RailS: Load Balancing for All-to-All Communication in Distributed Mixture-of-Experts Training

Heng Xu[1], Zhiwei Yu[2], Chengze Du[1], Ying Zhou[1], Letian Li[3], Haojie Wang[4], Weiqiang Cheng[4], Jialong Li[1]✉
[1] Faculty of Computer Science and Control Engineering, Shenzhen University of Advanced Technology
[2] Institute for Network Sciences and Cyberspace, Tsinghua University
[3]Department of Information Engineering, Chinese University of Hong Kong
[4]China Mobile
✉lijialong@suat-sz.edu.cn

***Abstract*—Training Mixture-of-Experts (MoE) models introduces sparse and highly imbalanced all-to-all communication that dominates iteration time. Conventional load-balancing methods fail to exploit the deterministic topology of Rail architectures, leaving multi-NIC bandwidth underutilized. We present RailS, a distributed load-balancing framework that minimizes all-to-all completion time in MoE training. RailS leverages the Rail topology's symmetry to prove that uniform sending ensures uniform receiving, transforming global coordination into local scheduling. Each node independently executes a Longest Processing Time First (LPT) spraying scheduler to proactively balance traffic using local information. RailS activates N parallel rails for fine-grained, topology-aware multipath transmission. Across synthetic and real-world MoE workloads, RailS improves bus bandwidth by 20%–78% and reduces completion time by 17%–78%. For Mixtral workloads, it shortens iteration time by 18%–40% and achieves near-optimal load balance, fully exploiting architectural parallelism in distributed training.**

***Index Terms*—Mixture-of-Experts models, load balancing, all-to-all communication, rail-optimized architecture**

I. Introduction

ARTIFICIAL intelligence (AI) training has shifted from dense models to mixture-of-experts (MoE) models, making cross-GPU communication a critical bottleneck. While early dense models [23], [12] used structured collective communication over millions of parameters, modern large language models (LLMs) [10], [58], [63] rely on multi-level parallelism and collective operations at the scale of billions of parameters. MoE models [34], [15], [14], [53], [40], [28], [21] introduce expert parallelism with tens of billions of parameters, where all-to-all communication generates sparse and highly imbalanced traffic across hundreds of GPUs.

This all-to-all communication makes load balancing a central challenge. Studies show that such communication dominates MoE iterations, with latency limiting distributed training throughput [39]. To mitigate this, the Rail architecture [44] employs multi-NIC direct-to-leaf links to relieve bandwidth bottlenecks in conventional spine–leaf networks. Nevertheless, existing deployments remain suboptimal for MoE's irregular traffic patterns and dynamic load-balancing needs, leaving hardware potential largely untapped.

Challenge 1: The inherent path selection conflicts in the Rail architecture can completely negate its hardware parallelism advantages.

Path selection [64], [43], [50], [67], [26], [70] critically affects communication efficiency. Rail offers two deterministic path options for all-to-all traffic, each with limitations. The first routes traffic through the spine layer. It works with traditional topologies but relies on hashing [25], [69], which ignores MoE traffic sparsity. This causes large flows to cluster on a few links, creating congestion while other paths remain idle. The second option uses NIC-direct rails. Without advanced transport support such as multipath transmission, flow splitting, reordering, and reassembly, the bottleneck stays at a single node. Spine paths fail due to architecture limits, and direct rails are underutilized because of immature transport protocols, restricting Rail's parallelism.

Challenge 2: Existing load balancing methods overlook topological characteristics, preventing their scheduling decisions from approaching the theoretical upper bound.

Current load balancing approaches ignore Rail's deterministic topology. ECMP and dynamic load balancers [6], [32], [24], [49], [60] treat multiple paths as interchangeable and select them at random. However, Rail has N parallel rails with a one-to-one mapping from each source NIC to a destination NIC. This predictable and symmetric structure offers prior knowledge useful for global optimization. Existing methods overlook it, so both static hashing and dynamic probing choose paths blindly, fail to exploit the topology, and cannot achieve the theoretical optimum, limiting system throughput.

Challenge 3: The granularity of traffic splitting involves an inherent trade-off, creating a dilemma between load balancing and system efficiency.

In load balancing, loss is inevitable due to the granularity of traffic splitting. Splitting traffic at the subflow level [55], [46] can lead to collisions and uneven utilization across N NIC paths, even when flows are further divided or recombined. Splitting traffic at the flowlet [59], [61], [29] or packet level [13], [33], [9] can cause excessive simultaneous arrivals at a shared link, overflowing buffers and resulting in packet loss throughout the network, as in all-to-all traffic patterns. Coarse splitting fails to balance load effectively, while overly fine splitting overwhelms NICs, breaks hardware offload, and increases CPU overhead, reducing communication-computation overlap in AI training. Designers must therefore navigate the trade-off between effective load balancing and hardware efficiency.

To address these challenges, we propose RailS, whose pri-

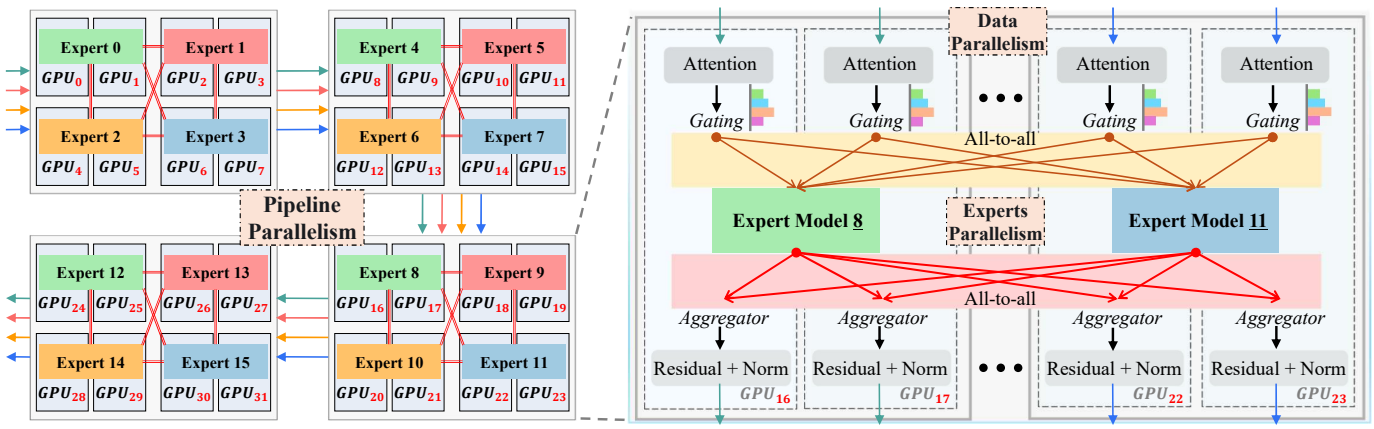


Fig. 1: Hybrid parallelism for a MoE model (data parallelism, pipeline parallelism, expert parallelism, and tensor parallelism).

mary contribution is resolving the fundamental path-selection conflicts. Our theoretical analysis leverages Theorem 1 to show that Rail provides N parallel logical channels between any pair of nodes, quantifying its maximum aggregate bandwidth. Building on this, Theorem 2 formulates the minimization of communication time as a min-max load balancing problem and establishes a standard linear programming solution. These results indicate that fully bypassing spine-layer forwarding, which is inefficient for MoE traffic, and exclusively using the N direct-connected rails constitutes the optimal strategy for all-to-all communication. RailS operationalizes this approach by activating these previously underutilized parallel paths, supporting up to 64 Queue Pairs (QPs).

Secondly, RailS addresses the "topological blindness" by exploiting the deterministic Rail topology. Theorem 3 shows a high symmetry between sending and receiving loads, so any strategy that balances sending loads also balances receiving loads. This provides a theoretical foundation for distributed scheduling, decomposing the global coordination problem into local subproblems where each node optimizes only its own sending load, thereby reducing communication and synchronization overhead. Consequently, RailS operates fully distributed, with each node executing Longest Processing Time first (LPT) scheduling based on local information while collaboratively achieving a near-optimal global solution.

Finally, RailS addresses the trade-off in splitting granularity through theory-driven traffic partitioning and scheduling. Underlying control coalescing mechanism executes low-level splitting, automatically dividing large flows into fixed-size atomic chunks. LPT approximation algorithm then performs proactive, upper-layer scheduling of them, replacing the default reactive logic. Theorem 4 shows that the algorithm's load imbalance is bounded by the maximum size $w_{\max}$, providing theoretical guidance for selecting an optimal granularity that balances load distribution and hardware efficiency.

We validate RailS in a large-scale, programmable datacenter simulator using a variety of synthetic workloads, including uniform, sparse, and skewed patterns, as well as real-world MoE traces. RailS consistently outperforms baselines: under sparse loads, bus bandwidth improves by 20%–78% across different sparsity levels and total completion time decreases by 17%–78%; in sender-skewed and receiver-skewed scenarios, it achieves the lowest mean squared error across NICs, indicating optimal load balance. For Mixtral workloads, RailS shortens iteration time by 18%–40%. In dense setups, it reduces collective completion time by 8.9%–44.3%, and in sparse setups, by 66.1%–80.4%. These results demonstrate RailS's ability to fully exploit architectural parallelism while maintaining consistent load balance across diverse and imbalanced workloads.

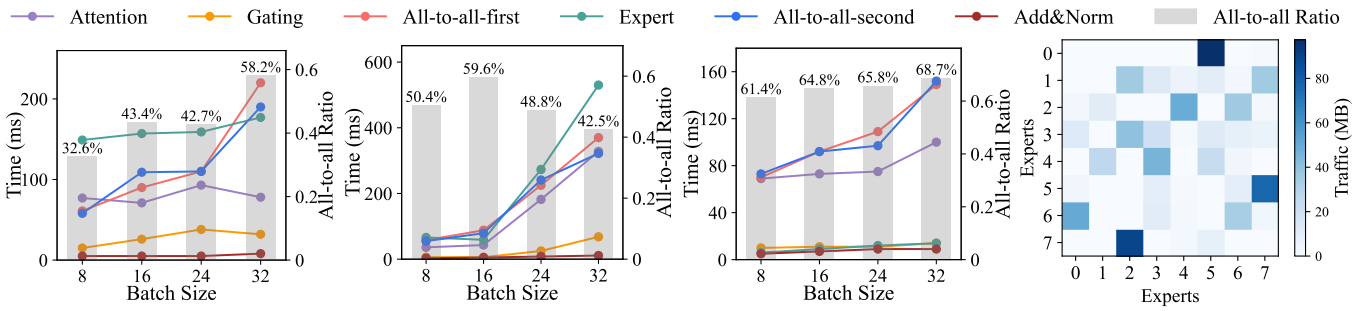
II. Background and Motivation

A. Research Background

Model Communication Challenges. AI training has evolved from deep neural networks (DNNs) with millions of parameters to LLMs with tens of billions, greatly increasing communication demands. Early data-parallel DNNs relied on allreduce for gradient synchronization with limited bandwidth pressure. In contrast, LLMs employ hybrid parallelism including data, tensor, and pipeline modes, where communication volume scales with model size and coordination complexity. In Fig. 1, MoE models further introduce sparsely activated experts, with only a subset active per sample. This produces highly sparse and dynamic all-to-all communication in Fig. 2, where all-to-all occupies 40%–60% of the iteration time, and its load imbalance confirms the impact on performance. Consequently, MoE training reveals the bottlenecks of communication-limited distributed systems.

