Crux: GPU-Efficient Communication Scheduling for Deep Learning Training

Jiamin Cao*, Yu Guan*, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, Ennan Zhai
Alibaba Cloud

Abstract

Deep learning training (DLT), *e.g.*, large language model (LLM) training, has become one of the most important services in multi-tenant cloud computing. By deeply studying in-production DLT jobs, we observed that communication contention among different DLT jobs seriously influences the overall GPU computation utilization, resulting in the low efficiency of the training cluster. In this paper, we present Crux, a communication scheduler that aims to maximize GPU computation utilization by mitigating the communication contention among DLT jobs. Maximizing GPU computation utilization for DLT, nevertheless, is NP-Complete; thus, we formulate and prove a novel theorem to approach this goal by *GPU intensity*-aware communication scheduling. Then, we propose an approach that prioritizes the DLT flows with high GPU computation intensity, reducing potential communication contention. Our 96-GPU testbed experiments show that Crux improves 8.3% to 14.8% GPU computation utilization. The large-scale production trace-based simulation further shows that Crux increases GPU computation utilization by up to 23% compared with alternatives including Sincronia, TACCL, and CASSINI.

CCS Concepts

• **Networks** → **Data center networks**; **Network control algorithms**; **Cloud computing**; • **Computing methodologies** → **Machine learning**.

Keywords

Communication Scheduling, Data Center Network, Deep Learning

ACM Reference Format:

Jiamin Cao*, Yu Guan*, Kun Qian, Jiaqi Gao, Wencong Xiao, Jianbo Dong, Binzhang Fu, Dennis Cai, Ennan Zhai, *Alibaba Cloud* . 2024. Crux: GPU-Efficient Communication Scheduling for Deep Learning Training. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3651890.3672239

1 INTRODUCTION

Deep learning has empowered many thriving fields. For example, in recent months, the emergence of large language models (LLMs) has revolutionized numerous natural language processing businesses such as Microsoft 365 Copilot [12], Firefly [8], and GitHub Copilot [11]. Due to their resource-intensive nature, deep learning training (or DLT) jobs run on large-scale clusters with tens of thousands of GPUs provided by cloud providers (*e.g.*, AWS, Azure, GCP, and Alibaba Cloud). Typically, a large number of DLT jobs co-execute in the clusters and share GPUs, as shown in Figure 1. GPU schedulers [36, 58] allocate GPUs for jobs.

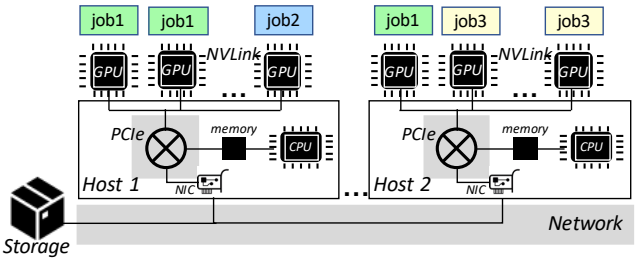


Figure 1: Allocation of GPUs on DLT jobs.

As a production DLT service provider, our goal is to maximize GPU computation utilization (or *GPU utilization* for short) of a GPU cluster, as it directly affects training throughput and cluster providers' profits (*e.g.*, training cost and customer expense). We observed that running jobs on the same GPU cluster increases the training time for each individual job (compared with the case monopolizing cluster), along with decreased GPU utilization. For example, the GPU utilization drops by 9.5% when co-executing a GPT job and a BERT job on the same cluster. A 9.5% decrease in GPU utilization means that 95 out of a 1,000-GPU cluster are wasted, which translates to millions of dollars.

We, therefore, profiled the entire DLT lifecycle in §2.2, and found that the performance interference between DLT jobs mainly stems from the *communication contention*. Communication happens when synchronizing model parameters, gradients, and optimizers among GPUs that reside in the same host or different hosts [37, 62], as shown in Figure 3. When co-executing multiple DLT jobs, we observed a remarkable iteration time increase because of communication contention. As a result, overcoming communication contention is vital for improving GPU utilization.

State of the arts. As shown in Figure 2, a comparison of state-of-the-art efforts, the prior work does not focus on solving the communication contention.

Job scheduler. Job schedulers optimize co-executed jobs by carefully allocating GPUs to each job and mitigating resource contention among them. Most job schedulers [20, 21, 30, 31, 33, 57, 58, 64]) treat communication (including inter-host network links and intra-host PCIe/NVLinks) as a blackbox and focus on computation contention for GPU resources. A few job schedulers [61, 63] consider communication contention. However, in a DLT cluster, due to the unpredictability of jobs (*e.g.*, the scale and start/end time), job schedulers

*Both authors contributed equally to this paper.

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08
https://doi.org/10.1145/3651890.3672239

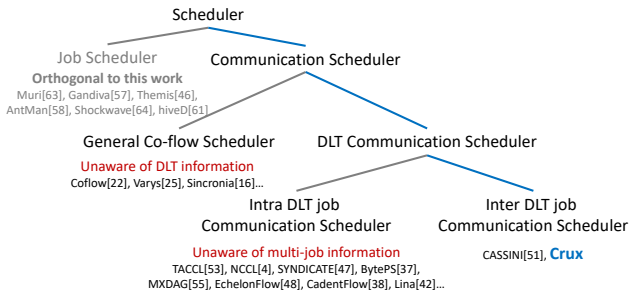
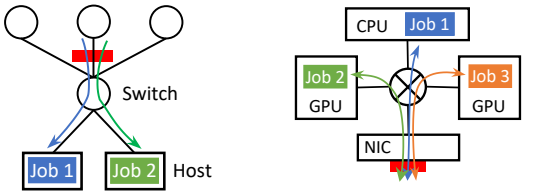


Figure 2: The state-of-the-art DLT schedulers.



(a) Contention on network forwarding paths

(b) Contention on intra-host PCIe links or NVLinks

Figure 3: Representative communication contention examples. Curves denote traffic. Red rectangles denote bottleneck links where contention happens.

cannot completely avoid communication contention. This paper focuses on solving communication contention with an existing job schedule, and is thus orthogonal to job schedulers.

Communication scheduler. Communication schedulers address the co-flow problem by determining flow priorities and selecting paths for multiple network flows. General co-flow schedulers [16, 22, 25] do not account for DLT-specific characteristics (*e.g.*, iterative and bursty data traffic, and communication-computation overlapping), so their insights and optimization strategies cannot work in this context. Most current DLT communication schedulers [4, 37, 38, 42, 46–48, 53, 55] solve the co-flow problem within a single DLT job. Their scheduling decisions are based on traffic patterns of a single job and fail to consider inter-job contention. CASSINI [51, 52] is an inter-job communication scheduler that proactively reduces contention by predicting each job's traffic patterns and applying a corresponding time-dimension offset. However, in a multi-tenant GPU cluster, the traffic patterns of each job are affected by the cluster itself and other jobs running concurrently. Therefore, simply shifting jobs based on traffic pattern predictions cannot eliminate communication contention.

Our approach: Crux. This paper proposes Crux, a communication scheduler to tackle inter-job communication contention. Crux introduces the concept of *GPU intensity* as a measure of a job's impact on GPU utilization. Using GPU intensity, Crux selects paths and assigns priorities for different jobs to mitigate communication contention. We make the following contributions.

Contribution 1. Analysis of our multi-tenant production training cluster reveals that 36.3% of DLT jobs may experience communication contention with other jobs, leading to significant GPU wastage. We argue that inter-job communication scheduling is necessary to improve GPU utilization (§2). We make our dataset publicly available at https://github.com/alibaba/alibaba-lingjun-dataset-2023 [9].

Contribution 2. We transform the challenge of maximizing GPU utilization, which is an NP-Complete (NPC) problem, into a *GPU intensity*-aware communication scheduling problem (§3). We design a system, Crux, to optimize GPU utilization in a DLT cluster. Crux introduces (1) a path selection algorithm to mitigate communication contention by choosing the least-congested path for jobs with higher GPU intensity (§4.1), (2) a priority assignment algorithm that considers DLT characteristics such as multiple iterations and communication-computation overlapping (§4.2), and (3) an efficient priority compression algorithm to adapt to limited priority levels on practical NICs and switches (§4.3).

Contribution 3. Our experimental testbed, consisting of 96 Nvidia A100 GPUs, demonstrated that Crux delivers up to a 14.8% increase in GPU utilization and 33% improvement in end-to-end performance with real-world models (*e.g.*, GPT, BERT, and ResNet) (§6.2). Our production trace (2,000+ GPUs, 5,000+ jobs) based simulation indicated that Crux improves GPU utilization by 5% to 23%, compared with state-of-the-art solutions (Sincronia [16], CASSINI [51], and TACCL [53]), under various cluster network architectures (§6.3).

2 BACKGROUND AND MOTIVATION

This section first describes the background of multi-tenant DLT clusters (§2.1). Then, §2.2 shows the popularity of inter-job communication contention in the in-production cluster, illustrating how it degrades GPU utilization and training throughput. §2.3 introduces why using GPU utilization as the goal. §2.4 presents our motivation to design an inter-job communication scheduler.

2.1 Background: Multi-Tenant DLT Clusters

Figure 1 presents a typical architecture of DLT clusters, including a *storage* that maintains training data, *hosts* equipped with many GPUs and CPUs, and a *network* (usually a multi-layer topology like Clos [32]) connecting the above entities. Each host consolidates multiple (*e.g.*, eight) GPUs for computation, together with CPUs and memory for data processing, and NICs for communication [27, 44]. The entities within hosts are connected by high bandwidth proprietary links (*e.g.*, NVLink) and PCIe fabric composed of PCIe switches (*PCIeSw* for short). Communication traffic of a DLT job usually needs to traverse NVLinks, PCIe links, and network links.

DLT jobs run for a large number of iterations to compute a pre-trained model. To accelerate the training process, or to fit larger models within the limited GPU memory, parallelism strategies (*e.g.*, data parallelism, pipeline parallelism, and tensor parallelism) distribute computation overload to multiple GPUs. In each iteration, GPUs communicate with each other to synchronize parameters, gradients, or optimizers of DLT models by collective communication operations (*e.g.*, AllReduce [49], Send/Recv, ReduceScatter, AllGather, and AllToAll).

