

Isolated Scheduling for Distributed Training Tasks in GPU Clusters

Xinchi Han¹, Weihao Jiang¹, Peirui Cao¹, Qinwei Yang¹,
Yunzhuo Liu¹, Shuyao Qi¹, Shengkai Lin¹, Shizhen Zhao^{1*}

¹Shanghai Jiao Tong University

Abstract

Distributed machine learning (DML) technology makes it possible to train large neural networks in a reasonable amount of time. Meanwhile, as the computing power grows much faster than network capacity, network communication has gradually become the bottleneck of DML. Current multi-tenant GPU clusters face network contention caused by hash-collision problem which not only further increases the overhead of communication, but also creates unfairness and affects the user experience. In this paper, we firstly analyse how network contention affects the training time in a cluster with 32 NVIDIA V100 GPUs. Then we propose vClos to eliminate network contention by jointly optimizing network topology and communication pattern in distributed training. An OCS-vClos which introduces a layer of optical circuit switches (OCSs) in the leaf-spine network is also proposed to reduce potential network resource fragmentation caused by resource allocation strategy in vClos. Testbed experiments and real-trace-based large-scale simulations are conducted to demonstrate the superiority of vClos over existing network resource scheduling strategies.

1 Introduction

The field of machine learning has experienced exponential growth overtime, leading to the emergence of Distributed Machine Learning (DML). As a result, the size of individual machine learning models has skyrocketed from 1.5B in 2020 to over 1000B at present. However, the expansion of network bandwidth has not kept pace with this growth, as illustrated in Fig. 1. In the past five years, the scale of machine learning models has increased by 2941 times, while single-chip computing power has only improved by 47 times. Consequently, more and more tasks now require multi-server training, which demands interconnected communications. Unfortunately, single-port bandwidth has merely increased by a factor of four, exacerbating the network bottleneck already

present in AI training. Open source datasets [18, 52] reveal that an increasing number of DML tasks now employ hundreds or even thousands of GPUs. Therefore, the network has gradually become the bottleneck in large-scale DML systems, with networking alone accounting for up to 90% of total training time [27].

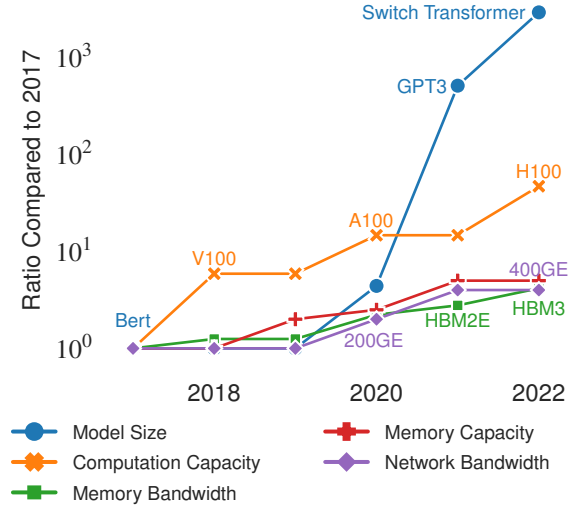


Figure 1: The growth rate of network capacity is much lower than the growth rate of computing capacity and model size.

Distributed training typically involves four types of parallelism: data parallelism, model parallelism, pipeline parallelism, and Mixture-of-Experts (MoE) parallelism. The training process usually involves two phases: forward propagation and backward propagation. To avoid the network from becoming a bottleneck in DML tasks, many training frameworks cover AllReduce communication overhead with computation overhead during backward propagation phases. However, the network still becomes the limiting factor for several reasons. 1. DML tasks such as NLP and Recommendation Systems require AlltoAll communication traffic during training. Com-

*Corresponding author

pared to AllReduce, cover AlltoAll communication overhead is more difficult, especially in large model training where AlltoAll communication overhead can account for 20 ~ 50% of total training time. 2. It is challenging to fully utilize network bandwidth in DML [57], and merely increasing bandwidth strategies may not adequately address the network bottleneck. 3. Modern data centers often use ECMP as the load balancing strategy, which faces hash collision problem, leading to bandwidth competition among different tasks in a multi-tenant cluster, resulting in higher communication overhead.

This paper focuses on investigating how network contention affects communication overhead and how to address the problem. An experimental testbed cluster with 32 NVIDIA V100 GPUs is set up to determine the order of severity of the hash collision problem and how it impacts communication overhead for a single DML task. Based on the findings, a *vClos* scheme that can address the hash collision problem in a multi-tenant GPU cluster is proposed. *vClos* aims to boost network resource utilization through eliminating network contention by jointly considering topology, routing, communication pattern, and GPU assignment.

The challenges of the contention-free task placement problem are two folds. There are mainly two sources of contention for DML tasks: intra-task contention and inter-task contention. **1. Intra-task contention:** The communication stage of a DML task may contain multiple phases, each with a different communication pattern. A straightforward approach to eliminate contention is to design different contention-free forwarding rules for different phases. However, updating forwarding rules in modern switches/routers takes at least tens of microseconds, which is slow and error-prone. **2. Inter-task contention:** When a network link is shared by multiple DML tasks, these tasks may contend for network bandwidth. One way to eliminate inter-task contention is to reserve a link-disjoint sub-topology for each DML task. However, this approach may incur additional resource fragmentation, i.e. even if there are enough GPU resources, a DML task still cannot be initiated due to insufficient network resources. Resource fragmentation can be detrimental to GPU resource utilization and increase the task waiting time.

Large model training typically involves multiple types of communication traffic, such as a typical large model involves data parallelism, MoE parallelism, model parallelism, and pipeline parallelism. Although previous work [51] has discussed partitioning mesh topology for tasks, AlltoAll communication in MoE parallelism performs poorly on mesh topology. DGX SuperPOD [31] also recommended to use the Leaf-Spine architecture, therefore, in this article, we consider partitioning clos sub-topology for each task.

vClos scheme we proposed addresses the challenges above for DML tasks in a Leaf-Spine network (i.e. a folded Clos). Given a single DML task, *vClos* is a smaller Leaf-Spine sub-topology and adopts a *Source Routing* strategy which can be proved to be contention-free for any *Leaf-wise Permutation*

Traffic Pattern defined in Section 5.2. By assigning GPUs properly, most communication patterns can be shaped to meet the pattern.

In a multi-tenant data center network, *vClos* can adapt an optional layer of Optical Circuit Switches (OCS) to reduce resource fragmentation while guaranteeing network isolation among different DML jobs. The power consumption of OCSes are negligible. When a DML task cannot be scheduled normally due to insufficient network resources, *OCS-vClos* can reconfigure the network topology by controlling the OCS layer to make the task deployable.

The key contributions are summarized below: 1. Detailed analysis of how network contention affects DML training time cost through testbed experiments on a cluster with 32 V100 GPUs; 2. Proves network contention-free communication is feasible for most DML jobs that follow a *Leaf-wise Permutation Traffic Pattern*; 3. Proposed a feasible *vClos* scheme that allocates an exclusive sub-topology for each job using the link reservation policy; 4. In *OCS-vClos* an additional OCS layer is introduced to reduce resource fragmentation caused by resource scheduling strategy in *vClos*. The feasibility of the scheme is verified through a cluster containing 32 v100 GPUs, and its performance in large-scale networks is demonstrated through simulation.

2 Background

2.1 With the development of large models, networks have gradually become potential bottlenecks

With the rise of DML, computing demand of large AI models doubles every 3.4 months since 2012 [16]. The BERT [11] model contains 300 million parameters, the GPT-2 [4] model contains 1.5B parameters, and GPT-3 [15] model even consists of 175B parameters. According to NVIDIA’s estimation, if GPT-3 is to be trained, even if the video/host memory can be installed, using 8 V100 GPUs (configured with a DGX-1 server) will take 36 years [40].

In recent years, GPU computing power has also been continuously increasing, the computing power of a single GPU has increased from 21.2 TFLOPS on P100 to 1000 TFLOPS on H100. However, the growing speed of AI-model computing demands outpaces Moore’s Law, which brings challenges to the development of AI technology. The growth rate of the model is greater than the growth rate of GPU computing power, so the number of GPUs required is increasing, which boost the market demand for large-scale GPU clusters.

However, the growth rate of network bandwidth is relatively slow compared with GPU computing power, and is far slower than the computing power demand of the model, so the bottleneck effect of the network is becoming increasingly apparent. The comparison of the growth rate between the

network and other major cluster components in Fig. 1 further illustrates this trend.

In order to meet the needs of training large-scale DNN models, distributed training comes to birth. Recent years, researchers have developed several useful frameworks for distributed training, including MXNet [6], TensorFlow [1], PyTorch [32], Horovod [42], MindSpore [45], etc. Distributed training typically consists of two phases: computation and communication. In the computation phase, each worker calculates the gradients of the model parameters in *forward and backward propagation* procedure [25] using *stochastic gradient descent* algorithm; in the communication phase, all the workers synchronize all the gradients and then update the model parameters used in the next iteration. Distributed training is highly expensive and time-consuming. [37] shows that for tasks with high communication to computation ratio α , network bandwidth is the primary factor that determines communication latency. In some extreme cases, 90% of the training time can be spent on networking communication in distributed training [27]. With the popularity of large model training [37, 47], the need for high communication efficiency is increasing.

Previous work [57] has mentioned that the network may be a bottleneck in DML due to the low utilization of the network caused by network protocol overhead, but there may be other factors as well. Different from [57], our testbed experiment is based on a 100Gbps RoCE instead of TCP, and with GDR enabling to maximize communication efficiency. In this case, hash-collision problem can result in a noticeable communication overhead. In a multi-tenant GPU cluster, network contention can cause two additional problems. First, the longer communication time of a single task will increase the waiting time of subsequent tasks, especially when the cluster load rate is high. Second, users may purchase services with different quality at the same price due to the network environments, which will affect the user experience.

3 Understand network contention for DML tasks

3.1 The severity of hash-collision problem in the network

Hash-collision is a common problem in modern data centers. Before measuring the impact of network collisions, we first need to know the probability of hash-collision occurring. Selecting the best combination from many hash factor and hash mode combinations is a common means of mitigating network contention. We conduct a testbed experiment based on two servers and two Huawei CE8850 switches as shown in Fig. 3. We randomly select 4 GPUs in each server to communicate with each other to generate 20 Ring-AllReduce communication tasks, and select 15 hash-mode-hash-factor combinations to calculate the probability of hash-collision. However, no

matter what hash mode-hash factor combination is chosen, hash-collision cannot be completely avoided. An example of hash collision is shown in Fig. 3. Under the optimal combination of hash factors, there is still an average 31.5% probability of network contention occurring, and even a 3% probability of all flows routing to the same port. Simulation tests on a large-scale cluster is also conducted based on the Leaf-Spine architecture in Fig. 4a to demonstrate the severity of hash-collision at different scales. Mmh3 is chosen as the hash function and the 5-tuple (src-ip,dst-ip,src-port,dst-port,port number) is chosen as the hash factor. Fig. 2 shows that the larger the cluster size, the greater the likelihood of network contention. In extreme cases, there is a 3.3% probability of 6 flows competing for bandwidth. So it is necessary to study the impact of hash-collision on large-scale GPU clusters.

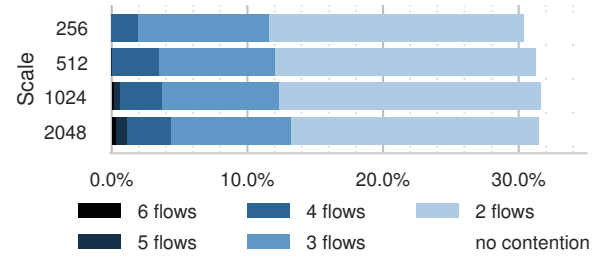


Figure 2: The proportion of flow contention under different scales (total is 100%)

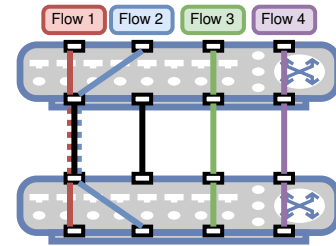


Figure 3: An example of hash collision.

