SpiderFlow: Efficient Topology-Aware Scheduling for LLM Training Across Decentralized GPU Clusters

Zihan Chang[12], **Shuibing He**[12*], **Bo Zhou**[12], **Sheng Xiao**[1], **Siling Yang**[1], **Rui Wang**[1], **Zhe Pan**[1]
[1]Zhejiang University, Hangzhou, China
[2]Zhejiang Lab, Hangzhou, China

Abstract

In response to increasing demands for large-scale machine learning training jobs, many organizations have deployed GPU clusters across geographically distributed regions. However, existing Integer Linear Programming (ILP)- or genetic-based cross-cluster training approaches largely overlook the topology of decentralized clusters, lacking both topology-aware task scheduling mechanisms and automated model parallelization strategies. As a result, naively applying these optimization-based methods in cross-cluster settings leads to prohibitive scheduling overhead, due to the drastically enlarged search space induced by complex inter-cluster topologies. To address these challenges, we propose SpiderFlow, a topology-aware scheduling system specifically designed for decentralized GPU clusters. We formulate cross-cluster task scheduling as a graph optimization problem and introduce SpinSearch, a low-overhead topology-aware scheduling algorithm. In addition, for automated model parallelization, we propose Topology-aware Parallelism Automation (TPA), a two-level scheduling framework that combines heuristic methods at the inter-cluster level with ILP-based optimization within clusters, effectively reducing the search space while maintaining high training throughput with substantially lower scheduling overhead. We evaluate SpiderFlow on a physical platform comprising 8 decentralized clusters, as well as on a simulation platform with up to 64 decentralized clusters. Experimental results demonstrate that SpiderFlow reduces job completion time (JCT) by 1.2-1.3×, improves throughput by 1.12-1.25×, and reduces scheduling overhead by 20-90× on average compared to state-of-the-art scheduling systems.

1 Introduction

With the rapid growth in the parameter scale of large language models (Kaddour et al., 2023; Zhao et al., 2024), training these models requires substantial GPU resources. When a single cluster cannot provide sufficient capacity, many organizations (Regions, 2024a,b,c) leverage GPU resources across multiple clusters to support LLM training.

Existing approaches can be broadly categorized into two classes. (1) *Task-level scheduling*. Lyra (Li et al., 2023) focuses on capacity lending from serving clusters to support cross-cluster training, but does not address model parallelism. Mast (Choudhury et al., 2024) enables global scheduling across clusters; however, it allocates workloads to individual clusters and does not leverage cross-cluster resources for training a single model. (2) *Automated model parallelization*. DT-FM (Yuan et al., 2022) and ALPA (Zheng et al., 2022) automate model parallelism using genetic and Integer Linear Programming (ILP) algorithms, respectively, but do not consider task-level scheduling. Overall, none of the above approaches conducts an in-depth analysis of cross-cluster topology in LLM training, nor do they simultaneously support topology-aware task-level scheduling and automated model parallelization.

We observe that the efficiency of LLM training is primarily determined by two factors: (1) the number of clusters spanned by a training job and (2) the configuration of model parallelism (Figure 1). Based on these observations, we propose SpiderFlow, a topology-aware decentralized training approach that minimizes the number of clusters involved in training jobs under multi-task workloads while enabling efficient automated model parallelization. However, compared to data-center environments, decentralized training exhibits significantly more complex topologies, which dramatically enlarge the scheduling search space; as a result, directly applying ILP- or genetic-algorithm-based solutions incurs prohibitive scheduling latency. Consequently, implementing SpiderFlow

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13603–13615

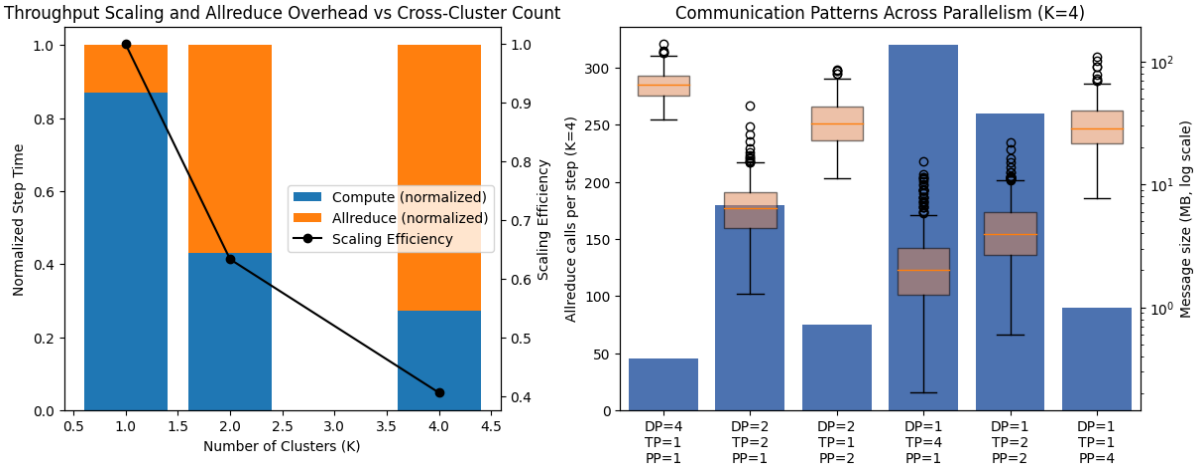

Figure 1: *(Left)* Throughput scaling efficiency and normalized step-time breakdown as the number of clusters increases when training GPT-2 1.3B with batch size = 1, using a fixed configuration of four A100 GPUs. Computation remains stable, while allreduce rapidly dominates the step time, indicating a transition to a communication-bound regime. *(Right)* Communication patterns of all valid parallel configurations at K=4 (4 decentralized A100 GPUs) when training GPT-2 1.3B with batch size = 8.

presents two key challenges: (1) designing an efficient scheduling strategy in multi-task scenarios that minimizes the number of clusters spanned by each training job, and (2) automatically configuring model parallelism to achieve high training throughput while keeping the automation overhead low. To address the first challenge, we design SpinSearch, which formulates the scheduling problem as a graph optimization task and employs an efficient topology-aware algorithm to minimize cross-cluster hops. To address the second challenge, we develop Topology-aware Parallelism Automation (TPA) framework, which combines heuristic scheduling at the inter-cluster level with ILP-based optimization within clusters, thereby sustaining high model throughput while significantly reducing the overall search overhead.