2.2 Inter-Job Communication Contention Seriously Degrades GPU Utilization

We studied the DLT jobs during two weeks in Aug. 2023 on one of our in-production DLT clusters [9]. This cluster contains more than 2,000 GPUs interconnected by a Clos network with a 1:1 oversubscription ratio, supporting more than 5,000 jobs. The models trained on this cluster include large-scale models like LLMs and also legacy models such as ResNet and BERT. Figure 4 shows that over 10% of jobs (belonging to GPT variant models) occupy a minimum of 128 GPUs, with the largest job consuming up to 512 GPUs. Figure 5

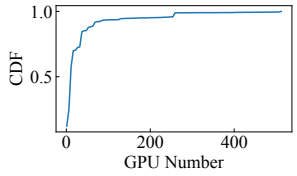
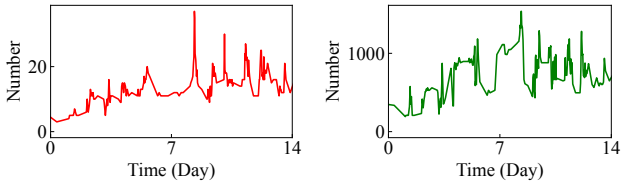
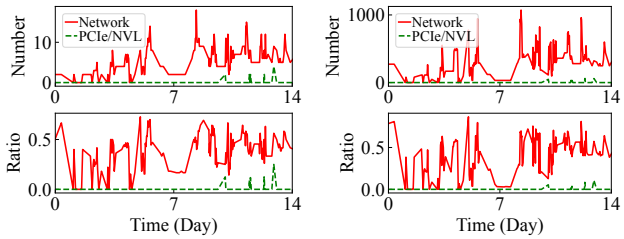


Figure 4: GPUs required by jobs in our cluster.



(a) The number of concurrent jobs. (b) The number of active GPUs.

Figure 5: The number of concurrent jobs and active GPUs in our cluster over two weeks.



(a) The number and ratio of jobs under the risk of communication contention. (b) The number and ratio of influenced GPUs.

Figure 6: Popularity of communication contention.

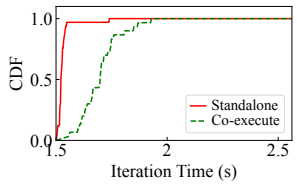


Figure 7: Impact of communication contention on iteration time of GPT.

highlights the concurrent execution of multiple jobs in the cluster over two weeks. In the peak hour, the number of concurrent jobs exceeds 30, occupying 1,000+ GPUs. Our cluster adopts an intuitive job scheduling approach which tries to allocate GPUs in the same host or under the same switch to a job.

Prevalence of inter-job communication contention in GPU clusters. Figure 6 shows the number of jobs and GPUs in our cluster that is at risk of communication contention (*i.e.*, sharing intra-host or inter-host links with others). As shown in the figure, among all jobs running in the cluster, 36.3% jobs (occupying 51% GPU) may suffer from communication contention during the life cycle. Only a minority of the contention occurs on intra-host PCIe links (as shown in Figure 3(b)), which results from resource fragmentation caused by the job scheduler's GPU allocation policies. Most contention occurs on network forwarding paths (as shown in Figure 3(a)). This is because network switches typically use Equal Cost Multi-Path (ECMP) -based forwarding by default, which inevitably leads to hash collisions and hence network link contention.

Why communication contention generally exists. Designing better job scheduling methods or increasing network capacities can help ease communication contention but cannot eliminate it. For job scheduling, the dynamic nature of jobs and their varying sizes can cause resource fragmentation (*i.e.*, resources are divided into smaller chunks), which increases the chance of communication contention. For example, a job may use GPU resources from several cluster units (pods) but may not use each pod completely. If multiple jobs share resources from the same pods, they would compete for the links connecting the pods. As for network capacity, even in networks with a 1:1 oversubscription, the communication contention may still exist as hash collision can lead to uneven traffic across links.

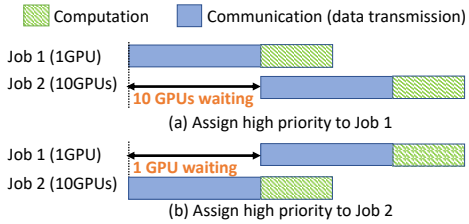


Figure 8: Example of why optimizing GPU utilization.

Impact of inter-job communication contention on GPU utilization. In our in-production GPU cluster, we sample a representative pair of jobs in communication contention and reproduce their training processes. Our evaluation shows that such contention leads to degradation in both GPU utilization and training throughput.

We run a 64-GPU large language model (a variant of GPT-3) and a 16-GPU language model (BERT) for this measurement. [1] GPT is distributed across eight hosts (H1 to H8) under two Top-of-Rack (ToR) switches, *i.e.*, four hosts under each switch. BERT is allocated to four hosts (H9 to H12) under the same two ToR switches, *i.e.*, using four GPUs in each host. They encounter communication contention on links between ToR switches and aggregation switches in their training processes. As shown in Figure 7, compared with executing GPT alone, when we launch BERT together, GPT's iteration time increases by 11.0%, from 1.53s to 1.70s during contention. The throughput of GPT and BERT reduces by 9.9% and 7.7%, respectively, which results in 9.5% reduction in GPU utilization.

2.3 Goal: Optimizing GPU Utilization

As a global DLT training provider, our business shows that a GPU cluster's goal is to optimize its overall GPU utilization. This is because the more computation (*e.g.*, floating-point operations) a cluster does, the greater profits the cluster provider can get. To do more computation in a given time duration, the most insightful way is to optimize the GPU utilization, benefiting the training throughput of the entire cluster. Suppose an 8-GPU server priced at $40 per hour. If a cluster has 10,000 GPUs, a 10% improvement in GPU utilization equates to a daily cost saving of $120,000. The overall GPU utilization, therefore, is the most important performance metric for our GPU cluster, which should be optimized.

There have been efforts that aim to optimize the job completion time [51, 64]. In the case of a single job, optimizing GPU utilization is equivalent to optimizing job completion time (JCT). However, in multi-DLT scenarios, naively optimizing JCT can lead to reduced GPU utilization. For example, in Figure 8, when two jobs contend over the same link, two different scheduling methods can result

[1] We apply the GPT-3 parameters from [13]. Considering training costs, we modify the number of transformer layers from 96 to 24, and hidden size from 12288 to 1024.

in the same average JCT, but the overall GPU utilization of the cluster can be completely different. Jobs with higher GPU workload usually have a greater impact on the overall GPU utilization, and thus should be scheduled with higher priorities. Notice that optimizing GPU utilization may introduce unfairness between jobs with various GPU workload, but this side effect is not severe (§7).

2.4 Optimizing GPU Utilization via Scheduling Inter-Job Communication Contention

Based on the above, communication contention between DLT jobs significantly degrades both training performance and GPU utilization. We, therefore, argue that an efficient communication scheduler for multi-job DLT is crucial. This paper tries to schedule inter-job communication for GPU clusters in two aspects.

Path selection. To reduce the probability of inter-job communication contention along network links, we select paths for each job based on traffic patterns and topologies. Note that communication within hosts typically uses the nearest NIC or NVLink directly for optimal performance, and thus path selection is not needed.

Priority assignment. To minimize losses in GPU utilization when communication contention is inevitable, Crux adopts a priority assignment approach which prioritizes jobs with a larger impact on GPU utilization.

3 METHODOLOGY AND SYSTEM OVERVIEW

This section presents our key methodology: maximizing GPU utilization is equivalent to scheduling more data flow of GPU-intensive DLT jobs in the network (§3.1 and §3.2). Based on this methodology, §3.3 presents the Crux's system overview.

3.1 Problem and Goal

Suppose a GPU cluster is connected by a network with topology $G =< \mathbf{V}, \mathbf{E} >$, where $\mathbf{V}$ is the set of GPUs and $\mathbf{E}$ is the set of links (including network links and intra-host links). The bandwidth of each link $e \in \mathbf{E}$ is denoted by B_e.

$\mathbf{J}$ denotes a set of iterative DLT jobs in this cluster. Each job $j \in \mathbf{J}$ occupies some GPUs in $\mathbf{V}$. In each iteration, job j generates computation workload W_j (in floating-point operations), which is distributed across GPUs it occupies. Meanwhile, job j generates a certain amount of network traffic. We define $M_{j,e}$ as the traffic produced by job j for link e in each iteration. Depending on parallelism strategies, a job's computation and communication may overlap with complex patterns (especially for large-scale DLT jobs).

Definition 1. **GPU utilization.** *In a given time period T, if GPU v processes computation workload L_v, the overall GPU utilization U_T of this GPU cluster is given by:*

$$U_T = \sum_{v \in \mathbf{V}} L_v \tag{1}$$

Our goal is thus to maximize U_T, by carefully scheduling the data flow of each DLT job (*e.g.*, by path selection or priority assignment) to avoid communication contention between them.

3.2 Deriving GPU Utilization Optimization to Flow Scheduling: A Single Link Case

Mathematically, the problem of maximizing U_T is a more complex version of maximum multi-commodity flow problem [14], an NPC problem.[2] Moreover, in our problem, each job is a DLT job with complex computation and communication behaviors, rather than a pure flow-transmission job whose transmission can be fully controlled. This adds further difficulty to communication scheduling.

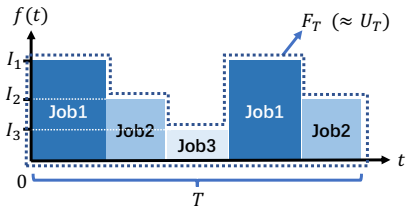


Figure 9: Deriving GPU utilization to a flow scheduling problem. Each rectangle is a job's flow scheduling. Its width indicates the flow transmission duration and its height indicates GPU intensity. The total area (enclosed by dotted line) is F_T, proved equivalent to U_T.

To solve this complex problem, we first introduce a new concept called *GPU intensity*, which reflects a job's impact on GPU utilization. As will be introduced in §5, the GPU intensity of a job can be obtained through measurement in the first few iterations of a job.

Definition 2. **GPU intensity** *of job j:*

$$I_j = \frac{W_j}{t_j} \tag{2}$$

where t_j is the maximum time a job's traffic takes to traverse any link e in $\mathbf{E}$, calculated as $t_j = \max_{e \in \mathbf{E}} \frac{M_{j,e}}{B_e}$.

> I_j indicates that, regardless of how communication and computation are overlapped, job j is always able to do W_j computation after t_j communication. Prioritizing the traffic of jobs with higher I_j is beneficial, as it unblocks more computation, thus improving GPU utilization.

We first consider a simple single-link scenario and prove that maximizing U_T is equivalent to maximizing the sum of GPU intensity of jobs on a given link. Specifically, we focus on a link e_0 with a constant bandwidth B_{e_0}. For other links $e \in \mathbf{E} \backslash e_0$, $B_e = \infty$. In this setting, the GPU utilization is affected by e_0.

Let $h(t)$ represent the jobs whose traffic is occupying link e_0 at time t, with $h(t) = \emptyset$ indicating the link is idle. We define $f(t)$ to reflect the intensity of the job occupying the link at time t as follows: $f(t) = \begin{cases} I_{h(t)} & h(t) \neq \emptyset \\ 0 & h(t) = \emptyset \end{cases}$, and $F_T = \int_{t \in T} f(t)$. Thus, F_T represents the total GPU intensity of all jobs transmitted by link e_0 during time period T, as shown in Figure 9. Then we can deduce Theorem 1, which is proved in Appendix A.