Network Architectures Evolution. To meet the growing bandwidth and latency demands of AI training, data center networks [4], [18], [20], [19], [56], [44] have evolved from Fat-tree to Spine-leaf and Rail architectures. Three-tier designs suffer from high latency and centralized bottlenecks, while spine–leaf architectures reduce latency over three-tier designs, they fall short under GPU-scale all-to-all workloads, causing congestion and imbalance. The Rail architecture, adopted by NVIDIA, connects multiple NICs to different leaf switches for single-hop GPU paths and parallel bandwidth use. Yet in MoE training, dynamic loads and path constraints still cause NIC congestion and bandwidth underutilization.

Multipath Transmission and Load Balancing. Multipath transmission and load balancing are essential for improving data center communication and supporting distributed node



(a) Mixtral-MoE (b) LLaMA-MoE (c) Qwen-MoE (d) Non-uniform all-to-all traffic

Fig. 2: Actual measured overhead [39] of different modules in MoE models under expert parallelism: (a) Mixtral 8×7B, (b) LLaMA 6.7B, and (c) Qwen 14.3B. The all-to-all ratio indicates the measured proportion of iteration time spent on all-to-all communication. (d) Empirically measured traffic matrix of Mixtral 8×7B.

coordination. Early work used flowlet-based TCP splitting, while protocols such as MPTCP [52] leverage multiple paths at the connection level, though their complexity limits large-scale deployment. With widespread RDMA [54] adoption in AI clusters, communication increasingly relies on single-path low-latency transmission, which restricts subflow coordination and amplifies bursty all-to-all loads in distributed MoE training, reducing global efficiency. Recent frameworks explore multipath optimization to better utilize available network paths, alleviate hotspots, and improve overall throughput and latency in distributed training.

B. Research Motivation

Traditional Topologies and MoE Traffic Incompatibility. In Spine-leaf networks, ECMP selects cross-leaf paths through static hashing, which performs well for uniform traffic but fails under MoE all-to-all patterns. Sparse expert activations create highly imbalanced traffic that ECMP cannot adapt to, causing NIC and uplink load disparities up to 5–10×. This imbalance prolongs critical all-to-all stages and limits MoE training throughput. Existing mitigations, such as disabling NIC congestion control [16], [68] or adopting dual-plane fabrics [48], risk deadlocks [27], [42] or incur high cost, without resolving the fundamental mismatch between ECMP and sparse traffic.

Rail Architecture Communication Bottlenecks. The Rail architecture connects multiple NICs directly to leaf switches, enabling inherent multipath parallelism. Yet in distributed MoE all-to-all training, current implementations underutilize this potential. RoCE [2], [3] enforces fixed NIC-level paths, preventing dynamic scheduling across NICs. NCCL [1] supports path selection but only within a single NIC, while UCCL [70] lacks topology-aware scheduling and ignores fixed NIC–leaf mappings. Consequently, cross-NIC bandwidth remains uneven and coordination efficiency constrained, leaving Rail's architectural advantages underutilized in MoE training.

Existing Load Balancing Limitations. Prior MoE load balancing studies focus on computation or expert placement while neglecting topological constraints and multipath characteristics. Some assume ECMP can evenly distribute traffic but overlook fixed NIC–leaf bindings in Rail, causing subflow mismatches. Others pursue software-based subflow scheduling without considering RDMA offloading, introducing overhead and disrupting communication–computation overlap. Lacking integrated modeling of topology and hardware, these approaches fail to mitigate sparse traffic hotspots, leaving load imbalance unresolved in Rail clusters.

III. OVERVIEW

This chapter provides a high-level overview of the proposed traffic scheduling system. First, we describe the core design premises and problem boundaries of the system. Next, we present the overall system design blueprint, including its key mechanisms and optimization objectives. Finally, this chapter summarizes the critical theoretical contributions introduced in subsequent chapters, providing readers with a clear roadmap.

A. Design Premises and Problem Scope

Before detailing our system design, this section first clarifies the core premises, model assumptions, and problem boundaries of our work.

Network Topology. In Fig. 3, this study focuses on the Rail network architecture, which equips each compute node with multiple NICs, each directly connected to a different leaf switch, providing a hardware foundation for multipath parallel communication. All subsequent theoretical analyses and system designs are built upon the intrinsic structural characteristics of this emerging topology.

Traffic Model. Our optimization target is the key performance bottleneck in current large-scale distributed training: all-to-all communication generated by MoE models. This traffic exhibits significant sparsity and imbalance, rendering traditional static load-balancing mechanisms inefficient. Our theoretical analysis and real-time scheduling are based on the known inter-GPU traffic matrix $D^{(1)}$ and inter-domain traffic matrix $D^{(2)}$.

System Abstraction. At the software level, the system adopts a scalable transport-layer framework that decouples the control and data paths of RDMA NICs. This separation enables flexible, software-defined implementation of transport-layer protocols on the host CPU. The extensibility provides the

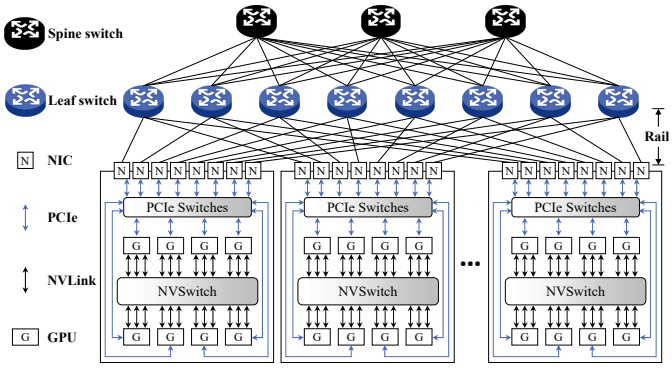


Fig. 3: Rail-optimized network topology.

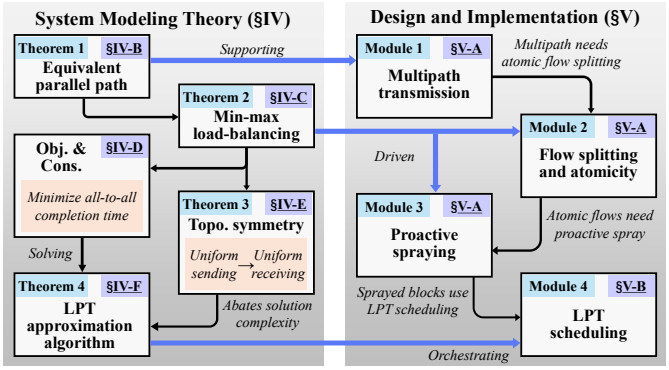


Fig. 4: Overview of system workflow.

technical foundation for integrating a custom LPT scheduler to enable proactive load balancing.

B. System Design Overview

We design and propose RailS, a distributed traffic load balancer specifically optimized for MoE communication under the Rail architecture. The system aims to minimize the communication completion time $T^{\star}$. Its core idea is to actively shape and schedule sparse, imbalanced flows at each sender node through a set of coordinated mechanisms, thereby approximating globally optimal load balancing.

The RailS's design comprises three core mechanisms forming an efficient splitting-scheduling-spraying pipeline:

Flow Splitting. Serving as the entry point of the pipeline, this mechanism decomposes large, coarse-grained messages generated by the upper-layer applications into a series of smaller, atomic flow units. This preprocessing step is crucial for fine-grained load balancing and for reducing the maximum flow size $w_{\max}$.

LPT Scheduling. Acting as the brain of the system and the core of the proactive spraying, this mechanism differs from traditional strategies that rely on passive feedback such as network latency. Using an LPT-based scheduler, it deterministically assigns each flow to the optimal target track based on the sizes of all pending atomic flows and the real-time local load state of each track, thereby actively constructing a highly balanced flow allocation.

Multipath Spraying. This mechanism provides the macroscopic strategy for flow distribution. It distributes the split atomic flows across all N available parallel communication tracks at each node. Unlike static-hash-based ECMP, spraying mechanism is fully dynamic and policy-driven.

Through the coordinated operation of these mechanisms, RailS effectively decomposes a complex global optimization problem into distributed local decisions that can be executed independently and efficiently at each node, ultimately translating the theoretical insights from Chapter 4 into practical system performance.

C. Core Theoretical Contributions Roadmap

This section outlines how a series of interconnected theoretical analyses establish the foundation for the subsequent system design of RailS. Fig. 4 illustrates the workflow.

Question 1 (§IV-B): What is the Theoretical Communication Capacity of the Rail Architecture? Our analysis begins by rigorously defining the fundamental capability of the Rail network topology. Using the max-flow min-cut method, we prove that the architecture provides N parallel, equal-capacity logical channels for arbitrary inter-domain communication (Theorem 1). This result not only quantifies the system's bandwidth upper bound but also establishes the feasibility and necessity of multipath load balancing.

Question 2 (§IV-C): How to Formulate the Time Minimization Problem? Having established the network capacity, we reformulate the core optimization objective of minimizing all-to-all communication time into a mathematical framework. We prove that this complex dynamic time optimization problem is equivalent to a more tractable min-max load-balancing problem (Theorem 2). This equivalence provides a clear and actionable optimization objective for designing allocation strategies.

Question 3 (§IV-E): What is the Optimal State for Load Balancing? To define an "ideal target" for practical scheduling algorithms, we first solve the problem under the idealized assumption that flows are arbitrarily divisible. We prove that the optimal strategy admits a concise closed-form solution, namely a perfectly uniform distribution across N rails (Theorem 3). This result not only reveals the intrinsic symmetry between sending and receiving loads in the Rail architecture but also sets the theoretical performance ceiling for all practical approximation algorithms.

Question 4 (§IV-F): How to Approximate the Optimum with Discrete Flows? Finally, we address the more realistic scenario of indivisible (atomic) flows. We propose the LPT approximation algorithm and prove that it efficiently approaches the ideal uniform state established in Step 3, with a guaranteed performance bound where the mean squared error is upper bounded by $w_{\max}^2$ (Theorem 4). This provides a solid theoretical basis for implementing efficient, low-overhead local scheduling in distributed nodes.

IV. Problem Formulation

In this chapter, we model the Rail-based network topology and reformulate the all-to-all communication optimization as an equivalent load-balancing problem. We derive closed-form

optimal strategies for both continuous and atomic flow scenarios and propose an efficient approximation algorithm with proven performance guarantees, establishing a solid theoretical foundation for system design.

A. System Model

We consider a large-scale GPU cluster system composed of M computing domains. The system is communication-constrained and supports distributed training of LLMs and MoE models. Each domain contains N GPUs interconnected via intra-domain networks. Any GPU in the system can be uniquely identified by a tuple (d, n), where $d \in \{1, \ldots, M\}$ is the domain index and $n \in \{1, \ldots, N\}$ is the GPU index within that domain. Each GPU (d, n) is connected to a dedicated network interface card, denoted as $\mathrm{NIC}_{d,n}$.