We validated the effectiveness of SpiderFlow using 8 geographically distributed servers with varying numbers of A100 GPUs. We also developed a simulator on the PAI platform (Alibaba, 2024) for larger-scale experiments. Experimental results show that SpiderFlow reduces JCT by 1.2–1.3×, improves throughput by 1.12–1.25×, and lowers scheduling overhead by 20–90× on average compared to state-of-the-art systems.

2 Background and Motivation

2.1 Analysis of existing methods

Traditional deep learning task scheduling systems (Bao et al., 2019; Chang et al., 2018, 2019; Gao et al., 2021; Gu et al., 2019; Hwang et al., 2021; Mahajan et al., 2020; Mohan et al., 2022; Qiao et al., 2021) primarily focus on resource management and task allocation within a single cluster environment, aiming to optimize the efficiency of computing resource usage and accelerate the model training process. However, as the complexity of models continues to increase, a single computing cluster often struggles to meet the demands of large-scale model training. Consequently, recent research has begun to explore how multiple clusters can collaborate to complete training.

Lyra (Li et al., 2023) focuses on capacity lending from serving clusters to support cross-cluster training, but does not address model parallelism. Mast (Choudhury et al., 2024) enables global scheduling across clusters; however, it allocates workloads to individual clusters and does not leverage cross-cluster resources for training a single model. In contrast, DT-FM (Yuan et al., 2022) and ALPA (Zheng et al., 2022) automate model parallelism using genetic algorithms and ILP, respectively, but do not consider task-level scheduling. Overall, none of these approaches analyzes how cross-cluster topology induces fundamental training bottlenecks for LLMs, nor do they jointly address topology-aware task-level scheduling and automated model parallelization.

2.2 Opportunities

2.2.1 Observations:

(1) **Cross-cluster count (K) is the primary factor driving performance degradation.** As illustrated in left panel of Figure 1, we evaluate cross-cluster training using GPT-2 1.3B with batch size = 1 un-

der identical model configuration and hardware settings, while varying the number of clusters K. As K increases from 1 to 4, the training throughput drops by more than 2×, even though the computation time per step remains largely unchanged. In contrast, the proportion of time spent in allreduce grows sharply, increasing from approximately 13% at K=1 to over 70% at K=4, indicating that allreduce communication becomes the dominant bottleneck as K increases. (2) **Parallelization strategy significantly affects communication efficiency across clusters.** As shown in right panel of Figure 1, we evaluate different parallelization strategies when training GPT-2 1.3B under a fixed cross-cluster configuration with K=4 and the same global parallelism. We find that the choice of parallelization strategy significantly affects communication efficiency by altering both the number of allreduce operations and the granularity of communicated messages. In particular, tensor-parallel configurations introduce substantially higher synchronization overhead, with more frequent allreduce calls and smaller message sizes, making them inefficient in cross-cluster settings; therefore, tensor parallelism is set to 1 by default in this work.

2.2.2 Our solutions:

To address the limitations of existing approaches, first, we propose SpinSearch, a topology-aware scheduling module that minimizes the number of clusters K spanned by a training job through compact, topology-aware placement. Second, we introduce a lightweight TPA module, which automatically determines parallelization configurations and GPU assignments based on cross-cluster topology, enabling efficient and fully automated training without manual tuning.

2.3 Scheduling as Graph Optimization

The scheduling strategy for multiple LLM training tasks is the most direct factor affecting the number of clusters K spanned during training. Selecting the optimal scheduling plan from all available cross-cluster resources is essentially a graph optimization problem.

For instance, consider the scenario depicted in Figure 2, where we have seven independent clusters, labeled from *a* to *g*, with black numbers above each cluster indicating the number of available GPUs. These clusters are interconnected via the network (represented by edges), with red numbers on the edges denoting network latency—smaller numbers

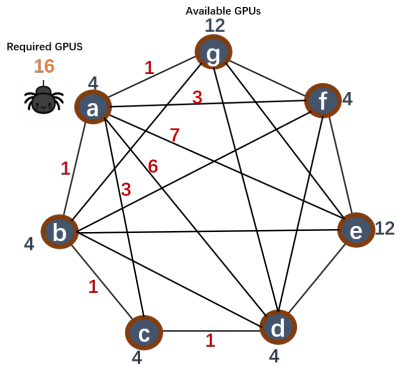

Figure 2: When a task (represented as a spider) arrives, the scheduling objective is to identify the minimal set of clusters that can satisfy the task's resource requirements. Corresponding to Table 1.

Table 1: Different crawling patterns of the spider correspond to different numbers of crossed clusters and varying crawling distances (network latency).

No.	trajectory	Nodes (clusters)	Distance(Latency)
1	a(4),b(4),c(4),d(4)	4	3
2	a(4),e(12)	2	7
3	a(4),g(12)	2	1

imply lower latency and higher bandwidth. Suppose that a cluster receives a task (known as spider) that requires 16 GPU cards, which is then processed by the SpinSearch module. Our goal is to design an efficient scheduling algorithm that minimizes the number of clusters traversed (denoted as K) by spider and the total distance traveled.

Table 1 outlines three possible crawling paths for spider. The first path covers clusters *a*, *b*, *c*, and *d*, with K =4 and a total latency of 3; the second path includes only clusters *a* and *e*, with K = 2 and a total latency of 7; the third path also involves clusters *a* and *g*, maintaining K = 2 and a total latency of 1. The third path delivers the highest performance.