Theorem 1. $\lim_{|T| \to \infty} \frac{F_T}{U_T} = 1$

[2] In the maximum multi-commodity flow problem, a network (where each link has a bandwidth constraint) needs to transfer data for multiple jobs. For each job, one path from a source to a destination must be assigned, to transfer arbitrary data size. The goal is to maximize the total amount of data the network can transfer. Our problem is more challenging in two aspects: (1) each job usually has multiple sources and destinations, and (2) our objective is to maximize total GPU utilization, which can be regarded as a weighted average of the data transferred for each job (see GPU intensity in Definition 2). Our problem can be strictly reduced to a maximum multi-commodity flow problem when each job has only one pair of source and destination, and all jobs have exactly the same computation-communication ratio. Thus, our problem is strictly no easier than the maximum multi-commodity flow problem.

In a long time duration T, maximizing U_T is equivalent to maximizing F_T, *i.e.*, the sum of *GPU intensity* of all jobs transmitted by the link.

Based on the above, a job with higher GPU intensity is more important in communication scheduling.

3.3 Extending to Networks: CRUX Overview

In this section we first extend the insight we obtained from the case of one single link (Theorem 1 in §3.2) to a whole network topology, and then present an overview of CRUX, which applies our insight in practical GPU cluster networks.

Extending Theorem 1 to complex network topologies. When multiple DLT jobs run concurrently in a GPU cluster with complex network topology, the communication throughput for each job during contention is exclusively determined by the bandwidth allocation of the most congested link, *i.e.* the bottleneck link. If we consider the bandwidth of all other links to be infinite, since the bandwidth of the bottleneck link remains unchanged, the communication performance of each job remains the same. Then the complex topology becomes identical to the single-link scenario described in §3.2 (e_0 corresponds to the bottleneck link). Therefore, the conclusions of §3.2 still hold in practical network environments.

Utilizing Theorem 1 in communication scheduling. In a cluster network, based on Theorem 1, the ideal scheduling is to keep the network transmitting data from the most GPU-intensive jobs at 100% time (*e.g.*, always scheduling job 1 in duration T in Figure 9).

However, it is not trivial to utilize this insight in a real-world DLT cluster. First, each DLT job generates data flow only part of the time and in part of network paths, following its traffic pattern. An adequate scheduler should prioritize GPU-intensive jobs while making full utilization of the GPU cluster network. Furthermore, commodity GPU clusters provide limited functions for user-defined traffic scheduling. For example, ECMP supports limited path selection based on 5-tuple, and Differentiated Services Code Point (DSCP) only supports a few unified and limited levels of job priorities across the entire cluster network. Both the problem complexity and practical constraints set further obstacles to implementing ideal communication scheduling.

Our approach, CRUX, is designed to bridge this gap. CRUX utilizes the concept of GPU intensity in practical flow scheduling mechanisms (*e.g.*, path selection and priority assignment), trying to improve GPU utilization in DLT clusters. CRUX proposes three techniques, illustrated in Figure 10:

- **GPU intensity-based path selection (§4.1).** Since contention between GPU-intensive jobs degrades overall GPU utilization more severely, CRUX presents a GPU intensity-based path selection method. CRUX tries to select different paths for GPU-intensive jobs to avoid contention, and is more tolerant of contention between jobs with lower GPU intensity.
- **Priority assignment with consideration of DLT characteristics (§4.2).** Due to the characteristics of DLT jobs (*e.g.*, multiple iterations and communication-computation overlap), directly assigning priorities to jobs based on their GPU intensity may produce uneven network overload in time-dimension (*e.g.*, sometimes there are multiple jobs contending for the network, and sometimes the network is idle for a while), leading to suboptimal performance. CRUX analyzes how these new characteristics influence network overload patterns, and presents a novel mathematical model to fine-tune the priority assignment.
- **Compressing priorities to limited levels (§4.3).** Because NICs and switches support limited priority levels (*e.g.*, no more than 8, which is much less than the number of jobs), after globally unique priority assignment, CRUX has to compress original priorities to limited priority levels, resulting in multiple jobs being mapped to the same level. This compression unavoidably leads to performance loss. Fortunately, we find that a well-designed priority compression significantly reduces this side effect. For jobs that are not GPU-intensive or do not share network paths, compressing their priority differences has a limited (or even no) side-effect on GPU utilization. Based on this insight, CRUX selectively compresses these unimportant priority differences.

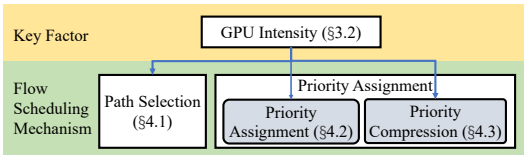


Figure 10: CRUX Overview.

4 CRUX DESIGN

This section presents our design of CRUX. §4.1 proposes CRUX's path selection method. §4.2 assigns a unique priority to each job in the cluster. §4.3 describes how to compress the assigned priorities to limited priority levels, to make it deployable in practical DLT clusters. §4.4 validates the effectiveness of CRUX using micro-benchmark evaluation.

4.1 GPU Intensity-Based Path Selection

Modern DLT clusters (*e.g.*, Clos) offer redundant links between multi-layer network switches (*e.g.*, ToR, aggregation, and core switch) and utilize ECMP-based hash mechanisms [2] to select random paths for DLT jobs. The network's load balance is then decided by the hash algorithm. As the number of concurrent jobs increases, hash collision is unavoidable. Consequently, careful path selection becomes critical to mitigate communication contention.

Why GPU intensity matters in path selection? As illustrated in §3.2, jobs with higher GPU intensity are more dominant for the cluster's GPU utilization. Therefore, the key idea of CRUX's path selection is to ensure that jobs with higher GPU intensity are not affected by communication contention. When high-intensity jobs compete with low-intensity jobs for the same link, as will be introduced later, high-intensity jobs generally have higher priority, so they will not be impacted by jobs with lower intensity. Consequently, the main challenge in path selection is to avoid communication contention between high-intensity jobs.

Our approach. CRUX proposes GPU intensity-based path selection. For multiple DLT jobs in the cluster, CRUX makes path selection starting from the most GPU-intensive jobs to the least. For each job, CRUX selects the least congested path from all available options at that moment. This strategy tries to select distinct paths for GPU-intensive jobs from one another. CRUX refreshes all path selections in dynamic job arrivals/completions (see §5).

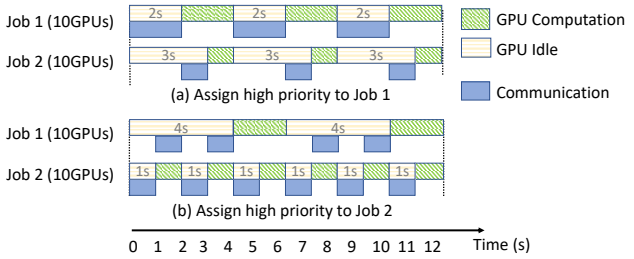


Figure 11: [Example 1] Iteration time influences priority.

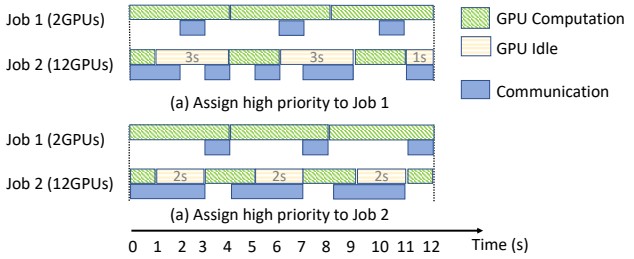


Figure 12: [Example 2] Computation-communication overlap influences priority.

4.2 Priority Assignment

This section assigns a unique priority P_j to each job j. We hope that when there is communication contention between j_1 and j_2:

(1) If $P_{j_1} > P_{j_2}$, prioritizing the communication of j_1 first should result in higher overall GPU utilization.

(2) If $P_{j_1} = P_{j_2}$, prioritizing the communication of either j_1 or j_2 should result in the same GPU utilization.

Why GPU intensity should be fine-tuned in priority assignment? Based on §3, we should always assign high priorities to GPU-intensive jobs. However, assigning priorities only based on GPU intensity (*e.g.*, $P_j \triangleq I_j$) may lead to uneven network overload in time-dimension (*e.g.*, sometimes there are multiple jobs contending for the network, and sometimes the network is idle for a while), similarly observed in CASSINI [51]. We will show that this is because DLT jobs have different iteration times and computation-communication overlap behaviors. Based on these new characteristics, fine-tuning our priority assignment can stagger the data-flow of different jobs, thereby achieving better performance.

We begin with two examples:

Example 1. Job 1 (W_1 = 10Gflops, $t_1 = 2s$) and Job 2 (W_2 = 5Gflops, $t_2 = 1s$) compete for one link. Both jobs require 10 GPUs. Job 1 has a large computation overload, so its computation time (2s) is twice that of Job 2 (1s). Based on Equation (2), they have equal GPU intensity. As shown in Figure 11, if Job 1 is prioritized, GPUs of Job 1 are idle for 6 seconds, and GPUs of Job 2 are idle for 9 seconds, so the overall GPU utilization is 37.5%. Conversely, if Job 2 is prioritized, the GPU utilization can reach 41.7%. This is because prioritizing a shorter-iteration job better utilizes the bandwidth to transmit more data, improving overall GPU utilization.

Example 2. Job 1 (W_1 = 10Gflops, $t_1 = 1s$) and Job 2 (W_2 = 30Gflops, $t_2 = 3s$) compete for one link. Job 1 requires 2 GPUs and takes 4s for computation per iteration. Job 2 requires 12 GPUs. Since Job 2 has 3x more computation overload but uses 6x more GPUs, its computation time is half that of Job 1, taking only 2s. Based on Equation (2), they have equal GPU intensity. For simplicity, we assume a job starts communication when it finishes 50% computation of one iteration (*e.g.*, after forward propagation which consists of about half of the computation, data communication can overlap with backward propagation). As shown in Figure 12, if Job 1 is prioritized, 12 GPUs of Job 2 are idle for 7 seconds; If Job 2 is prioritized, they are idle for only 6 seconds. This is because communication in Job 1 can be fully overlapped by its computation. Communication delay for $\leq$ 1s does not reduce its training throughput or GPU utilization. On the contrary, Job 2 is highly sensitive to communication delay since its communication can not be fully overlapped.

Modeling DLT characteristics in priority assignment. To measure how DLT characteristics affect priorities, we propose correction factor k_j for each job j, bridging the gap between I_j and P_j:

$$P_j \triangleq k_j I_j \tag{3}$$

Now, we compute the correction factor for each job. First, suppose job j is the job that generates the most network traffic in the cluster network. We define $k_j = 1$ and call job j the reference job. Then, correction factors of other jobs are computed by making comparisons with this reference job (as the reference job is most likely to contend against other jobs), like Figure 11 and 12.