3.2 Scaling Factor of DML Tasks

Scaling factor is often used to measure the scale-out ability, which is defined as Equation 1, where T is the throughput using single device and T_n is the throughput of a system with n devices. The ideal scaling factor would be 1, and communication bottlenecks contribute most decrement in actual models and environment.

$$ScalingFactor = \frac{T_n}{nT} \quad (1)$$

Previous work [57] has shown the scaling factor in a TCP environment. Compared with TCP, RoCE protocol result in lower communication overhead, and with GDR enabling can

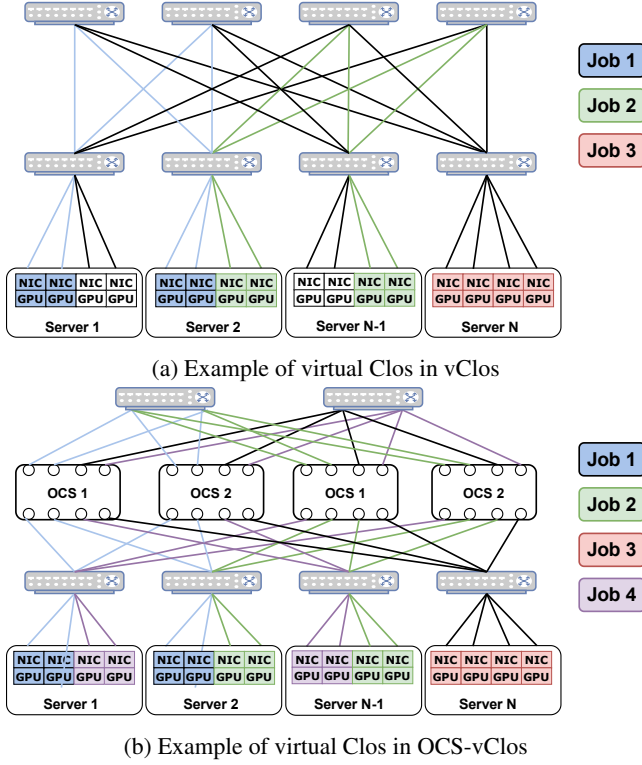


Figure 4: vClos/OCS-vClos physical architecture.

further improve communication efficiency, so a testbed experiment with 32 V100 GPU interconnected by a 100Gbps RoCE network is conducted, the network topology is the standard Leaf-Spine architecture in Fig. 4a.

We test two case: contention-free network and ECMP-based network. In the first case, we plan the route in advance by using ACL to ensure that there is no contention. In the second case, we select the 5-tuple (src-ip,dst-ip,src-port,dst-port,port number) as the hash factor and set the hash mode to 5 for testing. Fig. 5 shows the changes of scaling factor for different models, where the bar chart shows the SF values when using ECMP, the line chart shows the SF values under different bandwidths without contention.

When using ECMP, the scaling factor may decrease as the number of nodes increases, this may because the more nodes there are, the more severe the contention becomes. Fig. 5 indicates for some tasks such as VGG16 or BERT, network contention can significantly affect the scaling factor. The noticeable impact of hash-collision on the scaling factor can be seen in 100GE networks which suggests that hash-collision in large-scale clusters may have a visible impact on training performance. It is worth noting that these results are tested on V100 GPUs, and if the cluster adopts stronger GPUs like H100, the impact of hash-collision on performance could be worse.

3.3 How does network contention affect communication overhead for single DML task?

In Fig. 5, we note that different models have different sensitivities to network contention, a simple testbed experiment is conducted to illustrate this phenomenon, with a cluster consists of two machines each contains 4 V100, and bandwidth competition between the two flows is deliberately introduced through route planning.

Fig. 6 shows the impact of two flows contention on training time. We observed that: 1. For tasks as VGG16 and BERT, network contention may result in a obvious throughput decrement which may due to their relatively larger parameter number; 2. For all data parallelism models, the larger the batch size, the smaller the impact of contention, which may because for data parallelism DML with larger batch sizes, computational overhead is more easier to be covered; 3. Models such as DLRM and MoE are relatively sensitive to contention, which may because the AlltoAll communication is difficult to be covered; 4. DML tasks running in smaller bandwidths are more sensitive to network contention, which may be due to two reasons: firstly, the non-coverable communication overhead increases in smaller bandwidths, and secondly, the synchronization overhead of communication algorithms prevents the network bandwidth from being fully utilized, which implies that the severity of network contention has a non-linear impact on the performance of DML training, so avoiding hash-collision to reduce the number of flow contention is meaningful.

3.4 How does network contention affect multi-tenant Cluster

The case for multi-tenant clusters is more complex than the single-task scenario. Naturally, contention-affected tasks will increase the average DML job running time (JRT). However, the impact of network contention on the average completion time (JCT) of tasks may be more pronounced. The reason is that when a task runs longer, the wait time of all subsequent waiting tasks also becomes longer, so the average job waiting time (JWT) will also increase. In addition, the JCT of tasks with the same parameters may fluctuate due to the network environment, as a result, the service quietly with the same price are different, which affects the user experience.

4 Overview

4.1 Network Architecture

vClos is designed in a Leaf-Spine network architecture (Fig. 4a). The Leaf-Spine architecture has been widely adopted to build high-speed compute networks for GPU clusters [31].

In the Leaf-Spine architecture, each server contains a fixed number of GPUs (e.g. 8) and these GPUs are internally connected via Nvidia’s NVLinks and NVSwitches, which could

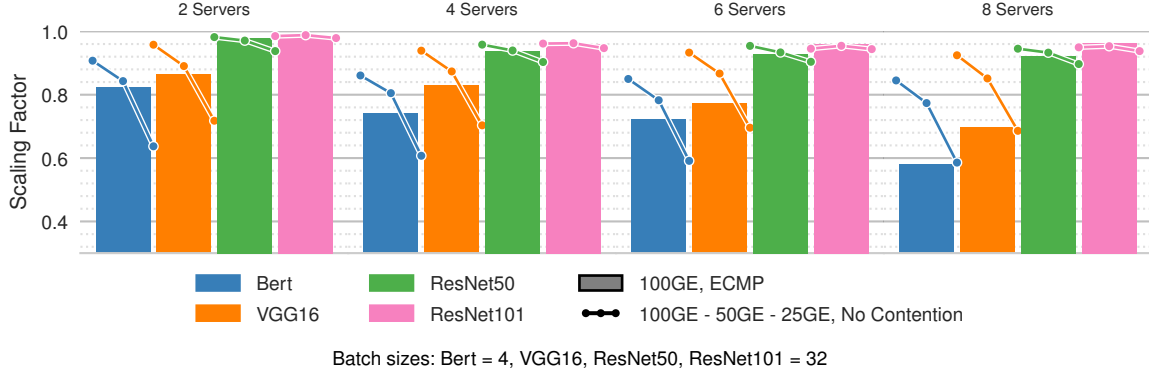


Figure 5: Scaling Factor may Decrement Due to Hash-collision.

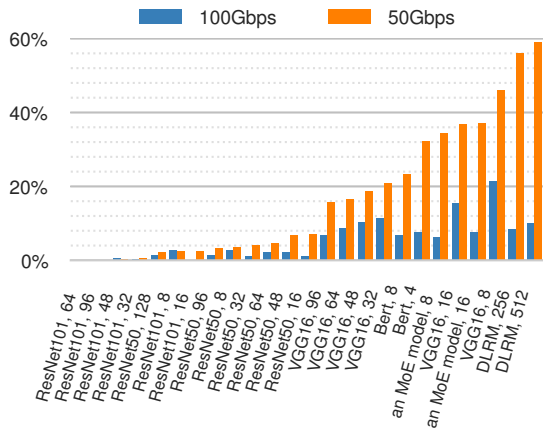


Figure 6: Expert parallel and some recommendation models such as DLRM are more sensitive to network contention. In extreme cases, two flow contention can result in a throughput decrease of about 60%

offer a Tbps-level bandwidth for intra-server communication between each GPU pair. GPUs from different servers communicate through the network via Network Interface Cards (NIC). A modern NIC, e.g. Mellanox CX5 [36], could offer a communication bandwidth of 100/200Gbps. Since a server contains multiple GPUs, to avoid PCIE contention, we bound each GPU to a single NIC. This design was also adopted in Alibaba’s GPU cluster EFLOPS [12]. To guarantee full bisection bandwidth, each Leaf switch has half of its ports connected to the server NICs and half of its ports connected to the Spine switches. The topology between all the Leaf switches and all the Spine switches is a uniform bipartite graph, with an equal number of links between every pair of Leaf-Spine switches. Such a Leaf-Spine architecture could support a small to medium scale GPU cluster. For example, if each Leaf or Spine switch has 64 ports, then the maximum number of GPUs is $32 \times 64 = 2048$.

4.2 Understanding the Communication Patterns of DML Tasks

Common DML tasks involves one or more of the following parallelism: data parallelism, model parallelism, pipeline parallelism, and expert parallelism. Different parallel paradigms have different communication patterns.

In data parallelism each GPU stores the same model and processes different part of data set, and GPUs need to synchronize parameters which is usually realized through AllReduce. Typical data parallelism DML tasks contains forward computation, backward computation and AllReduce communication. Common AllReduce communication algorithms include Ring-AllReduce (Ring) in Fig. 8a, recursive half-doubling(HD) in Fig. 8c, hierarchical ring-AllReduce and other algorithms. The data size of AllReduce flow is generally around a few megabytes or tens of megabytes, and most of the communication overhead can be covered by backward computation. However, communication overhead may not be well covered up when multiple flows competing for bandwidth.

In model parallelism each GPU store different parts of the same model. Different model segmentation methods may result in different communication characteristics. Communication in model parallelism cannot be covered by computation, and in order to efficiently utilize the higher bandwidth within the machine, model parallelism is usually executed within the machine [14].

Pipeline parallelism [14] improves parallelism by dividing a large batch into multiple small batches and synchronizing parameters after all small batches have been calculated. Only send/receive communication between neighbour GPUs exists in pipeline parallelism. Compared to data parallelism, pipeline parallelism has a smaller amount of communication size, but the communication overhead cannot be covered up by computation overhead.

Expert parallelism is widely used in large model training. A MoE layer exists in expert parallelism which consists of multiply experts placed on different workers. MoE layer requires All2All communication between different experts. Compared

with data parallelism, the communication overhead in All2All can not be covered. The data size of All2All flows is generally around tens of megabytes, so expert parallelism is sensitive to network contention.

A typical large model training process may involve multiple parallel methods mentioned above. We analyze the training overhead of a large model with 600 billion parameters without preventing network contention, in which All2All communication overhead accounting for 25.8% and allreduce communication accounting for 4.2%.

4.3 Objective

Since Leaf-Spine architecture is rearrangeable-nonblocking, it seems possible to plan routes based on different communication patterns to avoid contention, but there are two reasons that make this difficult to achieve: 1. If modifying routing table affects the existing tasks, packet loss may occur and may result in PFC storms in large-scale networks; 2. It is difficult to directly perceive user traffic characteristics in a public cloud environment due to privacy considerations. Therefore, we hope to eliminate network contention without perceiving communication patterns. The objective of *vClos* is to eliminate link contention for DML tasks in a multi-tenant GPU clusters. We focus on two scenarios in this paper.

Single Task Case The common communication patterns of DML tasks is analyzed and network contention can be eliminated by designing a content free routing in Leaf-Spine networks. Specifically, in Leaf switch, a scheduler plan the route when some tasks arrives by issuing ACL flow table, and in Spine switch, the route table is directly generated through BGP or static routing. In this way, traffic contention within the task can be eliminated.

Multi-Tasks Case Compared with single task case, multi-tasks scenario is more complex. Therefore, the scheduler attempt to partition sub-topologys *vClos* for each task in a Leaf Spine network to isolate traffic between tasks. In order to reduce potential network resource fragmentation, which means even if the cluster has sufficient GPU resources, tasks still need to wait due to the inability to find suitable network resources to partition *vClos*, we also consider introducing a layer of OCS between Leaf switches and Spine switches to reduce fragmentation of network resources through OCS reconfiguration.

Overview Fig. 7 shows an overview of proposed Isolated scheduling. The DML task first enter a waiting queue after it is submitted, and the task scheduler will select a task from the waiting queue and attempt to allocate resources. The most commonly used task scheduling algorithm in the industry is First-in-First-out (FIFO), and this paper mainly considers the

case of FIFO. We also conducted experiments on the cases of Smallest-First(SF) and Early-Deadline-First(EDF) in Section 9.7 to analyse the sensitivity to task scheduling. Resource Scheduler is the key of this paper, which assign GPU resource, issue flow tables to the switch to plan the route, and configure the VRF command to generate virtual Leaf/virtual Spine to generate *vClos* for each chosen job. Performance indicators such as task running time and waiting time are also monitored to evaluate the performance of different solutions.