2.4 Challenge

2.4.1 The challenge of scheduling strategies

Despite its intuitive formulation, identifying the optimal cross-cluster scheduling strategy is fundamentally challenging. First, the search space grows combinatorially with the number of clusters and heterogeneous resource configurations, making exhaustive exploration intractable. Second, cluster interconnections exhibit highly irregular topologies with non-uniform bandwidth and latency, which breaks the assumptions of traditional homogeneous or fully-connected settings. Third, minimizing the number of spanned clusters (K) and communication cost often leads to conflicting objectives, requiring careful trade-offs between locality and

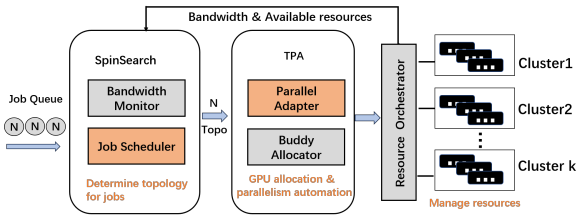

Figure 3: SpiderFlow architecture.

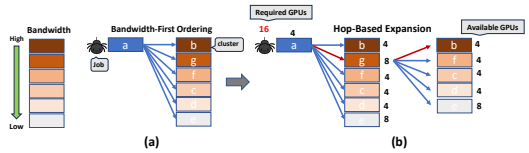

Figure 4: Example for SpinSearch: (a) Bandwidth monitor sort the clusters by bandwidth from high to low (b) Search for the minimum number of clusters that satisfy the task's resource requirements via hop expansion.

resource sufficiency. Finally, the dynamic nature of multi-tenant environments further complicates scheduling decisions, as resource availability and network conditions may change over time. These challenges make it difficult to design a scheduling algorithm that is both efficient and robust in practice.

2.4.2 The challenge of Parallelism Automation

In cross-domain multi-cluster training, the composition of network and computational resources is significantly more complex than that in datacenter-based training, resulting in a search space that is an order of magnitude larger. Although traditional ILP-based parallelization automation methods achieve effective parallel configurations in datacenter settings, directly applying them to cross-cluster training incurs prohibitively high scheduling overhead (as detailed in Table 2). Consequently, designing a low-overhead approach that can still obtain optimal parallelization configurations becomes a key challenge.

3 SpiderFlow overview

3.1 Architecture

The overall architecture of SpiderFlow is illustrated in Figure 3. The Resource Orchestrator, deployed on the head node, indexes k decentralized clusters, collects available GPU resources and inter-cluster bandwidth information, and constructs a graph-structured state information table that is continuously updated and accessible across all clusters. Upon task submission, training jobs are queued following a FIFO policy and dispatched to the SpinSearch module, which leverages the state information table to rank clusters based on network bandwidth and selects $m < k$ clusters to jointly execute each task according to its GPU requirement N. SpinSearch outputs the selected cluster IDs and the corresponding per-cluster GPU allocations. Based on this output, TPA determines the model parallelization strategy by considering cluster topology, generates the parallel configuration, and assigns specific GPUs through the Parallel Adapter and Buddy Allocator (As detailed in Appendix A) to reduce communication overhead. Finally, the Resource Orchestrator deploys the task according to the generated configuration and updates the global state information table upon successful allocation.

4 SpiderFlow design

4.1 SpinSearch

4.1.1 Problem Setting

We consider a multi-cluster environment represented as an undirected graph $G = (V, E)$, where each node $i \in V$ denotes a cluster with $g_i \in \mathbb{Z}_{\geq 0}$ available GPUs, and each edge $(i, j) \in E$ is annotated with a link bandwidth $b_{ij} > 0$. A task t requires $d_t \in \mathbb{Z}_{>0}$ GPUs, which may be allocated across multiple clusters. The scheduling objective is to minimize cross-cluster communication cost, prioritizing allocations that span fewer hops, while secondarily maximizing the effective bandwidth among the participating clusters.

The communication capacity between two nodes is modeled as the *widest-path bandwidth*:

$$\beta(u \to v) = \max_{p \in \mathcal{P}(u,v)} \min_{e \in p} b_e, \tag{1}$$

where $\mathcal{P}(u, v)$ denotes the set of all paths between nodes u and v. This captures the bottleneck link of the best available path.

4.1.2 Bandwidth-First Ordering

Bandwidth monitor actively probes inter-cluster links by transmitting test packets and records the measured bandwidths into a state information table. Each cluster is then assigned a global *bandwidth priority score* B_i. A simple definition is the sum of incident link capacities:

$$B_i = \sum_{j:(i,j) \in E} b_{ij}. \tag{2}$$

Alternatively, if the task communicates with a designated source cluster r, we may use the widest-path

bandwidth from r:

$$B_i = \beta(r \to i). \tag{3}$$

Clusters are ordered in descending order of B_i. Given this order, we apply a *First-Fit Decreasing (FFD)* procedure: iteratively allocate GPUs to the task starting from the highest-ranked cluster, assigning

$$\Delta_i = \min\{g_i,\ R\}, \tag{4}$$

where R is the residual GPU demand, until either $R = 0$ (successful allocation) or all clusters have been considered.

4.1.3 Hop-Based Expansion

If the initial FFD traversal fails to satisfy the demand, we select as *anchor* the cluster with the highest bandwidth score:

$$s^\star = \arg\max_{i \in V} B_i. \tag{5}$$

We then expand the search radius in terms of hop distance. For radius r, the candidate set is

$$\mathcal{C}(r) = \{\, i \in V \mid \mathrm{dist}(s^\star, i) \le r \,\}. \tag{6}$$

Within $\mathcal{C}(r)$, nodes are sorted by a lexicographic key:

$$(\beta(s^\star \to i),\ g_i,\ B_i), \tag{7}$$

favoring higher bottleneck bandwidth, larger residual capacity, and higher global score. The FFD procedure is repeated on $\mathcal{C}(r)$, and the radius is increased incrementally ($r \mapsto r + 1$) until the task demand is fully satisfied or the graph diameter is reached.

