PXN.

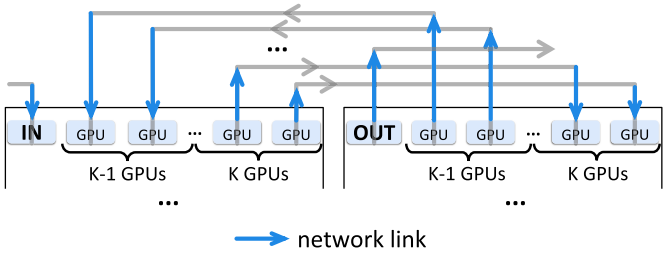
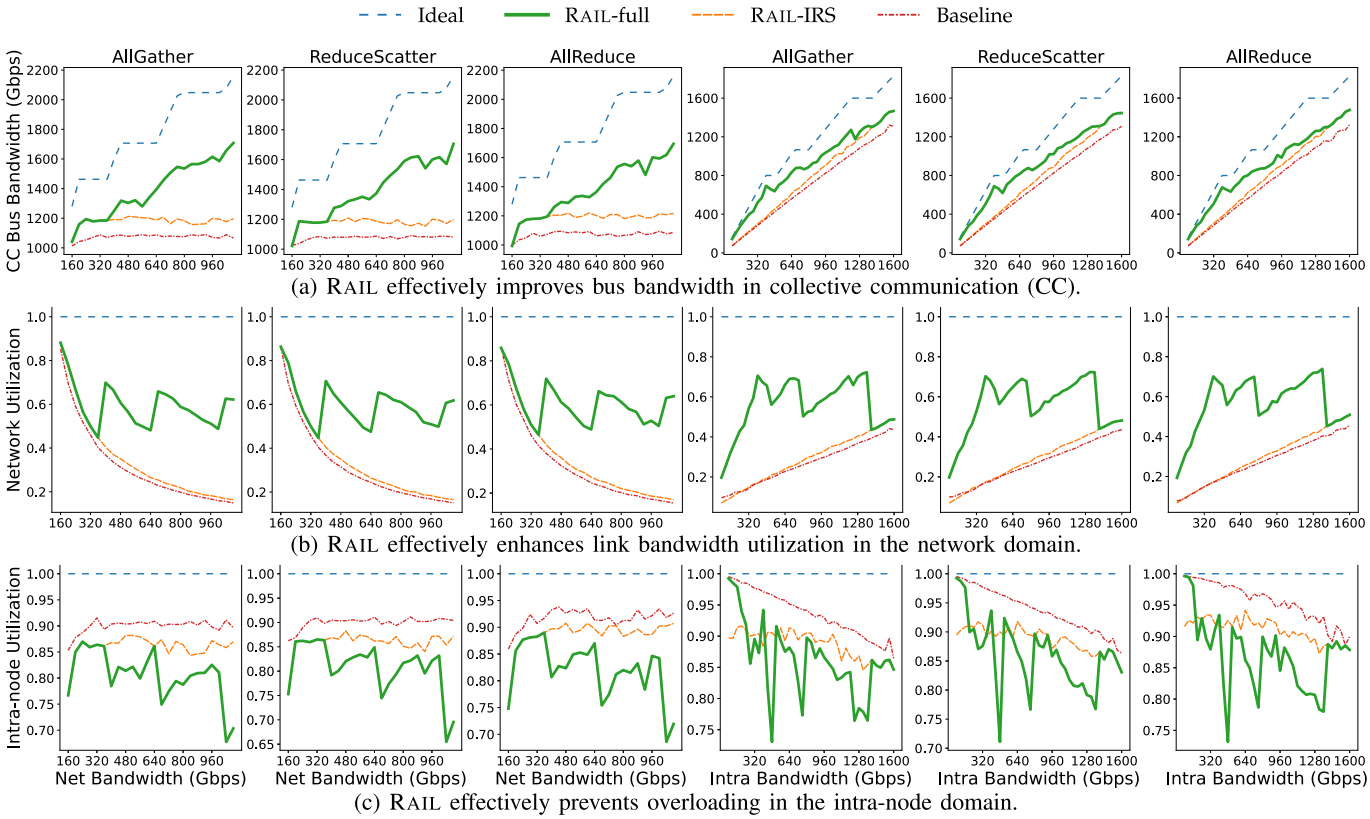


Fig. 7. After dividing each intra-node domain into K logical nodes, RAIL-IRS rearranges the network domain links of the $2K$ ingress and egress GPUs from these logical nodes to eliminate cross-rail traffic.