The inter-domain communication architecture follows a Rail-based design. Specifically, for any fixed intra-domain index n, the n-th NIC of all domains (i.e., $\mathrm{NIC}_{1,n}, \mathrm{NIC}_{2,n}, \ldots, \mathrm{NIC}_{M,n}$) connects to the same Leaf switch S_n. Hence, the system contains N such switches, forming N parallel communication "rails." The Spine and Leaf layers remain fully connected.

To precisely analyze this system, we define the following key concepts.

Network Link Rates. Two types of link rates are considered. The intra-domain forwarding rate R_1 represents the data exchange rate among GPUs within a single domain. The inter-domain forwarding rate R_2 corresponds to the forwarding rate of domain NICs or switches and reflects the data processing capability of switch ports.

Traffic Matrix. To quantify communication demands, we define two levels of traffic matrices. The GPU-to-GPU traffic matrix $D^{(1)}$ has elements $D^{(1)}_{(d,n),(f,m)}$ representing the traffic from source GPU (d, n) to destination GPU (f, m). Aggregating these yields the domain-to-domain traffic matrix $D^{(2)}$, where each element $D^{(2)}_{d,f}$ denotes the total traffic from domain d to domain f:

$$D^{(2)}_{d,f} = \sum_{n=1}^{N} \sum_{m=1}^{N} D^{(1)}_{(d,n),(f,m)} \tag{1}$$

Allocation Matrix. The allocation matrix P is a three-dimensional decision variable describing the routing of traffic. Its elements $P_{k,f,n}$ denote the proportion of total traffic from source domain k to destination domain f assigned to the n-th communication rail. The matrix and constraints are given by:

$$P = [P_{k,f,n}] \in \mathbb{R}^{M \times M \times N} \tag{2}$$

$$\text{s.t.} \quad \sum_{n=1}^{N} P_{k,f,n} = 1, \quad P_{k,f,n} \geq 0, \quad \forall k, f, n \tag{3}$$

Load Matrix. Given an allocation matrix P, the communication load on each NIC can be computed. We define the sending load matrix $\mathbf{S} \in \mathbb{R}^{M \times N}$ and receiving load matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$:

$$\mathbf{S}_{k,n} = \sum_{f=1}^{M} D^{(2)}_{k,f} P_{k,f,n} \tag{4}$$

$$\mathbf{R}_{f,n} = \sum_{k=1}^{M} D^{(2)}_{k,f} P_{k,f,n} \tag{5}$$

All-to-All Completion Time. In the all-to-all communication pattern, the total completion time T is defined as the time from the start of communication until all inter-domain data transfers are finished. It is determined by the most heavily loaded NIC (either sending or receiving) and serves as the key metric of system communication performance. Minimizing the time $T^\star$ is one of the main optimization objectives.

Load Balancing Mean Squared Error. To quantify the deviation between actual NIC loads and the ideal target within a domain, we define the mean squared error (MSE). Let the ideal target load T_{opt} be the average load of the domain, and let L_j denote the load assigned to the j-th NIC. The MSE is:

$$\text{MSE} = \frac{1}{N} \sum_{j=1}^{N} (L_j - T_{\text{opt}})^2 \tag{6}$$

B. Inter-Domain Communication Capacity Analysis

We now analyze the fundamental limits of inter-domain communication in the Rail-based network. This section quantifies the theoretical upper bound of cross-domain throughput and establishes its equivalence-in-use to a set of parallel logical rails, which provides the analytical foundation for subsequent flow allocation and optimization.

Theorem 1. *Under the defined system model and assuming that the intra-domain forwarding rate is higher than the inter-domain forwarding rate ($R_1 > R_2$), the maximum achievable unidirectional throughput between any two distinct domains k and f is:*

$$Cap_{k \to f} = N \cdot R_2 \tag{7}$$

Proof. We represent the Rail-based interconnect as a directed capacitated graph $G = (V, E)$. Each node corresponds to either a GPU, a NIC, or a switch, and each directed edge $e \in E$ has transmission capacity c_e. Let $\mathbf{c} = (c_e)_{e \in E}$ and $\mathbf{f} = (f_e)_{e \in E}$ satisfy:

$$0 \leq f_e \leq c_e, \quad \forall e \in E \tag{8}$$

For inter-domain analysis, each domain d is modeled as a cluster of N GPUs with dedicated NICs $(\mathrm{NIC}_{d,1}, \ldots, \mathrm{NIC}_{d,N})$. We contract the source domain k to a super-source s and the destination domain f to a super-sink t; the validity of this contraction will be justified by a no-saturation argument under $R_1 > R_2$ below. Let $B \in \mathbb{R}^{|V| \times |E|}$ be the node–edge incidence matrix. Flow conservation is:

$$B\mathbf{f} = \mathbf{b}, \qquad \mathbf{b} = \big(\ldots, \underbrace{+x}_{v=s}, \ldots, \underbrace{-x}_{v=t}, \ldots\big)^\top, \tag{9}$$

where x denotes the unidirectional throughput from k to f. The feasible region is:

$$\mathcal{F} = \{(x, \mathbf{f}) \ : \ B\mathbf{f} = \mathbf{b}, \ \mathbf{0} \leq \mathbf{f} \leq \mathbf{c}\} \tag{10}$$

a) Cut-Based Upper Bound: For any s–t cut $(C, \bar{C})$ with $s \in C$, $t \in \bar{C}$, the forward cut-set is $\delta^+(C)$. By the Max-Flow–Min-Cut theorem, it has:

$$x \le \mathrm{Cap}(C) := \sum_{e \in \delta^+(C)} c_e \tag{11}$$

Choose C to include all nodes of domain k. The forward edges leaving C are exactly the N inter-domain uplinks:

$$E_{k \to \mathrm{fabric}} = \left\{ e_n : \ \mathrm{NIC}_{k,n} \to S_n \right\}_{n=1}^{N} \tag{12}$$

where S_n denotes the leaf switch attached to $\mathrm{NIC}_{k,n}$ (and symmetrically to $\mathrm{NIC}_{f,n}$). Each e_n has capacity R_2, so

$$\mathrm{Cap}(C) = \sum_{n=1}^{N} c_{e_n} = N \cdot R_2 \tag{13}$$

Therefore:

$$Cap_{k \to f} \le N \cdot R_2 \tag{14}$$

b) Lower Bound: Let $\mathcal{P}$ denote the set of simple $s \to t$ paths. We explicitly construct N rail-aligned paths:

$$\Pi_n : s \to \mathrm{NIC}_{k,n} \to S_n \to \mathrm{NIC}_{f,n} \to t, n = 1, \ldots, N \tag{15}$$

where each path Π_n uses a distinct NIC pair $(\mathrm{NIC}_{k,n}, \mathrm{NIC}_{f,n})$ and its associated leaf S_n. The NIC–leaf attachments are fixed per rail, so intra-domain edges used by different rails are edge-disjoint.

Assign rate $\alpha_n = R_2$ to each Π_n. Define the path–edge incidence variable $a_{e,n} \in \{0, 1\}$, which equals 1 if edge e lies on path Π_n and 0 otherwise. The induced edge load is:

$$f_e = \sum_{n=1}^{N} \alpha_n a_{e,n} = \sum_{n=1}^{N} R_2 \, a_{e,n} \tag{16}$$

By construction, all inter-domain edges are edge-disjoint across $\{\Pi_n\}_{n=1}^{N}$, and each such edge has capacity R_2, hence $f_{e_n} = R_2 \le c_{e_n} = R_2$.

Let E_{intra} denote the set of intra-domain edges. For intra-domain segments on Π_n, every traversed edge has capacity at least R_1 and carries only the rate of its own rail $\alpha_n = R_2$. Therefore, for each $n = 1, \ldots, N$ and each intra-domain edge $e \in \Pi_n \cap E_{\mathrm{intra}}$, we have:

$$f_e = R_2, \quad c_e \ge R_1 > R_2 \ \Rightarrow \ f_e < c_e \tag{17}$$

No intra-domain edge saturates under $R_1 > R_2$. Hence, contracting each domain to a super-node preserves the max-flow value. The aggregate throughput achieved by $\{\Pi_n\}$ is:

$$x = \sum_{n=1}^{N} \alpha_n = N \cdot R_2 \tag{18}$$

which proves:

$$Cap_{k \to f} \ge N \cdot R_2 \tag{19}$$

Combining (14) and (19) yields the equality.

c) Engineering Note: The theorem only assumes $R_1 > R_2$, whereas practical systems usually have $R_1 \gg R_2$ because intra-domain networks (e.g., NVLink) exceed NIC-based inter-domain links by a large margin, often close to an order of magnitude. Hence, inter-domain links form the bottleneck in practice, validating our parallel-rails view. ■

Summary. Rail-based network provides N independent logical rails, each with capacity R_2, for any domain pair (k, f). Therefore, the overall system capacity scales linearly with N, and subsequent optimization focuses on how to allocate traffic across these N rails to achieve global load balance and minimize all-to-all completion time.

C. Completion Time as a Load Balancing Problem

Building on Theorem 1, which establishes that inter-domain communication relies on N parallel rails, we now analyze the global all-to-all communication pattern. The objective is to minimize the overall completion time $T^\star$. Since direct optimization of $T^\star$ is intractable, the problem is reformulated as an equivalent resource allocation problem in the form of load balancing across rails, which provides a more tractable analytical framework.

The all-to-all communication demand is described by the inter-domain traffic matrix $D^{(2)}$, while a specific flow allocation is characterized by the allocation matrix P. These jointly determine the per-domain sending and receiving loads $\mathbf{S}_{d,n}$ and $\mathbf{R}_{d,n}$ on each rail n. The system completion time is ultimately governed by the maximum processing time among all such loads. The following theorem formalizes this relationship between completion time and load distribution.

Theorem 2. *Under the defined model and given the traffic matrix $D^{(2)}$, the minimum completion time $T^\star$ of the all-to-all communication is equivalent to solving a min-max load balancing problem over all NICs in both sending and receiving directions. If the rate of all inter-domain links is R_2, then:*

$$T^\star = \frac{1}{R_2} \min_{P} \left\{ \max_{k,f,n} \left(\mathbf{S}_{k,n}, \mathbf{R}_{f,n} \right) \right\} \tag{20}$$

where $\min_P$ denotes the optimization over the strategy space formed by all feasible allocation matrices P, and $\max_{k,f,n}$ indicates the maximum taken over all sending and receiving loads.

Proof. This proof aims to demonstrate that any achievable completion time T is necessarily bounded below by the maximum load across all NICs, and that this lower bound can be attained by an optimal allocation.

For any given allocation strategy P, the corresponding load matrices $\mathbf{S}$ and $\mathbf{R}$ are fully determined. Completion of the all-to-all communication task implies that all sending and receiving operations on every track for every domain must be finished. Therefore, the total completion time T must be at least as large as the time required for any individual NIC to complete its assigned workload.