4.1.4 Illustrative Scheduling Example

As depicted in Figure 4(a), Firstly, the bandwidth monitor dynamically sorts clusters connected to the head node in descending order based on their available bandwidth. Suppose there is a job (spider) that requires 16 GPU cards, step one: spider will check if local resources are sufficient for distribution and will allocate them immediately if they are adequate. If local resources are insufficient, then step two is executed: it will visit clusters from *b* to *e* in order of bandwidth, until it finds a cluster with sufficient resources to meet the demand, and then proceed with the allocation. If no cluster satisfies the requirements after traversing all clusters, step three is implemented: pre-allocate all resources from cluster *a* (4 GPUs), and consequently, the resource requirement for the task (spider) is adjusted from 16 to 12 GPU cards. It will then continue to traverse the remaining clusters in descending order of bandwidth. If a cluster fulfills the criteria, it will

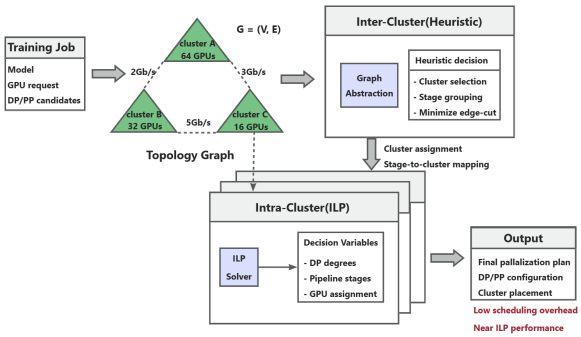

Figure 5: Two-level parallelization automation framework for cross-cluster training.

assign the remaining 12 GPU cards. As illustrated in Figure 4(b), if none of the clusters visited can satisfy the available GPU requirement, SpinSearch ranks the clusters by bandwidth in descending order and selects the cluster *g* (with 8 GPUs) that has the largest number of available GPUs as the next candidate cluster. It then performs hop expansion to enlarge the search radius and repeats the search among the remaining clusters, ultimately, cluster *a*, *b* and *g* would train this job together.

4.1.5 Complexity and Approximation Analysis

Let $n = |V|$ denote the number of clusters and $m = |E|$ the number of links. SpinSearch consists of two lightweight stages.

Bandwidth-first ordering. Computing the priority score B_i for all clusters requires scanning incident links once, which takes $O(m)$ time. Sorting clusters by B_i costs $O(n \log n)$.

Hop-based expansion. Starting from the anchor cluster, SpinSearch incrementally expands the search frontier by hop distance using BFS. Since each node and edge is visited at most once, the expansion phase costs $O(n + m)$.

Therefore, the total scheduling complexity is:

$$T_{\text{SpinSearch}} = O(m + n \log n), \tag{8}$$

which is substantially lower than global ILP or genetic-search methods whose worst-case complexity grows exponentially with cluster count.

Approximation intuition. SpinSearch optimizes a lexicographic objective: (i) minimizing the number of participating clusters, and (ii) maximizing effective connectivity among selected clusters. For the first objective, the First-Fit style allocation achieves the same classical bound as bin-packing heuristics, using at most $(11/9)\mathrm{OPT} + 1$ clusters in the worst case. For the second objective, the hop-based local expansion greedily prioritizes nearby high-bandwidth clusters, which empirically approx-

imates low-cut communication placements.

Although SpinSearch is heuristic rather than globally optimal, it provides a favorable trade-off between solution quality and scheduling overhead, which is particularly desirable in large decentralized GPU environments.

4.2 Topology-aware Parallelism Automation (*TPA*)

Parallel Adapter

As illustrated in Figure 5, parallelization automation for cross-cluster training can be formulated as a graph partitioning problem. The physical training environment is abstracted as a weighted graph $G = (V, E)$, where each vertex represents a compute unit (e.g., a cluster), and each edge is associated with a communication cost derived from bandwidth and latency heterogeneity. Given a training job with GPU demand r, the objective is to determine a parallelization configuration and a placement strategy that minimize the end-to-end training time while satisfying resource capacity constraints. In this formulation, overall performance is largely dominated by communication across high-cost edges, motivating a communication-aware partitioning strategy.

We adopt a two-level parallelization automation design to address the scalability challenges of global optimization. The core idea is to decompose the problem into two stages: inter-cluster coordination using lightweight heuristic methods and intra-cluster optimization using ILP. This hierarchical decomposition enables scalable decision making while preserving high-quality parallelization configurations.

At the intra-cluster level, we employ ILP to jointly optimize data parallelism (DP) and pipeline parallelism (PP). Let (d, p) denote the DP degree and the number of pipeline stages, respectively, subject to the constraint $d \cdot p \leq r$. The ILP determines the assignment of GPUs to pipeline stages, the grouping of GPUs into DP replicas, and the placement of these replicas within a cluster. The optimization objective minimizes an estimated per-step execution time,

$$\min\ T_{\text{cluster}} = T_{\text{comp}} + \alpha \cdot T_{\text{dp}} + \beta \cdot T_{\text{pp}},$$

where T_{comp} models computation time, T_{dp} captures gradient synchronization cost under DP, and T_{pp} accounts for pipeline communication and bubble overhead. By restricting ILP optimization to the intra-cluster scope, the number of decision variables and constraints is significantly reduced, enabling efficient exploration of the DP and PP search space while retaining near-optimal solution quality.

At the inter-cluster level, we use lightweight heuristic methods to coordinate resource allocation across clusters. Based on the topology graph G, the heuristic selects clusters and groups pipeline stages to reduce expensive cross-cluster communication. The resulting inter-cluster communication cost can be expressed as

$$T_{\text{inter}} \propto \sum_{(u,v)\in E_{\text{cut}}} w(u, v)\, x_{uv}, \tag{9}$$

where E_{cut} denotes the set of edges induced by pipeline stages or DP replicas placed in different clusters, $w(u, v)$ represents the communication cost between clusters u and v, and x_{uv} is the traffic volume crossing the corresponding link. Minimizing T_{inter} exactly is equivalent to a capacitated graph-partitioning problem, which becomes computationally expensive as the number of clusters increases. Therefore, instead of exhaustively solving a global ILP, our heuristic incrementally places communication-intensive components onto clusters with sufficient residual capacity and favorable connectivity, while keeping strongly coupled stages within the same cluster whenever possible. This greedy low-cut placement strategy effectively preserves most of the benefit of global optimization, while achieving substantially lower scheduling overhead.

Compared to a monolithic ILP formulation spanning all clusters, the two-level approach significantly reduces the overall search space by confining ILP optimization to individual clusters. While global ILP suffers from a combinatorial explosion in variables and constraints as the number of clusters increases, the proposed hybrid design leverages heuristics to efficiently handle inter-cluster decisions. As a result, this approach achieves solution quality comparable to full ILP-based optimization, while substantially reducing scheduling overhead, making it well suited for large-scale and bandwidth-constrained cross-cluster training environments.