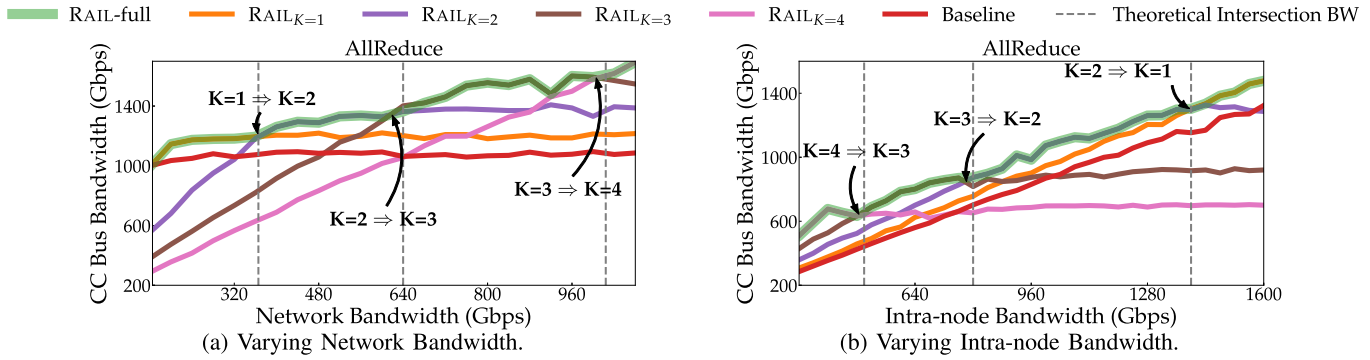


(a) RAIL effectively improves bus bandwidth in collective communication (CC).

(b) RAIL effectively enhances link bandwidth utilization in the network domain.

(c) RAIL effectively prevents overloading in the intra-node domain.

Fig. 8. The bus bandwidth and link bandwidth utilization of Baseline, RAIL-IRS, RAIL-full, and Ideal under different domain bandwidths when using ALLGATHER, REDUCESCATTER, and ALLREDUCE as the workloads, corresponding to subsection VI-A, answering Q1. The number of GPUs in the cluster is set to 128, with the remaining setup following subsection V-B. In the left three subfigures, the horizontal axis sets the intra-node domain link bandwidth to 1280 GBps, with RNIC bandwidth varying from 160 to 1080 Gbps. In the right three subfigures, the RNIC bandwidth is fixed at 400 Gbps, and the intra-node domain link bandwidth varies from 80 to 1600 Gbps, both with a 40 Gbps step. The collective communication data size is set to 256MB.



(a) Varying Network Bandwidth.

(b) Varying Intra-node Bandwidth.

Fig. 9. Two subfigures in Figure 8a for ALLREDUCE, illustrating how bus bandwidth changes with domain bandwidth for different K values in LDS plus RAIL-IRS, answering Q3. The black arrows indicate shifts in the optimal K value across tested logical node counts, showing the corresponding domain bandwidth and bus bandwidth for collective communication. The gray dashed lines denote intersection points calculated from Equation 3. RAIL-full consistently selects the optimal K across various cluster configurations, closely aligning with the theoretically optimal K values.

larger $b^r(K)$. Thus, RAIL calculates the optimal K under different bandwidths of the two domains according to the above LDS, and partitions the logical nodes accordingly. Precisely partitioning the logical nodes is crucial, as improper partitioning may result in suboptimal optimization or even significant performance degradation (see Figure 9).

C. Intra-Rail Strategy

After partitioning the logical nodes, we obtained the optimal load distribution scheme and matched the load transmission capability theoretically. Additionally, we need to rearrange links in the network domain, such as the link between GPU01 and GPU02, and GPU03 and GPU08 in Figure 5a, to avoid cross-rail traffic. Therefore, RAIL employs the Intra-rail strategy to minimize cross-rail traffic between logical nodes.

Based on the characteristics of the rail-optimized topology, we label all nodes as $D_0, D_1, \ldots, D_i, \ldots, D_{M-1}$, and label the rails shown in Figure 1 as $R_0, R_1, \ldots, R_j, \ldots, R_{N-1}$. All GPUs are numbered according to their node i and rail j as $U_{(i,j)}$. For any given ring, to ensure that all network domain

traffic remains intra-rail, we have **Constraint 1**: **For any two adjacent GPUs $U_{(d_1,r_1)}$ and $U_{(d_2,r_2)}$ in the ring, they should either be within the same node (i.e., $d_1 = d_2$) or on the same rail (i.e., $r_1 = r_2$).**

We define the GPUs that transition from within a logical node to the network domain and from the network domain back into the logical node as the exit unit and entry unit of the logical node, respectively. Clearly, since all logical nodes must communicate with other logical nodes through the network domain, we can derive **Constraint 2**: **The number of exit and entry units in a physical node (i.e., an intra-node domain) within a ring is equal to the number of logical nodes K.**

Based on the two constraints, we propose the intra-rail strategy. As shown in Figure 7, and following Constraint 2, IRS first selects K GPUs as entry units and K GPUs as exit units from the left physical node, denoted as D_a. This selection occurs before the logical node partitioning using LDS. Next, according to Constraint 1, the logical nodes in D_a must communicate with the logical nodes in the right physical node, D_b, via the network domain while ensuring intra-rail traffic. To achieve this, IRS selects $2K$ GPUs from D_b such that they align with the $2K$ GPUs in D_a, ensuring $r_1 = r_2$, with entry and exit units having opposite roles.