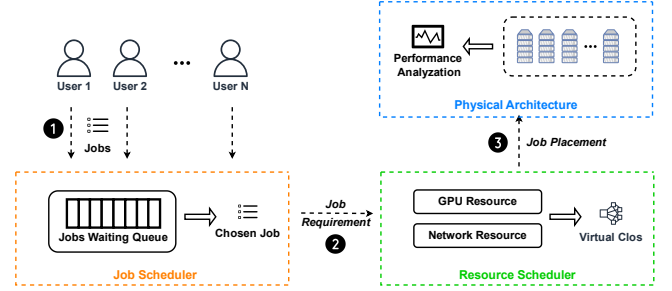


Figure 7: Overview of Isolated Scheduling

5 Preliminaries to Eliminate Network Contention

5.1 Difficulty of Eliminating Contention

We aim to design a contention-free resource scheduler. **Since the Leaf-Spine architecture is rearrangeably nonblocking, any permutation traffic pattern can be routed in the Leaf-Spine architecture without link contention.** As we will see shortly, the communication pattern of the collective-communication based distributed training tasks belongs to the category of permutation patterns. Hence, if we were able to update the switch forwarding table as soon as the communication pattern changes, then we can completely eliminate network link contention. However, this approach incurs significant control overhead. First, computing a feasible set of contention-free forwarding rules for a given permutation pattern in a rearrangeably-nonblocking Leaf-Spine architecture takes time. Second, updating switch forwarding tables can be error-prone and may interrupt the on-going flows. Third, the communication patterns of some distributed training tasks may change on the order of tens of microseconds. Even though the detailed communication patterns of distributed training tasks are predictable, switch forwarding rule update cannot be accomplished in such short amount of time.

Another strategy to achieve contention-free is to build a strictly-nonblocking Leaf-Spine architecture. **This approach stills creates a uniform bipartite graph between all the Leaf switches and all the Spine switches,** but the number of uplinks p_u of each Leaf switch and the number of downlinks p_d of each Leaf switch must satisfy $p_u \geq 2 * p_d - 1$. Then, when a

new flow arrives, there must exist a Spine switch, such that both the link from the source Leaf to this Spine switch and the link from this Spine switch to the destination Leaf are not occupied. Using this strategy, it is easy to compute forwarding rules for new flows, and programming these new forwarding rules to switches does not affect the on-going flows. However, this architecture still faces the following problems. First, adding new forwarding rules takes non-negligible amount of time. Second, the strictly-nonblocking Leaf-Spine architecture is less efficient than the rearrangeably-nonblocking Leaf-Spine architecture. For example, using 64-port switches, we can build a rearrangeably-nonblocking Leaf-Spine network with $32 \times 64 = 2048$ GPUs using $32 + 64 = 96$ switches. In contrast, using 64-port switches, we can only build a strictly-nonblocking Leaf-Spine network with $21 \times 64 = 1344$ GPUs using $(21 \times 2 - 1) + 64 = 105$ switches.

5.2 Contention-Free Routing in Leaf-Spine Networks

In this section, a sufficient condition that guarantees contention free in Leaf-Spine networks is provided. The results in this section will motivate the design of *vClos*.

Consider the following routing strategy. Note that each Leaf switch has an equal number of server-facing ports and spine-facing ports. In the m -th Leaf switch \mathcal{L}_m , we create an arbitrary one-to-one mapping $f_m(\cdot)$ from the server-facing ports to the spine-facing ports, such that no two server-facing ports map to the same spine-facing port. Then, we can route a packet from a source GPU G_s to a destination GPU G_d based on the following strategy:

Case 1: s and d connect to the same Leaf switch. We can directly forward the packet from s to d .

Case 2: s and d connect to different Leaf switches. Assume that s connects to the i -th server-facing port of the m -th Leaf switch. We will first forward the packet to the $f_m(i)$ -th spine-facing port of the m -th Leaf switch, and then from that Spine switch, we forward the packet to the destination. (Note that there is a unique path from any Spine switch to any GPU.)

In a Leaf-Spine network, any Spine switch can be used to forward a packet from its source to its destination. Note that in the above routing strategy, it is the source that determines the route a packet takes. Hence, in the rest of this paper, we call the above routing strategy as *Source Routing*.

Definition 1 (Leaf-wise Permutation Traffic Pattern). *Assume that there are N servers and L Leafs in the network. Any permutation $(G_0, G_1, \dots, G_{N-1})$ of $(0, 1, \dots, N-1)$ defines a permutation pattern: the s -th GPU G_s only sends traffic to the d -th GPU G_d . The above permutation pattern is called a Leaf-wise permutation pattern, if there exist another permutation*

$(l_0, l_1, \dots, l_{L-1})$ of $(0, 1, \dots, L-1)$, such that any flow from G_i to G_d satisfies one of the following two conditions: 1) G_i and G_d belong to the same Leaf switch; 2) there exists j , such that G_i directly connects to the j -th Leaf l_j and G_d directly connects to the j^ -th Leaf l_{j^*} where $j \neq j^*$. For any G_i and G_{s^*} , if G_i and G_{s^*} directly connects to different Leaf switches, the G_d and G_{d^*} directly connects to different Leaf switches.*

A source routing strategy can be completely determined by L mapping functions $(f_0(\cdot), f_1(\cdot), \dots, f_{L-1}(\cdot))$. For any source routing strategy, we have the following lemma.

Lemma 5.1. *A source routing strategy can guarantee contention-free for any Leaf-wise permutation traffic pattern.*

Proof. If G_i and G_d directly connect to the same Leaf switch, the proposed source routing strategy can naturally guarantee contention-free, we prove another case by contradiction, where two flows cross Leafs competes bandwidth. Suppose there exists permutation $(s_0, s_1, \dots, s_{S-1})$ of $(0, 1, \dots, S-1)$, such that G_i directly connects to j -th Leaf l_j and G_d directly connects to the k -th Leaf l_k where $j \neq k$. l_j connect to the j -th port of m -th Spine switch s_m and l_k connect to the k -th port of s_m .

Suppose there exists another pair of GPUs G_{i^*} and G_{d^*} competes bandwidth with G_i and G_d . G_{i^*} directly connects to j^* -th Leaf l_{j^*} and G_{d^*} directly connects to the k^* -th Leaf l_{k^*} where $j^* \neq k^*$. l_{j^*} connect to the j^* -th port of m^* -th Spine switch s_m^* and l_{k^*} connect to the k^* -th port of s_m^* .

There are two cases when network contention happens: 1) $j = j^*$, where G_i and G_{i^*} directly connects to the same Leaf but send data to same Spine switch, which is contrary to source routing. 2) $k = k^*$ and $t = t^*$, which means two GPUs aim at the same Leaf are routed through the same Spine, which is contrary to Definition 1. □

5.3 Eliminate Network Contention for a Single Distributed Training Task in a Leaf-Spine Network

We analyzed the communication algorithms often involved in GPU cluster in recent years [2, 10, 13, 20, 23, 24, 32, 41, 42, 44, 46, 51, 53, 55], and found that most of the communication algorithms follow the traffic pattern in Definition 1. We consider a case where GPU G_i sends flow to G_d , the communication path can be represented as $G_i \rightarrow l_j \rightarrow s_m \rightarrow l_k \rightarrow G_d$. We will show that for these communication algorithms for any other communication path $G_{i^*} \rightarrow l_{j^*} \rightarrow s_{m^*} \rightarrow l_{k^*} \rightarrow G_{d^*}$, if $j! = j^*$ and $m = m^*$, then $k! = K^*$.

Data parallelism: With data parallelism, each GPU stores a copy of the whole model and handle different part of the dataset. In each iteration, GPUs aggregate their gradients locally and synchronize weights through communication algorithms such as ring allreduce. There are in general two

paradigms for data parallelism: 1) parameter server [44] and 2) collective communication [37, 38, 49, 53]. In the parameter server mode, many workers may send gradients to the same parameter server at the same time, thus creating a communication bottleneck. Collective communication could evenly distribute the communication workload among all the workers, and thus can be more efficient for distributed training. Using Lemma 5.1, we show that many collective-communication based distributed training tasks can be contention-free if properly deployed. AllReduce is one of the most frequently used collective communication procedures in distributed training. AllReduce performs a global reduction operation (e.g., SUM, AVERAGE, MAX, etc.) on values from all the workers and then distributes the result back to every worker. Distributed training uses AllReduce operations to average all the gradients computed by all the workers in each iteration. There are two widely adopted AllReduce operations which are suitable for small-scale and large-scale scenarios respectively: Ring AllReduce(Ring) and Recursive Half-Doubling(HD).

At the start of the Ring-allreduce [7, 42] phase, each node is assigned with a rank and form a logical ring pattern according to the rank of the nodes.

For a DML occupies a whole cluster which contains N GPUs, the communication phase is divided into two steps: first is scatter-reduce and second is all-gather. Scatter-reduce phase contains $N - 1$ rounds and each GPU divide its local data into N chunks, in round t , GPU G_i sends its $(i - t) \bmod N$ -th chunk to GPU $G_{(i+1) \bmod N}$. After $N - 1$ rounds, the $(i + 1) \bmod N$ chunk of GPU G_i contains the local aggregated result. In all-gather phase, G_i send its local aggregated in $(i + 1 - t) \bmod N$ -th chunk to GPU $G_d = G_{(i+1) \bmod N}$ and after $N-1$ rounds all nodes have N synchronized data chunks. Ring-allreduce follows the Leaf-wise permutation pattern. Suppose GPU G_i directly connects to l_j , where $j = \lfloor i/n \rfloor = \lfloor i * L/N \rfloor$, n is the number of ports of one Leaf and $n = N/P$. $G_d = G_{(i+1) \bmod N}$ directly connects to $l_k = l_{((i+1) \bmod N)/n}$, and $\lfloor (i+1)/n \rfloor = \lfloor i/n \rfloor$.

There exists two cases, 1) $\lfloor (i+1)/n \rfloor = \lfloor i/n \rfloor$ and $(i+1) < N$, in this case $k = \lfloor (i+1)/n \rfloor = \lfloor i/n \rfloor + 1 = j + 1$. 2) $(i+1) \geq N$, however $i \leq N - 1$, so we have $i = N - 1$ and $l_k = l_0$. Suppose there exists another G_i^* where $\lfloor i^*/n \rfloor = \lfloor i/n \rfloor$ send data to $G_{d^*} = G_{(i^*+1) \bmod N}$, G_i^* connects to $l_j^* = l_{\lfloor i^*/n \rfloor}$. G_{d^*} connects to Leaf $l_k^* = l_{((i^*+1) \bmod N)/n}$. If $k^* = k = 0$, then $j = j^* = N - 1$. If $k^* = k$ and $k^* \neq 0$, $j = j^* = k - 1$. So if $k^* = k$ then we must have $j = j^*$ which contradicts the hypothesis, so Ring-allreduce follows *Leaf-wise Permutation Traffic Pattern*.

For hierarchical rings or rings with multiple channels, we only need to construct an independent communication plane for each ring, where data flow in each plane follows the *Leaf-wise Permutation Traffic Pattern*.

Recursive Half-Doubling (HD) [39] is usually used when N is an idempotent of two in large-scale cluster. As for HD, it contains the communication in two phases: Reduce-Scatter

and AllGather. In the Reduce-Scatter phase, it has $\log_2 N$ steps. In step t , i -th rank exchanges data with a rank that is a distance $2^{t-1}(-1)^{\lfloor i/(t+1) \rfloor}$ away: Each rank sends different half of the data to its peer, and performs the reduction operation on the received data. This procedure continues recursively, halving the data communicated at each step. After the last step, each rank holds $1/N$ reduced data. The AllGather phase is analogous to the reduce-scatter phase. In the first step, ranks that are distance $N/2$ apart exchange their data. In the second step, ranks that are distance $N/4$ apart exchange their own data as well as the data they received in the previous step. And in step t , ranks that are distance $N/2^t$ apart exchange their data as well as the data they received in all the previous steps. After the last step, each rank hold all the reduced data. Compared to ring, HD has lower synchronization times, making it more suitable for large-scale scenarios.

In both reduce-scatter phase and AllGather phase, GPU G_i send data to GPU $G_{i \oplus 2^t}$ in step t . Since when $t \leq \log_2 n$, GPU G_i and GPU $G_{i \oplus 2^t}$ directly connects to the same Leaf, so we only consider $t > \log_2 n$. Take reduce-scatter phase as an example, suppose G_i directly connects to l_j , where $l_j = b_{\lfloor i/n \rfloor}$. $G_d = G_{i \oplus 2^t}$ directly connects to $k = \lfloor (i \oplus 2^t)/n \rfloor$ -th Leaf. Since $\lfloor (i \oplus 2^t)/n \rfloor = \lfloor (i + 2^t - 2(i \wedge 2^t))/n \rfloor$. There are two cases, the first is $i \wedge 2^t = 0$, we have $\lfloor i/n \rfloor \wedge 2^{t-\log_2 n} = 0$ and $\lfloor (i + 2^t - 2(i \wedge 2^t))/n \rfloor = \lfloor (i + 2^t)/n \rfloor = \lfloor i/n + 2^{t-\log_2 n} \rfloor = \lfloor i/n \rfloor + 2^{t-\log_2 n} = \lfloor i/n \rfloor \oplus 2^{t-\log_2 n}$. The second is $i \wedge 2^t = 1$, we have $\lfloor i/n \rfloor \wedge 2^{t-\log_2 n} = 1$ and $\lfloor (i + 2^t - 2(i \wedge 2^t))/n \rfloor = \lfloor (i - 2^t)/n \rfloor = \lfloor i/n - 2^{t-\log_2 n} \rfloor = \lfloor i/n \rfloor - 2^{t-\log_2 n} = \lfloor i/n \rfloor \oplus 2^{t-\log_2 n}$. All in all, $k = j \oplus 2^{t-\log_2 n}$, which means some l_j only communicate with one Leaf $l_{j \oplus 2^{t-\log_2 n}}$ at t -th step, so HD also follows Leaf-wise permutation pattern.