5 Evaluation

5.1 Setup

Real Tested: In our physical testbed, a cluster is defined as a geographically or logically isolated resource domain connected through lower-bandwidth inter-cluster links, rather than by GPU count alone. Accordingly, we deployed 8 decentralized clusters

(servers), each equipped with 2, 2, 2, 2, 2, 4, 1, and 1 NVIDIA A100 GPUs, respectively, to emulate heterogeneous cross-cluster environments. The inter-server connectivity was established through Ethernet links, offering data transfer rates from 1 Gb/s to 15 Gb/s. Within the first three servers, internal communication between components was facilitated by PCIe 4.0, providing a bandwidth capacity of approximately 4 GB/s. In contrast, from fourth to sixth server featured an advanced 25 GB/s NVLink for optimized GPU-to-GPU communication. Our distributed computing environment was constructed using the Ray framework, designating the first server as the head node and the remaining servers as worker nodes.

Simulator: We extended the PAI's simulator (Ali, 2020) by going beyond basic task-level events such as submission, arrival, waiting, computation, and completion. In particular, we incorporated the decentralized intra-cluster communication bandwidth and modeled the inter-cluster bandwidth within the simulator. To capture communication delays in data parallelism, we employed the latency of AllReduce operations measured on physical machines under different bandwidths. Moreover, we analyzed the task throughput and GPU utilization of multiple models executed across physical clusters. These empirical measurements were integrated as parameter inputs to the simulator, enabling it to accurately reproduce the time overhead, model throughput, and GPU utilization of cross-cluster training, with deviations of less than 5% compared to training results on real clusters. This same-scale validation ensures that the simulator reliably captures both intra- and inter-cluster behavior before being used for larger-scale evaluations with richer cluster topologies. Our simulator utilized approximately 3,584 GPU cards, 64 clusters and 448 nodes with 8 GPUs each.

Training Tasks: We used *NewWorkload* for real test, consisting of GPT-2 350M, 1.3B, 4.2B (Radford et al., 2019), OPT 6.7B, 13B and BERT (Devlin et al., 2019) 110M, 340M models with different sizes and various batch sizes, to create 1000 LLM task queues. Additionally, we incorporated the *Philly* (Jeon et al., 2019) dataset, based on over 100,000 jobs from Microsoft's multi-tenant GPU cluster, providing insights into managing large-scale environments. We also used *Helios* (Hu et al., 2021) from SenseTime, which contains data from four clusters with over 3.3M tasks.

5.2 Baseline

Opportunistic Scheduling Strategy: Opportunistic scheduling strategy (Li et al., 2023) always prioritize the use of servers with higher bandwidth for incoming tasks. The opportunistic scheduling followed an FCFS policy, greedily allocating idle hardware resources to each newly submitted task.
DT-FM: DT-FM (Yuan et al., 2022) is a method for model training that utilizes clusters distributed across different regions of the world. Primarily, it optimizes parallel model execution strategies at the model level.
Lyra: Lyra (Li et al., 2023) is a cross-cluster training method that leverages both inference and training clusters for model training. Primarily, it focuses on resource borrowing between two clusters at the cluster level for cross-cluster training.
Mast: Mast (Choudhury et al., 2024) is a cross-cluster scheduling approach designed for ML training; however, it restricts each training job to execute within a single cluster.
ALPA: ALPA (Zheng et al., 2022) is a system that uses integer linear programming (ILP) to automatically configure parallelization strategies for deep learning workloads.

5.3 General result

We evaluate average throughput (samples per second) and JCT on a physical testbed consisting of eight decentralized servers with a total of 16 NVIDIA A100 GPUs, using the top 300 tasks from *NewWorkload*.

The left panel of Figure 6 shows the throughput scalability of four cross-cluster schedulers as the number of concurrent jobs increases. Although all methods initially benefit from additional cross-cluster resources, throughput growth gradually slows due to rising communication overhead and resource contention. Across the entire workload range, *SpiderFlow* improving throughput by 1.12-1.25× compared to the baselines, which demonstrates superior scalability. This advantage is attributed to SpinSearch, which enables low-overhead and near-optimal cross-cluster scheduling, and TPA, which provides efficient parallelism configuration with ILP-comparable effectiveness at significantly lower cost.

The right panel of Figure 6 reports the average JCT under increasing workload intensity. Although JCT increases for all methods with growing job concurrency, *SpiderFlow* achieves a 1.2-1.3× re-

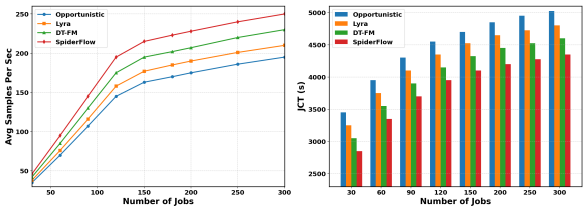

Figure 6: Performance comparison of cross-cluster training schedulers of Opportunistic scheduling, Lyra, DT-FM and SpiderFlow. The left figure shows the average number of samples processed per job per second, while the right figure reports the JCT, both measured over the top 300 tasks.

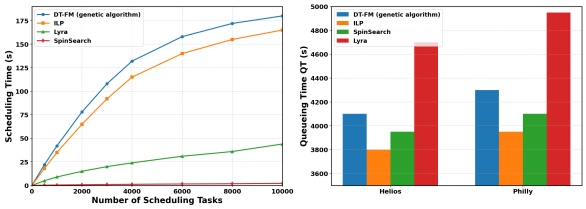

Figure 7: Scheduling overhead and queueing delay of cross-cluster training schedulers. The left figure reports the scheduling time as the number of scheduling tasks increases, while the right figure shows the average QT on two real-world workloads.

duction in JCT across all workload scales. This improvement stems from the combined effect of reduced scheduling overhead and efficient parallelism automation, allowing SpiderFlow to translate improved cross-cluster utilization into shorter completion times, particularly under high system load.