Subsequently, the remaining GPUs are randomly assigned to K logical nodes using LDS, or other collective communication path scheduling algorithms are employed to determine the inter-GPU paths within logical nodes. Finally, the logical nodes are connected according to the preselected entry and exit units and the physical nodes are merged, and a fully intra-rail ring is formed. An example of a ring formed with IRS and LDS applied simultaneously on two physical nodes with $N = 8$ and $K = 2$ is shown in Figure 5b. Compared to the ring generated with only LDS in Figure 5a, RAIL-full ensures all network domain traffic remains intra-rail. Bringing all of the above together, RAIL rearranges the inter-GPU paths according to the pseudo code shown in Algorithm 1.

Algorithm 1 The Path-Finding Algorithm in RAIL

Input: M: The No. of intra-node domains
N: The No. of GPUs in a intra-node domain
K: The No. of logical nodes computed by RAIL-LDS
Output: R: A set of rings, each expressed as a list of GPUs

Procedure SearchRingChannels(M, N, K)
  $R \leftarrow \emptyset$
  start $\leftarrow [0]$
  // Calculate the start index of logical nodes in the ring
  for $i \leftarrow 1$ **to** $2K - 1$ **do**
    $n_i \leftarrow \lfloor \frac{N}{K} \rfloor$ // Size of logical node i
    if $i = 2K - 1$ **then**
      $n_i \leftarrow n_i + N \% K$
    start.append(*start*$[i-1] + n_i$)
  for $i \leftarrow 0$ **to** $N - 1$ **do** // Find N rings
    $r \leftarrow$ List of size $M \times N$
    // Calculate the ordering of GPUs for the first pair of intra-node domains
    firstDomainUnits $\leftarrow \{0, 1, \ldots, N-1\}$
    for $j \leftarrow 0$ **to** $2K - 1$ **do**
      // Add the entry and exit units
      $r[(start[j] + 2N - 1)\ \%\ 2N] \leftarrow (i + j)\ \%\ N + (j\%2) * N$
      $r[start[j]] \leftarrow (i + j)\ \%\ N + (j\%2) * N$
      firstDomainUnits.remove($(i + j)\ \%\ N$)
    // Add the intermediate units in arbitrary order
    for $j \leftarrow 0$ **to** $K - 1$ **do**
      for $l \leftarrow 1$ **to** $start[2j+1] - start[2j] - 2$ **do**
        $u \leftarrow$ *firstDomainUnits*.pop()
        $r[start[2j] + l] \leftarrow u$
        $r[start[2j+1] + l] \leftarrow u + N$
    // Reuse the GPU ordering for the remaining intra-node domains with a shift
    for $j \leftarrow 2N$ **to** $MN - 1$ **do**
      $r[j] \leftarrow r[j - 2N] + 2N$
    R.append(r)
  return R

V. METHODOLOGY

This section outlines the methodology employed in this study, focusing on the simulation platform and the system modeling and configuration.

A. Simulation Platform and System Modeling

We extended ns-3 [21], a discrete-event simulator for network systems, to simulate and model the collective communication Ring algorithm used in distributed training.

We modeled the baseline algorithm based on version 2.22.3 of the NCCL [17] and implemented RAIL's rearrangement strategy of ring algorithm in ns-3, supporting various communication primitives. Benefiting from the regular architecture of the training clusters and the predictable network traffic patterns of distributed training, a comprehensive network system simulator like ns-3 can accurately model and match the collective communication in real-world systems. While ensuring high-fidelity modeling and simulation, we also integrated UNISON [43] into our simulation platform to efficiently verify the performance of RAIL in larger-scale training clusters.

The simulation-based modeling and experimentation allowed us to gain deeper insights into the detailed communication behavior between GPUs during collective communication, helping to identify performance bottlenecks and validate RAIL's algorithmic design. Additionally, due to the differences in development speed and technological barriers between the two domains, using simulation as our methodology enables us to show the necessity of RAIL in future systems, where the trends in section III are expected to become more pronounced.

B. Workload and Configuration

We set up the topology shown in Figure 1, where the rail-optimized network domain and the switch-based fully connected intra-node domain are connected in a GPU-centric manner. RAIL also applies to intra-node domains using other interconnects, which will be discussed in section VII. Given that the maximum number of GPUs in the same rail is limited by the port count of the leaf switch in real clusters, we

TABLE III
COMPARISON OF LINKS ARRANGEMENT METHOD

Method	Comment
Baseline	A ring algorithm implementation modeled based on NCCL source code, utilizing PXN technology to avoid cross-rail traffic.
RAIL-IRS	Applying only RAIL's Intra-Rail Strategy (IRS) to ensure intra-rail traffic without using PXN.
RAIL-full	Combines both the Load Distributing Strategy (DSS) and Intra-Rail Strategy (IRS), representing the full design of RAIL as described in section IV.
Ideal	All inter-GPU links in collective communication maintain complete synchronization for chunk transmission start and end, with no idle links. The communication time for a single primitive is minimized to $chunkSize/linkBw \times n$, achieving 100% BW utilization.

divided all intra-node domains into groups. Intra-rail traffic within a group remains two hops away via the leaf switch (i.e. the corresponding rail switch), while intra-rail traffic between groups requires forwarding via the spine switch. The uplink oversubscription ratio at the leaf switch layer is 2:1. The propagation delays for the links in the network domain and intra-node domain are set to 1.7 microseconds and 0.7 microseconds, respectively.