When the number of nodes N is not an idempotent of two, it is necessary for i -th node G_i , where $i = 0, 1, 2, \dots, N - 2^{\log N}$ communicate with d -th node $G_d = G_{i+2^{\log N}}$ in advance during the reduce scatter stage and the all gather phase, so that the remaining $2^{\log N}$ nodes is an idempotent of two. Suppose GPU G_i sends data to GPU G_d through $i \% n$ -th Spine, G_i connects to j -th Leaf l_j , where $l_j = l_{\lfloor i/n \rfloor}$ and n is the number of ports of one Leaf. G_d connects to $k = \lfloor (i + 2^{\log N})/n \rfloor$ -th Leaf, and $\lfloor (i + 2^{\log N})/n \rfloor = \lfloor i/n \rfloor$. Suppose there exists another G_i^* send data to $G_{d^*} = G_{(i^*+2^{\log N})}$ through $i^* \% n$ -th Spine, where $\lfloor i/n \rfloor = \lfloor i^*/n \rfloor$, G_{d^*} connects to $k^* = \lfloor (i^* + 2^{\log N})/n \rfloor$ -th Leaf. If $i \% n = i^* \% n$, we have that $i^* - i = p * n$, where $-L < p < L$ is some non-zero integer, so $k^* = \lfloor (i + p * n + 2^{\log N})/n \rfloor = k + p$, so $k^* \neq k$, so it also follows *Leaf-wise Permutation Traffic Pattern*.

Model parallelism With model parallelism algorithms, different part of the model are partitioned over multiple GPUs. In which each GPU is only responsible for updating the parameter weights of the neurons to which it is assigned. Different model segmentation methods may result in different communication characteristics, but in many cases [14] the complete

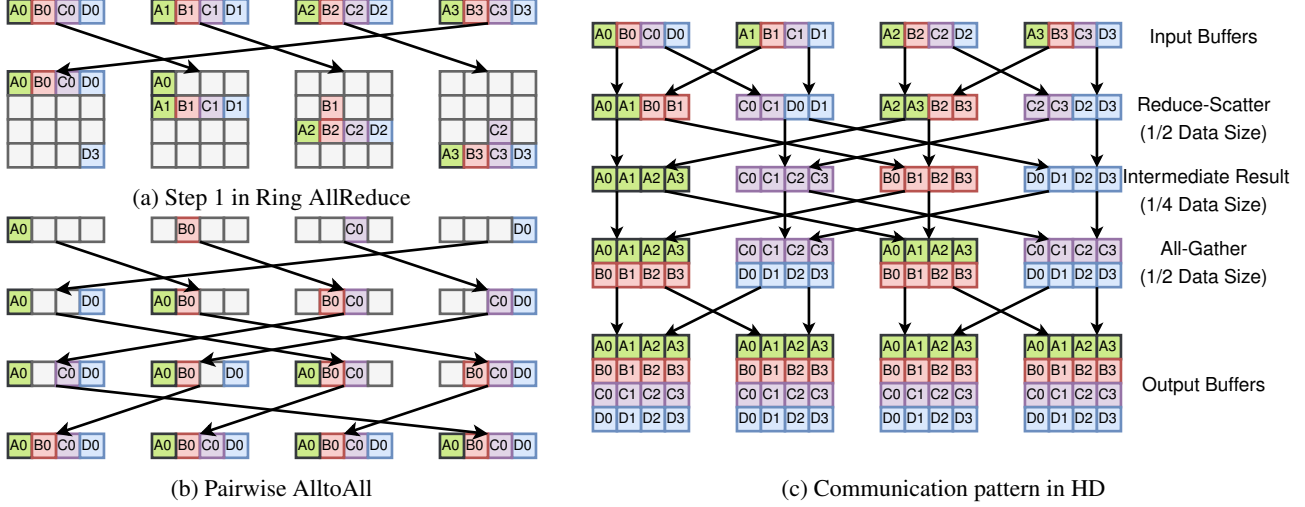


Figure 8: Communication Patterns

model will not be distributed across servers, so model parallelism usually results in intra-server traffic pattern. Even if the model has to occupy multiple servers, assigning sub-topology for each task will not lead to more serious network contention.

Pipeline parallelism With pipeline parallelism, one layer of model is shared by multiple GPUs. GPU G_i send data to GPU $G_d = G_{i+1}$ in forward phase and send data to GPU G_{i-1} in backward phase. Pipeline parallelism requires send/receive between adjacent nodes. Suppose G_i connects to j -th Leaf l_j , where $l_j = l_{\lfloor i/n \rfloor}$ and n is the number of ports of one Leaf and $i < N$. $G_{(i+1)}$ connects to $k = \lfloor (i+1)/n \rfloor$ -th Leaf, and $\lfloor (i+1)/n \rfloor! = \lfloor i/n \rfloor$. G_i send data to G_d through $i\%n$ -th Spine. Suppose there exists another G_i^* send data to $G_d^* = G_{(i^*+t+1)}$ through $i^*\%n$ -th Spine, if $i\%n = i^*\%n$, then we have $-N < i^* - i = p * n < N$, so $k^* = \lfloor (i^* + t + 1)/n \rfloor = \lfloor (i + p * n + t + 1)/n \rfloor = \lfloor (i + t + 1)/n \rfloor + p$, which means $k! = k^*$, so it follows *Leaf-wise Permutation Traffic Pattern*.

Expert parallelism With Expert Parallelism, different groups of *experts* are partitioned over multiple GPUs, and all GPUs should communicate with all other GPUs. In many cases, the communication patterns of MoE parallelism are based on pair-wise exchange algorithm. The pair-wise All2All contains $N - 1$ steps. In step t , GPU G_i sends data to GPU $G_{(i+t+1) \bmod N}$ through $i\%n$ -th Spine. Suppose G_i connects to j -th Leaf l_j , where $l_j = l_{\lfloor i/n \rfloor}$ and n is the number of ports of one Leaf. $G_{(i+t+1) \bmod N}$ connects to $k = \lfloor ((i+t+1) \bmod N)/n \rfloor$ -th Leaf, and $\lfloor ((i+t+1) \bmod N)/n \rfloor! = \lfloor i/n \rfloor$. Suppose there exists another G_i^* send data to $G_d^* = G_{(i^*+t+1) \bmod N}$ through $i^*\%n$ -th Spine, where $\lfloor i/n \rfloor! = \lfloor i^*/n \rfloor$, G_d^* connects to $k^* = \lfloor ((i^*+t+1) \bmod N)/n \rfloor$ -th Leaf. If $i\%n = i^*\%n$, we have that $i^* - i = p * n$, where $-L < p < L$ is some integer, then we have $-N < (i^* + t + 1) - (i + t + 1) = p * n < N$,

so $k^* = \lfloor ((i^* + t + 1) \bmod N)/n \rfloor = \lfloor ((i + t + 1 + p * n) \bmod N)/n \rfloor$, which means $k! = k^*$, so pair-wise exchange algorithm also follows *Leaf-wise Permutation Traffic Pattern*.

For DML tasks that do not follow the *Leaf-wise Permutation Traffic Pattern*, using *Source Routing* can still bring performance improvements. If a Leaf Spine architecture cluster contains L Leafs and S Spines, in extreme cases, *Source Routing* will cause at most L streams to compete for bandwidth, while the ECMP strategy will cause at most $L * S$ streams to compete for bandwidth. Take Double Binary Trees based AllReduce [20] as an example, it organizes ranks into two binary tree, and independently conducts a tree AllReduce of half data on two Binary tree. Each rank is a Leaf node in only one of the two binary tree. A cluster of N nodes using tree communication will generate $2 * N$ flows, so it is not possible to achieve exclusive links for each flow. However, through simulation, we find that in a cluster with 2048 GPUs, when using *Source Routing*, the number of link contention will not exceed 3. An OCS layer can also be introduced to generated a small double-binary tree sub-topology to eliminate network contention.

When there are multiple tasks in a Leaf-Spine network, the communication pattern may not be a Leaf-wise permutation pattern. Then, we cannot guarantee contention-free. Fig. 9 shows an example, link 0 is shared when GPU0 sends flows to GPU1 and GPU3 sends flows to GPU2, link is shared when GPU2 sends flows to GPU3 and GPU1 sends flows to GPU0.

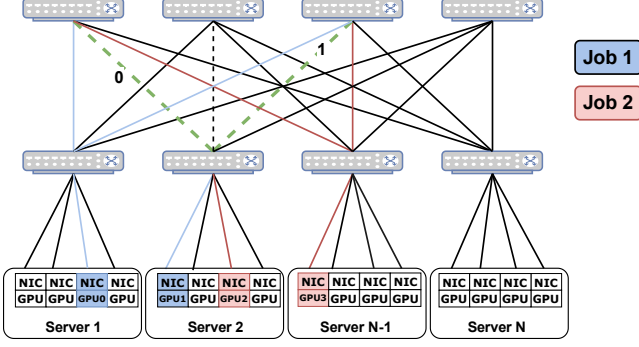


Figure 9: Example of network contention when multiple tasks run in a Leaf-Spine network.

6 vClos Design

6.1 vClos: Contention-Free Multi-Task Resource Scheduling in a Leaf-Spine Network

Single task can be deployed in a Leaf-Spine network without network contention as long as it follows *Leaf-wise Permutation Traffic Pattern*. This motivates us to assign each DML task a individual virtual network which can naturally eliminate inter-task contention. To this end, we propose *vClos*¹. To rigorously formulate the network resource scheduling problem in *vClos*, we introduce the following definitions. Mathematical notations of *vClos* is shown in Table 1.

Let's first introduce the general process of *vClos*. *vClos* method runs on a cluster contain L Leaf switches, S Spine switches and $L * S$ GPUs. T GPUs on each server support NVlink to reduce communication overhead. Each arriving i -th task requiring N_i GPUs is assigned with a sub-topology according to the current resource usage. Meeting locality is a common condition in the industry to ensure service quality, which means for some i -th task with $N_i \leq T$, the scheduler try to place it into a single server; some i -th task with $N_i > T$ is assigned with $\lceil N_i/T \rceil$ servers and a sub-topology *vClos* containing l virtual Leafs and s virtual Spines. The detail process of generating *vClos* can be divided into two stages which is shown in Stage 2 and Stage 3. During the scheduling process, scheduler strictly followed the locality constraint. We also conducted simulation experiments on the relaxation of locality to demonstrate the necessity of meeting the locality conditions in Table. 5.

It is worth noting that the prerequisite for successfully generating *vClos* is that **the number of GPUs required N is a prime number**. Fortunately, through analysis of the Helios [18] dataset, we found that in the vast majority of cases, N is a prime number if $N > T$. In extreme cases, we can also find an N^{new} , where N^{new} is the smallest composite number greater than N , and generate a *vClos* contains N^{new} GPUs.

¹A Leaf-Spine network can be viewed as a folded three-layered clos network. That is why we call our solution *vClos*.

Stage 0: Attempt to allocate resources within one server

Due to the locality constraint, i -th task with $N_i \leq T$ will be placed into one server. The scheduler selects the server with the least remaining number of GPUs among all servers which are capable of placing tasks. If such a server cannot be found, the task will enter the waiting phase.

Stage 1: Attempt to allocate resources within one Leaf switch

For tasks with $N_i > T$, scheduler enter into Stage 1. In order to minimize the occupation of network resources as much as possible, we first attempt to allocate tasks under the same Leaf. The advantage of this strategy is that it can reduce the occupation of the Spine port, thus reducing the fragmentation of network resources. **To be more specific, scheduler considers all Leaf switches connected to more than N_i/T idle servers as candidate switches**. Among these candidate switches, scheduler select the one connecting least idle servers. The detailed scheduling strategy is shown in Algorithm 1.