5.4 Impact of Individual Techniques

5.4.1 SpinSearch

To evaluate the scheduling effectiveness of SpinSearch, we enable only the SpinSearch module and conduct experiments using the first 300 tasks from *NewWorkload*. Experiments are performed on a decentralized platform consisting of eight physical servers with a total of 16 NVIDIA A100 GPUs, and the results are compared against state-of-the-art scheduling approaches.

The left panel of Figure 7 shows the scheduling time of different cross-cluster schedulers as the number of scheduling tasks increases from 0 to 10,000. Optimization-based methods, including DT-FM (genetic algorithm) and ILP, exhibit rapidly increasing scheduling overhead, reaching up to 180 seconds at large scales. Lyra shows moderate growth, whereas SpinSearch consistently maintains sub-second to 2-5s scheduling time even at the largest scale. This performance gap arises because DT-FM and ILP rely on expensive global optimization, while SpinSearch adopts a low-overhead search mechanism tailored for cross-cluster environments.

The right panel of Figure 7 presents the average job queueing time (QT) on top 300 NLP workloads from Helios and Philly. Lyra incurs the highest QT, while DT-FM and ILP achieve lower QT. SpinSearch yields QT slightly higher than ILP but comparable to DT-FM across both workloads. Taken together with the left panel, these results demonstrate that SpinSearch achieves a favorable balance between scheduling quality and efficiency, delivering near-optimal queueing performance with substantially lower scheduling overhead in cross-cluster training scenarios.

5.4.2 TPA

To validate the effectiveness of *Heuristic-ILP* in TPA for automated parallelization, we construct computation graphs with varying numbers of nodes and edges using an Intel Xeon 96-core CPU-based simulator, emulating large-scale decentralized clusters. We evaluate graph partitioning performance using 1,000 tasks from *NewWorkload*, recording runtime and edge-cut quality for each method, where lower edge-cut indicates reduced inter-partition communication cost and values normalized to ILP (1.0) represent near-optimal solutions.

Table 2 compares different automated parallelization approaches on multi-cluster computation graphs. The ILP-based solver used in ALPA achieves optimal edge-cut quality but incurs prohibitive runtime and fails to scale to larger graphs. DT-FM, which adopts a genetic algorithm for global graph partitioning, reduces runtime compared to ILP but still suffers from high optimization cost and inferior edge-cut quality. In contrast, *Heuristic-ILP* attains near-optimal partition quality while significantly reducing runtime.

This advantage stems from the hybrid design of *Heuristic-ILP*, which applies ILP only within individual clusters—where the search space remains manageable—while relying on lightweight heuristic scheduling across clusters. By preserving ILP's effectiveness for intra-cluster parallelism and avoiding the exponential complexity of global optimization, *Heuristic-ILP* achieves a favorable balance between solution quality and efficiency, making it well suited for large-scale cross-cluter training.

Table 2: Comparison of automated parallelization methods on multi-cluster graphs.

Method	Graph Size (nodes/edges)	Runtime (s)	Edge-Cut Quality (lower is better)
ILP Solver (ALPA)	128 / 512	182.4	1.00 (baseline)
DT-FM (genetic algorithm)	128 / 512	145.7	1.18
`Heuristic-ILP` (ours)	128 / 512	3.37	1.05
ILP Solver (ALPA)	512 / 2048	>3600 (timeout)	–
DT-FM (genetic algorithm)	512 / 2048	812.6	1.23
`Heuristic-ILP` (ours)	512 / 2048	12.15	1.08

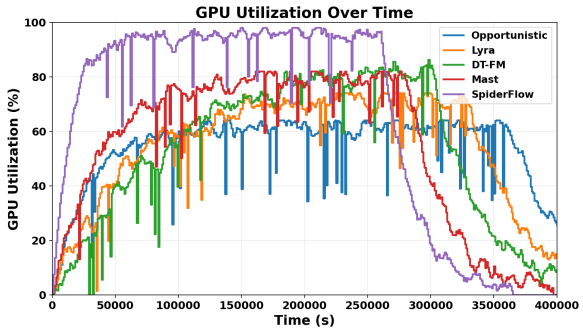

Figure 8: GPU utilization over time under different schedulers. SpiderFlow achieves the fastest ramp-up, highest peak utilization, and earliest completion. Opportunistic and Lyra show lower utilization and longer runtime, while Mast and DT-FM achieve moderate gains.

5.5 Large-scale test for SpiderFlow

We evaluate SpiderFlow using a simulator-based multi-cluster setup comprising six independent clusters, each with 64 nodes and 8 GPUs per node, for a total of 3,072 GPUs. To emulate hierarchical communication, the intra-node GPU bandwidth is set to 25 GB/s, the inter-node bandwidth within each cluster to 1 GB/s, and the inter-cluster bandwidth to 5 Gb/s. The evaluation uses the top 10,000 tasks from the Philly and Helios traces, with each task requesting between 0 and 256 GPUs.

Figure 8 presents GPU utilization over time under different scheduling strategies. SpiderFlow achieves the fastest ramp-up and the highest peak utilization, demonstrating its ability to rapidly and efficiently occupy available GPU resources. In contrast, Opportunistic and Lyra show slower utilization growth and larger fluctuations, indicating less effective resource coordination, while Mast and DT-FM yield moderate improvements. SpiderFlow also completes the workload earlier than all baselines, as reflected by its utilization dropping to zero significantly sooner. Overall, these results show that improved scheduling efficiency simultaneously increases average GPU utilization and reduces overall workload makespan.