We use REDUCESCATTER, ALLGATHER, and ALLREDUCE based on the Ring algorithm as our workload. According to subsection II-B, these primitives cover the majority of training workloads, reflecting the end-to-end communication overhead. Since communication time far exceeds computation time, we estimate the computation time for ReduceScatter based on the size of communication data and the FLOPS of the corresponding GPU. ALLREDUCE is simply the execution of REDUCESCATTER followed by ALLGATHER.

We evaluate collective communication performance using the bus bandwidth (`busbw`) from the `nccl-tests` [8], which provides a unified bandwidth metric for different data sizes and primitives. If the communication size for a collective operation is s, the completion time is t, and the number of participating nodes is n, then the bus bandwidth (b_{bus}) for REDUCESCATTER and ALLGATHER is given by the formula: $b_{bus} = \frac{s}{t} \cdot \frac{n-1}{n}$. For ALLREDUCE, the bus bandwidth is given by the formula: $b_{bus} = \frac{s}{t} \cdot \frac{2(n-1)}{n}$.

Table III presents the four inter-GPU links arrangement methods, using NCCL with PXN as the baseline and gradually incorporating RAIL's two strategies, ultimately comparing with the ideal method. The corresponding evaluation can be found in subsection VI-A.

In GPU-centric clusters, the number of rings constructed for a distributed training task is a multiple of the number of GPUs or NICs within a node, denoted as N. When the number of rings exceeds N, the additional rings are directly duplicated from the existing ones. In our subsequent evaluations, the number of rings is set to 8. RAIL is also applicable to other intra-node configurations, as discussed in section VII.

VI. EVALUATION

In this section, we will comprehensively evaluate RAIL, focusing on the following key questions:

- **Q1: Can** RAIL's two strategies improve bus bandwidth and link bandwidth utilization? How does it compare to the baseline and ideal cases? (subsection VI-A)
- **Q2: How** does RAIL work in a single collective communication operation compared to cross-rail and the baseline? How are the improvements from the previous point achieved? (subsection VI-B)
- **Q3: When** does RAIL perform effectively under different communication data sizes, varying network and intra-node link bandwidth, different numbers of logical nodes K? (subsection VI-C)

A. Rail Improves Bus Bandwidth and Link Bandwidth Utilization

Figure 8 shows that in current cluster configurations (network domain bandwidth ranging from 200 Gbps for Mellanox CX-5/6 to 800 Gbps for CX-8, and intra-node domain bandwidth from 256 Gbps for PCIe-Gen4 $\times$ 16 to 1600 Gbps for Tesla H800 [32]), RAIL-full improves bus bandwidth by 8.59% to 96.4% compared to the baseline (the latter with both domains at 400 Gbps, which can occur when high interconnect numbers lead to such intra-node domain bandwidth performance [11]), averaging a 30.7% improvement. The network domain bandwidth utilization increases by 8.73% to 315.5%, while the absolute utilization stays below 70%, avoiding congestion. The intra-node domain utilization is reduced from over 90% to a more reasonable 80%.

Similarly, RAIL-IRS, using only the Intra-Rail Strategy of RAIL, shows notable improvements, independent of domain bandwidth ratios, making it plug-and-play across clusters. Compared to the baseline, RAIL-IRS boosts bus bandwidth by 3.77% to 14.7%, with an average improvement of 9.75%, and increases network domain utilization by an average of 8.64%.

When the network-to-intra-node bandwidth ratio is low, RAIL uses only IRS to avoid overloading the network domain, as adding LDS could worsen performance.

For Ideal, Table III outlines its theoretical performance, which requires discussion across different underlying topologies. In rail-optimized clusters, Ideal assumes that link bandwidth utilization in both domains is maximized while ensuring that no link becomes a bottleneck. We calculate Ideal bus bandwidth by multiplying the maximum bus bandwidth per ring (derived from Equation 4) by the number of rings, assuming no congestion and thus 100% link utilization in both domains.

Due to real-cluster factors like asynchrony of chunk transmission and the physical constraint $0 < K \leq N/2$ where K is an integer, RAIL-full shows a gap compared to Ideal, with reduced link utilization over the duration of whole collective communication compared to Figure 2c.

B. Rail Ensures Data Chunks in the Ring Follow the Optimal Path

RAIL-IRS. Figure 2a(i) shows the transmission rate over time for a chunk traveling through the ring when a cross-rail link is present, as shown in Figure 4(a), without taking the cluster architecture features into account. Figure 2a(ii) shows

the transmission rate of a chunk when it follows a path guided by RAIL's IRS, considering the topological characteristics of the rails. Comparing (i) and (ii), cross-rail communication severely degrades the transmission rate in the network domain, resulting in an approximately 75.5% decrease. In the intra-node domain, the load generated by RAIL-IRS and NCCL without PXN is similar, with the average transmission rate difference within 8%. By avoiding cross-rail traffic in the network domain, RAIL-IRS reduces the completion time of a single ALLGATHER by 22.6%. In larger-scale training clusters, cross-rail communication increases the likelihood of hash collisions, and the uplink oversubscription further hampers the performance of cross-rail traffic.