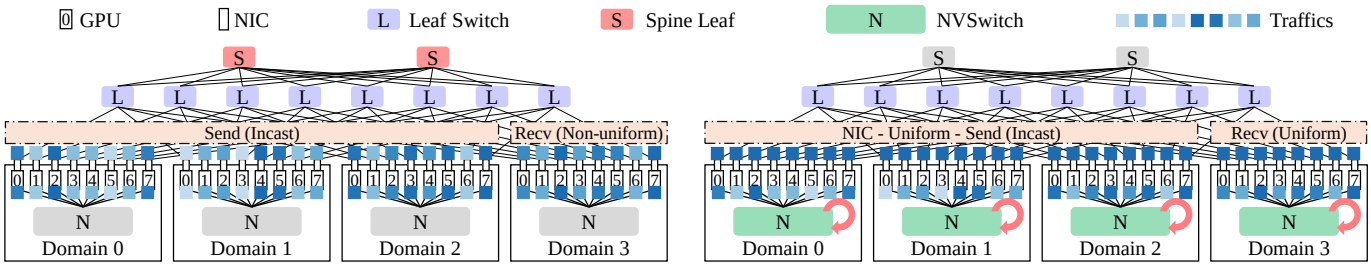


Fig. 5: Illustration of symmetry mechanism. The left figure does not leverage the topology. The right figure leverages the intra-domain forwarding and topological properties, where uniform NIC sending results in uniform receiving.

For any $\mathrm{NIC}(d, n)$, the time to complete its sending load $\mathbf{S}_{d,n}$ is $\mathbf{S}_{d,n}/R_2$, and the time to complete its receiving load $\mathbf{R}_{d,n}$ is $\mathbf{R}_{d,n}/R_2$. Hence, we have:

$$T \geq \frac{\mathbf{S}_{d,n}}{R_2} \quad \text{and} \quad T \geq \frac{\mathbf{R}_{d,n}}{R_2} \tag{21}$$

Considering all NICs, the completion time under this strategy must satisfy:

$$T \geq \frac{1}{R_2} \max_{k,f,n} (\mathbf{S}_{k,n}, \mathbf{R}_{f,n}) \tag{22}$$

This inequality holds for all feasible allocation strategies P. Therefore, the global minimum completion time $T^\star$ cannot be smaller than the optimal value obtained by minimizing this maximum load over all strategies:

$$T^\star \geq \frac{1}{R_2} \min_{P} \left\{ \max_{k,f,n} (\mathbf{S}_{k,n}, \mathbf{R}_{f,n}) \right\} \tag{23}$$

The above derivation establishes a lower bound. When traffic can be continuously divided, the search for an optimal allocation matrix P minimizing the load $\max(\mathbf{S}, \mathbf{R})$ reduces to a linear programming problem. Standard linear programming theory ensures the existence of an optimal allocation $P^\star$ whose induced maximum load attains the min–max optimum. Under this allocation, the system completion time reaches the lower bound, and the inequality becomes an equality.

In summary, the problem of minimizing the all-to-all time is rigorously proven to be equivalent to minimizing the maximum sending and receiving load across all NICs. ■

Summary. This theorem transforms the dynamic performance metric into a optimizable load objective, guiding the search for an optimal traffic allocation strategy P that minimizes the maximum NIC load. To solve this min-max problem systematically, it must first be formulated as a mathematical program. The next section presents the linear programming formulation that underlies the optimal allocation strategy.

D. Objectives and Constraints

Based on the conclusions of the previous section, we formulate the min-max problem of all-to-all minimum completion time as a standard linear programming (LP) model to facilitate formal solution. The model takes the allocation matrix P as the core decision variable and introduces an auxiliary variable t representing the upper bound of all loads.

The optimization problem is formulated as:

$$\begin{aligned} \min_{P,t} \quad & t \\ \text{s.t.} \quad & \sum_{f=1}^{M} D_{k,f}^{(2)} \cdot P_{k,f,n} \leq t, \quad \forall k, n \\ & \sum_{k=1}^{M} D_{k,f}^{(2)} \cdot P_{k,f,n} \leq t, \quad \forall f, n \\ & \sum_{n=1}^{N} P_{k,f,n} = 1, \quad \forall k, f \\ & P_{k,f,n} \geq 0, \quad \forall k, f, n \end{aligned} \tag{24}$$

In this LP formulation, the auxiliary variable t represents the objective to be minimized and is defined as the global upper bound of all NIC sending loads $\mathbf{S}_{k,n}$ and receiving loads $\mathbf{R}_{f,n}$. Minimizing t therefore corresponds to minimizing the system's bottleneck load. The remaining constraints guarantee a valid allocation matrix P, ensuring complete traffic assignment (normalization) and correct directionality (non-negativity). As both the objective and constraints are linear, the problem can be solved optimally using standard LP methods.

This LP formulation provides a numerical solution to the problem, while the next section will further explore and present a more insightful analytical solution.

E. Topological Symmetry of Load

The all-to-all minimum completion time problem has been formulated as a linear program. Here we derive a closed-form analytical solution and prove its optimality. The key insight is that in the Rail-based architecture, uniform sending loads inherently yield uniform receiving loads in Fig. 5.

Theorem 3. *In the studied system, any allocation strategy that achieves uniformization of sending loads is globally optimal. Specifically, if for every source domain k, the sending load is uniformly distributed as:*

$$\mathbf{S}_{k,n} = \frac{1}{N} \sum_{f=1}^{M} D_{k,f}^{(2)}, \quad \forall n = 1, \ldots, N \tag{25}$$

then the receiving load for any destination domain f is automatically uniformized:

$$\mathbf{R}_{f,n} = \frac{1}{N} \sum_{k=1}^{M} D_{k,f}^{(2)}, \quad \forall n = 1, \ldots, N \tag{26}$$

This strategy simultaneously minimizes both sending and receiving bottlenecks, thereby achieving the all-to-all minimum completion time $T^\star$.

Proof. The core of this proof relies on the system structure revealed in Theorem 1. Communication from any source domain k to destination domain f is equivalent to N parallel, independent paths, with endpoints $\mathrm{NIC}_{k,n}$ and $\mathrm{NIC}_{f,n}$ for $n = 1, \ldots, N$. This implies that traffic sent from $\mathrm{NIC}_{k,n}$ along the n-th track is uniquely directed to $\mathrm{NIC}_{f,n}$.

We first construct a sending-uniform strategy. This strategy evenly distributes the total outgoing traffic of each source domain k, $\sum_{f=1}^{M} D_{k,f}^{(2)}$, across its N NICs. Specifically, the sending load on the n-th NIC is set equal to the average load. By adopting the allocation strategy:

$$P_{k,f,n}^{\star} = \frac{1}{N} \tag{27}$$

we obtain the following equation:

$$\mathbf{S}_{k,n} = \sum_{f=1}^{M} D_{k,f}^{(2)} P_{k,f,n}^{\star} = \frac{1}{N} \sum_{f=1}^{M} D_{k,f}^{(2)} \tag{28}$$

This strategy successfully uniformizes the sending load, thereby minimizing the maximum processing time among all sending tasks.

Next, we examine the impact of this strategy on receiving loads. The receiving load $\mathbf{R}_{f,n}$ on the n-th track for any destination domain f is the sum of the traffic from all source domains k through that track. Based on the one-to-one correspondence of paths in the Rail architecture, it has:

$$\mathbf{R}_{f,n} = \sum_{k=1}^{M} D_{k,f}^{(2)} P_{k,f,n}^{\star} = \frac{1}{N} \sum_{k=1}^{M} D_{k,f}^{(2)} \tag{29}$$

This shows that the receiving load is automatically uniformized, with each receiving NIC carrying exactly the average of the domain's total incoming traffic.

Thus, a simple sending-uniform strategy, due to the intrinsic system architecture, inevitably leads to uniform receiving loads. Uniform sending minimizes the sending bottleneck, while the resulting uniform receiving minimizes the receiving bottleneck. Since this single strategy simultaneously optimizes both sending and receiving loads, it is globally optimal and achieves the all-to-all minimum completion time $T^\star$.

Consequently, the optimal allocation matrix achieving this state has a simple closed-form solution:

$$P_{k,f,n}^{\star} = \frac{1}{N}, \quad \forall k, f, n \tag{30}$$

which satisfies the normalization constraint:

$$\sum_{n=1}^{N} P_{k,f,n}^{\star} = 1 \tag{31}$$

The proof is complete. ■

Summary. This theorem provides the optimal allocation strategy under the ideal assumption that traffic can be arbitrarily split. However, in practical systems, traffic often exists as discrete units that cannot be fully subdivided. The next section investigates how to effectively approximate this ideal uniform allocation strategy under such discrete constraints.

F. LPT-based Load Balancing for Atomic Flows

The previous section derived the optimal allocation under the ideal assumption of arbitrarily divisible traffic. In practice, however, communication between GPUs is scheduled as indivisible units, making continuous allocation infeasible. Even when flows are split, they remain discrete, as large flows are only divided into several smaller ones (packets or flowlets) rather than continuous segments (bytes or bits).

This section addresses the load balancing problem under such discrete constraints. We aim to minimize the mean squared error between the actual loads and the ideal uniform loads guided by Theorem 3. To this end, we propose an efficient approximate allocation algorithm and provide theoretical guarantees for its performance.

We assume that each flow is indivisible or has been split into the smallest atomic units. The inter-domain total traffic matrix $D^{(2)}$ is known, i.e., the flow between each source domain k and destination domain f, $D_{k,f}^{(2)}$, is given. Based on a fixed allocation matrix P, let p_n denote the fraction of traffic ideally assigned to the n-th NIC. Then the target load for each flow on NIC n is defined as:

$$T_{k,n} = \sum_{f=1}^{M} D_{k,f}^{(2)} p_n \tag{32}$$

Each GPU-to-GPU flow must be assigned entirely to a single NIC. The problem can thus be formalized as allocating a set of discrete flows with weights w_i to N NICs, such that the actual load on each NIC

$$L_j = \sum_i w_i x_{i,j}, \quad x_{i,j} \in \{0, 1\} \tag{33}$$

is as close as possible to its target load T_j. The binary assignment variables $x_{i,j}$ satisfy:

$$x_{i,j} = \begin{cases} 1, \text{flow } i \text{ on NIC } j \\ 0, \text{otherwise} \end{cases}, \sum_{j=1}^{N} x_{i,j} = 1, \ \forall i \tag{34}$$

To quantify the load deviation, we introduce the MSE as the metric:

$$\mathrm{MSE} = \frac{1}{N} \sum_{j=1}^{N} (L_j - T_j)^2 \tag{35}$$

Under this definition, the GPU-to-GPU flow allocation problem with fixed allocation fractions p_n can be formalized as a combinatorial optimization problem:

$$\begin{aligned} \min_{x_{i,j}} \quad & \frac{1}{N} \sum_{j=1}^{N} (L_j - T_j)^2 \\ \text{s.t.} \quad & \sum_{j=1}^{N} x_{i,j} = 1, \quad \forall i, \\ & L_j = \sum_i w_i x_{i,j}, \quad \forall j, \\ & x_{i,j} \in \{0, 1\}, \quad \forall i, j \end{aligned} \tag{36}$$

This problem is a discrete combinatorial optimization task, aiming to minimize the mean squared deviation between the actual load and the target load on each NIC. Since each flow

is assigned to exactly one NIC, directly solving this problem incurs high computational complexity in large-scale systems. To reduce computational overhead, we adopt the LPT for approximate allocation.