Table 1: Notations used in vClos/OCS-vClos Physical Architecture

L	Num of Leafs
S	Num of Spines
T	Num of GPU per server
L_n	n -th Leaf
S_m	m -th Spine
O_k	k -th OCS
$C_{n,m}$	Unused links between L_n and S_m
N_i	Num of GPU required by i -th job
R_n	Num of remaining idle servers in E_n
r_n^i	Num of servers to be used in E_n for i -th job
l	Num of Virtual Leafs to be used for i -th job
s	Num of Virtual Spines to be used for i -th job

Stage 2 in vClos: Attempt to generate virtual Clos When the scheduler is unable to place tasks under the same Leaf, it constructs *vClos* through link and port reservation. In Stage 2, scheduler attempts to generate virtual Clos with s virtual Spines and l virtual Leafs. There are many combinations of s and l , but as we show in Fig. 14, there is no sample path optimal solution to this problem, which means no matter how virtual clos are currently generated, there will always be task sequences that cause network resource fragmentation in the cluster in future. A direct solution to find a accessible solution is firstly choosing l Leaf switches which directly connects to N_i/T idle servers, then for each Leaf switch l_j generating a set s_j which contains all Spine switches which directly connect to l_j by a unused link. **If $|\cup s_j| \geq s$** , we can choose s Spine switches and then we can generate a virtual clos. Theoretically, we can try many combination of l Leaf

Algorithm 1: vClos Algorithm

Input: Required job size N
Output: $gpus, c_{n,m}$
 /* Calculate the required size within one server */
 1 $ReqSize \leftarrow \min(N, T)$;
 /* Calculated the required number of servers */
 2 $ReqSevNum \leftarrow \lceil N/ReqSize \rceil$;
 3 $validLeafList \leftarrow []$;
 /* Step1: Find valid Leafs with least idle servers */
 4 **for** $n \leftarrow 1$ **to** P **do**
 5 **if** $E_n.validServer(ReqSize, ReqSevNum)$ **then**
 6 $validLeafList.append(E_n)$;
 7 $validLeafList.sort()$;
 8 **if** $len(validLeafList) \geq 1$ **then**
 9 /* Step1: Find valid servers with least idle GPUs */
 9 $E_n^* \leftarrow validLeafList[0]$;
 10 $gpus = E_n^*.ChooseGpu(ReqSize, ReqSevNum)$;
 11 **return** $gpus, none$;
 /* Step2: Find valid vClos using ILP */
 12 $gpus, c_{n,m} \leftarrow FindvClos(N, Q)$;
 13 **return** $gpus, c_{n,m}$;

switches and s Spine switches by enumerating, but it is time consuming. In Algorithm 1 we linearize this problem into an ILP problem and give a solution with an average time cost of 1 second in a cluster containing 2048 GPUs. If vClos cannot be generated, the task will enter the waiting phase.

7 OCS-vClos Design

As Section 6 shows, a simple Leaf-Spine architecture is not sufficient to eliminate network contention without incurring additional overhead. A simulation experiments where 5000 training tasks run on a cluster with 512 GPUs is conducted to analyze the impact of vClos on the fragmentation of resources. The severity of network resource fragmentation is measured through simulation in Table 2, in which the arrival time of the task follows the Poisson distribution, and we modify the task arrival rate by modifying the λ variable of the Poisson distribution. It can be seen that when the task arrival rate is high, the network resource fragmentation introduced by vClos may be severe.

In order to reduce the additional overhead, a layer of optical circuit switches (OCS) is introduced to rewire links between Leaf switches and Spine switches. Fig. 4b shows an example of OCS-vClos. In [59], integer linear programming(ILP) was used to generate re-configuring strategies for OCS, and

Table 2: Fragmentation Count with Causes

$\lambda(s)$	vClos		OCS-vClos	
	GPU	Network	GPU	Network
100	1311	264	1180	172
110	988	250	1047	161
120	878	218	920	129
130	726	145	705	98

our work drew on similar methods to rewire links to minimize network fragmentation as much as possible. It is worth mentioning that OCS switching requires a delay of 50ms, so occupied links can not be rewired during AI training to prevent package loss. Therefore, only idle links can be rewired to use fragmented resources. Simulation experiment in Table 2 shows that the number of fragmentation caused by network resources can be greatly reduced when using OCS-vClos.

Before introducing the details of OCS-vClos, let's briefly introduce the scheduling process. To generate OCS-vClos, the behavior of scheduler in stage0 and stage1 is similarly to those in vClos. The scheduler places tasks with $N_i \leq T$ into one server in stage 0 and try to place tasks with $N_i > T$ under one Leaf in stage 1. However, due to the flexibility brought by the OCS layer, the schedule's strategy for building virtual Clos is slightly different. In stage2, the scheduler tries to place the task under single Spine through link rewiring. If there is no valid Spine, scheduler generate OCS-vClos using multiply Spines through rewiring links in stage 3.

7.1 Stage 0 and Stage 1 in OCS-vClos

When dealing with tasks with $N_i \leq T$, the scheduler enter into Stage 0 and tries to place the task into one server with least idle GPUs. For tasks with $N_i > T$, in Stage 1, the scheduler tries to select the Leaf switch with the least number of idle servers connected among all leaves that can place tasks. If there is no valid Leaf, scheduler tries to place the task under one Spine in Stage 2.

7.2 Stage 2 in OCS-vClos: Attempt to generate virtual Clos within one Spine

In Stage 2 OCS-vClos tries to generate virtual Clos within one Spine. Benefit from the flexibility of OCS reconfiguration, OCS-vClos can generate virtual Clos within one Spine. Stage 2 can be divided into 2 steps, firstly OCS-vClos choose the Spine with least but enough ports. When the Spine S_m is chosen, OCS O_k can be used to connect to some server which belongs to Leaf L_n and this Spine S_m as long as there are idle ports connecting to L_n and S_m . Secondly, OCS-vClos choose the servers by rewiring links without affecting current tasks. It is worth mentioning that if a task only uses two Leafs,

a special *OCS-vClos* as shown in Fig. 3 can be generated directly through OCS reconfiguration without using Spines. This can reduce the usage of Spine ports and further reduce network resource fragmentation.

7.3 Stage 3 in OCS-vClos: Attempt to generate virtual Clos

If virtual Clos can not be generated within one Spine, *OCS-vClos* tries to generate *vClos* with l Leafs and s Spines. Similarly in *vClos*, we can try to enumerate all possible combinations of p , q and OCS configuration, but it is time consuming. In Algorithm 2 we linearize this problem into an ILP problem and give a solution with an average time cost of 2 second in a cluster containing 2048 GPUs.

8 Implementation

8.1 Simulation Setup

Traning hardware. Our experiment contains both testbed experiment and simulation experiment. The testbed experiment is conducted on 8 Huawei FusionServer G5500 servers. Each server contains 2 Intel Xeon processors and 64GB memory. In order to avoid potential PCIe conflicts, we placed 4 NVIDIAS Tesla V100 GPUs on each server, with each GPU bound to a Mellanox-5 network card that can provide 100gbps of bandwidth. We use four 32 port Huawei CE8850 switches to form a Leaf Spine network, and we add a MEMS-OCS between the Leaf and Spine layers.

Generation of Task Datasets in Testbed Experiments.

Our test dataset includes 100 tasks including data parallel tasks (VGG16, ResNet50, ResNet101), MoE and DLRM, where the batch size of different DML is shown in Table 3. The communication algorithm is randomly selected among Ring, hierarchical Ring, and HD. Communication is based on PyTorch’s built-in communication library DDP. The number of GPUs required for the task is randomly selected from 2, 4, 8, and 16. FIFO is chosen as the default job scheduler.

Implementation of Communication Algorithm. For data parallel tasks, we used Horovod 0.19.2 and a modified version of NCCL 2.7.8 for distributed training. For some commonly used communication algorithms in the industry, such as hierarchical ring and HD algorithms, NCCL does not have native support. Therefore, we have added a custom communication algorithm design layer on top of the NCCL 2.7.8 communication library to support hierarchical ring and HD algorithms. For tasks that require All2All communication such as DLRM and MoE, Torch 1.8.1 is used to run and NCCL and pytorch ddp is used for communication.

Implementation of Routing Scheme. We tested four different routing schemes, namely *OCS-vClos*, *Source Routing*, ECMP, and the ECMP strategy of adding additional 50%

Algorithm 2: OCS-vClos Algorithm

Input: Required job size N_i
Output: $gpus, c_{n,m}^k$

```

/* Calculate the required size within one server */
1  $ServerSize \leftarrow \min(N_i, T)$ ;
/* Calculated the required number of servers */
2  $ReqSevNum \leftarrow \lceil N_i / ServerSize \rceil$ ;
/* Step1: Find valid Leafs with least idle servers */
3  $validLeafList \leftarrow []$ ;
4 for  $n \leftarrow 1$  to  $L$  do
5   if  $L_n.validServer(ServerSize, ReqSevNum)$  then
6      $validLeafList.append(L_n)$ ;
7  $validLeafList.sort()$ ;
8 if  $len(validLeafList) \geq 1$  then
9   /* Step1: Find servers Leafs with least idle GPUs */
10   $L_n^* \leftarrow validLeafList[0]$ ;
11   $gpus = L_n^*.ChooseGpu(ServerSize, ReqSevNum)$ ;
12  return  $gpus, none$ ;
/* Step2: Find vClos within one Spine */
13  $validSpineList \leftarrow []$ ;
14 for  $m \leftarrow 1$  to  $S$  do
15   if  $S_m.freePort > N_i$  then
16      $validSpineList.append(S_m)$ ;
17  $validSpineList.sort()$ ;
18 /* Step3: Find valid vClos using ILP */
19 if  $len(validSpineList) \geq 1$  then
20    $S_m^* \leftarrow validSpineList[0]$ ;
21    $gpus, c_{n,m}^k = S_m^*.ChooseGpu(T, ReqSevNum)$ ;
22   return  $gpus, c_{n,m}^k$ ;
23  $gpus, c_{n,m}^k \leftarrow OCS - FindClos(N_i, S)$ ;
24 return  $gpus, c_{n,m}^k$ ;

```

Table 3: mini-BatchSize set in testbed experiment

TaskType	VGG16	ResNet50/101	BERT	MoE	DLRM
BatchSize	16,32	32,64	4,8	8,16	256,512

ports. Because we don’t have enough switches to test large-scale networking, we use VRF to flexibly virtualize a switch into multiple switches. In *OCS-vClos* and *Source Routing* experiment, we configure the ACL command on the incoming port of the switch to realize *Source Routing*. Four independent Routing table are configured in the Spine switch through VRF, and one Spine switch is virtualized into four logical Spine switches. Inter-task contention may occur when using *Source*

Table 4: Performance on Testbed Environment(s)

Strategy	ECMP	Redundance	SR	vClos
<i>Avg.JRT</i>	95.12	86.23	76.92	74.72
<i>Avg.JWT</i>	72.64	60.73	50.19	45.56

Routing.

The scheduler will reconfigure the network based on the current task requirements and network status. OCS' port re-configuration in our experiment relies on a Telnet-based protocol. A trivial procedure has been developed, allowing for the declarative alteration of OCS links while preserving the integrity of connections that are presently in use, ensuring uninterrupted connectivity and re-configurability throughout the network.

Due to the stochastic nature of the hashing algorithms used in ECMP, contention is inevitable, especially when communication pattern are arbitrary and cannot be determined a priori. As the specific hashing algorithm remains undisclosed, we undertake a series of experiments as referred in Section 3.1 to determine an optimum combination of hashing mode and factors. In these experiments, we randomly select communication peers and analyze the distribution of data flows among the interconnected links that possess equal costs. The 5-tuple (src-ip,dst-ip,src-port,dst-port,port number) is chosen as the hash factor and hash mode is set to 5.

8.2 Performance Analysis in Testbed Experiment

In the testbed experiment, we additionally tested the performance of ECMP when there is an increase of 50% links between the leaves and Spines. The results are shown in Table 4 and Fig. 10, and it can be seen that the performance of *OCS-vClos*, *vClos*, and *Source Routing* is better than ECMP. Even if there are more redundant ports when using ECMP-redundance (rECMP) strategy, network contention still cannot be avoided.

What is more, we found that when using ECMP, *Avg.JRT* for tasks requiring 8 GPUs increase by 8.1%, while *Avg.JRT* of tasks requiring 16 GPUs increased by 17.9%, which may be because AI communication is an all or nothing task, and as long as there exists two flow compete bandwidth, the training time will be affected. This phenomenon suggests that our large tasks may be more susceptible to network contention.

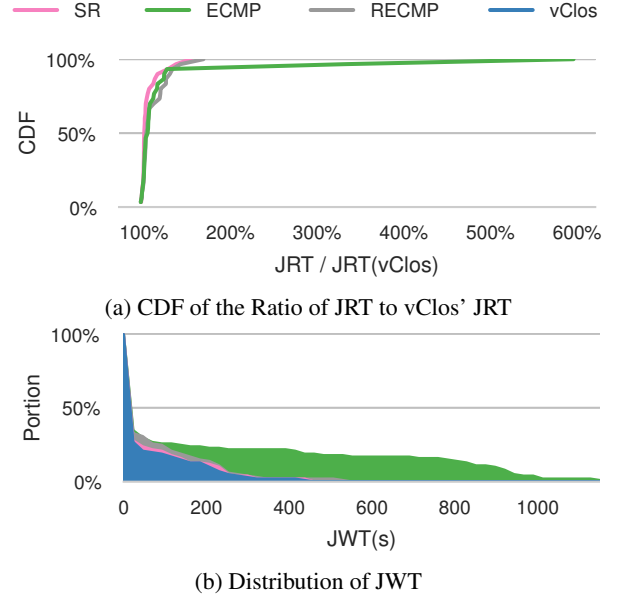


Figure 10: Tasks Performance for DML requires more than 8 GPUs.