RAIL-full bandwidth. Figure 2b(i) and Figure 2b(ii) illustrate the "journey" of a chunk in the baseline and RAIL-full configurations, respectively. Under the guidance of RAIL-full, the chunk more frequently utilizes the less congested network domain rather than the crowded intra-node domain. By following this strategy, part of the load in the intra-node domain is shifted to the network domain, leading to more balanced performance across both domains and allowing the chunk to complete its journey faster. Compared to the baseline, RAIL-full reduces the completion time of ALLGATHER by 16.2%, with a corresponding improvement in bus bandwidth.

RAIL-full link-utilization. Figure 2c(i) and Figure 2c(ii) show the average link bandwidth utilization of all network domain RNICs and the intra-node domain links between high-bandwidth switch and GPUs during an ALLGATHER operation for the baseline and RAIL-full, respectively, with a sampling granularity of 5 microseconds. In the baseline, the intra-node domain consistently maintains high utilization, while the network domain utilization fluctuates around 50%, resulting in significant idle time in the network domain. In contrast, RAIL-full, through the Load Distributing Strategy (LDS), avoids excess load in the intra-node domain while maximizing link bandwidth utilization in both domains.

In the cluster setup shown in Figure 2, RAIL sets the number of logical nodes K to 2 based on LDS. By rearranging the links, RAIL theoretically reduces the load on the intra-node domain by $K/N = 25\%$ compared to the baseline, while increasing the load on the network domain by a factor of $K = 2$. The micro-benchmark results demonstrate that RAIL closely aligns with these theoretical predictions, achieving corresponding improvements in bus bandwidth, primitive completion time, and link bandwidth utilization.

C. Rail Excels in Various Clusters

K-sensitivity. The number of logical nodes K determined by LDS affects the performance of RAIL, as shown in Figure 9. The optimal K value varies across different cluster configurations, and the bus bandwidth of ALLREDUCE shows significant variation with different K values. When the intra-node domain bandwidth is 1280 Gbps and the network domain bandwidth is below 640 Gbps, or when the network domain bandwidth is 400 Gbps while the intra-node domain bandwidth exceeds 738 Gbps, aggressive K values may result in lower bus bandwidth than the baseline. Therefore, accurately choosing the number of logical nodes based on domain bandwidth is crucial. The positions of the black arrows and gray dashed lines in the figure indicate that RAIL's bandwidth when switching K values closely aligns with the theoretical switching points, with a deviation of less than 10 Gbps. RAIL selects the optimal K value, which closely aligns with the theoretical analysis, achieving precise trade-offs between domains and maximizing performance gains.

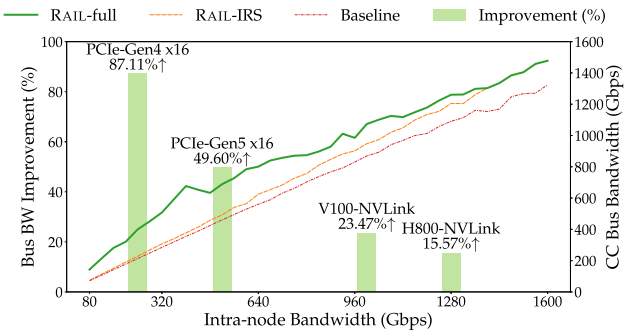


Fig. 10. The improvement in bus bandwidth of RAIL-full compared to the baseline when using configurations from different vendors in the intra-node domain, with the ALLREDUCE primitive and CX-7 as the network domain.

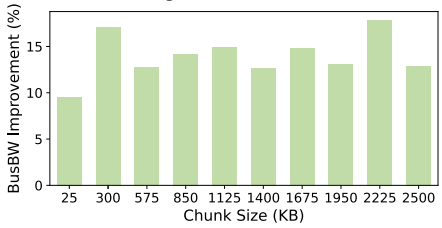


Fig. 11. The improvement in bus bandwidth of RAIL-full compared to the baseline when using different chunk sizes with the ALLREDUCE communication primitive.

BW-sensitivity. As discussed in subsection III-A and subsection III-A, the imbalances in load between the two domains shape the trade-off space between them. AI startups further expand this trade-off space with strategies that reduce costs and improve efficiency when building clusters. In this context, we propose RAIL to strike a balance between the two domains, and achieve improvements in bus bandwidth across configurations from different vendors in the intra-node domain, as shown in Figure 10. As the trade-off space between the two domains narrows, the performance gains brought by RAIL-IRS gradually surpass those of RAIL-LDS, delivering stable intra-rail benefits for better-configured intra-node domains.

Size-sensitivity. A chunk is the data unit of collective communication, and the total data volume in collective communication is equal to the chunk size multiplied by the number of chunks (typically equal to the number of GPUs). As shown in Figure 11, with all other configurations identical to those in Figure 8, RAIL-full achieves over a 10% increase in bus bandwidth for chunk sizes ranging from 300 KB to 2.5 MB, which aligns with the bandwidth optimal characteristics of the ring algorithm.