The LPT method first sorts all flows in descending order of weights w_i and then sequentially assigns each flow to the NIC with the currently smallest total load until all flows are allocated. According to Theorem 3, the sending-uniform strategy ensures that the target load T_j already matches the ideal average load $L_{\text{opt}} = \sum_i w_i/N$, so the LPT algorithm effectively performs discrete allocation while maximizing the utilization of uniformly balanced NIC loads.

Theorem 4. *For the discrete-flow load balancing problem described above, when the LPT algorithm is employed for allocation, the MSE between the actual load and the ideal target load is upper-bounded by the square of the largest single flow weight* $w_{\max}$*:*

$$MSE \leq w_{\max}^2 \tag{37}$$

Proof. Classical LPT theory [17] guarantees that, for the standard discrete load balancing problem, the maximum load $L_{\max}$ satisfies:

$$L_{\max} \leq L_{\text{opt}} + w_{\max} \tag{38}$$

where $w_{\max} = \max_i w_i$ denotes the weight of the largest single flow. The LPT approximation ratio is given by:

$$R = \frac{L_{\max}}{L_{\text{opt}}} \leq \frac{4}{3} - \frac{1}{3N} \tag{39}$$

Consequently, the deviation of the actual load on each NIC from the target load is bounded by:

$$|L_j - L_{\text{opt}}| \leq L_{\max} - L_{\text{opt}} \leq w_{\max}, \quad \forall j. \tag{40}$$

This directly leads to the following upper bound on MSE:

$$\text{MSE} = \frac{1}{N}\sum_{j=1}^{N}(L_j - L_{\text{opt}})^2 \leq w_{\max}^2 \tag{41}$$

Thus, by sorting flows in descending order and greedily assigning them to the NIC with the current minimum load, LPT effectively minimizes the load deviation of each NIC and indirectly controls the MSE. The computational complexity of the algorithm is dominated by sorting and assignment operations, which are $O(F \log F)$ and $O(FN)$, respectively, yielding an overall complexity of $O(F \log F + FN)$, where F denotes the number of flows. Compared to the exponential complexity of an exact ILP solution, LPT provides a significant reduction in computational overhead, making it suitable for large-scale multi-GPU systems. ■

Summary. This section extends the continuous-flow theory to practical systems with indivisible flows. Through the LPT algorithm, we show that discrete allocation can still achieve predictable load balancing. The key insight is that load balancing effectiveness depends on the maximum flow granularity, motivating the next focus on designing flow-splitting and scheduling mechanisms that control granularity to approach the theoretical optimum.

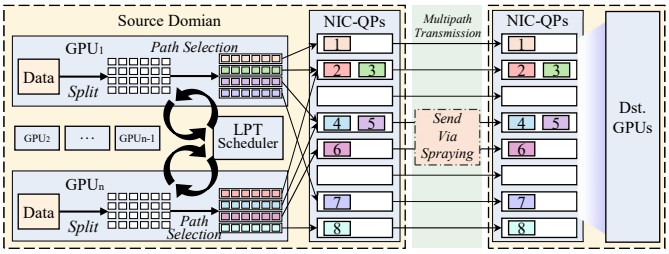


Fig. 6: System design and implementation.

V. Implementation

Building upon the theoretical framework established in the previous chapter, this section presents the concrete design and implementation of an all-to-all communication system optimized for the Rail architecture. We first introduce the overall framework adopted to achieve load balancing, encompassing multi-path transmission, flow splitting, and the spraying mechanism. Subsequently, we provide a detailed description of the design and implementation of the LPT scheduler.

A. Multi-Path spraying Framework

To fully exploit the N parallel communication lanes provided by the Rail architecture, our system employs multi-path packet spraying as the core traffic distribution mechanism in Fig. 6. Spraying is an advanced multi-path transmission technique designed to actively disperse the units of a logical data flow (e.g., packets or subflows) across multiple physical paths. For the sparse and uneven all-to-all traffic generated by MoE models, the spraying mechanism provides finer-grained and more dynamic load balancing compared to traditional ECMP static hashing, effectively alleviating link congestion caused by hash collisions on large flows.

Our system builds on a scalable software transport framework that supports flexible, multi-path data transmission across NICs. Algorithm 1 presents the procedure for system operation in MoE training. Its primary advantage lies in its successful decoupling of the RDMA NIC's control and data paths, enabling flexible implementation and innovation of transport-layer protocols in software on the host CPU. This design provides a natural entry point for integrating a custom LPT scheduler. The framework mainly exploits some key features.

Multipath Transmission. The system realizes multipath communication by establishing and managing multiple QPs between each pair of communicating NICs. In our Rail architecture, each node is equipped with N NICs, and this mechanism is used to construct N parallel logical communication lanes, perfectly corresponding to the N parallel channels in our theoretical model. By default, it supports up to 256 QPs for parallelization and load balancing, providing abundant path resources for traffic spraying.

Flow Splitting and Atomicity. Theorem 4 indicates that the effectiveness of load balancing is closely related to the granularity of traffic. The underlying control coalescing mechanism naturally supports flow splitting. It segments large application-layer messages into fixed-size data chunks (e.g., 32 KB by default), and makes independent transmission decisions for

Algorithm 1: MoE distributed training with DP/EP/PP and LPT-based all-to-all communication.

Input: Local GPU g; Number of rails N; model parameters Θ; Expert assignment $\mathcal{E}$
Output: Updated model parameters Θ after one training iteration

```
// Step 0: Preprocessing / Input preparation
```
Load mini-batch input data for this GPU.
```
// Step 1: Attention and gating computation
```
Compute attention outputs: $AttnOut \leftarrow Attention(Inputs, \Theta)$
Compute gating decisions: $Gate \leftarrow Gate(AttnOut)$
```
// Step 2: First All-to-All (inputs to experts)
```
Select input slices for local experts according to $Gate$ and $\mathcal{E}$
Split inputs into atomic flows $\{w_i\}$
Call **LPT Scheduler** to assign flows to rails
Transmit flows via spraying
```
// Step 3: Expert computation (distributed EP)
```
Compute local expert outputs $ExpOut_e$
If expert spans multiple GPUs, aggregate partial results
```
// Step 4: Second All-to-All (expert outputs aggregation)
```
Split expert outputs into atomic flows $\{w_i\}$
Call **LPT Scheduler** to assign flows to rails
Transmit flows via spraying
```
// Step 5: Add & Norm computation
```
$Output \leftarrow AddNorm(AttnOut, ExpOut)$
```
// Step 6: Output propagation
```
$Out_{MoE} \leftarrow Output$
Pass Out_{MoE} to the next pipeline stage

Algorithm 2: LPT scheduler.

Input: Number of rails N; set of local GPUs $\mathcal{G}$
Output: Path assignment $\{(w_i, j^*)\}$ for all flows; load MSE

```
// Initialization before each All-to-All
```
Initialize $LoadState[1 \dots N] \leftarrow 0$.
Initialize empty flow set $\mathcal{W} \leftarrow \emptyset$.
```
// Step 1: Flow collection from local GPUs
```
for *each GPU* $g \in \mathcal{G}$ **do**
  Receive atomic flows $\{w_{g,1}, w_{g,2}, \dots\}$ from GPU g.
  Tag each $w_{g,k}$ with source ID g.
  Append them into $\mathcal{W}$.
```
// Step 2: LPT sorting
```
Sort $\mathcal{W}$ by descending weight.
Break ties by GPU index.
```
// Step 3: Iterative allocation
```
for *each flow* $w_i \in \mathcal{W}$ **do**
  $j^* \leftarrow \arg\min_{j \in [1,N]} LoadState[j]$.
  Record mapping (w_i, j^*) into assignment table.
  Notify source GPU of w_i with selected NIC j^*.
  $LoadState[j^*] \leftarrow LoadState[j^*] + w_i$.
```
// Step 4: Spraying execution
```
for *each assignment* (w_i, j^*) **do**
  Select port p from NIC j^* by round-robin.
  Map (j^*, p) to a QP in NIC's QP set.
  Bind w_i to the chosen QP.
  Transmit w_i via transport engine.
```
// Step 5: Completion & Feedback
```
for *each flow* w_i **do**
  Await completion signal from NIC j^*.
  Confirm delivery to destination GPU.
  Update scheduler statistics with actual transmission time and throughput.
```
// Step 6: Performance evaluation (MSE)
```
Compute average load: $\mu \leftarrow \frac{1}{N} \sum_{j=1}^{N} LoadState[j]$
Compute $MSE \leftarrow \frac{1}{N} \sum_{j=1}^{N} (LoadState[j] - \mu)^2$
Log MSE for analysis and scheduler tuning.

each chunk. In our system, these generated fixed-size chunks, or small application-layer messages, are treated as indivisible atomic flows. These atomic flows serve as the fundamental units for load balancing operations.

Proactive Spraying. Current spraying strategies are often reactive, such as randomly sampling path RTTs and probabilistically steering traffic to currently lower-latency paths. While these approaches can adapt to network dynamics, they do not necessarily achieve the globally balanced load that the theoretical analysis guarantees as optimal. Our core design principle is to implement a proactive, theory-guided spraying mechanism on top of the multipath framework. Using a deterministic scheduling algorithm, the system directly computes the optimal target path for each atomic flow. The next section presents the key component that realizes the LPT scheduler.

B. LPT Scheduler

To realize the proactive, load-aware spraying strategy proposed in the previous section, we design and implement the LPT Scheduler for path selection logic. The scheduler is deployed as a lightweight software module on each source domain and serves as the core decision engine for path selection, translating the theoretical algorithm from Chapter IV into an efficient engineering practice.

Architecture and State Management. The LPT Scheduler runs independently on each sending node. Its core component is a local load state array of size N, denoted as $LoadState[N]$. Each entry $LoadState[j]$ maintains the cumulative number of bytes already assigned to the j-th communication lane (i.e., the j-th local NIC) for the current all-to-all operation. To adapt to the dynamic nature of MoE traffic, the array is reset at the start of each new all-to-all round, ensuring that scheduling decisions are always based on the current communication round's status.

Workflow and Implementation. Upon the initiation of an all-to-all communication by the upper-layer application, the generated traffic is split into a set of atomic flows $\{w_i\}$. The LPT Scheduler then takes over the path selection for these flows, following the LPT algorithm 2 strictly:

- **Flow Sorting:** The scheduler first sorts all atomic flows $\{w_i\}$ to be sent from the local node in descending order of their weights.
- **Iterative Assignment:** The scheduler then iterates over the sorted flow list. For each flow w_i, it performs the following core steps:
  - *Path Selection:* Identify the index of the path with the currently minimum cumulative load in the local $LoadState$ array $j^* = \arg\min_j LoadState[j]$.
  - *Decision Output:* Assign w_i to the selected path j^* and return this decision to transport engine.