For example, in Figure 11, suppose Job 1 is the reference job (*i.e.*, $P_{j_1} = 1$), given the per-iteration computation/communication time of the two jobs, we can try both priority assignments and make a comparison. When Job 1 is prioritized, the network spends 6 seconds transmitting Job 1's data and 3 seconds on Job 2's. Conversely, when Job 2 is prioritized, the network transmits 4 seconds for Job 1's data and 6 seconds for Job 2's. According to the definition of priority, if $P_{j_1} = P_{j_2}$, prioritizing Job 1 (transmit Job 1's data for 2 more seconds) or Job 2 (transmit Job 2's data for 3 more seconds) should yield the same computation overload. Based on Equation (2), Job 1 should have $3s/2s$ = 1.5x GPU intensity of Job 2 (*i.e.*, $I_{j_1} = 1.5I_{j_2}$). Then, according to Equation (3), we have $k_{j_2} = 1.5k_{j_1} = 1.5$.

The definition of correction factor does not depend on which characteristic we focus on (*e.g.*, iteration time). We can also measure the impact of computation-communication overlap on priority assignment in a similar method. In general, we can always compute the correction factor of one job related to another. The difference between two jobs in iteration length, computation-communication overlap, and maybe other aspects, are included in the correction factor. Note that the correction factor is different when selecting a different job as the reference job, and we will discuss in §7.1.

4.3 Priority Compression

In real-world GPU clusters, NICs and switches typically support limited (*e.g.*, 8) physical priority levels. Some of these levels are already reserved for dedicated usages (*e.g.*, TCP traffic, instruction transmission, and proactive network diagnosis). Thus, priority compression is needed, *i.e.*, some jobs may have to be assigned to the same priority, leading to random communication contention. The challenge lies in how to compress priorities to reduce the GPU utilization loss caused by such random communication contention.

Example. As shown in Figure 13, Job 1-4 have decreasing priorities, and we need to map them onto two priority levels, *i.e.*, high (1) and low (0). Sincronia [16] assigns Job 1 to the high level, and others to the low level. Varys [25] takes a more balanced priority

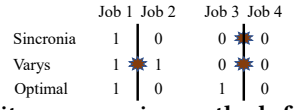


Figure 13: Priority compression methods for existing work and the optimal compression. The explosion mark indicates contention between jobs of the same priority level.

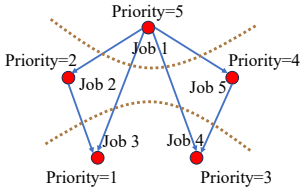


Figure 14: Example for priority compression. The DAG describes communication contention among jobs.

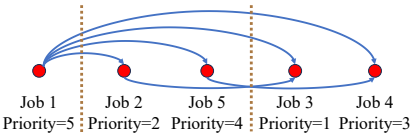


Figure 15: A topological order of DAG in Figure 14. The dotted line shows a corresponding 3-Cut, equivalent to the 3-Cut for DAG in Figure 14.

compression method, assigning Job 1-2 to the high level, and Job 3-4 to the low level. Both strategies are reasonable in the absence of further information. However, if we know that Job 1-2 share the same link, while Job 3-4 share another link, and there is no contention on other links, the optimal priority compression would assign Job 1,3 to the high level, and Job 2,4 to the low level.

Starting from this example, to further reduce the performance loss in priority compression, we also need to consider the GPU intensity of each job, and how seriously they contend over the links with others. Our simulation shows that with careful priority compression, the GPU utilization loss caused by random compression can be significantly reduced (see §4.4).

Our approach. We model the problem of minimizing GPU utilization loss in priority compression as a *K-cut problem* in a directed acyclic graph (DAG) and propose an effective algorithm to solve it.

Problem definition and modeling. A *valid priority compression* can be defined as partitioning job set $\mathbf{J}$ into K subsets $\{\mathbf{J}_1, \mathbf{J}_2, ..., \mathbf{J}_K\}$, with priority level $P(\mathbf{J}_1) > P(\mathbf{J}_2) > ... > P(\mathbf{J}_K)$, where K is the number of available priority levels. Jobs in the same subset have the same priority level. To utilize our priority assignment in §4.2, a valid compression should ensure that, *if two jobs share the same links, the one with a higher priority should be mapped to a priority level that is not lower than the other.*

Crux employs a DAG to model the potential GPU utilization loss for each pair of DLT jobs in priority compression, based on GPU intensity of each job. Define DAG $D =< \mathbf{V}^D, \mathbf{E}^D >$, where each node in $\mathbf{V}^D$ represents a job in $\mathbf{J}$, and each edge in $\mathbf{E}^D$ represents the contention between two jobs. For any two jobs j_1, $j_2 \in \mathbf{V}^D$ which share the same network links, assuming j_1 is assigned a higher priority in §4.2, there will be an edge e from j_1 to j_2. The weight of this edge, w_e or w_{j_1,j_2}, is determined by GPU intensity of j_1, *i.e.*, I_{j_1}. We call D Communication Contention DAG. Figure 14 gives an example. In this case, the optimal solution is mapping Job 1 to a high priority, Job 2&5 to medium priorities, and Job 3&4 to low priorities. Such partition cuts all edges in the DAG.

Algorithm 1: Priority Compression

Input: D =< V^D, E^D >: Communication Contention DAG.
Output: $OutputCut$: a K-Cut for DAG D.

```
for cases ← 1 to m do
    {a_1, a_2, ..., a_n} = RandomTopoOrder(D)
    #Preprocessing matrix C
    for i ← 1 to n do
        for k ← 1 to K do
            S_{i,k} ← S_{i-1,k} + S_{i,k-1} − S_{i-1,k-1} + w_{a_i,a_k}
    for i ← 1 to n do
        for j ← i+1 to n do
            C_{i,j} ← S_{i,j} − S_{i,i}
    #Compute f(n,K) by dynamic programming
    for i ← 1 to n do
        for k ← 1 to K do
            f(i,k) ← max_{g(i−1,k)≤j<i} f(j,k−1) + C_{j,i}
            g(i,k) ← arg max_{g(i−1,k)≤j<i} f(j,k−1) + C_{j,i}
            Cut(i,k) ←
              Cut(g(i,k), k−1) + {a_{g(i,k)+1}, a_{g(i,k)+2}, ..., a_i}
    #Update the current MaxCut
    if f(n,K) > MaxCut then
        MaxCut ← f(n,K)
        OutputCut ← Cut(n,K)
```

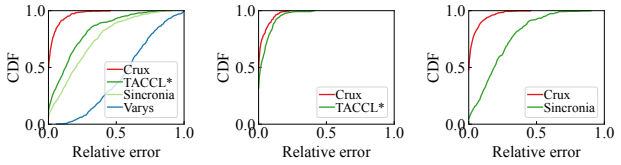
The insight of this model is that, each edge in DAG D describes the GPU utilization loss if the two connected jobs are assigned the same priority level. If there is an edge from j_1 to j_2, it indicates potential communication contention between them. If j_1 and j_2 are mapped to different priority levels, only j_2 suffers from GPU utilization loss since its communication is preempted by j_1. If they are mapped to the same level, both jobs suffer from GPU utilization loss due to contention. The difference between these two cases is whether j_1 loses GPU utilization. Thus, the weight of w_{j_1,j_2} equals to I_{j_1}, since the loss is proportional to j_1's GPU intensity.

Optimization goal. The objective is to minimize GPU utilization loss during compression. Given a valid priority compression for $\mathbf{J}$, the corresponding node set $\mathbf{V}^D$ of DAG D is also partitioned into K subsets. For each edge $e \in \mathbf{E}^D$, if the two corresponding nodes are in the same subset, GPU utilization is lost by w_e. Therefore, the optimal partition should minimize the total weight of links whose nodes are in the same subset. In other words, the goal is to maximize the link cut by K-partitioning.

Thus, our priority compression problem is equivalent to the Max K-Cut problem in DAG: dividing V^D to $V_1, V_2, ..., V_K$, s.t. there is no edge from V_i to V_j for $i > j$. The goal is to maximize the sum of edge weights from V_i to V_j for all $i < j$.

Algorithm: approximating max K-Cut for DAG. Although finding the optimal K-Cut for DAG D is difficult, the problem can be solved in the constrained solution space if we impose constraints on the original problem. We first solve the problem in the constrained solution space, then expend the space to approximate the original solution space, and finally approximate the optimal solution.

Suppose sequence $\{a_n\} = \{a_1, a_2, ..., a_n\}$ is a topological order of DAG D, where $n = |\mathbf{V}^D|$. Then $\{a_n\}$ can be partitioned to K consecutive subsequences, $B_1, B_2, ..., B_K$, where $B_1 = \{a_1, a_2, ..., a_{i_1}\}, ..., B_K =$



(a) Priority assignment of Crux vs. baselines (b) Path selection of Crux vs. TACCL* (c) Priority compression of Crux vs. Sincronia

Figure 16: Relative error of Crux and baselines, compared with the global optimal solution.

$\{a_{i_{K-1}+1}, a_{i_{K-1}+2}, ..., a_n\}$. We call $B_1, B_2, ..., B_K$ a K-Cut of sequence $\{a_n\}$. There is a surjection from $\{a_n\}$'s K-Cut to D's K-Cut, illustrated in Figure 15, proved by Theorem 2 and 3 (Appendix B).

Compared with the Max K-Cut problem in DAG D, it is much easier to find the Max K-Cut for the sequence $\{a_n\}$. Given a topological order $\{a_n\}$ of D, we define $f(i,k)$ as the max k-Cut for $\{a_1, a_2, ..., a_i\}$. Then we have:

$$f(i,k) = \max_{1 \le j < i} f(j, k-1) + C_{j,i} \tag{4}$$

where $C_{j,i}$ denotes the sum of edge weights from $\{a_1, .., a_j\}$ to $\{a_{j+1}, .., a_i\}$ in DAG D.

Then $f(n,K)$ is Max K-Cut for $\{a_1, .., a_n\}$, and can be computed using a dynamic programming algorithm. f has nK states. Constant matrix C can be pre-computed in $O(n^2)$. f's state conversion is $O(n)$, and can be optimized to $O(1)$, as the optimal choice j in $f(i,k)$'s state conversion can be proven to be monotonic with i (We omit the proof due to space limit, see Quadrangle Inequalities [59]). Therefore, given a topological order sequence, we can get its max K-cut in $O(n^2)$.

Back to the original problem in DAG. Since each topological order of D has its own optimal K-Cut, equivalent to the optimal K-Cut in a constraint solution space, we approximate the global Max K-Cut for DAG by sampling enough different topological order sequences of D.

Put it all together, the pseudocode of the priority compression algorithm is shown in Algorithm 1. Crux generates m (in practice we set $m = 10$) random topological orders of D by Breadth First Search [10] (line 1-2), computes the max K-Cut for each topological order (line 3-13), and chooses the maximal one (line 14-16).

4.4 Effectiveness Validation

Due to the complexity of the problem, we can not prove the optimality of Crux mathematically. Instead, we present a microbenchmark to compare each part of Crux design with the optimal scheduling through simulation.

Simulation settings. We construct 1,500 small-scale cases to validate the effectiveness of Crux. In each case, we have at most 20 hosts (each with 8 GPUs), connected by a 2-layer Clos network with 2-4 ToR switches and 2 aggregation switches. We define 5 different DLT jobs running in this cluster and assign it with 3 priority levels. Therefore, we need §4.1 to select paths, §4.2 to assign priorities for each job, and §4.3 to finally compress 5 priorities to 3. In these small-scale cases, we can get the global optimal priority assignment and path selection by enumeration. Then we can compare Crux's scheduling with optimal scheduling.