VII. DISCUSSIONS AND LIMITATIONS

P2P-based intra-node interconnects. To fully utilize NIC bandwidth in training tasks while ensuring concurrent communication of the same data, the number of *channels*

(rings in NCCL) is typically a multiple of the number of GPUs/NICs within a node. This is also supported by NCCL's default `NCCL_MIN_NCHANNELS` parameter (16 for versions 2.3/2.4, 32 for 2.5 and later). For P2P-based topologies of intra-node, like pure NVLink and InfinityFabric, when the number of rings is sufficient to ensure that the load on each P2P link is balanced, the strategies employed by RAIL remain applicable.

Mechanisms of Intra-Node Bandwidth Degradation. The performance degradation in the intra-node domain, despite NVSwitch's high theoretical capacity, is an emergent result of micro-architectural contention under the adverse traffic patterns created by default collective algorithms. The distinction lies between the switch's theoretical aggregate bandwidth and the *effective throughput*, which is diminished by three primary mechanisms:

- Head-of-Line (HOL) Blocking, where a single stalled data flow blocks its entire input queue, starving other flows destined for available output ports; [44]
- Incast and Output Port Contention, where multiple simultaneous flows overwhelm a single output buffer, causing it to fill rapidly and trigger flow control; [45]
- Propagation of Backpressure, where credit-based flow control signals from congested output ports travel backward, throttling transmission rates directly at the source GPUs. [46]

In essence, topology-agnostic algorithms create a hostile traffic pattern that triggers these bottlenecks. RAIL is designed to circumvent these issues by creating a contention-aware communication pattern that respects the distinct capabilities of the heterogeneous domains.

Deployment Considerations and Practicality. RAIL is designed for straightforward deployment as a targeted enhancement to existing collective communication libraries like NCCL. A practical implementation requires only replacing the default graph search logic (e.g., in NCCL's 'src/graph/search.cc') with our lightweight, constant-time path-finding algorithm (algorithm 1).

Our approach seamlessly leverages NCCL's robust, built-in topology discovery (e.g., 'ncclTopoSearchInit'), which already gathers all necessary hardware interconnect config. The integrated RAIL logic is then automatically invoked during the channel construction phase. This makes RAIL a surgical, high-impact enhancement that utilizes existing infrastructure rather than a new, complex system to be managed, ensuring its path to real-world deployment is clear and practical.

Limitations and Future Work. We acknowledge two primary limitations in our work. First, regarding the algorithmic foundation, RAIL focuses on optimizing ring-based collectives. While robust for high throughput, rings entail linear step complexity, making them inherently suboptimal for latency-sensitive or small-message scenarios where tree-based algorithms often dominate. Second, RAIL's Intra-Rail Strategy relies on the structural properties of rail-optimized topologies. Although this implies a specific dependency, such architectures have established themselves as the prevailing consensus for hyperscale AI training, evidenced by widespread adoption by industry leaders [20], [34], [35], [36]. Furthermore, the core principle of RAIL's Load Distributing Strategy regarding inter-domain traffic balancing remains theoretically generalizable to other network designs. In terms of future directions, while our current evaluation utilizes high-fidelity simulations due to the prohibitive resource requirements of accessing full-scale spine-leaf clusters, we plan to conduct deployments on production GPU clusters to validate RAIL's performance in real-world environments. Additionally, we intend to extend our bottleneck analysis methodology to emerging network architectures, such as those discussed in [47], to investigate how RAIL's scheduling principles can be adapted to optimize efficiency in next-generation topologies.

VIII. Related Works

Current collective communication algorithms can be broadly categorized into three classes based on their topology awareness and routing mechanisms: search-based synthesis, tree-based hierarchical methods, and ring-based approaches.

Search-based and synthesis algorithms, represented by works such as SCCL [48], TACCL [11], TECCL [49], SyCCL [50], and ForestColl [51], formulate collective communication as a graph optimization problem. By leveraging global topology information, these methods utilize advanced mathematical techniques—ranging from integer linear programming to spanning tree packing—to derive bandwidth-optimal schedules. This class of algorithms is particularly effective in arbitrary or highly heterogeneous network fabrics, as it enables complex multi-path routing that exploits all available physical links, thereby theoretically overcoming the bottleneck limitations inherent in rigid, static topology assumptions.

Tree-based and hierarchical algorithms have evolved from latency-focused binary trees to throughput-oriented designs. Recent advancements, such as the Pipelined Aggregation Tree (PAT) algorithm introduced in recent NCCL versions [52] and hierarchical approaches like BlueConnect [53], aim to combine the low latency of trees with the bandwidth saturation of pipelining. These methods are increasingly adopted to mitigate the latency overhead inherent in linear ring traversals for latency-sensitive operations.

Despite the emergence of these advanced topologies, ring-based algorithms remain a foundational primitive in collective communication. As highlighted in Meta's NCCLX [54], ring algorithms inherently regulate concurrent network traffic by limiting communication strictly to immediate neighbors. This characteristic is crucial for preventing congestion in over-subscribed networks with limited bisection bandwidth, where complex multi-path routing might exacerbate switch buffer contention. While standard rings are performance-bounded by the bottleneck link in heterogeneous environments, our proposed RAIL demonstrates—through theoretical analysis and simulation—that ring-based schedules can be effectively optimized for heterogeneity. RAIL thus preserves the congestion-control benefits and structural simplicity of rings while mitigating their throughput limitations.