  - *State Update:* Immediately update the local load state $LoadState[j^*] \leftarrow LoadState[j^*] + w_i$.
- **Spraying Execution:** Upon receiving the path index j^*, the transport engine transmits the atomic flow w_i through the set of QPs associated with the j^*-th lane. Repeating this process for all atomic flows completes a global, deterministic spraying, systematically distributing the node's total outbound traffic across the N parallel lanes according to the LPT-optimized allocation.

The LPT Scheduler effectively translates the theoretical model of Theorem 4 into a fully distributed low-complexity engineering implementation. Unlike naive random spraying, it performs strategic flow assignments aimed at minimizing load variance, thereby fully exploiting the multi-NIC parallelism provided by the Rail architecture, even under the irregular and sparse traffic patterns typical of MoE workloads.

VI. Evaluation

A. Experimental Setup

Simulation Setup. We implement a scalable datacenter simulation environment on Mininet 2.3.1b4 with Soft-RoCE, using a Rail-optimized topology. The network comprises multiple spine switches and at least eight fully connected leaf switches. Each computing domain connects in parallel to different leaf switches via multiple NICs, forming Rail-based links. The system scales to 128 compute domains with eight GPUs each interconnected through NVLINK. Network forwarding uses Open vSwitch with OpenFlow for fine-grained control, while a Ryu-based distributed controller installs flow rules and manages ECMP routing.

Workloads. We construct four representative workloads using MoE outputs from the SimAI AICB library [62] with controlled token and gating distributions. **Uniform Workload** sends equal data from each sender to all receivers, yielding balanced link loads without "elephant" or "mice" flows, and serves to measure the gap between achieved and optimal bandwidth utilization. **Sparse Workload** applies a Top-K expert selection matrix with column-wise sparsity, where higher sparsity increases load concentration as fewer active experts handle proportionally more traffic. **Skewed Workload** models Zipfian-driven imbalance in two forms: receiver skew, where many senders target a few experts (incast), and sender skew, where uneven input causes asymmetric expert loads. **Real MoE Workload** replays traffic traces from MixNet [39] to reproduce realistic communication patterns. Table I summarizes the corresponding distributions.

Evaluation Metrics. We evaluate performance using some key metrics. **Collective Completion Time (CCT)** measures MoE all-to-all efficiency across average, 80th, 95th, and 99th percentiles, with the 99th percentile approximating total transfer completion. **Bus Bandwidth (BusBw)** quantifies effective link utilization and enables direct comparison with peak hardware bandwidth to assess algorithmic efficiency. **NIC Transmission/Reception Volume** captures per-NIC send and receive traffic via a matrix representation, assessing load balance across interfaces. **Normalized MSE of Load** evaluates intra-domain NIC load balance on a 0–1 scale, where 0 denotes perfect uniformity. **Iteration Time** measures the total duration of one training iteration.

TABLE I: The MoE workloads' token and gating distributions.

Type / Distribution	**Uniform**	**Sparse**	**Sender-skewed**	**Receiver-skewed**	**Real workload**
Token input	Uniform	Uniform	Zipf	Uniform	Uniform
Gating	Uniform	Top-K	Uniform	Zipf	Training -based

Baseline Methods. We propose the **RailS** and compare it with four baseline methods: **ECMP** [25] is a widely used equal-cost multipath routing mechanism in datacenters that binds flows to a single path via hashing, providing traditional load balancing and serving as a performance benchmark. **MinRTT** [46] is a multipath scheduling algorithm based on RTT measurements. It selects the subflow with the minimum RTT to transmit packets, enabling multipath bandwidth aggregation and load balancing. MinRTT was first implemented in MPTCP. **PLB** [49] is a flowlet-based load balancing scheme that uses IPv6 flow label to dynamically switch paths during idle periods, balancing loads while minimizing packet reordering. **REPS** [9] is a per-packet spraying load balancing algorithm that minimally manages state to reroute packets around congestion hotspots and unreliable links.

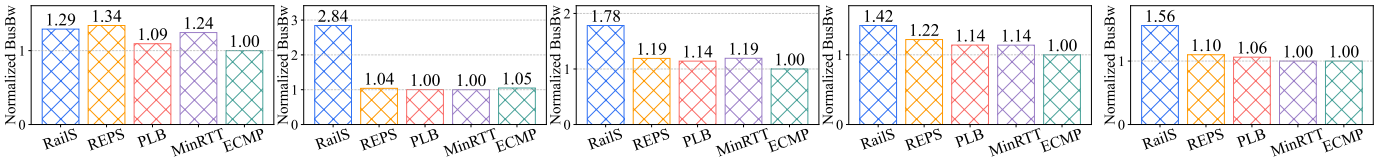
B. Performance under Uniform Load

The uniform-load scenario simulates an ideal condition in which each sender transmits an equal amount of data to all receivers, resulting in balanced link utilization without the presence of "elephant" or "mice" flows.

BusBw. Fig. 7a shows the normalized BusBw of different schemes relative to ECMP under the uniform-load scenario. RailS achieves a normalized BusBw of 1.29, representing an improvement of approximately 29% over ECMP, demonstrating its superior bandwidth utilization. REPS and MinRTT achieve normalized bandwidths of 1.34 and 1.24, respectively, which are close to that of RailS. PLB's normalized BusBw is 20% lower than that of RailS, indicating relatively average performance.

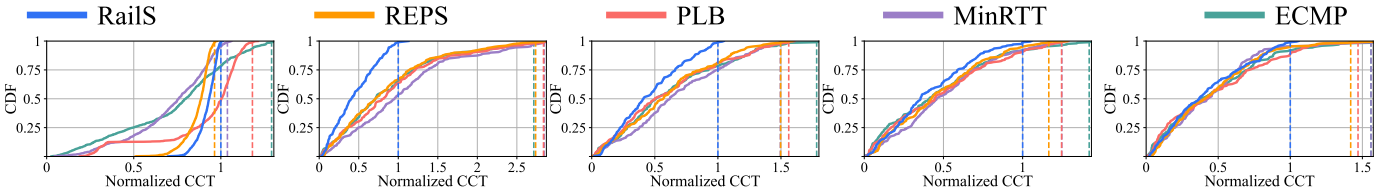
CDF of CCT. Fig. 8a illustrates the CDF of normalized CCT for different schemes relative to RailS under the uniform-load scenario. Specifically, RailS and REPS exhibit similar and concentrated trends, reflecting excellent load-balancing capability. MinRTT completes some flows faster, but its overall completion time is slightly worse than RailS. PLB follows, showing better performance than ECMP.

CCT Values. Fig. 9a further presents the mean, 80th, 95th, and 99th percentile values of normalized CCT for different schemes under uniform load. RailS demonstrates very close mean and tail percentiles, indicating good load-balancing capability. REPS performs consistently with RailS. Although MinRTT achieves the lowest mean CCT, its overall completion time is 4% worse than RailS. The tail flows of PLB finish relatively simultaneously, with an overall completion time 18% longer than RailS but still 11% better than the worst-performing ECMP.



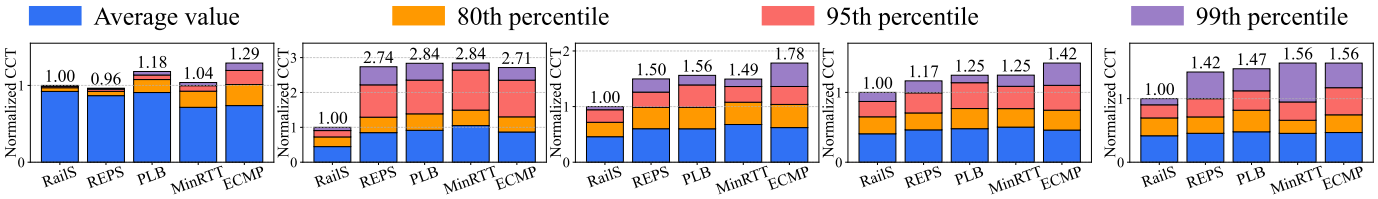
(a) Uniform (b) Sparse-0.6 (c) Sparse-0.4 (d) Sparse-0.2 (e) Sparse-0 (Fully dense)

Fig. 7: Normalized BusBw of different schemes under various all-to-all workloads: (a) Uniform matrix, (b) Sparse-0.6, (c) Sparse-0.4, (d) Sparse-0.2, and (e) Sparse-0 (fully dense).



(a) Uniform (b) Sparse-0.6 (c) Sparse-0.4 (d) Sparse-0.2 (e) Sparse-0 (Fully dense)

Fig. 8: CDF of normalized CCT for different schemes under various all-to-all workloads: (a) Uniform matrix, (b) Sparse-0.6, (c) Sparse-0.4, (d) Sparse-0.2, and (e) Sparse-0 (fully dense).



(a) Uniform (b) Sparse-0.6 (c) Sparse-0.4 (d) Sparse-0.2 (e) Sparse-0 (Fully dense)

Fig. 9: Normalized CCT of different schemes under various all-to-all workloads, showing average, 80th, 95th, and 99th percentiles: (a) Uniform matrix, (b) Sparse-0.6, (c) Sparse-0.4, (d) Sparse-0.2, and (e) Sparse-0 (fully dense).

C. Performance under Sparse Load

The sparse-load scenario is constructed using a Top-K ($K = 2$) expert selection matrix to simulate non-uniform communication patterns. By adjusting the sparsity, only a subset of receivers participates in data processing, forming hotspot experts and unbalanced link loads. This setup is used to evaluate the load-balancing capability of different schemes under non-ideal conditions.

BusBw. Fig. 7b, 7c, 7d and 7e show normalized BusBw of different schemes under various sparsity levels (0.6, 0.4, 0.2, and fully dense). RailS consistently maintains the highest BusBw across all sparsity levels, and its advantage becomes more pronounced as the load becomes sparser. At a sparsity of 0.6, RailS achieves a BusBw 2.8× higher than other schemes. Under other sparsity levels, RailS still achieves 20%–78% performance improvements, indicating that RailS fully utilizes the multi-path bandwidth of the Rail architecture to maintain high BusBw.

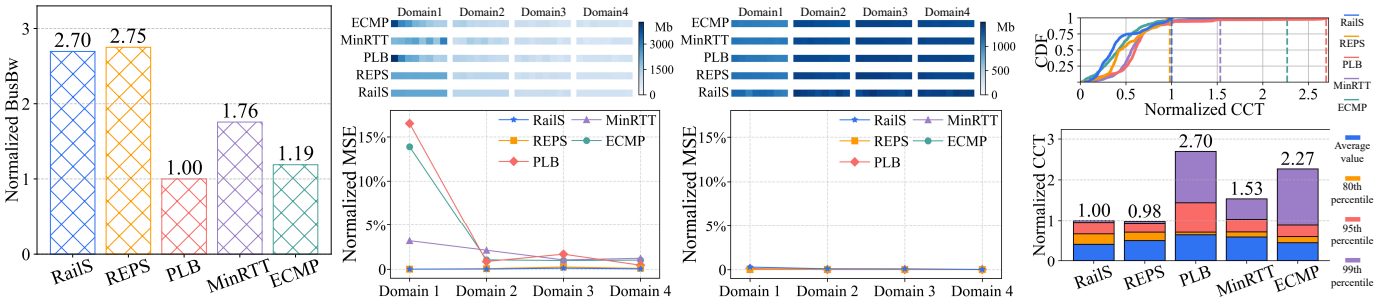
CDF of CCT. Fig. 8b, 8c, 8d and 8e illustrate the CDF of normalized CCT for different schemes under sparse load. It is evident that as sparsity decreases, RailS completes flows faster and with greater advantage. With increasing sparsity, the short-flow performance of RailS becomes closer to other schemes, yet the longest flows still complete the fastest. This demonstrates that RailS achieves optimal load balancing and maintains excellent completion time performance.