Baselines. To better visualize our improvement compared with existing methods, we evaluate an intra-job communication scheduler

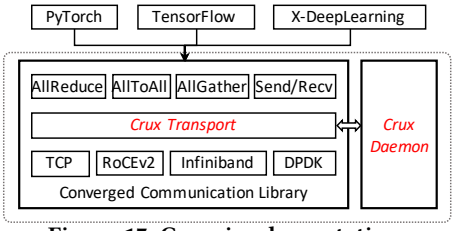


Figure 17: Crux implementation.

(*i.e.*, TACCL [53]). Due to TACCL's specification in intra-job communication, we leverage its key insights and implement TACCL* for inter-job communication scheduling.[3] In priority assignment, we evaluate Sincronia [16] and Varys [25]. In priority compression, we evaluate Sincronias compression algorithm.

To make an ablation study, when evaluating one mechanism (*e.g.*, priority compression), we apply the optimal solution to the other two scheduling mechanisms (*e.g.*, optimal path selection and priority assignment) for all methods.

Performance comparison with optimal. Figure 16 shows the CDF of the error rate from optimal for each solution. Crux provides 97.69%, 97.24%, and 97.12% performance in path selection (§4.1), priority assignment (§4.2), and priority compression (§4.3), compared with optimal. In each part of Crux design, Crux is much closer to the optimal performance compared with other solutions. We discuss the limitations of Crux algorithms in §7.1.

5 IMPLEMENTATION

Figure 17 shows Crux's implementation (7K lines of code) for widely-used DLT frameworks, including PyTorch [5], TensorFlow [15], X-DeepLearning [7]. The converged communication library (CoCoLib) supports RoCEv2 [6], TCP, *etc.*. CoCoLib provides collective communication APIs (*e.g.*, AllReduce) as well as the Send/Receive API, and satisfies various DLT models. Jobs can use Crux by replacing the original communication libraries with CoCoLib.

Crux Daemon (CD) and Crux Transport (CT) are two core modules in Crux, which are deployed in each host in the cluster. CD runs as a daemon process, collecting information and making communication scheduling decisions. CT executes the scheduling decisions. For efficiency, in each job, only a leader CD makes scheduling decisions and synchronizes with others. In the real-world deployment, Crux only costs <0.01% network bandwidth for these information synchronizations and scheduling decisions.

In communication scheduling, Crux requires information on the network topology (*i.e.*, paths) and each active DLT job in the cluster, and then executes scheduling decisions for each job. Now we show that Crux can transparently do these processes, without any modification of DLT model codes.

Path information probing. Crux requires path information for path selection, including both intra-host and inter-host paths. For intra-host paths (*e.g.*, PCIe) paths, Crux directly uses existing tools (Intel PCM [3] and AMD uProf [1]). For inter-host paths, Crux collects path information between each pair of hosts by sending probing packets. Specifically, considering that GPU clusters typically use ECMP [2] to forward flows through multiple candidate paths by hashing the five-tuple, we need to find a suitable 16-bit UDP

[3]Based on TACCL's insight on routing and scheduling, TACCL* selects the least congested link for each job and prioritizes the traffic with longer transmission distances.

source port for each candidate path. To achieve this, we can send probing packets with varied source ports until all candidate paths can be reached. In Crux, we employ INT [39] (widely deployed in current switches [18, 43, 56]) to insert per-hop information into the probing packets. Note that INT is not imperative, and we can obtain path information through alternative methods, such as calculation based on the hash algorithm.

Job information measurement. Crux requires computation overload W_j and communication overload t_j of each job j to compute GPU intensity (Equation 2). To ensure accurate measurement of computation and communication overload, Crux assigns a unique highest priority to a job during profiling. This prevents this job from contending with others. Then, Crux utilizes hardware monitoring (*e.g.*, GPU, NIC, and PCIe), to measure computation and communication overloads. For computation overload, Crux directly sums up the GPU overload during a fixed monitoring period (*e.g.*, 30s). For communication overload, Crux sums up the duration of data transfers. Both overloads are divided by the number of iterations within that period to finally get the per-iteration overload W_j and t_j. Crux employs a mathematical speculation to estimate the duration of one iteration. Given that the communication pattern of a job is consistent across iterations, Crux applies the Fourier Transform to convert the communication from the time domain to the frequency domain and then estimates the duration of a single iteration.

Execute scheduling decisions. Each time a new job arrives, Crux first conducts the above probing and measurement processes for the new job and then reassigns paths and priorities for all existing jobs. To execute path selection, CTs invoke *ibv_modify_qp* to set UDP source ports of RoCEv2 connections, and the packets will be delivered via certain paths based on ECMP. To execute priority assignment in inter-host networks, CTs set the IP traffic classes corresponding to the assigned priority level via *ibv_modify_qp* to enter different packet queues of NIC and switches. To execute intra-host priority assignment, CDs maintain semaphores for PCIe links, and block PCIe communication of lower-priority jobs when higher-priority jobs use the PCIe links. CTs access the semaphores via inter-process shared memory. If a job is rescheduled (*e.g.*, using new resources) or finishes, Crux treats this as a new status and reassigns paths and priorities to the remaining jobs. Considering that the duration of a DLT job can range from hours to months, the overhead of profiling and re-scheduling communication, which takes less than one minute for each job's arrival or completion, is considered negligible.

6 EVALUATION

In this section, we show that:

- In a 96-GPU testbed, Crux improves GPU utilization by up to 14.8%, and reduces up to 33% job completion time (JCT) for GPU-intensive jobs.
- In the 2,000-GPU production trace-based simulation, Crux improves GPU utilization by 4% to 23% compared with state-of-the-arts under various network architectures.
- As a communication scheduler, Crux is compatible with state-of-the-art job schedulers, and further improves GPU utilization and DLT performance by 11% to 23%.

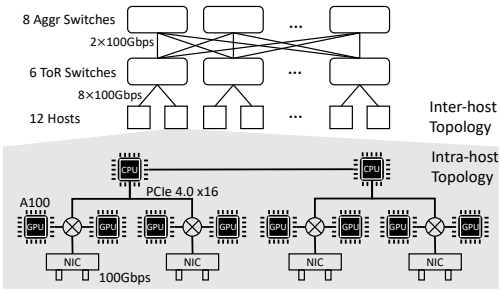
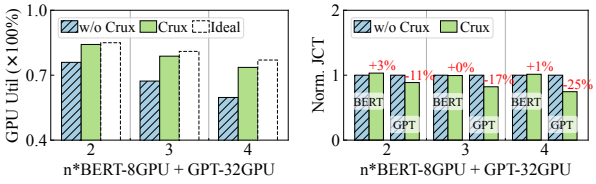
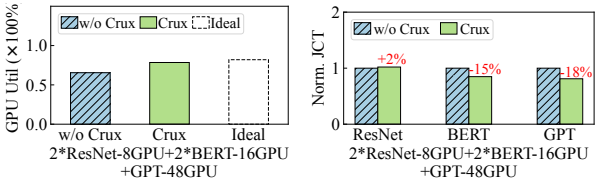


Figure 18: The testbed is composed of 12 hosts, each with 8 Nvidia A100 GPUs and 4×200Gbps RDMA NIC. Hosts are connected through a two-layer Clos network. Besides PCIe links, GPUs in the same hosts are also connected via NVLinks that are not drawn for brevity.



(a) Overall average GPU Utilization (b) Per-job end-to-end performance

Figure 19: Contention on network paths between a GPT model and multiple BERT models.



(a) Overall average GPU Utilization (b) Per-job end-to-end performance

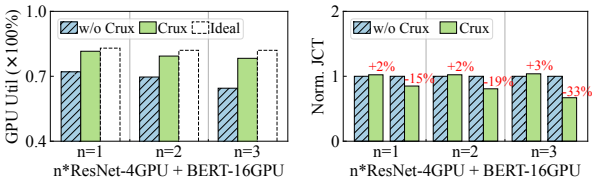
Figure 20: Contention on network paths among a GPT model, two BERT models, and two ResNet models.

6.1 Experiment Setup

Real-world testbed. In our testbed shown in Figure 18, each host (with eight GPUs) is connected to one ToR switch via four links, with every two GPUs connected to one switch via a shared link (*e.g.*, GPU 0&1 connects to switch 1 via link 1, GPU 2&3 connects to switch 2 via link 2). If GPUs of different hosts need to communicate, as they may not be connected to the same ToR switch, they would require communication through aggregation switches.

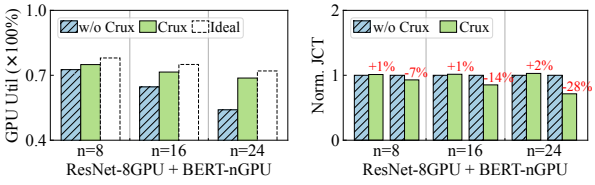
Simulated GPU cluster. We implemented a simulator for GPU cluster networks to simulate the computation and communication processes of multiple jobs. For computation simulation of a DLT job, we directly obtain the per-iteration computation time of by running models on actual GPUs. For communication simulation, we use the alpha-beta model [34]. This model considers the transmission delay over a link to include both the physical link delay and the delay associated with the data size and bandwidth. The simulator assumes 8 priority levels are available.

Using our simulator, we conduct a large-scale simulation based on the production trace collected from 2,000+ GPUs for two weeks (described in §2.2), and apply two network topologies: (1) Double-sided topology, consisting of 6 ToR switches, 12 aggregation switches, and 32 core switches. Each host is connected to two ToR switches



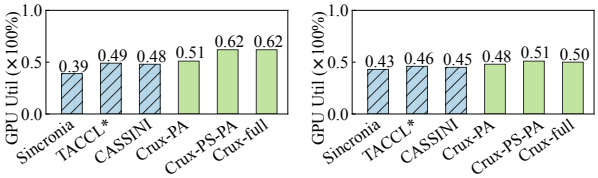
(a) Overall average GPU Utilization (b) Per-job end-to-end performance

Figure 21: Contention on PCIe between a BERT model and multiple ResNet models.



(a) Overall average GPU Utilization (b) Per-job end-to-end performance

Figure 22: Contention on PCIe between a ResNet model and a BERT model with varying number of GPUs.



(a) Clos topology (b) Double-sided topology

Figure 23: Comparison of average GPU utilization across different communication schedulers.

via eight links. It is exactly the actual topology used in the trace. (2) Two-layer Clos topology, consisting of 173 ToR switches and 16 aggregation switches. Each host is connected to one ToR.

Performance metrics. As the optimization goal of this paper, we evaluate the overall GPU utilization as the performance metric of the GPU cluster. To show how each individual job performs, we also evaluate the end-to-end training performance, *i.e.*, the job completion time (JCT, which is inversely proportional to throughput).