IX. Conclusion

In this paper, we re-examine the imbalanced load between the network domain and the intra-node domain caused by

large-scale model training workloads in GPU-centric clusters. To match load and transmission capacity between the two domains, eliminate cross-rail traffic and guarantee the scalability of strategies in heterogeneous clusters, we propose RAIL.

RAIL trades off the utilization of the two domains by partitioning logical nodes and rearranging inter-GPU links, while leveraging topology characteristics to select optimal paths between the logical nodes. RAIL effectively mitigates congestion in both domains and achieves an average 30.7% improvement in collective communication bus bandwidth compared to the widely-used cutting-edge NCCL solution.

ACKNOWLEDGMENT

The authors would like to thank editors and anonymous reviewers for their valuable comments.

REFERENCES

[1] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, no. 120, pp. 1–39, 2021.

[2] A. Sapio et al., "Scaling distributed machine learning with in-network aggregation," in *Proc. 18th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2021, pp. 785–808.

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

[4] W. Deng, J. Pan, T. Zhou, D. Kong, A. Flores, and G. Lin, "DeepLight: Deep lightweight feature interactions for accelerating CTR predictions in ad serving," in *Proc. 14th ACM Int. Conf. Web Search Data Mining*, Mar. 2021, pp. 922–930.

[5] NVIDIA.(2023). *NVlink High-Speed GPU Interconnect*. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/

[6] S. Naffziger, K. Lepak, M. Paraschou, and M. Subramony, "2.2 AMD chiplet architecture for high-performance server and desktop products," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2020, pp. 44–45.

[7] W. Wang, M. Ghobadi, K. Shakeri, Y. Zhang, and N. Hasani, "Rail-only: A low-cost high-performance network for training LLMs with trillion parameters," 2023, *arXiv:2307.12169*.

[8] *Performance Reported By NCCL Tests*. Accessed: 2025. [Online]. Available: https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md#bus-bandwidthmymargin{

[9] I. Sutskever. (2024). *Sequence To Sequence Learning With Neural Networks: What a Decade*. [Online]. Available: https://www.youtube.com/watch?v=1yvBqasHLZs

[10] A. Liu et al., "DeepSeek-V3 technical report," 2024, *arXiv:2412.19437*.

[11] A. Shah et al., "TACCL: Guiding collective algorithm synthesis using communication sketches," in *Proc. 20th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2023, pp. 593–612.

[12] A. Li et al., "Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 1, pp. 94–110, Jan. 2020.

[13] Y. Li et al., "HPCC: High precision congestion control," in *Proc. ACM Special Interest Group Data Commun.*, Aug. 2019, pp. 44–58.

[14] G. Kumar et al., "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proc. Annu. Conf. ACM Special Interest Group Data Commun. Appl., Technol., architectures, protocols Comput. Commun.*, Jul. 2020, pp. 514–528.

[15] Y. Zhang, Q. Meng, C. Hu, and F. Ren, "Revisiting congestion control for lossless Ethernet," in *Proc. 21st USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2024, pp. 131–148.

[16] S. Rashidi, W. Won, S. Srinivasan, S. Sridharan, and T. Krishna, "Themis: A network bandwidth-aware collective scheduling policy for distributed training of DL models," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 581–596.

[17] NVIDIA.(2021). *NCCL*. [Online]. Available: https://github.com/nvidia/nccl/

[18] H. Kim, J. Ryu, and J. Lee, "TCCL: Discovering better communication paths for PCIe GPU clusters," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, vol. 3, Apr. 2024, pp. 999–1015.

[19] J. Cao et al., "Crux: GPU-efficient communication scheduling for deep learning training," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 1–15.

[20] NVIDIA.(2022). *Doubling All2all Performance With Nvidia Collective Communication Library 2.12*. [Online]. Available: https://developer.nvidia.com/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12

[21] NSNAM.(2024). *NS-3*. [Online]. Available: https://www.nsnam.org/

[22] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds Mach.*, vol. 30, no. 4, pp. 681–694, 2020.

[23] A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 240, pp. 1–113, 2023.

[24] J. Kaplan et al., "Scaling laws for neural language models," 2020, *arXiv:2001.08361*.

[25] Z. Zhang, C. Chang, H. Lin, Y. Wang, R. Arora, and X. Jin, "Is network the bottleneck of distributed training?," in *Proc. Workshop Netw. Meets AI ML*, Aug. 2020, pp. 8–13.

[26] L. Dai, H. Qi, W. Chen, and X. Lu, "High-speed data communication with advanced networks in large language model training," *IEEE Micro*, vol. 44, no. 2, pp. 31–40, Mar. 2024.

[27] NVIDIA.(2018). *Nvswitch: Leveraging Nvlink to Maximum Effect*. [Online]. Available: https://developer.nvidia.com/blog/nvswitch-leveraging-nvlink-to-maximum-effect/

[28] H. Liao et al., "Ascend: A scalable and unified architecture for ubiquitous deep neural network computing: Industry track paper," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*, Feb. 2021, pp. 789–801.