5.6 Breakdown Analysis

We conduct a breakdown analysis on real traces from the Philly and Helios datasets to evaluate

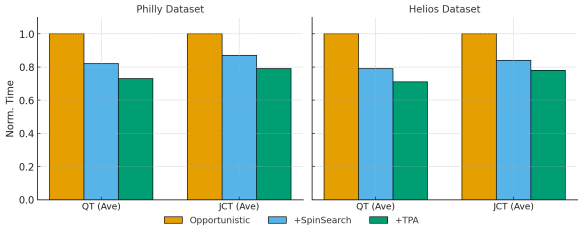

Figure 9: Breakdown analysis on the Philly and Helios datasets with normalized QT and JCT.

the effectiveness of the proposed optimization modules, using the Opportunistic method as the baseline with normalized QT and JCT set to 1.0. The +SpinSearch variant augments the baseline with the SpinSearch module, while +TPA further incorporates the TPA module on top of SpinSearch. Experiments are performed on 8 isolated clusters, each equipped with 1–4 NVIDIA A100 GPUs, using the top 1,000 jobs from each dataset. As shown in Figure 9, both datasets exhibit consistent performance improvements as additional modules are introduced: on the Philly dataset, QT decreases from 1.00 to 0.82 and then to 0.73, while JCT is reduced from 1.00 to 0.87 and then to 0.79; similar trends are observed on the Helios dataset, with QT decreasing from 1.00 to 0.79 and then to 0.71, and JCT from 1.00 to 0.84 and then to 0.78. These results demonstrate that SpinSearch and TPA jointly improve scheduling efficiency, yielding shorter queueing and completion times across different workloads.

6 Conclusion

We design SpiderFlow, an efficient topology-aware scheduling system for LLM training across decentralized GPU clusters. We first analyze the performance bottlenecks inherent in cross-cluster training. Guided by these insights, we develop low-overhead task scheduling and parallelism automation strategies that maximize training efficiency.

Acknowledgments

We thank the anonymous reviewers for their constructive comments that improved this manuscript. This work was supported by the Major Projects of Zhejiang Province (LD24F020012), National Science Foundation of China (U25A20423), Hangzhou Chengxi Sci-Tech Innovation Corridor Special Development Fund (K20241957), Key Research Project of Zhejiang Lab (2025SSYS0005), and the Science and Technology Program of Zhejiang Province (2026SDXT012).

7 Limitations

While our current design focuses on communication-efficient scheduling in decentralized environments, it does not yet incorporate tensor parallelism (TP) into the parallelization strategy, nor does it fully support training across heterogeneous GPU types in cross-cluster settings. These design choices are primarily motivated by the high communication cost and system complexity introduced by fine-grained synchronization and heterogeneity in bandwidth-constrained environments.

As future work, we plan to extend our framework to enable TP within single clusters equipped with high-bandwidth interconnects, where its benefits can be effectively realized. In addition, we aim to support training across heterogeneous GPU resources and explore more flexible parallelization strategies, including expert parallelism (EP) and hybrid schemes, to further improve scalability and resource utilization in decentralized multi-cluster systems.

References

2020. clusterdata.

2024. Nvidia v100 tensor core gpu.

2024. Tensorboard.

Alibaba. 2024. Alibaba pai (alibaba machine learning platform for ai.).

Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep learning-based job placement in distributed machine learning clusters. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, page 505–513. IEEE Press.

Zihan Chang, Yang Zhang, and Wenbo Chen. 2018. Effective adam-optimized lstm neural network for electricity price forecasting. In *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, pages 245–248. IEEE.

Zihan Chang, Yang Zhang, and Wenbo Chen. 2019. Electricity price prediction based on hybrid model of adam optimized lstm neural network and wavelet transform. *Energy*, 187:115804.

Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, Ritesh Tijoriwala, Denis Samoylov, and Chunqiang Tang. 2024. MAST: Global scheduling of ML training across Geo-Distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 563–580, Santa Clara, CA. USENIX Association.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 609–623, New York, NY, USA. Association for Computing Machinery.

Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 485–500, Boston, MA. USENIX Association.

Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15.

Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic resource sharing for distributed deep learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 721–739. USENIX Association.

Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of {Large-Scale}{Multi-Tenant}{GPU} clusters for {DNN} training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 947–960.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *Preprint*, arXiv:2307.10169.

Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2023. Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 835–850, New York, NY, USA. Association for Computing Machinery.

Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 289–304, Santa Clara, CA. USENIX Association.

Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking beyond

GPUs for DNN scheduling on Multi-Tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 579–596, Carlsbad, CA. USENIX Association.

Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 1–18. USENIX Association.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

AWS Regions. 2024a. [link].

Azure Regions. 2024b. [link].

Google Cloud Regions. 2024c. [link].

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-lm: Training multi-billion parameter language models using model parallelism. *Preprint*, arXiv:1909.08053.

Binhang Yuan, Yongjun He, Jared Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy S Liang, Christopher Ré, and Ce Zhang. 2022. Decentralized training of foundation models in heterogeneous environments. In *Advances in Neural Information Processing Systems*, volume 35, pages 25464–25477. Curran Associates, Inc.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2024. A survey of large language models. *Preprint*, arXiv:2303.18223.

Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P Xing, and 1 others. 2022. Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578.

A Appendix

A.1 GPU Buddy Allocator in TPA

The key to designing resource allocation strategy in intra-cluster topology lies in optimizing the communication between GPUs. It is well known that GPU interconnectivity can be achieved through various means, including NVIDIA's Nvlink, PCIe, switches, or traditional network connections. Typically, GPUs that are physically closer to each other can provide higher communication bandwidth, which is crucial for parallel tasks that require a significant amount of data transfer.

We design a buddy allocation scheme within each cluster to manage GPUs in a topology-aware manner. The cluster's GPUs are first organized into a balanced binary tree, where each leaf node corresponds to a single GPU and each internal node represents a contiguous block of GPUs. This hierarchical structure allows the allocator to split or merge GPU blocks dynamically, ensuring both flexibility and efficient use of resources.

During initialization, the system builds the binary tree based on the intra-cluster topology (such as NVLink connectivity, NUMA domains, or rack-level groupings). Free lists are maintained at different locality levels, including node, rack, and global scope. This design ensures that allocations always prefer GPUs that are physically close to each other before spilling into wider scopes with higher communication costs.

When a task requests a number of GPUs, the allocator rounds the request to the nearest suitable block size and searches the free lists in order of locality. If an exact block is available, it is allocated directly. Otherwise, a larger free block is selected and recursively split into smaller sub-blocks until the requested size is reached. The unused sub-blocks are returned to the free lists, maintaining efficient resource utilization.

On deallocation, the system returns the block to the free lists and attempts to coalesce it with its buddy, i.e., the sibling block under the same parent in the binary tree. If both buddies are free, they are merged back into a larger block. This recursive coalescing reduces fragmentation and helps preserve larger contiguous GPU blocks for future allocations.