6.2 Real-World Evaluation

In the real-world testbed, we co-locate multiple DLT jobs with and without Crux. Each job runs a ResNet, BERT, or GPT model, which represents small, medium, and large DLT jobs, respectively. To better visualize Crux's improvement, we also evaluate the training performance of each DLT job when running alone, regarded as ideal training performance.

Contention on network paths. We co-locate a 32GPU GPT job with multiple 8GPU BERT jobs, producing communication contention on network paths. Figure 19(a) indicates that as communication contention increases, GPU utilization decreases. However, with Crux, overall GPU utilization is improved by 8.3% to 12.9%, and is close to the ideal case. Figure 19(b) analyzes this performance gain by showing the job completion time (JCT) of each job. Based on Crux's key idea, GPT has higher GPU intensity than BERT, and should be prioritized in communication scheduling. With Crux scheduling, GPT gets much shorter JCT (-11% to -25%) at the expense of a small increase in BERT's JCT (+0% to +3%). The increase in BERT's JCT is not significant. This is because Crux can mitigate some contention through effective path selection. Since GPT is more important than BERT, this reasonable trade-off significantly improves the overall GPU utilization.

We also co-locate a 48GPU GPT job with two 8GPU ResNet jobs and two 16GPU BERT jobs. In this case, GPT has the highest GPU intensity and ResNet has the lowest. Figure 20(b) shows that with Crux scheduling, GPT's JCT decreases by 18%, BERT's JCT decreases by 15%, and ResNet's JCT increases by 2% to reserve network bandwidth for other two models. As a whole, with Crux, the GPU utilization increases by 13.9% (Figure 20(a)).

Contention on PCIe. We produce two PCIe contention cases in Figure 3(b). The first case co-locates a 16GPU BERT job with multiple 4GPU ResNet jobs, shown in Figure 19. The second case co-locates the 8GPU ResNet with a BERT in varying GPU numbers (8, 16, and 24), shown in Figure 20. Figures 19(a) and 20(a) show that in both cases Crux gets higher GPU utilization (+9.5% to +14.8%), and is close to the ideal results. Figures 19(b) and 20(b) show that with Crux scheduling, BERT's job completion time (JCT) is significantly decreased (-7% to -33%) due to its high GPU intensity, and ResNet's JCT is slightly increased (+1% to +3%).

6.3 Trace-based Simulation

We compare Crux with different communication schedulers, including a general co-flow scheduler (Sincronia [16]), an intra-job communication scheduler (TACCL* [53] described in §4.4), and an inter-job communication scheduler (CASSINI [51]). For Crux, we also separately evaluate the performance under three combinations: using only priority assignment (Crux-PA), combining path selection with priority assignment (Crux-PS-PA), and integrating priority assignment and compression and path selection (Crux-full).

In the simulation, 11 different models are evaluated, including five open-source models (BERT, GPT [13], ResNet, NMT [54], and Multi-Interests [40]) and their five variants, along with two in-house models for Click-Through-Rate and transformer-based NLP.

Performance under different GPU clusters. Figure 23 shows the performance comparison in a two-layer Clos network and a three-layer double-sided network. Compared with baselines, Crux improves GPU utilization by 13% to 23% and 4% to 7%, respectively.

GPU intensity in the network *vs.* GPU utilization. To prove that Crux's performance gain comes from the better scheduling for GPU-intensive jobs (§4), we zoom in the detail of the simulation in Clos topology (Figure 23(a)). Figure 24 presents the real-time distribution of jobs' GPU intensity whose data flow is transmitted in the network. Dark color represents the data flow of jobs with higher GPU intensity and light color represents data flow of jobs with low GPU intensity. The white color indicates that a part of network links are idle. The GPU intensity distribution in intra-host PCIe-NIC links, NIC-ToR switch links, and ToR-Aggregation switch links is presented separately. We show the real-time GPU utilization of the whole cluster with the same timeline.

First, we compare Sincronia (Figure 24(a)), TACCL* (Figure 24(b)), CASSINI (Figure 24(c)), and Crux-PA (Figure 24(d)). We find that the GPU intensity distribution of Crux-PA has more dark colors, since Crux's priority assignment (§4.2) tries to prioritize GPU-intensive jobs during communication contention. Focusing on the solid rounded rectangles in Figure 24(a),24(b), 24(c) and 24(d), on the

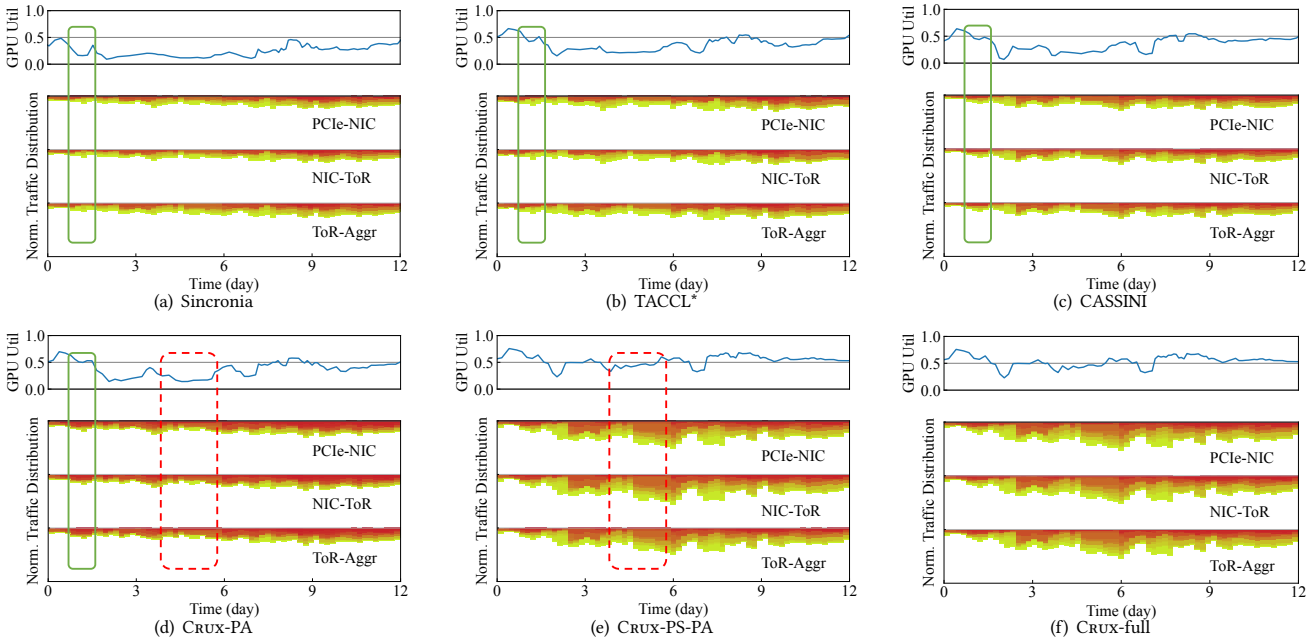


Figure 24: The real-time distribution of jobs' GPU intensity whose data flow is being transmitted in the network. A darker color represents higher GPU intensity.

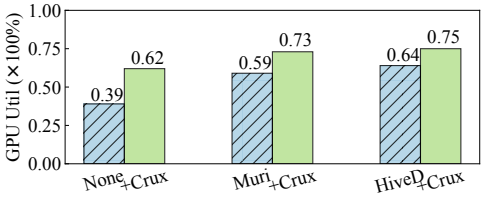


Figure 25: GPU utilization comparison between using job schedulers alone and combining them with Crux.

first day of the trace, the baselines' data flow distribution is generally in light color, and Crux's color is much darker, corresponding to 26%, 14%, and 5% average GPU utilization improvement.

Second, we compare Crux-PA (Figure 24(d)) and Crux-PS-PA (Figure 24(e)). We find that Crux-PS-PA has a much larger area with a non-white color, indicating that network paths are better utilized with path selection (§4.1). Focusing on the dashed rounded rectangles in Figures 24(d) and 24(e), the increase in the non-white area brings a 97% increase in network utilization.

Third, we compare Crux-PS-PA (Figure 24(e)) and Crux-full (Figure 24(f)). Their GPU intensity distribution is highly consistent. This indicates that although the priorities of 5,000 jobs are compressed to only eight levels, Crux's compression algorithm (§4.3) significantly reduces the performance loss, introducing nearly no side-effect on overall GPU utilization.

Putting the above observations together, we prove that Crux avoids potential communication contention by path selection, and prioritizes GPU-intensive jobs, achieving higher GPU utilization.

6.4 Working Together with Job Schedulers

As explained in §1, Crux is orthogonal to job schedulers, and can work together with them. Figure 25 shows how Crux performs with and without job scheduling. We evaluate two job schedulers, HiveD [61] and Muri [63]. HiveD allocates GPUs based on physical affinity to minimize communication overhead, while Muri schedules jobs by reducing idle links. Compared to not performing job scheduling ("None" in Figure 25), Muri and HiveD improve GPU utilization by 20% and 25%, respectively. Together with Crux, GPU utilization can be further improved by 14% and 11%.

The result shows that, with state-of-the-art job schedulers, there still exists communication contention between DLT jobs which degrades GPU utilization, so there is still sufficient room for Crux to produce performance gain.

7 DISCUSSIONS

7.1 Limitations in Crux Algorithm

Although the definition of GPU intensity helps us greatly improve the communication scheduling decisions, there is still a gap between our algorithm (§4) to the optimal. The gap mainly comes from the complex communication pattern of DLT jobs, and Crux makes several simplifications in §4.2.

First, §4.2 assumes the communication and computation are both continuous in one iteration of DLT job, and their overlap is of a simple pattern, like Figure 12. In actual DLT jobs, computation and communication may occur alternately many times, some communication kernels can be partially overlapped by computation and some are not, generating complex overlap patterns. Fortunately, we find that the most important factor affecting communication scheduling is the overlap ratio, *i.e.*, the proportion of overlap between computation and communication. Jobs with less overlap between computation and communication are usually much more sensitive to communication latency. So we believe that our assumption is sufficient to reflect the overlap ratio.

Second, to compute the correction factor for each DLT job, Crux only chooses the job with the most network traffic as the reference job. Mathematically, the reference job should be all possible combinations of jobs which have communication contention with the given job, rather than one job. Thus, a better solution should be enumerating all other job combinations as reference jobs, computing a correction factor based on each of them, and making a weighted average based on the frequency of communication contention between them. This requires exponential computational complexity and fine-grained network monitoring, which is not practical in real-world cluster networks. Instead, we simplify the computation of correction factor for easy-deployment.

Third, the communication performance of DLT jobs is also related to some other factors, such as storage traffic, the timing of each job's communication, and specific collective communication algorithms. For example, regular communication traffic may be mixed with storage-related traffic, such as checkpointing or dataset loading. Storage traffic may vary across different iterations, and the actual measurement of t_j (Equation 2) may be affected by storage traffic. Fortunately, modern GPU clusters typically adopt a compute/storage separation architecture, and the impact of storage traffic on performance tends to be limited. Our evaluation also proves that Crux can make scheduling decisions close to the optimal.