[29] NVIDIA.(2023). *Nvidia L20 Enterprise 48Gb*. [Online]. Available: https://viperatech.com/shop/nvidia-l20-pcie/

[30] (2023). *Nvidia L40s*. [Online]. Available: https://www.nvidia.com/zh-tw/data-center/l40s/

[31] (2023). *Nvidia A800 40Gb Active Graphics Card*. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/a800/

[32] (2024). *Nvidia Tesla H800*. [Online]. Available: https://advdownload.advantech.com/productfile/PIS/SKY-TESL-H800-80P/file/SKY-TESL-H100-80P_H800-80P_DS(092623)20230926111757.pdf

[33] (2024). *Nvidia Tesla V100*. [Online]. Available: https://www.nvidia.com/en-gb/data-center/tesla-v100/

[34] Z. Jiang et al., "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *Proc. 21st USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2024, pp. 745–760.

[35] Q. Meng et al., "Astral: A datacenter infrastructure for large language model training at scale," in *Proc. ACM SIGCOMM Conf.*, Sep. 2025, pp. 609–625.

[36] K. Qian et al., "Alibaba HPN: A data center network for large language model training," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 691–706.

[37] D. Narayanan et al., "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2021, pp. 1–15.

[38] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2020, pp. 1–16.

[39] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *J. Parallel Distrib. Comput.*, vol. 69, no. 2, pp. 117–124, Feb. 2009.

[40] NVIDIA.(2024). *NVIDIA H100 Tensor Core GPU*. [Online]. Available: https://www.nvidia.com/en-us/data-center/h100/

[41] (2024). *Connectx-7 400G Adapters*. [Online]. Available: https://resources.nvidia.com/en-us-accelerated-networking-resource-library/connectx-7-datasheet

[42] Intel.(2020). *Production Brief for Intel® Ethernet Controller E810-CAM2/CAM1/XXVAM2*. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/615503

[43] S. Bai et al., "Unison: A parallel-efficient and user-transparent network simulation kernel," in *Proc. 19th Eur. Conf. Comput. Syst.*, Apr. 2024, pp. 115–131.

[44] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, Apr. 2002.

[45] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, Aug. 2009, pp. 73–82.

[46] S. Werner, J. Navaridas, and M. Luján, "A survey on optical network-on-chip architectures," *ACM Comput. Surveys*, vol. 50, no. 6, pp. 1–37, Nov. 2018.
[47] H. Liao et al., "UB-mesh: A hierarchically localized nD-fullmesh datacenter network architecture," 2025, *arXiv:2503.20377*.
[48] Z. Cai et al., "Synthesizing optimal collective algorithms," in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, Feb. 2021, pp. 62–75.
[49] X. Liu et al., "Rethinking machine learning collective communication as a multi-commodity flow problem," in *Proc. ACM SIGCOMM Conf.*, Aug. 2024, pp. 16–37.
[50] J. Cao et al., "SyCCL: Exploiting symmetry for efficient collective communication scheduling," in *Proc. ACM SIGCOMM Conf.*, Sep. 2025, pp. 645–662.
[51] L. Zhao et al., "ForestColl: Throughput-optimal collective communications on heterogeneous network fabrics," 2024, *arXiv:2402.06787*.
[52] S. Jeaugey, "PAT: A new algorithm for all-gather and reduce-scatter operations at scale," 2025, *arXiv:2506.20252*.
[53] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter, "BlueConnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy," *IBM J. Res. Develop.*, vol. 63, no. 6, pp. 1:1–1:11, Nov. 2019.
[54] M. Si et al., "Collective communication for 100k+ GPUs," 2025, *arXiv:2510.20171*.

Haixin Nan received the B.E. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2023. He is currently pursuing the M.S. degree with the Department of Computer Science and Technology, Nanjing University, Nanjing, China. His research interests include networks for AI and data center networks.

Jun Xu, photograph and biography not available at the time of publication.

Peirui Cao received the bachelor's degree from Southeast University in June 2015, the master's degree from Beihang University in March 2018, and the Ph.D. degree from Shanghai Jiao Tong University in June 2024. He is currently a Research Assistant Professor with the School of Computer Science, Nanjing University (NJU). His research interests include reconfigurable data center networks/AI clusters, large-scale network simulator and network bottleneck analysis. He is committed to closing the gap between theory and the real network system.

Zhaochen Zhang, photograph and biography not available at the time of publication.

Yizhi Wang, photograph and biography not available at the time of publication.

Zhehao Lin, photograph and biography not available at the time of publication.

Yuhang Li, photograph and biography not available at the time of publication.

Chengyuan Huang, photograph and biography not available at the time of publication.

Wentao Fan, photograph and biography not available at the time of publication.

Xiaohu Xu, photograph and biography not available at the time of publication.

Zhongming Ji, photograph and biography not available at the time of publication.

Shengju Zhang, photograph and biography not available at the time of publication.

Dongxu Wang, photograph and biography not available at the time of publication.

Lingkun Meng, photograph and biography not available at the time of publication.

Rong Gu (Member, IEEE), photograph and biography not available at the time of publication.

Guihai Chen (Fellow, IEEE), photograph and biography not available at the time of publication.

Chen Tian (Senior Member, IEEE), photograph and biography not available at the time of publication.