Overall, this intra-cluster buddy allocation approach ensures that GPU assignments are made quickly, maintain locality for high-bandwidth communication, and minimize internal fragmentation through systematic splitting and coalescing. The detail process is shown in Algorithm 1.

A.2 More performance analysis of cross-cluster training

we conducted experiments using the identical GPT-2 1.3B (Radford et al., 2019) model, employing only data parallelism and deployed on 8 V100 32GB (Nvi, 2024) GPUs distributed across 4 decentralized clusters, each connected with 10 Gbps

Algorithm 1 GPU buddy allocation within clusters (binary tree)

```
Class GPUBlock(id, size, gpus, parent, left, right)
Class BuddyAlloc(gpus, root, free_blocks)
function GPUBuddyAlloc(num_gpus, intra_topo)
    gpu_topology ← init_intra_cluster_topology()
    root ← build_tree(flatten(gpu_topology), None)
    populate_blocks(root, gpu_topology, free_blocks)
    allocator ← BuddyAlloc(gpus, root, free_blocks)
    task_block ← allocator.allocate(num_gpus)
    allocator.deallocate(task_block)
end function
function build_tree(gpu_list, parent)
    if |gpu_list| = 1 then
        return new GPUBlock()
    end if
    mid ← |gpu_list|/2
    block ← new GPUBlock()
    block.left ← build_tree(gpu_list[0:mid], block)
    block.right ← build_tree(gpu_list[mid:], block)
    return block
end function
function allocate(num_gpus)
    size ← smallest power of 2 such that size ≥ num_gpus
    for level in {node, rack, global} do
        if free_blocks[level] has a block of size size then
            block ← pop(free_blocks[level][size])
            while block.size > num_gpus do
                split_block(block)
                block ← block.left
            end while
            block.is_free ← False
            return block.gpus
        end if
    end for
end function
```

bandwidth. The experiments compared centralized training (single cluster)(Shoeybi et al., 2020), and decentralized training(Yuan et al., 2022) across two and four clusters. As shown in Figure 10, we can observe that training efficiency declines in a nearly linear manner as the number of clusters grows. This indicates that we should minimize the number of clusters for scheduling policy.

To delve deeper into the specific factors causing such significant performance differences, We conducted an in-depth operator-level analysis across the three experimental topologies using TensorBoard (Ten, 2024). We detailed the execution times of the main operators and visualized them in the bar chart. As shown in Figure 11, the analysis showed that

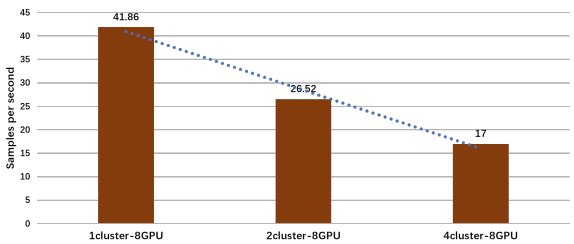

Figure 10: Efficiency of using the GPT-2 model on different topologies.

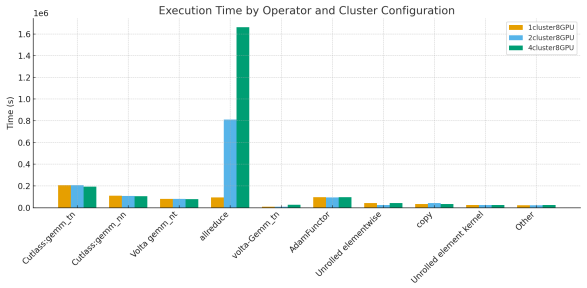

Figure 11: Execution time (s) of different operators under various cluster configurations.

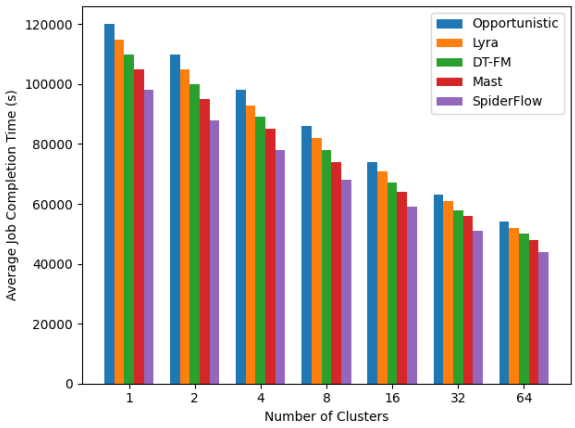

Figure 12: Average job completion time (JCT) under different cluster scales (1–64 clusters). SpiderFlow consistently achieves lower JCT than Opportunistic, Lyra, DT-FM, and Mast across all cluster counts.

while the execution times for most operators (eg., GEMM, Copy, etc.) remained largely unchanged in numerical values, one operator's execution time varied significantly — the communication operator for gradient aggregation, *AllReduce*.

A.3 Scalability Analysis of SpiderFlow

Figure 12 compares the average job completion time (JCT) under different cluster scales using a simulator and the top 10,000 jobs from the Helios workload. As the number of clusters increases from 1 to 64, every cluster has 7 nodes and 56 GPUs, the JCT of all methods decreases, since additional cross-cluster resources help alleviate resource contention. However, SpiderFlow consistently outperforms Opportunistic, Lyra, DT-FM, and Mast across all cluster scales, achieving the lowest JCT in each configuration. The performance gap becomes more pronounced at larger scales, indicating that SpiderFlow can more effectively exploit increased cluster availability. This result demonstrates the scalability of SpiderFlow in large-scale decentralized training and its ability to translate expanded cross-cluster resources into shorter job completion times.

A.4 Overhead Analysis

We further analyze the total scheduling overhead introduced by SpiderFlow to ensure that the observed performance gains are not offset by additional runtime costs. Experimental measurements show that the scheduling overhead of SpiderFlow is less than 1% of the total job execution time, indicating that its impact is negligible. The lightweight design of its topology-aware scheduler and the efficient implementation of the TPA module ensure that scheduling decisions are made within milliseconds, even across multiple clusters. As a result, SpiderFlow achieves substantial performance improvements while maintaining minimal coordination and computation overhead, demonstrating its practicality for large-scale, real-world deployments.