9 Large-Scale Simulation Result

9.1 Simulation Experiment Accuracy Adjustment

Since fine-grained simulation runs very slowly facing plentiful flows generated by large-scale DML jobs, we develop a coarse-grained simulator RapidNetSim [5] for large-scale simulation. RapidNetSim is a flow-level network simulation framework that enables fast large-scale simulation with minimal accuracy loss. Compared to traditional data centers, GPU clusters typically enable policies such as PFC, while no deadlocks occur in *vClos* and *OCS-vClos*, so the network environment is relatively simple, and it is possible to use RapidNetSim in large-scale GPU cluster.

Fig. 6 shows the proportion of different types of tasks slowed down by network communication, so the uncoverable communication proportion α is essential for a reliable simulation results. We set α value for different tasks in simulation according to the testbed result.

Fig. 11 shows the distribution of *JRT* and *JWT* on RapidNetSim for 100 jobs tested in testbed experiment, the results of the two experiments are basically consistent.

9.2 Simulation Setup

The simulation experiment is conducted on clusters of 512 GPU CLUSTER512 and 2048 GPU CLUSTER2048 with a coarse grained simulator RapidNetSim. The dataset which contains 5000 DML tasks is based on Helios [18]. Although

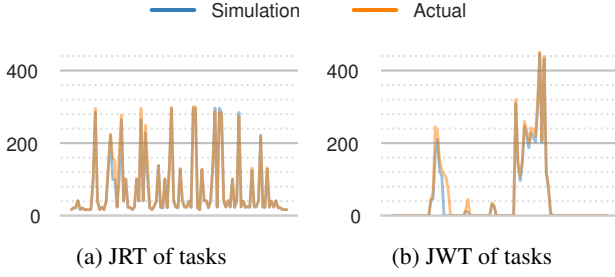


Figure 11: Consistency between Simulation and Actual Results, SR Strategy

the Helios dataset contains task arrival time information, the cluster size we tested is different from that in [18], so in our simulation the arrive time of each tasks follows a Poisson distribution with $\lambda = 120s$. FIFO is chosen as the default job scheduler. We have also tried to conduct simulations based on the fine-grained simulator OMNET, but the simulation speed is too slow. Therefore, we chose RapidNetSim for large-scale networking simulation experiments.

9.3 Baselines for Comparison

We compare our algorithm *vClos* and *OCS-vClos* with *Source Routing*(SR) and ECMP. We also tested two strategies in simulation experiment: i) *Balanced ECMP*(*Balanced*): Beyond traditional ECMP [8, 17], we assume that we can sense the current link load of the network, we randomly select the least congested ports to transmit instead of directly random ports. Its performance is better than traditional ECMP; ii) *Best*: we assume that there is an electric switch with enough ports to connect all servers. In this case, network contention can be completely avoided. This is also the upper bound of performance among all strategies. We measure cluster performance using *Avg.JRT*, *Avg.JCT*, and *Avg.JWT*, and measure the *Stability* of tasks by the average standard deviation of *JCT* of DML with same parameter.

9.4 Performance Analysis of large-scale clusters.

Simulation experiment is conducted to analyse performance of large-scale clusters, where we set $\lambda = 120s$ for CLUSTER512 to keep the system in a steady state with task queuing. The results in Fig. 13a shows that both *vClos* and *OCS-vClos* policy can reduce *Avg.JRT* by 3.1% compared with *Source Routing* and 6.17% compared with *Balanced ECMP*, indicating that the elimination of network contention can reduce the training time of DML tasks.

In Fig. 12b, 12c, and 13a, although *Avg.JRT* only reduces by 3.18% when using *OCS-vClos* compared with SR, *Avg.JWT* is reduced by 65.65%. The reason is that according

to queuing theory, even a slight change in arrival rate may cause the system to enter a non-stationary state for a period of time, resulting in a significant increase in average waiting time. The introduction of OCS layer can reduce *Avg.JWT* by 20.35%, which shows reconfiguring network connections by OCS can effectively reduce network resource fragmentation. *Avg.JCT* is only increased by 3.38% compared with *Best* when using *OCS-vClos*, which indicates that the performance when using *OCS-vClos* is very close to the theoretical upper limit.

Among all strategies, when using *ECMP*, distribution of *JWT* has a significant long tail as shown in Fig. 12a, indicating that hash-collision can cause significant performance fluctuations in certain tasks and affect user experience. Although *Source Routing* and *Balanced ECMP* can effectively reduce hash collisions, it still cannot completely eliminate network contention.

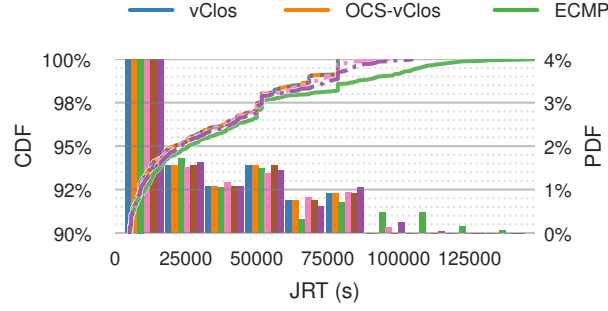
Fig. 12d shows the *Stability* of different kind of jobs under different strategies. The *Stability* is mainly influenced by the *JRT*, and the small task and the task using 160GPU have lower *Stability* due to the large variance of the running time. Some tasks, such as those using 96GPU, have high *Stability* due to the small sample size. However, regardless of the type of task, the *Stability* of the task is relatively low when using the ECMP strategy, which is consistent with the results of our testbed experiment.

9.5 Sensitivity to Cluster Size

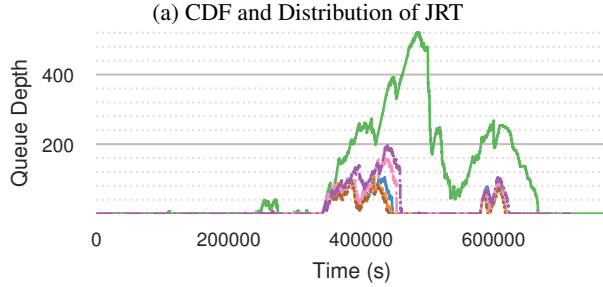
We also conducted simulations on a larger cluster, CLUSTER2048, which contains 2048 GPUs, and switches and servers of the same model as CLUSTER512. In order to keep the system in a reasonable steady state, *lambda* is set to 15s. Similarly to the case in CLUSTER512, Fig. 13b shows that *OCS-vClos* can reduce *Avg.JRT* by 1.59% and *Avg.JWT* by 64.91% compared with *Source Routing*. However, *Avg.JWT* when using *vClos* is only increased by 12.06% compared with *OCS-vClos*, which may be due to more available resources in larger-scale scenarios, resulting in a corresponding reduction in network resource fragmentation. This suggests that using *vClos* in large-scale scenarios is sufficient to ensure task performance.

9.6 Sensitivity to workload

In the simulation experiment, the task arrival time is randomly generated according to the Poisson distribution, and the *Stability* of the queuing system is mainly determined by the arrival rate. Therefore, we try to change the task arrival rate by changing the parameters λ of the Poisson distribution, and analyze the performance of *vClos* and *OCS-vClos* under different arrival rates. To be more specific, we choose $\lambda = 100s, 110s, 120s, 130s, 140s$. Results in table 5 shows that the larger the λ , the more sensitive *Avg.JCT* is to network

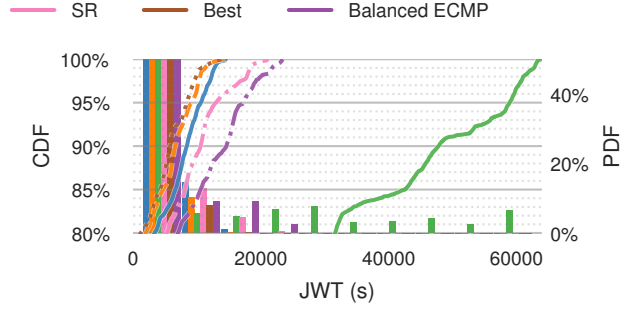


Long Tail effect caused by hash-collision affects *Stability*.

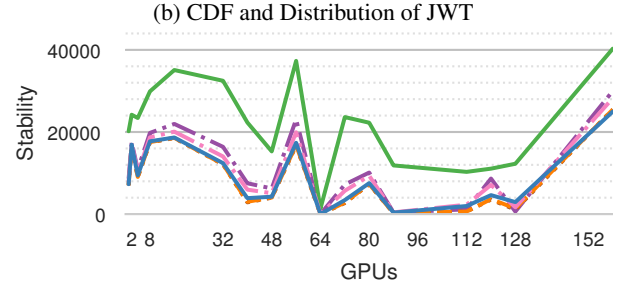


Hash-collision may cause a significant increase in waiting queue depth, affecting JWT and user experience.

(c) Queue Depth



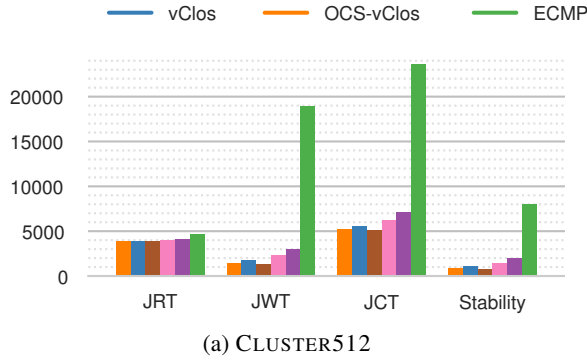
Long Tail effect is less observed in *OCS-vClos* or *vClos*.



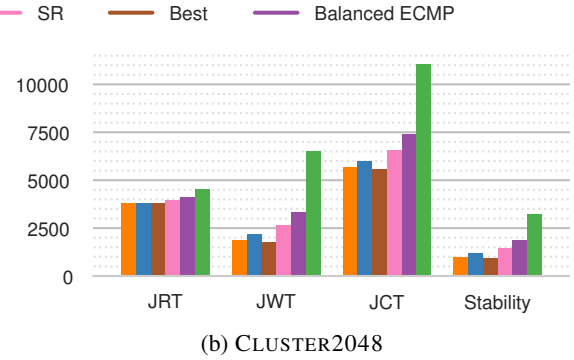
Eliminating network contention can effectively improve task *Stability*, especially for DML with more GPUs.

(d) Stability

Figure 12: Performance of 5000 tasks on CLUSTER512



(a) CLUSTER512



(b) CLUSTER2048

Figure 13: Key Performance Indicators Average on CLUSTER512 and CLUSTER2048.

contention, so it is more necessary to eliminate network contention in a clusters with high utilization. At the same time, when λ is large, the advantage of *OCS-vClos* relative to *vClos* is more obvious, which may be because the probability of network resource fragmentation is also higher when the resource utilization rate is higher. What is more, results of *OCS-Releax* indicate that relaxing the locality limit may greatly increase training time. Under various workload conditions, the performance gap between *OCS-vClos* and *Best* remains within 4%, which verifies that *OCS-vClos* is not sensitive to workload.

9.7 The impact of job scheduler

Many previous articles mentioned the impact of job scheduler, our proposed algorithm mainly focuses on the invocation of network resources, and we wondered whether our framework still powerful under different job schedulers. We implemented fewest GPU first(FF), early deadline first(EDF) in simulation experiments. The results in Table 6 show that when the scheduling algorithm performs better, the sensitivity of *Avg.JCT* to network contention and network fragmentation becomes lower, this is because well performing scheduling algorithms can reduce *Avg.JWT*. But the *vClos* strategy still

Table 5: Avg. JCT(s) of different strategy under different workload on Cluster512

$\lambda(s)$	OCS-vClos	vClos	Best	SR	Balanced	ECMP	OCS-Releax
100	7800.5	9799.6	7514.3	10083.5	12267.1	35032.9	24313.3
110	6368.0	6981.5	6131.8	7449.6	8622.5	25055.6	13768.4
120	5229.9	5550.6	5053.0	6228.6	7105.4	23545.4	11291.4
130	4491.5	4710.4	4340.7	4955.6	5436.9	13118.7	7586.6
140	4025.4	4105.1	3986.5	4209.3	4764.5	11395.6	5917.9

outperforms other strategies.