CCT Values. Fig. 9b, 9c, 9d and 9e further present the mean, 80th, 95th, and 99th percentile values of normalized CCT for different schemes under sparse load. At low sparsity, RailS maintains closely aligned tail flow completion times. As sparsity increases, the range of tail completion times expands. Compared with other schemes, RailS consistently achieves the best mean and percentile performance in normalized CCT. At a sparsity of 0.6, RailS improves total completion time by more than 1.5×. With higher sparsity levels, the improvement remains within 17%–78%, demonstrating the superior load-balancing performance of RailS.

D. Performance under Skewed Sender Load

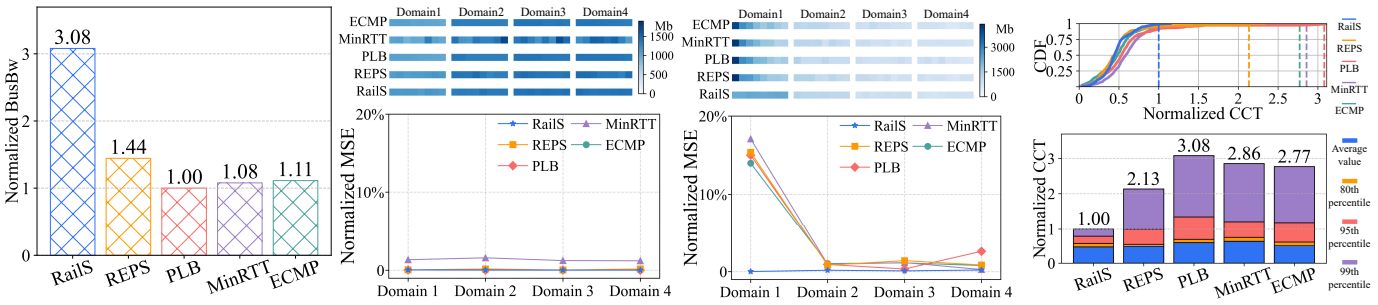
The sender-skewed scenario employs a Zipf distribution to make a small number of domain experts carry the majority of traffic, forming "hotspot senders." This simulates a situation where non-uniform input leads to extremely unbalanced outgoing loads among experts. It is used to examine each scheme's bandwidth utilization and congestion mitigation capability under sudden source-side pressure.

BusBw. Fig. 10a shows that RailS and REPS achieve BusBw 2.7× and 2.75× higher than PLB, respectively. RailS



(a) Normalized BusBw (b) Sender load matrix and MSE (c) Receiver load matrix and MSE (d) Normalized CCT

Fig. 10: Performance of different schemes under sender-skewed workloads: (a) normalized BusBw, (b) sender load matrix and its MSE, (c) receiver load matrix and its MSE, and (d) normalized CCT.



(a) Normalized BusBw (b) Sender load matrix and MSE (c) Receiver load matrix and MSE (d) Normalized CCT

Fig. 11: Performance of different schemes under receiver-skewed workloads: (a) normalized BusBw, (b) sender load matrix and its MSE, (c) receiver load matrix and its MSE, and (d) normalized CCT.

improves bandwidth by 126% compared with ECMP and by 58% compared with MinRTT, demonstrating the effectiveness of the spraying mechanism under sender-skewed load.

Sender Load Matrix and MSE. The heatmap in Fig. 10b shows that RailS and REPS maintain balanced traffic distribution across NICs within each domain, with an inter-domain MSE below 0.01, indicating excellent sender load balancing. MinRTT shows moderate balance with an MSE of about 0.03, while PLB and ECMP have the highest MSE, around 15%.

Receiver Load Matrix and MSE. Fig. 10c shows that although the sending side is skewed, the receiving traffic distribution remains uniform. As a result, the actual NIC-level receiving load of all schemes is balanced, and the bottleneck lies on the sender side.

CCT. As shown in Fig. 10d, the CDF curves indicate that RailS and REPS achieve the best CCT performance, reducing completion time by 55.9%–62.9% compared with ECMP and PLB, and by 34.6% compared with MinRTT.

In summary, RailS and REPS effectively mitigate sender-side bottlenecks under skewed load through the spraying mechanism, achieving balanced transmission. MinRTT benefits from multipath utilization and thus outperforms PLB and ECMP. However, since the bottleneck is not within the network and PLB cannot exploit the Rail architecture's advantages, its performance is the lowest.

E. Performance under Skewed Receiver Load

The receiver-skewed scenario models an *incast* situation, where most senders concentrate on a few hot experts following a Zipf distribution, leading to sudden many-to-one traffic bursts. This pattern reflects the real training condition in which "hot experts" are frequently invoked. It is used to evaluate each scheme's ability to suppress tail latency and maintain load balance when the target-side load increases sharply.

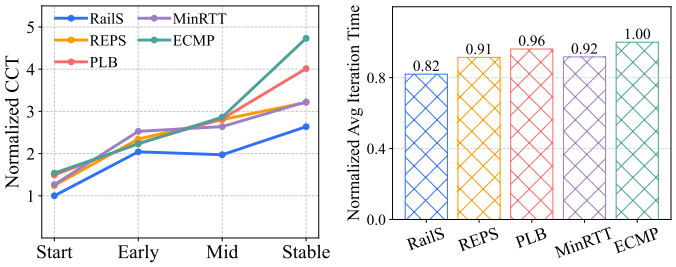
BusBw. As shown in Fig. 11a, RailS achieves 3.08× normalized BusBw, representing a 2× improvement over ECMP, MinRTT, and PLB, significantly alleviating receiver hotspots. REPS reaches only 1.44× that of PLB, less than half of RailS.

Sender Load Matrix and MSE. Fig. 11b shows that although the receiver load is skewed, the sending traffic distribution remains uniform. Therefore, all schemes exhibit balanced NIC-level sending and receiving loads, with the bottleneck located on the receiver side.

Receiver Load Matrix and MSE. Fig. 11c demonstrates that RailS maintains balanced receiving traffic across domains, with an MSE below 0.01, while all other schemes have MSE values around 15%. The heatmap clearly shows that although Domain 1 experiences a strong receiving skew, RailS effectively evens it out. This indicates that RailS achieves simultaneous load balancing on both sender and receiver sides.

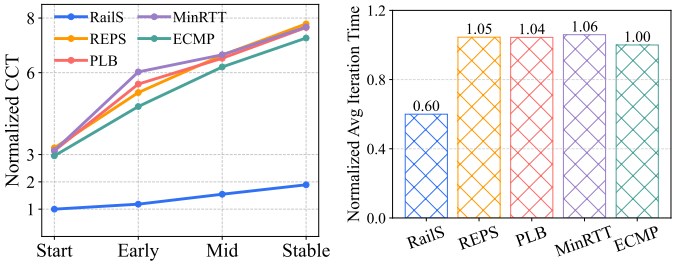
CCT. As shown in Fig. 11d, RailS achieves the best CCT performance. Compared with other schemes, RailS reduces completion time by more than 50%, benefiting from balanced NIC utilization on the receiver side, which other schemes fail to achieve.

In summary, although RailS also employs a spraying mechanism, its uniform sending pattern naturally induces uniform receiving distribution, making it significantly superior to REPS.



(a) Various phases for CCT (b) Training iteration time

Fig. 12: The Mixtral 8×7B model under dense mode exhibits the following metrics: (a) all-to-all completion time and (b) training iteration time.



(a) Various phases for CCT (b) Training iteration time

Fig. 13: The Mixtral 8×7B model under sparse mode exhibits the following metrics: (a) all-to-all completion time and (b) training iteration time.

Since the bottleneck lies on the single receiver side rather than within the network, PLB and MinRTT even perform worse than ECMP, as ECMP avoids unnecessary path reorganization in this case.

F. Real-World Workload Performance

We replayed the training and communication emulation of the Mixtral 8×7B MoE training trace. Each expert was deployed across multiple GPUs. The original data to be transmitted by each expert were distributed on GPUs in two different ways to perform all-to-all communication. In the **dense** setup, each expert's original data were evenly distributed across multiple GPUs for parallel data exchange. In the **sparse** setup, the original data were aggregated on a single GPU, and the all-to-all exchange was performed on that GPU.

During training, four phases of the iteration process were identified: **Start** denotes the first iteration, **Early** represents the early stage, **Mid** refers to the middle stage, and **Stable** indicates the stable stage. The input distribution on each GPU remained uniform, while the total data volume increased as the iteration progressed. For example, in the Start phase, the total transmission volume per expert was around 100 MB, whereas in the Stable phase, it reached 256 MB.

Dense Setup. In Fig. 12, the normalized CCT analysis indicates that RailS reduces the CCT by 19.5%–34.9% in the Start phase, 8.9%–19.2% in the Early phase, 25.3%–31.2% in the Mid phase, and 17.9%–44.3% in the Stable phase. Overall, RailS shortens the iteration time by up to 18% compared with other schemes.

Sparse Setup. In Fig. 13, the normalized CCT shows that RailS reduces the CCT by 66.1%–69.2% in the Start phase, 75.1%–80.4% in the Early phase, 75.1%–76.8% in the Mid phase, and 73.9%–75.7% in the Stable phase. Throughout the entire training process, RailS achieves at least a 40% reduction in iteration time compared with other schemes.

These results demonstrate that the sparser the load matrix, the better the performance of the RailS scheme. This is because, under sparse workloads, bottlenecks tend to occur at the receiver side of the Rail architecture rather than within the network. Although some schemes can exploit multipath transmission, they fail to leverage parallel reception, thereby creating receiver-side bottlenecks and performing worse than ECMP. The uniform-sending and uniform-receiving characteristics of RailS fully utilize architectural parallelism and ensure NIC load balancing, achieving optimal performance.

VII. Related Work

The core objective of load balancing is to efficiently distribute network traffic across multiple paths to enhance system throughput and resource utilization. Existing schemes exhibit significant differences in traffic granularity. Traffic granularity can range from individual packets [9], [22], [70], [33] to flowlets [6], [59], [61], [29], to subflows [51], [46], [41], [38] within a single connection, and up to entire connections or flows [25], [69], [5]. The choice of granularity impacts scheduling accuracy and load balancing, with fine-grained partitioning improving allocation precision and coarse-grained partitioning reducing overhead.