7.2 Job Fairness in Scheduling

The goal of Crux is to optimize GPU utilization for a cluster, which means that some jobs have to sacrifice their performance for more GPU-intensive jobs (usually involving more GPUs). For a GPU cluster provider, GPU-intensive jobs should be treated more carefully (which is what we have been doing all along) because the customer pays significantly more money to rent GPUs for them. We acknowledge that this could be detrimental to fairness, but for a GPU cluster provider, this weakness is far outweighed by the benefits brought about by optimizing GPU utilization.

Based on our evaluation, although the performance of some jobs is reduced due to Crux's scheduling, no job is starved. For example, jobs with the lowest priority experience a 55.5% decrease in training throughput in §6.3, instead of a complete halt in training. This is because the traffic patterns of DLT jobs are periodic and bursty, which means that the network links are idle for a significant portion of the time. As a result, it is impossible for any jobs to be completely starved of resources and unable to communicate. Instead, they just experience delayed communication during contention.

Crux can be easily extended to also consider fairness if one really wants to make a trade-off. For example, we can calculate a weighted average of GPU intensity and the recent decrease in throughput for each job due to communication contention as the final priority assignment, or compute a Pareto-optimal frontier.

7.3 Adaptability to Other Topologies

In addition to the two topologies evaluated in §6, Crux has the potential to be adapted to other topologies. First, Crux schedules communication based on GPU intensity, an inherent characteristic of DLT jobs, which is independent of network topologies. Compared with joint optimization that considers more factors like network topology, although our topology-independent design may sacrifice some potential gains, it provides significant simplicity for real-world deployment. Thus, Crux can be applied to any topology. Second, the Clos and double-sided topologies evaluated in this paper are representative of multi-layered cluster architecture. Although there are other less commonly deployed topologies, such as Torus, they also face the communication contention problem in multi-tenant scenarios. Therefore, Crux has the potential to improve GPU utilization across various topologies.

8 RELATED WORK

Job scheduler. A series of work [20, 21, 30, 31, 33, 46, 57, 58, 63, 64] focus on optimizing the overall training performance by scheduling GPU for different DLT jobs. Crux is orthogonal to these methods, which targets a similar object but does not touch the high-level arrangements of jobs. Crux can also work together with them.

Communication scheduler. Communication schedulers can be divided into three categories. First, general co-flow schedulers solve the mathematical problem of multi-flow scheduling, including path selection and priority assignment. Coflow [22], Orchestra [24], Aalo [23], Varys [25], Sincronia [16], Weaver [35], PIAS [17], Amoeba [60], Karuna [19], and Baraat [26] propose solutions for optimizing the average completion time or minimizing job tardiness (*i.e.*, enable more jobs to meet their deadlines). Since they focus on general data flow, they are unaware of how different co-flows affect the end-to-end DLT performance or the GPU utilization.

Second, intra-job communication schedulers target optimizing multiple flows in one DLT job. BytePS [37], ATP [41], ByteScheduler [50], MXDAG [55], EchelonFlow [48], CadentFlow [38], Lina [42], SYNDICATE [47], BGL [45], Janus [29], ACCL [28], and TACCL [53] allocate different priorities or different paths to flows in a single DLT job. These methods utilize the job's flow pattern to optimize scheduling decisions, and are unaware of communication contention among multiple jobs. Crux is orthogonal to these methods, as Crux focuses on the inter-job contention problem.

Third, CASSINI [51] is an inter-job communication scheduler that aims to reuse different network links over time by assigning a time-dimension offset to each job. However, dynamic networks and jobs can affect the actual traffic pattern of each job. Thus, merely setting a time offset for each job cannot eliminate communication contention among jobs.

9 CONCLUSION

This paper presents Crux, a communication scheduler that tackles inter-job communication contention and optimizes cluster GPU utilization. Crux proposes the concept of GPU intensity, and reduces the GPU utilization problem to a flow optimization problem. To approximate the optimal flow scheduling in real-world multi-tenant GPU clusters, Crux proposes GPU intensity-based path selection, priority assignment, and priority compression for DLT jobs. Our evaluation demonstrates that compared with state-of-the-art, Crux improves the cluster GPU utilization by up to 23%.

Ethics. This work does not raise any ethical issues.

Acknowledgements

We thank our shepherd Sujata Banerjee, and the SIGCOMM reviewers for their insightful comments. We thank Yu Zhou for implementing the earliest version of Crux. We also thank Xin Jin and Minlan Yu for their valuable feedback on earlier drafts of this paper. Ennan Zhai is the corresponding author.

References

[1] 2022. AMD uProf. https://www.amd.com/en/developer/uprof.html.

[2] 2022. Equal-cost multi-path routing (ECMP). https://en.wikipedia.org/wiki/Equal-cost_multi-path_routing.

[3] 2022. Intel Performance Counter Monitor. https://github.com/intel/pcm.

[4] 2022. NVIDIA Collective Communications Library (NCCL). https://developer.nvidia.com/nccl.

[5] 2022. PyTorch. https://pytorch.org/.

[6] 2022. RDMA over Converged Ethernet. https://en.wikipedia.org/wiki/RDMA_over_Converged_Ethernet.

[7] 2022. X-DeepLearning. https://github.com/alibaba/x-deeplearning.

[8] 2023. Adobe Firefly. https://www.adobe.com/sensei/generative-ai/firefly.html.

[9] 2023. Alibaba GPU Cluster Trace 2023. https://github.com/alibaba/alibaba-lingjun-dataset-2023.

[10] 2023. Breadth-first search. https://en.wikipedia.org/wiki/Breadth-first_search.

[11] 2023. Github Copilot. https://github.com/features/copilot.

[12] 2023. Microsoft365. https://www.microsoft.com/en-us/microsoft-365.

[13] 2024. Megatron GPT3 MODEL. https://github.com/NVIDIA/Megatron-LM/tree/main/examples/gpt3.

[14] 2024. Multi-commodity flow problem. https://en.wikipedia.org/wiki/Multi-commodity_flow_problem.

[15] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *OSDI*.

[16] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David B. Shmoys, and Amin Vahdat. 2018. Sincronia: near-optimal network design for coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*. ACM, 16–29. https://doi.org/10.1145/3230543.3230569

[17] Wei Bai, Kai Chen, Hao Wang, Li Chen, Dongsu Han, and Chen Tian. 2015. Information-Agnostic Flow Scheduling for Commodity Data Centers. In *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*. USENIX Association, 455–468. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/bai

[18] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minlan Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic In-band Network Telemetry. In *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*. ACM, 662–680. https://doi.org/10.1145/3387514.3405894

[19] Li Chen, Kai Chen, Wei Bai, and Mohammad Alizadeh. 2016. Scheduling Mix-flows in Commodity Datacenters with Karuna. In *Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016*. ACM, 174–187. https://doi.org/10.1145/2934872.2934888

[20] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. (2016), 681–696. https://doi.org/10.1145/2872362.2872368

[21] Yangrui Chen, Yanghua Peng, Yixin Bao, Chuan Wu, Yibo Zhu, and Chuanxiong Guo. 2020. Elastic parameter server load distribution in deep learning clusters. In *SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, October 19-21, 2020*. ACM, 507–521. https://doi.org/10.1145/3419111.3421307

[22] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: a networking abstraction for cluster applications. In *11th ACM Workshop on Hot Topics in Networks, HotNets-XI, Redmond, WA, USA - October 29 - 30, 2012*. ACM, 31–36. https://doi.org/10.1145/2390231.2390237

[23] Mosharaf Chowdhury and Ion Stoica. 2015. Efficient Coflow Scheduling Without Prior Knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*. ACM, 393–406. https://doi.org/10.1145/2785956.2787480

[24] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I. Jordan, and Ion Stoica. 2011. Managing data transfers in computer clusters with orchestra. In *Proceedings of the ACM SIGCOMM 2011 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Toronto, ON, Canada, August 15-19, 2011*. ACM, 98–109. https://doi.org/10.1145/2018436.2018448

[25] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with Varys. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*. ACM, 443–454. https://doi.org/10.1145/2619239.2626315

[26] Fahad R. Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony I. T. Rowstron. 2014. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*. ACM, 431–442. https://doi.org/10.1145/2619239.2626322

[27] Jianbo Dong, Zheng Cao, Tao Zhang, Jianxi Ye, Shaochuang Wang, Fei Feng, Li Zhao, Xiaoyong Liu, Liuyihan Song, Liwei Peng, Yiqun Guo, Xiaowei Jiang, Lingbo Tang, Yin Du, Yingya Zhang, Pan Pan, and Yuan Xie. 2020. EFLOPS: Algorithm and System Co-Design for a High Performance Distributed Training Platform. In *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 610–622. https://doi.org/10.1109/HPCA47549.2020.00056

[28] Jianbo Dong, Shaochuang Wang, Fei Feng, Zheng Cao, Heng Pan, Lingbo Tang, Pengcheng Li, Hao Li, Qianyuan Ran, Yiqun Guo, Shanyuan Gao, Xin Long, Jie Zhang, Yong Li, Zhisheng Xia, Liuyihan Song, Yingya Zhang, Pan Pan, Guohui Wang, and Xiaowei Jiang. 2021. ACCL: Architecting Highly Scalable Distributed Training Systems With Highly Efficient Collective Communication Library. *IEEE Micro* 41, 5 (2021), 85–92. https://doi.org/10.1109/MM.2021.3091475

[29] Adam Dunkels, Richard Gold, Sergio Angel Marti, Arnold Pears, and Mats Uddenfeldt. 2005. Janus: An Architecture for Flexible Access to Sensor Networks. In *Proceedings of the 1st ACM Workshop on Dynamic Interconnection of Networks* (Cologne, Germany) *(DIN '05)*. Association for Computing Machinery, New York, NY, USA, 48–52. https://doi.org/10.1145/1080776.1080792

[30] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. (2014), 455–466. https://doi.org/10.1145/2619239.2626334

[31] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. USENIX Association, 81–97. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_graphene

[32] Albert G. Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Barcelona, Spain, August 16-21, 2009*. ACM, 51–62. https://doi.org/10.1145/1592568.1592576

[33] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Harry 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 2019, Boston, MA, February 26-28, 2019*. USENIX Association, 485–500. https://www.usenix.org/conference/nsdi19/presentation/gu

[34] Roger W. Hockney. 1994. The Communication Challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.* 20, 3 (1994), 389–398. https://doi.org/10.1016/S0167-8191(06)80021-9

[35] Xin Sunny Huang, Yiting Xia, and T. S. Eugene Ng. 2020. Weaver: Efficient Coflow Scheduling in Heterogeneous Parallel Networks. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), New Orleans, LA, USA, May 18-22, 2020*. IEEE, 1071–1081. https://doi.org/10.1109/IPDPS47924.2020.00113

[36] 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 2019, Renton, WA, USA, July 10-12, 2019*. USENIX Association, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[37] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[38] Sangeetha Abdu Jyothi, Sayed Hadi Hashemi, Roy H. Campbell, and Brighten Godfrey. 2020. Towards An Application Objective-Aware Network Interface. In *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020*. USENIX Association. https://www.usenix.org/conference/hotcloud20/presentation/jyothi

[39] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band network telemetry via programmable dataplanes. In *SIGCOMM*.