Table 6: Avg. JCT(s) under different job scheduler

Sched.	OCS-vClos	vClos	Best	SR	Balanced	ECMP
FIFO	5229.9	5550.6	5053.0	6228.6	7105.4	23545.4
EDF	4198.5	4401.4	4176.5	4685.5	5020.9	7658.6
FF	4081.6	4137.2	4029.9	4250.5	4607.8	6667.6

9.8 The impact of Job Distribution

In recent years, with the increasing demand for computing power in tasks, the number of GPUs required for a single task has also been increasing. To verify the scalability of *vClos*, we conducted experiments by regenerating a dataset based on the data in the TPUv4 paper. The experimental results are shown in Table 7, and it can be seen that the performance of *vClos* is same as *OCS-vClos*. This may be because the tasks in TPUv4 are mainly large tasks, so the occupation of network resources is relatively regular, resulting in less fragmentation of network resources, which also implies that as the number of tasks increases, *vClos* is sufficient to ensure task performance.

Table 7: Avg. JRT(s), Avg. JWT(s) and Avg. JCT(s) under different job distribution

Crit.	OCS-vClos	vClos	Best	SR	Balanced	ECMP
JRT	3856.1	3856.1	3856.1	3858.0	3861.6	4968.8
JWT	8101.9	8102.8	8101.9	8115.7	8160.2	17839.8
JCT	11958.1	11958.9	11958.1	11973.7	12021.8	22808.7

10 Relative Work

How to reduce the impact of network overhead on performance has been a hot topic recent years. There exist many schemes. The first scheme is to use parameter compression to reduce the amount of communication data during parameter synchronization. However, lossy compression, such as gradient thinning and gradient quantization [26] [3], may reduce the accuracy of the final AI model.

The second scheme is to adopt the pipeline mode [19, 24, 29, 33, 38, 43, 50], that is, divide the AI model into multiple groups of tensors, so that the tensor group that has completed the calculation can start communication ahead of time while calculating a group of tensors, and the communication time can be hidden in the calculation process. However, those work does not care about conflict network environment on a multi-tenant cluster.

The third solution is to reduce the impact of communication bottleneck between nodes/servers on distributed training tasks by reasonably deploying AI distributed training tasks [18, 21, 30, 35, 52, 54, 58]. But these work generally treat the network as a black box and fail to take into account the potential problems of the network itself.

The fourth solution is to reduce communication overhead by improving the bandwidth utilization. [28, 34, 48, 56] reduce the impact of network contention by traffic management, [51] proposes intelligent cluster network-efficient scheduling methods by resource estimation, but they do not totally eliminate network contention. [9, 22] are designed to avoid the low bandwidth between servers slowing down the entire communication process by designing a multi-stage communication mode. BLink [49] and Plink [27] aim at generating the optimal communication scheme for distributed training through bandwidth detection. EFLOPS [12] try to avoid network contention of a single task by jointly optimizing network topology and AllReduce communication in distributed training. These works help to improve the network bandwidth utilization efficiency of AI distributed training tasks, but cannot fundamentally eliminate the bandwidth bottleneck in a multi-tenant GPU cluster.

11 Discussion

11.1 Stability of MEMS-OXC

The MEMS-OXC we used in the experiment is a first-generation prototype of a certain company, and it is very sensitive to noise. When conduct testbed experiment, we often encounter network instability, which leads to frequent disconnection of RDMA connections. To alleviate this situation, we add soundproofing cotton to MEMS-OXC and regularly adjust the lenses to ensure signal stability. In future work, we will try to use a more reliable MEMS OXC.

11.2 Low communication efficiency of hierarchical algorithms

Although we realize HD and Hierarchical Ring in NCCL, the communication efficiency of these two hierarchical algorithms is relatively slow. We found that the reason for this phenomenon is that NCCL sends data based on a block during each communication, with a block size of 4MB by default. However, in the layered algorithm, our point-to-point communication volume is smaller each time, which can lead to a smaller actual block size and affect communication efficiency. We will further explore this phenomenon in future work.

12 Conclusion

In this paper, we first propose a routing strategy suitable for AI training, and then propose *vClos* to isolate traffic between tasks to eliminate network contention in a multi-tenant cluster. Then we propose *OCS-vClos*, which further reduces fragmentation of resources through network configuration. Our testbed and simulation experiments demonstrate that the proposed strategy can effectively reduce *Avg.JRT*, *Avg.JCT*, and *Avg.JWT* and improve task *Stability* to enhance user experience.

References

- [1] Martín Abadi. Tensorflow: learning functions at scale. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pages 1–1, 2016.
- [2] Ammar Ahmad Awan, Jereon Bédorf, Ching-Hsiang Chu, Hari Subramoni, and Dhabaleswar K Panda. Scalable distributed dnn training using tensorflow and cuda-aware mpi: Characterization, designs, and performance evaluation. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CC-GRID)*, pages 498–507. IEEE, 2019.
- [3] Youhui Bai, Cheng Li, Quan Zhou, Jun Yi, Ping Gong, Feng Yan, Ruichuan Chen, and Yinlong Xu. Gradient compression supercharged high-performance data parallel dnn training. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSp ’21*, page 359–375, New York, NY, USA, 2021. Association for Computing Machinery.
- [4] Pawel Budzianowski and Ivan Vulic. Hello, it’s GPT-2 - how can I help you? towards the use of pretrained language models for task-oriented dialogue systems. *CoRR*, abs/1907.05774, 2019.
- [5] Peirui Cao. Rapidnetsim. <https://github.com/caopeirui/rapidnetsim/>.
- [6] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.
- [7] Zehua Cheng and Zhenghua Xu. Bandwidth reduction using importance weighted pruning on ring allreduce. *CoRR*, abs/1901.01544, 2019.
- [8] Marco Chiesa, Guy Kindler, and Michael Schapira. Traffic engineering with equal-cost-multipath: An algorithmic perspective. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1590–1598, 2014.
- [9] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. volume 1, pages 241–251, 2019.
- [10] Ching-Hsiang Chu, Pouya Kousha, Ammar Ahmad Awan, Kawthar Shafie Khorassani, Hari Subramoni, and Dhabaleswar K. (D K) Panda. Nv-group: Link-efficient reduction for distributed deep learning on modern dense gpu systems. In *Proceedings of the 34th ACM International Conference on Supercomputing, ICS ’20*, New York, NY, USA, 2020. Association for Computing Machinery.
- [11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
- [12] 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. Eflows: Algorithm and system co-design for a high performance distributed training platform. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 610–622, 2020.
- [13] Nikoli Dryden, Naoya Maruyama, Tim Moon, Tom Benson, Andy Yoo, Marc Snir, and Brian Van Essen. Aluminum: An asynchronous, gpu-aware communication library optimized for large-scale training of deep neural networks on hpc systems. In *2018 IEEE/ACM Machine Learning in HPC Environments (MLHPC)*, pages 1–13, 2018.
- [14] Jiawei Fei, Chen-Yu Ho, Atal N. Sahu, Marco Canini, and Amedeo Sapia. Efficient sparse collective communication and its application to accelerate distributed deep learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference, SIGCOMM ’21*, page 676–691, New York, NY, USA, 2021. Association for Computing Machinery.
- [15] Luciano Floridi and Massimo Chiriatti. Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.
- [16] Karen Hao. The computing power needed to train ai is now rising seven times faster than ever before. Accessed 6 May 2019, 2019.
- [17] Christian E. Hopps. Analysis of an equal-cost multipath algorithm. RFC 2992, RFC Editor, 2000.
- [18] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.
- [19] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, Hyoungho Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhiheng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Proceedings of the 33rd*

International Conference on Neural Information Processing Systems, Red Hook, NY, USA, 2019. Curran Associates Inc.

- [20] Sylvain Jeaugey. Massively scale your deep learning training with nccl 2.4. 2019, *nVIDIA Developer Blog*, 2019.
- [21] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, unjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant gpu clusters for dnn training workloads. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '19, page 947–960, USA, 2019. USENIX Association.
- [22] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *CoRR*, abs/1807.11205, 2018.
- [23] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.
- [24] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479. USENIX Association, November 2020.
- [25] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.
- [26] Hyeontaek Lim, David G. Andersen, and Michael Kaminsky. 3lc: Lightweight and effective traffic compression for distributed machine learning. *CoRR*, abs/1802.07389, 2018.
- [27] Liang Luo, Peter West, Arvind Krishnamurthy, Luis Ceze, and Jacob Nelson. Plink: Discovering and exploiting locality for accelerated distributed training on the public cloud. In *MLSys*, 2020.
- [28] Luo Mai, Chuntao Hong, and Paolo Costa. Optimizing network performance in distributed machine learning. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.
- [29] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15, 2019.
- [30] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-aware cluster scheduling policies for deep learning workloads. In *OSDI*, 2020.
- [31] NVIDIA. NVIDIA DGX SuperPOD: Next generation scalable infrastructure for ai leadership. [Online]. Available: <https://shorturl.at/mxzB2>. Accessed Dec., 2023.
- [32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [33] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. A generic communication scheduler for distributed dnn training acceleration. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 16–29, 2019.
- [34] Qiang Qi, Fei Xu, Li Chen, and Zhi Zhou. Rationing bandwidth resources for mitigating network resource contention in distributed dnn training clusters. *CCF Transactions on High Performance Computing*, 3(2):171–185, 2021.
- [35] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *OSDI*, 2021.
- [36] D. Raghavan, P. A. Levis, M. A. Zaharia, and I. Zhang. Breakfast of champions: towards zero-copy serialization with nic scatter-gather. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2021.
- [37] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. Themis: A network bandwidth-aware collective scheduling policy for distributed training of dl models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 581–596, 2022.
- [38] Joshua Romero, Junqi Yin, Nouamane Laanait, Bing Xie, M Todd Young, Sean Treichler, Vitalii Starchenko,

- Albina Borisevich, Alex Sergeev, and Michael Matheson. Accelerating collective communication in data parallel training across deep learning frameworks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1027–1040, 2022.
- [39] Martin Ruefenacht, Mark Bull, and Stephen Booth. Generalisation of recursive doubling for allreduce. In *European Mpi Users Group Meeting*, pages 23–31, 2016.
- [40] Ram Sagar. How to take full advantage of gpus in large language models. Accessed 6 May 2021. <https://rb.gy/tr4hu>, 2021.
- [41] Jo Sanghoon, Hyojun Son, and John Kim. Logical/physical topology-aware collective communication in deep learning training. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 56–68. IEEE, 2023.
- [42] Alexander Sergeev and Mike Del Balso. Horovod: fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.
- [43] Shaohuai Shi, Xiaowen Chu, and Bo Li. Exploiting simultaneous communications to accelerate data parallel distributed deep learning. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.
- [44] Indu Thangakrishnan, Derya Cavdar, Can Karakus, Piyush Ghai, Yauheni Selivonchyk, and Cory Pruce. Her-ring: Rethinking the parameter server at scale for the cloud. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE, 2020.
- [45] Zhihao Tong, Ning Du, Xiaobo Song, and Xiaoli Wang. Study on mindspore deep learning framework. In *2021 17th International Conference on Computational Intelligence and Security (CIS)*, pages 183–186. IEEE, 2021.
- [46] Yuichiro Ueno and Rio Yokota. Exhaustive study of hierarchical allreduce patterns for large messages between gpus. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 430–439. IEEE, 2019.
- [47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [48] Raajay Viswanathan, Arjun Balasubramanian, and Aditya Akella. Network-accelerated distributed machine learning for multi-tenant settings. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 447–461, 2020.
- [49] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Nikhil Devanur, Jorgen Thelin, and Ion Stoica. Blink: Fast and generic collectives for distributed ml. *Proceedings of Machine Learning and Systems*, 2:172–186, 2020.
- [50] Shuai Wang, Dan Li, and Jinkun Geng. Geryon: Accelerating distributed cnn training by network-level flow scheduling. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1678–1687. IEEE, 2020.
- [51] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Zhijiao Jia, Dheevatsa Mudigere, Ying Zhang, Anthony Kewitsch, and Manya Ghobadi. Topoopt: Optimizing the network topology for distributed dnn training. *arXiv preprint arXiv:2202.00433*, 2022.
- [52] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. {MLaaS} in the wild: Workload analysis and scheduling in {Large-Scale} heterogeneous {GPU} clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, 2022.
- [53] Zhenguo Wu, Liang Yuan Dai, Ziyi Zhu, Asher Novick, Madeleine Glick, and Keren Bergman. Sip architecture for accelerating collective communication in distributed deep learning. In *2023 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3. IEEE, 2023.
- [54] 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. Gandiva: Introspective cluster scheduling for deep learning. In *OSDI*, 2018.
- [55] Zimu Xu, Wei Tian, Yingxin Liu, Wanjuan Ning, and Jingjin Wu. A ring topology-based optimization approach for federated learning in d2d wireless networks. *arXiv preprint arXiv:2212.02830*, 2022.
- [56] Weihong Yang, Yang Qin, Zukai Jiang, and Xiaowen Chu. Traffic management for distributed machine learning in rdma-enabled data center networks. In *ICC 2021-IEEE International Conference on Communications*, pages 1–6. IEEE, 2021.
- [57] Zhen Zhang, Chaokun Chang, Haibin Lin, Yida Wang, Raman Arora, and Xin Jin. Is network the bottleneck of distributed training? In *Proceedings of the Workshop on Network Meets AI & ML*, pages 8–13, 2020.
- [58] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis CM Lau,

Yuqi Wang, Yifan Xiong, et al. Hived: sharing a {GPU} cluster for deep learning with guarantees. In *OSDI*, 2020.