Local load balancing distributes traffic across equivalent paths and can be implemented at the host or switch level. Host-level schemes [49], [31], [52], [66] offer greater flexibility, easier scaling, and faster adaptation to network changes, supporting multipath scheduling and improving system performance. Switch-level [7], [11], [55], [8] schemes react quickly to transient congestion and reduce host overhead but provide limited flexibility for policy customization and scaling.

Traffic forwarding decisions in load balancing rely on network state information. Many schemes utilize connectivity information to identify currently available equivalent paths. Significant differences exist in how load information is applied. Some schemes perform static allocation without relying on load information [24], [13], [45]. Others dynamically adjust traffic distribution based on local load conditions to optimize link utilization [65], [30], [57]. Certain schemes employ global load information for unified scheduling to achieve overall performance optimization [47], [32]. Recent works further extend load balancing to reconfigurable data center networks, addressing dynamic topologies and time-varying link conditions [37], [36], [35]. These differences determine the performance of load balancing schemes in terms of throughput, latency, and network resource utilization.

VIII. Conclusion

Driven by the sparse, dynamic, and imbalanced communication of MoE models, we propose RailS, a load balancer for MoE all-to-all communication. RailS exploits the deterministic

topology and symmetry of the Rail architecture to resolve path conflicts and structure multi-NIC resource allocation. Theoretically, we prove that sender-side balancing inherently ensures receiver-side balance, reducing global coordination to locally solvable subproblems. System-wise, an LPT-based scheduler efficiently realizes this principle, achieving near-optimal performance through proactive, fine-grained traffic planning. Experiments demonstrate that RailS enhances bus bandwidth, shortens collective communication and training iteration times, and improves overall GPU and network utilization. We hope this work inspires future exploration of topology-aware load balancing and transport co-design, encouraging broader investigation into efficient and scalable communication for large MoE systems.

REFERENCES

[1] NVIDIA Collective Communications Library (NCCL). https://github.com/NVIDIA/nccl.

[2] Infiniband architecture specification release 1.2.1 annex a16: Roce. *InfiniBand Trade Association*, 2010.

[3] Infiniband architecture specification release 1.2.1 annex a17: Rocev2. *InfiniBand Trade Association*, 2014.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[5] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, et al. Hedera: dynamic flow scheduling for data center networks. In *Nsdi*, volume 10, pages 89–92. San Jose, USA, 2010.

[6] Mohammad Alizadeh, Tom Edsall, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, 2014.

[7] Mohammad Alizadeh, Shuang Yang, Milad Sharif, et al. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

[8] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Microte: Fine grained traffic engineering for data centers. In *Proceedings of the seventh conference on emerging networking experiments and technologies*, pages 1–12, 2011.

[9] Tommaso Bonato, Abdul Kabbani, Ahmad Ghalayini, et al. Reps: Recycled entropy packet spraying for adaptive load balancing and failure mitigation. *arXiv preprint arXiv:2407.21625*, 2025.

[10] Tom Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. dcpim: Near-optimal proactive datacenter transport. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 53–65, 2022.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[13] Advait Dixit, Pawan Prakash, Y Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in data center networks. In *2013 proceedings ieee infocom*, pages 2130–2138. IEEE, 2013.

[14] Nan Du, Yanping Huang, Andrew M Dai, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pages 5547–5569. PMLR, 2022.

[15] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[16] Adithya Gangidi, Rui Miao, Shengbao Zheng, et al. Rdma over ethernet for distributed training at meta scale. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 57–70, 2024.

[17] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2):416–429, 1969.

[18] Albert Greenberg, James R Hamilton, Navendu Jain, et al. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 51–62, 2009.

[19] Chuanxiong Guo, Guohan Lu, Dan Li, et al. Bcube: a high performance, server-centric network architecture for modular data centers. SIGCOMM '09, page 63–74, New York, NY, USA, 2009. Association for Computing Machinery.

[20] Chuanxiong Guo, Haitao Wu, Kun Tan, et al. Dcell: a scalable and fault-tolerant network structure for data centers. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, page 75–86, New York, NY, USA, 2008. Association for Computing Machinery.

[21] Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[22] Mark Handley, Costin Raiciu, Alexandru Agache, et al. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 29–42, 2017.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Keqiang He, Eric Rozner, Kanak Agarwal, et al. Presto: Edge-based load balancing for fast datacenter networks. *ACM SIGCOMM Computer Communication Review*, 45(4):465–478, 2015.

[25] Christian Hopps. Analysis of an equal-cost multi-path algorithm. Technical report, 2000.

[26] Jinbin Hu, Chaoliang Zeng, Zilong Wang, et al. Load balancing with multi-level signals for lossless datacenter networks. *IEEE/ACM Transactions on Networking*, 32(3):2736–2748, 2024.

[27] Shuihai Hu, Yibo Zhu, Peng Cheng, et al. Deadlocks in datacenter networks: Why do they form, and how to avoid them. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 92–98, 2016.

[28] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[29] Abdul Kabbani, Balajee Vamanan, Jahangir Hasan, and Fabien Duchene. Flowbender: Flow-level adaptive routing for improved latency and throughput in datacenter networks. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 149–160, 2014.

[30] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 37(2):51–62, 2007.

[31] Naga Katta, Aditi Ghag, Mukesh Hira, et al. Clove: Congestion-aware load balancing at the virtual edge. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 323–335, 2017.

[32] Naga Katta, Mukesh Hira, Changhoon Kim, et al. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, pages 1–12, 2016.

[33] Yanfang Le, Rong Pan, Peter Newman, et al. Strack: A reliable multipath transport for ai/ml clusters. *arXiv preprint arXiv:2407.15266*, 2024.

[34] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, et al. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

[35] Jialong Li, Federico De Marchi, Yiming Lei, Raj Joshi, Balakrishnan Chandrasekaran, and Yiting Xia. Unlocking diversity of fast-switched optical data center networks with unified routing. *IEEE Transactions on Networking*, 2025.

[36] Jialong Li, Haotian Gong, Federico De Marchi, Aoyu Gong, Yiming Lei, Wei Bai, and Yiting Xia. Uniform-cost multi-path routing for reconfigurable data center networks. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 433–448, 2024.

[37] Jialong Li, Yiming Lei, Federico De Marchi, Raj Joshi, Balakrishnan Chandrasekaran, and Yiting Xia. Hop-on hop-off routing: A fast tour across the optical data center network for latency-sensitive flows. In *Proceedings of the 6th Asia-Pacific Workshop on Networking*, pages 63–69, 2022.

[38] Jialong Li, Shreyansh Tripathi, Lakshay Rastogi, Yiming Lei, Rui Pan, and Yiting Xia. Optimizing mixture-of-experts inference time combining model deployment and communication scheduling. *arXiv preprint arXiv:2410.17043*, 2024.

[39] Xudong Liao, Yijun Sun, Han Tian, et al. Mixnet: A runtime reconfigurable optical-electrical fabric for distributed mixture-of-experts training. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 554–574, 2025.

[40] Aixin Liu, Bei Feng, Bing Xue, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[41] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, et al. {Multi-Path} transport for {RDMA} in datacenters. In *15th USENIX symposium on networked systems design and implementation (NSDI 18)*, pages 357–371, 2018.

[42] Radhika Mittal, Alexander Shpiner, Aurojit Panda, et al. Revisiting network support for rdma. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 313–326, 2018.

[43] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2002.

[44] DGX NVIDIA. SuperPOD: Next Generation Scalable Infrastructure for AI Leadership. https://docs.nvidia.com/https:/docs.nvidia.com/dgx-superpod-reference-architecture-dgx-h100.pdf, 2023.

[45] Vladimir Olteanu, Haggai Eran, Dragos Dumitrescu, et al. An edge-queued datagram service for all datacenter traffic. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 761–777, 2022.

[46] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pages 27–32, 2014.

[47] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, et al. Fastpass: A centralized" zero-queue" datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 307–318, 2014.

[48] Kun Qian, Yongqing Xi, Jiamin Cao, et al. Alibaba hpn: A data center network for large language model training. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 691–706, 2024.

[49] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, et al. Plb: congestion signals are simple and effective for network load balancing. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 207–218, 2022.

[50] Costin Raiciu, Sebastien Barre, Christopher Pluntke, et al. Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):266–277, 2011.

[51] Costin Raiciu, Dragos Niculescu, Marcelo Bagnulo, and Mark James Handley. Opportunistic mobility with multipath tcp. In *Proceedings of the sixth international workshop on MobiArch*, pages 7–12, 2011.

[52] Costin Raiciu, Christoph Paasch, Sebastien Barre, et al. How hard can it be? designing and implementing a deployable multipath {TCP}. In *9th USENIX symposium on networked systems design and implementation (NSDI 12)*, pages 399–412, 2012.

[53] Samyam Rajbhandari, Conglong Li, Zhewei Yao, et al. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR, 2022.

[54] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification. Technical report, 2007.

[55] Siddhartha Sen, David Shue, Sunghwan Ihm, and Michael J Freedman. Scalable, optimal flow routing in datacenters via local link balancing. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 151–162, 2013.

[56] Arjun Singh, Joon Ong, Amit Agarwal, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. SIGCOMM '15, page 183–197, New York, NY, USA, 2015. Association for Computing Machinery.

[57] Cha Hwan Song, Xin Zhe Khooi, Raj Joshi, et al. Network load balancing with in-network reordering support for rdma. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 816–831, 2023.

[58] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[59] Erico Vanini, Rong Pan, Mohammad Alizadeh, et al. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 407–420, 2017.

[60] Ying Wan, Haoyu Song, Yu Jia, et al. Laps: Joint load balancing and congestion control on unequal-cost multi-path data center networks. In *Proceedings of the 2nd Workshop on Networks for AI Computing*, pages 11–18, 2025.

[61] Peng Wang, Hong Xu, Zhixiong Niu, et al. Expeditus: Congestion-aware load balancing in clos data center networks. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 442–455, 2016.

[62] Xizheng Wang, Qingxu Li, Yichi Xu, et al. {SimAI}: Unifying architecture design and performance tuning for {Large-Scale} large language model training with scalability and precision. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 541–558, 2025.

[63] An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[64] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[65] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. Detail: Reducing the flow completion time tail in datacenter networks. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 139–150, 2012.

[66] Hong Zhang, Junxue Zhang, Wei Bai, et al. Resilient datacenter load balancing in the wild. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2017.

[67] Junxue Zhang, Wei Bai, and Kai Chen. Enabling ecn for datacenter networks with rtt variations. In *Proceedings of the 15th international conference on emerging networking experiments and technologies*, pages 233–245, 2019.

[68] Chenggang Zhao, Shangyan Zhou, Liyue Zhang, et al. Deepep: an efficient expert-parallel communication library. *https://github.com/deepseek-ai/DeepEP*, 2025.

[69] Junlan Zhou, Malveeka Tewari, Min Zhu, et al. Wcmp: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14, 2014.

[70] Yang Zhou, Zhongjie Chen, , et al. An extensible software transport layer for gpu networking. *arXiv preprint arXiv:2504.17307*, 2025.