[40] Taesup Kim, Inchul Song, and Yoshua Bengio. 2017. Dynamic Layer Normalization for Adaptive Neural Acoustic Modeling in Speech Recognition. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*. ISCA, 2411–2415. https://doi.org/10.21437/INTERSPEECH.2017-556

[41] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M. Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*. USENIX Association, 741–761. https://www.usenix.org/conference/nsdi21/presentation/lao

[42] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. 2023. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference, USENIX ATC 2023, Boston, MA, USA, July 10-12, 2023*. USENIX Association, 945–959. https://www.usenix.org/conference/atc23/presentation/li-jiamin

[43] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: high precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019, Beijing, China, August 19-23, 2019*. ACM, 44–58. https://doi.org/10.1145/3341302.3342085

[44] Kefei Liu, Zhuo Jiang, Jiao Zhang, Haoran Wei, Xiaolong Zhong, Lizhuang Tan, Tian Pan, and Tao Huang. 2023. Hostping: Diagnosing Intra-host Network Bottlenecks in RDMA Servers. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 15–29. https://www.usenix.org/conference/nsdi23/presentation/liu-kefei

[45] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. 2023. BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 103–118. https://www.usenix.org/conference/nsdi23/presentation/liu-tianfeng

[46] 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 2020, Santa Clara, CA, USA, February 25-27, 2020*. USENIX Association, 289–304. https://www.usenix.org/conference/nsdi20/presentation/mahajan

[47] Kshiteej Mahajan, Ching-Hsiang Chu, Srinivas Sridharan, and Aditya Akella. 2023. Better Together: Jointly Optimizing ML Collective Scheduling and Execution Planning using SYNDICATE. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 809–824. https://www.usenix.org/conference/nsdi23/presentation/mahajan

[48] Rui Pan, Yiming Lei, Jialong Li, Zhiqiang Xie, Binhang Yuan, and Yiting Xia. 2022. Efficient flow scheduling in distributed deep learning training with echelon formation. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets 2022, Austin, Texas, November 14-15, 2022*. ACM, 93–100. https://doi.org/10.1145/3563766.3564096

[49] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distributed Comput.* 69, 2 (2009), 117–124. https://doi.org/10.1016/J.JPDC.2008.09.002

[50] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*. ACM, 16–29. https://doi.org/10.1145/3341301.3359642

[51] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. (2024). https://www.usenix.org/conference/nsdi24/presentation/rajasekaran

[52] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. 2022. Congestion control in machine learning clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets 2022, Austin, Texas, November 14-15, 2022*. ACM, 235–242. https://doi.org/10.1145/3563766.3564115

[53] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 593–612. https://www.usenix.org/conference/nsdi23/presentation/shah

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[55] Weitao Wang, Sushovan Das, Xinyu Crystal Wu, Zhuang Wang, Ang Chen, and T. S. Eugene Ng. 2021. MXDAG: A Hybrid Abstraction for Cluster Applications. *CoRR* abs/2107.07442 (2021). arXiv:2107.07442 https://arxiv.org/abs/2107.07442

[56] Weitao Wang, Masoud Moshref, Yuliang Li, Gautam Kumar, T. S. Eugene Ng, Neal Cardwell, and Nandita Dukkipati. 2023. Poseidon: Efficient, Robust, and Practical Datacenter CC via Deployable INT. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 255–274. https://www.usenix.org/conference/nsdi23/presentation/wang-weitao

[57] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*. USENIX Association, 595–610. https://www.usenix.org/conference/osdi18/presentation/xiao

[58] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 533–548. https://www.usenix.org/conference/osdi20/presentation/xiao

[59] F. Frances Yao. 1980. Efficient Dynamic Programming Using Quadrangle Inequalities. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*. ACM, 429–435. https://doi.org/10.1145/800141.804691

[60] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. 2015. Guaranteeing deadlines for inter-datacenter transfers. In *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*. ACM, 20:1–20:14. https://doi.org/10.1145/2741948.2741957

[61] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis C. M. Lau, Yuqi Wang, Yifan Xiong, and Bin Wang. 2020. HiveD: Sharing a GPU Cluster for Deep Learning with Guarantees. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 515–532. https://www.usenix.org/conference/osdi20/presentation/zhao-hanyu

[62] Mark Zhao, Niket Agarwal, Aarti Basant, Bugra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*. ACM, 1042–1057. https://doi.org/10.1145/3470496.3533044

[63] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. 2022. Multi-resource interleaving for deep learning training. In *SIGCOMM '22: ACM SIGCOMM 2022 Conference, Amsterdam, The Netherlands, August 22 - 26, 2022*. ACM, 428–440. https://doi.org/10.1145/3544216.3544224

[64] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*. USENIX Association, 703–723. https://www.usenix.org/conference/nsdi23/presentation/zheng

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A PROOF OF THEOREM 1

Below we prove Theorem 1 in §3.2:

$\lim_{|T|\to\infty} \frac{F_T}{U_T} = 1$,

where U_T denotes the GPU utilization of the GPU cluster over the time interval T, and F_T represents the cumulative GPU intensity of all jobs transmitted by a link within the time interval.

Proof. We define the function $g_j(t)$ for a job j at time t as follows:

$$g_j(t) = \begin{cases} I_{h(t)} & \text{if } h(t) = j, \\ 0 & \text{if } h(t) \neq j. \end{cases}$$

Let $G_{j,T} = \int_{t\in T} g_j(t)\, dt$. Hence, we have $F_T = \sum_{j\in \mathbf{J}} G_{j,T}$, where $\mathbf{J}$ is the set of all jobs.

Let S_j represent the time spent transmitting the data flow of job j, yielding $G_{j,T} = I_j S_j$. Assuming it takes t_j time to transmit one iteration of job j's data, S_j can cover $N_j = S_j/t_j$ iterations (not necessarily an integer).

For time duration T, let job j complete computation for N_j' iterations (also not necessarily an integer). As illustrated in Figure 26, in a DLT job training scenario, computation and communication for an iteration always occur in pairs, regardless of whether they are overlapped. Irrespective of T's starting point, the discrepancy between the iterations for computation and communication within T will not exceed 1 (whether it is +1 or -1). This leads us to the following inequality:

$$N_j - 1 \leq N_j' \leq N_j + 1 \tag{5}$$

By replacing N_j by S_j/t_j in Equation (5), we obtain:

$$\frac{S_j}{t_j} - 1 \leq N_j' \leq \frac{S_j}{t_j} + 1 \tag{6}$$

Since the computation workload on the GPU cluster is equal to the sum of the workloads of all jobs, we can express U_T as follows:

$$U_T = \sum_{v\in \mathbf{V}} L_v = \sum_{j\in \mathbf{J}} W_j N_j' \tag{7}$$

where L_v represents the computation workload of GPU v in the time duration T, and W_j represents the per-iteration GPU workload of job j, as defined in §3.1.

Substituting the bounds for N_j' from Inequality (6) into Equation (7), we have:

$$\sum_{j\in \mathbf{J}} W_j \left(\frac{S_j}{t_j} - 1\right) \leq U_T \leq \sum_{j\in \mathbf{J}} W_j \left(\frac{S_j}{t_j} + 1\right) \tag{8}$$

Since $\sum_{j\in \mathbf{J}} \frac{W_j S_j}{t_j} = \sum_{j\in \mathbf{J}} I_j S_j = \sum_{j\in \mathbf{J}} G_{j,T} = F_T$, Inequality (8) simplifies to:

$$F_T - \sum_{j\in \mathbf{J}} W_j \leq U_T \leq F_T + \sum_{j\in \mathbf{J}} W_j \tag{9}$$

$$1 - \frac{\sum_{j\in \mathbf{J}} W_j}{U_T} \leq \frac{F_T}{U_T} \leq 1 + \frac{\sum_{j\in \mathbf{J}} W_j}{U_T} \tag{10}$$

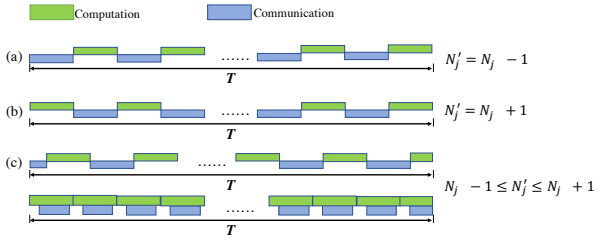


Figure 26: Numerical relationship between N_j and N_j'.

From Inequality (10), as $|T| \to \infty$, U_T tends to infinity because there is always jobs running on GPUs. Therefore, we obtain:

$$1 = \lim_{|T|\to\infty} \left(1 - \frac{\sum_{j\in \mathbf{J}} W_j}{U_T}\right) \leq \lim_{|T|\to\infty} \frac{F_T}{U_T} \leq \lim_{|T|\to\infty} \left(1 + \frac{\sum_{j\in \mathbf{J}} W_j}{U_T}\right) = 1 \tag{11}$$

Equation (11) yields the result that $\lim_{|T|\to\infty} \frac{F_T}{U_T} = 1$. □

B PROOF OF THEOREMS IN §4.3

Below we prove that there is a surjection from the K-Cut of a topological order to the K-Cut of its corresponding DAG in §4.3.

Theorem 2. *Any K-Cut of sequence $\{a_n\}$ is a valid K-Cut for the directed acyclic graph (DAG) D.*

Proof. By the definition of a topological ordering, there cannot exist an edge from a_j to a_i for $i < j$. Consequently, for all $1 \leq i < j \leq K$, there are no edges from nodes in B_j to nodes in B_i. Therefore, the partition $B_1, B_2, \ldots, B_K$ forms a valid K-Cut of the DAG D. □

Theorem 3. *For any valid K-Cut for D, there exists at least one topological order sequence $\{a_n\}$ with a K-Cut on it, s.t. this K-Cut of $\{a_n\}$ is equivalent to K-Cut on D.*

Proof. Assume $\{V_1, V_2, \ldots, V_K\}$ represents a K-Cut of D. Define the sub-DAG $D_i =< V_i, E_i >$, where E_i includes all edges with both nodes in V_i. Let $B_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,|V_i|}\}$ be a topological order of D_i. According to the definition of a valid K-Cut for D, there are no edges in E^D from any node in B_i to any node in B_j for $i > j$. Therefore, the concatenated sequence $\{b_{1,1}, b_{1,2}, \ldots, b_{1,|V_1|}, \ldots, b_{K,1}, b_{K,2}, \ldots, b_{K,|V_K|}\}$ forms a topological ordering of D, and the partition $\{B_1, B_2, \ldots, B_K\}$ is a K-Cut on it, equivalent to the K-Cut $\{V_1, \ldots, V_K\}$. □