- [59] Shizhen Zhao, Rui Wang, Junlan Zhou, Joon Ong, Jeffrey C Mogul, and Amin Vahdat. Minimal rewiring: Efficient live expansion for clos data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 221–234, 2019.

A Appendix

A.1 There is no sample path optimal solution to generate virtual Clos in leaf-spine architecture

To illustrate that There is no sample path optimal solution to generate $vClos$ in leaf-spine architecture, we present a simple example depicted in Figure 14. Initially, the configuration is set as shown in Figure 14a, where Job 1 occupies GPU 1-2 and GPU 2-2, and Job 2 occupies GPU 1-3 and GPU 2-3. Subsequently, Job 3 arrives and occupies GPU 3-2 and GPU 4-2. At this point, two potential solutions emerge for generating a $vClos$ for Job 3. If we opt for the first solution as 14b, wherein Job 1 finishes in the next slot and releases the corresponding GPUs, a $vClos$ cannot be generated for Job 4. Likewise, the second solution, as shown in Figure 14c, encounters the same issue if Job 2 finishes in the following slot. Consequently, it becomes evident that no sample path optimal solution exists for this problem.

A.2 ILP in $vClos$ Stage 2

In this section we introduce a FINDVCLOS Algorithm to generate a $vClos$ with accessible time cost. l_n^i and s_m^i represent the number of virtual leaf and spine generated in L_n and S_m for i -th job, and $c_{n,m}$ represents whether the physical links between L_n and S_m will be used to form the virtual Clos. Our goal in $vClos$ Generation Algorithm is to find the feasible solution of server resource parameter r_n^i and network resource parameter $l_n^i, s_m^i, c_{n,m}$. For ease of reference, notations are summarized in Table 1 and Table 8.

Table 8: Notations used in $vClos$ Stage 2

l_n^i	Num of virtual leaf in L_n for i -th job
s_m^i	Num of virtual spine in S_m for i -th job
$c_{n,m}$	Num of links to be used between between L_n and S_m

Through experiments, it was found that most network fragmentation occurs because suitable leaf port resources cannot be found. Therefore, we first try to ensure that leaf port resources are occupied as neatly as possible, that is, the number of ports in each virtual leaf is as large as possible to a power of 2. The scheduler starts from the initial value of $l = \max\{1, \frac{2^{\lfloor \log_2 N_i \rfloor}}{S}\}$, $s = (l = 1) ? 0 : \frac{N_i}{l}$, and tries every possible combination of p and q until it finds a suitable solution. For each given l and s $vClos$ tries to solve a $vClos$ -ILP which is defined as Algorithm 11.

$vClos$ -ILP Constraints In $vClos$, at most one virtual leaf/virtual spine can be generated from each leaf/spine for one training task, and the number of generated virtual

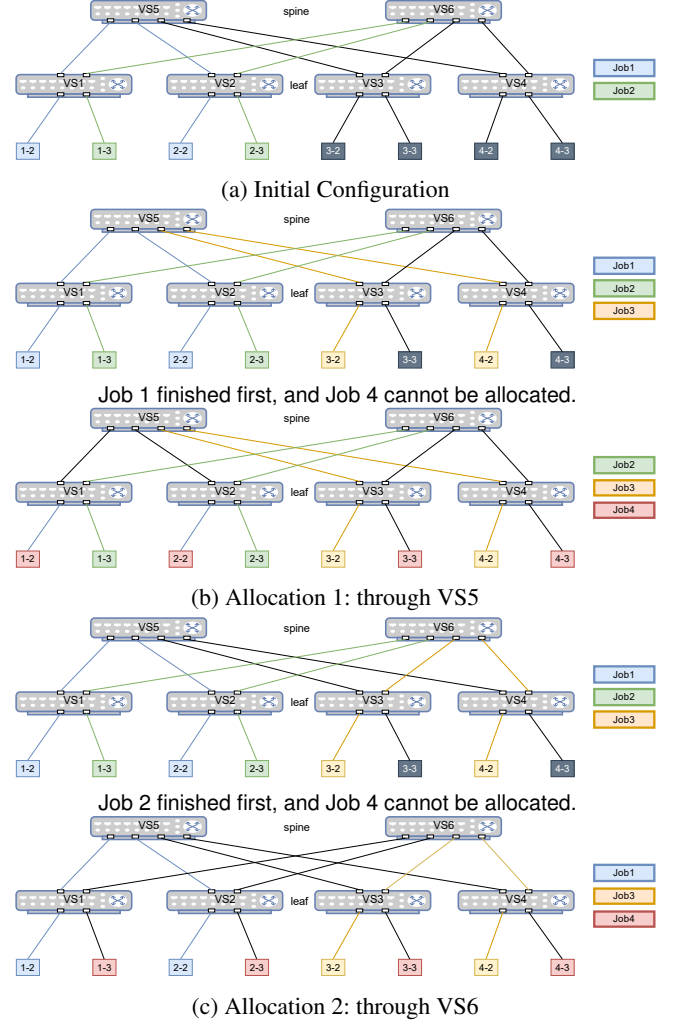


Figure 14: A Simple Example

Algorithm 3: FINDVCLOS Algorithm

Input: Required job size N_i

Output: $gpus, c_{n,m}$

```

1  $l \leftarrow \max\{1, \frac{2^{\lfloor \log_2 N_i \rfloor}}{S}\};$ 
2  $s \leftarrow (l = 1) ? 0 : \frac{N_i}{l};$ 
3 for  $n \leftarrow 1$  to  $S$  do
4    $success, l_n^i, s_m^i, c_{n,m} = vClosILP(l, s);$ 
5   if  $success$  then
6     for  $n \leftarrow 1$  to  $L$  do
7        $gpus.extend(L_n.ChooseGpu(T, \frac{l_n^i * s}{T}));$ 
8     return  $gpus, c_{n,m};$ 
9    $l \leftarrow l \times 2;$ 
10   $s \leftarrow \frac{N_i}{l};$ 
11 return  $none, none$ 
```

leafs/spines should meet the number requirement. i.e.,

$$\begin{cases} \sum_n l_n^i = l \\ \sum_m s_m^i = s \end{cases} \quad (2)$$

By clos network definition, the number of uplinks must be equal to the number of virtual spine switch and the number of downlinks must be equal to the number of virtual leaf switch:

$$\begin{cases} \forall n, \sum_m c_{n,m} = s & \text{if } l_n^i = 1 \\ \forall m, \sum_n c_{n,m} = l & \text{if } s_m^i = 1 \end{cases} \quad (3)$$

Recall that $C_{n,m}$ is the total number of unused link between L_n and S_m . Clearly, the link number to be used $C_{n,m}$ must be smaller than l_n and s_m :

$$\forall n \forall m, c_{n,m} \leq \min\{C_{n,m}, l_n^i, s_m^i\} \quad (4)$$

Due to strict locality constraint, only idle servers can be used to generate $vClos$ and the number of to be used GPU in each virtual leaf should equal to the number of virtual spine:

$$\forall i, \begin{cases} r_n^i \leq R_n \\ r_n^i \geq l_n \\ T * r_n^i = s & \text{if } l_n^i = 1 \end{cases} \quad (5)$$

vClos-ILP Objective By experience, the scheduler should look for solutions that have the least impact on idle leaf switches and spine switches, specifically, we use $RPN(S_m)$ defines the free ports number of S_m before scheduling and $RSN(L_n)$ defines the free servers number of L_n before scheduling. $vClos$ chooses leafs with least free servers and spines with least free ports among all valid leafs/spines, i.e.,

$$\begin{aligned} & \min \sum_m RPN(S_m) * s_m + \sum_n RSN(L_n) * T * l_n \\ & \text{s.t. (2), (3), (4), (5)} \end{aligned} \quad (6)$$

A.3 ILP in OCS-vClos Stage 3

l_n^i and s_m^i represent the number of virtual leaf and spine generated in L_n and S_m , and $c_{n,m}^k$ represents whether the physical links between L_n and S_m through k -th OCS O_k will be used to form the $OCS-vClos$. Recall in Equation 2, each virtual leaf and virtual spine is connected by 1 link, but in OCS-vClos, we can reconfigure the link number between each leaf and spine as long as $\sum_{n,m,k} c_{n,m}^k = l_n^i * s_m^i$, unfortunately this constraint is nonlinear. To linearize this constraint, we define an intermediate variable $L_{n,a}^i$, where $\forall t < a, l_{n,t}^i = 1$ means a virtual leafs are generated from L_n . Naturally, we have that $\sum_a L_{n,a}^i = l_n^i$. For ease of reference, notations are summarized in Table 9.

For given l and s OCS-vClos generates virtual Clos by solving following OCS-vClos-ILP:

Table 9: Notations used in OCS-vClos

l_n^i	Num of virtual leaf in L_n for i -th job
s_m^i	Num of virtual spine in S_m for i -th job
$L_{n,a}^i$	A boolean intermediate variable to represent virtual leaf generated in L_n
$c_{n,a,m,k}$	Whether the physical links between $L_{n,a}^i$ and s_m^i through OCS O_k will be used

Physical Topology Constraints: By clos network definition, the number of generated virtual leaf switches and spine switches should meet the required number, i.e.,

$$\begin{cases} \sum_a L_{n,a}^i = l_n^i \\ \sum_n \sum_a L_{n,a}^i = l \\ \sum_m s_m = s \\ \forall a > 0, L_{n,a}^i \leq l_{n,a-1}^i \end{cases} \quad (7)$$

Due to the locality constraint, the ports number of all virtual leaf switches occupies in L_n should be equal to the ports number of all idle servers connected by L_n , i.e.,

$$\forall n \forall a, L_{n,a}^i * s = T * r_n^i \quad (8)$$

The number of links to be used should be equal to the number of GPUs the user requires, and the number of links between each $l_{n,a}$ and S_m should be either 0 or the number of virtual spines in S_m , i.e.

$$\begin{cases} \sum_{n,a,m,k} c_{n,a,m}^k = N^i \\ \forall n \forall a \forall m, \sum_k c_{n,a,m}^k \leq \min\{L_{n,a}^i, s_m^i\} \end{cases} \quad (9)$$

All links out from $L_{n,a}$ should equal to the used port in $L_{n,a}$, i.e.

$$\begin{cases} \forall n \forall a, \sum_{m,k} c_{n,a,m}^k = L_{n,a}^i * s \\ \forall m \sum_{n,a,k} c_{n,a,m}^k = s_m^i * l \end{cases} \quad (10)$$

Link numbers between all $L_{n,a}^i$ and S_m through O_k can not larger than the potential links between L_n and S_m through O_k , and the number of links between all generated $L_{n,a}^i$ and O_k can be larger than the number of links between L_n and O_k , similarly, the number of links between all generated S_m and O_k can be larger than the number of links between S_m and O_k , i.e.

$$\begin{cases} \forall a \forall k, \sum_{n,m} c_{n,a,m}^k \leq C_{n,m}^k \\ \forall n \forall k, \sum_{m,a} c_{n,a,m}^k \leq C_n^k \\ \forall m \forall k, \sum_{n,a} c_{n,a,m}^k \leq C_m^k \end{cases} \quad (11)$$

OCS-vClos-ILP Objective: the objective function is same as it is in $vClos$, i.e.,

$$\begin{aligned}
& \min \sum_m RPN(S_m) * s_m + \sum_n RSN(L_n) * T * l_n \\
& \text{s.t. } (8), (9), (10), (11)
\end{aligned} \tag{12}$$

Algorithm 4: OCSFINDCLOS Algorithm

Input: Required job size N^i
Output: $gpus, c_{n,m}^k$

```

1  $l \leftarrow \max\{1, \frac{2^{\lfloor \log_2 N_i \rfloor}}{S}\};$ 
2  $s \leftarrow (l = 1) ? 0 : \frac{N_i}{l};$ 
3 while  $l \leq L$  do
4    $success, l_n^i, s_m^i, c_{n,m}^k \leftarrow OCS - vClosILP(l, s)$  if
      $success$  then
5     for  $n \leftarrow 1$  to  $S$  do
6        $gpus.extend(L_n.ChooseGpu(T, \frac{l_n^i * s}{T}))$ 
7     return  $gpus, c_{n,m}^k$ .
8    $l \leftarrow l \times 2;$ 
9    $s \leftarrow \frac{N_i}{l};$ 
10 return  $none, none$ .
```
