Scheduling Multiple AI models in Optical Data Center Networks with a time-division multiplexing-based Multi-Stage Strategy

Binjun Tang, Xiaoliang Chen and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, Hefei, China
†Email: {zqzhu}@ieee.org

***Abstract*—With the significant increase in the size of training datasets and the number of parameters in current models, training models on a single GPU has led to prolonged training times and memory constraints that hinder the accommodation of the entire model's parameters. To address these challenges, distributed parallel training methods, such as data parallelism (DP) and expert parallelism (DP), have been proposed. As the improvement of GPU computing capabilities and maturation of parallel technologies, the proportion of communication during model training has risen, gradually shifting the training bottleneck towards communication. Training large distributed models using optical data center networks (ODCN) presents clear advantages over traditional data center networks based on electrical packet switched (EPS), positioning it as a promising direction for the development of large model training.**

In this paper, we consider scenarios in which various jobs employing different parallelization schemes coexist within an optical data center network, and we model both the network and the jobs. A simple and efficient multi-stage job scheduling scheme based on time-division multiplexing is proposed to schedule the communication and training processes for multiple jobs jointly. Simulation results indicate that our scheduling scheme, compared to a non-time-division multiplexing approach, achieves an average performance improvement of...

***Index Terms*—ODCN, distributed training, parallelism, collective communication, time-division multiplexing, network schedule**

I. Introduction

With the advancement of artificial intelligence (AI), the parameter scale and training dataset of large language model (LLM) have reached astonishing sizes. For instance, GPT-4 has a parameter count of 1.8 trillion and is trained on a dataset comprising approximately 13 trillion tokens [1].Training models using a single GPU often encounters challenges such as excessively long training times and insufficient memory capacity to accommodate the entire model. Consequently, current AI training often employs distributed training schemes [2] within clusters (such as Meta [3] and Colossus), distributing both models and training tasks across each GPU in the cluster. As the computational speed of individual GPU increases, the rate of improvement in training speed for AI models within clusters has gradually slowed, with the proportion of collective communication latency between GPUs increasing in iteration time and becoming a bottleneck in the training process progressively.

As illustrated in *Figure* 1, traditional data center networks utilize EPS for model training in clusters, resulting in inflexible bandwidth allocation between pods that fails to meet the demands of time-varying and skewed traffic. Furthermore, the use of Equal-Cost Multi-Path (ECMP) protocol in traditional networks can lead to hash polarization during training [4], significantly increasing traffic skew and prolonging training times. In contrast, in ODCN illustrated in *Figure* 2, the flexibility to adapt the network topology enables it to accommodate various traffic patterns effectively. Additionally, by employing circuit switching instead of packet switching, this approach eliminates the risk of hash polarization arising from the existence of multiple paths. Consequently, this network design proves to be more suitable for AI model training within clusters.

When employing distributed methods for training models, it is necessary to allocate the dataset and model to each GPU for parallel training. Common parallelization strategies include DP and EP. In practical data center networks, various parallel training tasks often occur simultaneously.

In this paper, we model the iterative process of jobs employing different parallel scheme and aim to jointly schedule the training and collective communication processes of these jobs. Given that the topology reconfiguration delay in the ODCN is significant, on the order of hundreds of microseconds [5], we must also consider the scheduling of reconfiguration time and schemes, which notably increases the complexity of the problem compared to traditional data center networks. If all jobs start their computing process simultaneously and then proceed to collective communication only after completing their training, this cyclical approach would allow for a straightforward derivation of job scheduling plans and Optical Cross-Connect (OXC) topology. However, this strategy entirely separates computation and training times, resulting in underutilization of network resources; thus, it often yields suboptimal results in scenarios where communication demands are high.To rapidly and efficiently schedule various AI training tasks within the network, we propose a multi-stage algorithm based on time-division multiplexing. This algorithm is designed to jointly schedule the training and communication processes of different jobs by overlapping the communication and training times between jobs and obtaining OXC topology reconfiguration scheme of each stage.

The contribution of this paper is as follow:

- In the network model under consideration, we categorize the training patterns of three types of jobs and illustrate that each job can be viewed as a cyclical process comprising four distinct stages.
- A time-division multiplexing-based four-stage strategy is proposed for these jobs, aiming to achieve an easily schedulable and efficient solution within polynomial time complexity.
- Our simulations demonstrate that this approach improves training efficiency by *** compared to aggregation scheduling strategies that do not employ time-division multiplexing.

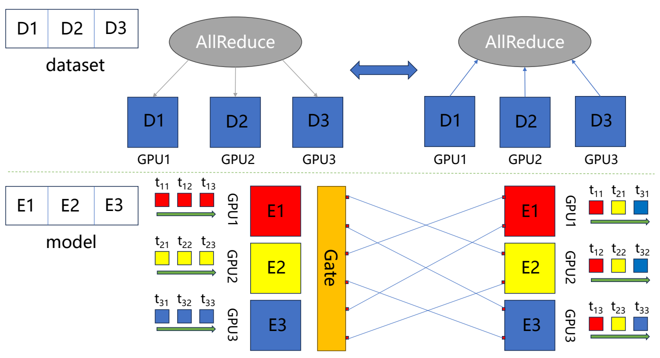

Fig. 3: Training process of DP and EP

II. Problem Description

A. Network model

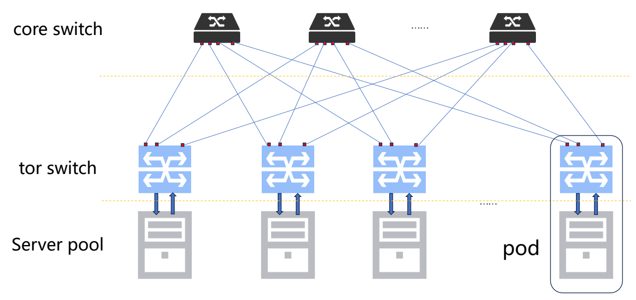

Fig. 1: Traditional data center network

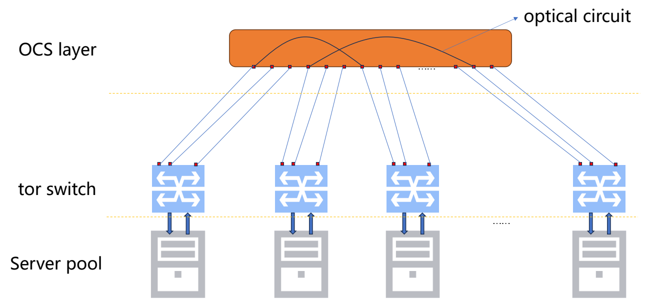

Fig. 2: Optical data center network

Figure 2 illustrates the three-layer Clos network architecture under consideration, which primarily consists of the Optical Circuit Switching (OCS) layer, the Top-of-Rack (ToR) layer, and the server layer. The OCS layer is responsible for facilitating optical interconnection communication between ToR switches, comprising one or more OXCs. Unlike packet switching, the OCS layer requires the establishment of a one-to-one connection between ports prior to communication. Once a communication demand arises, inter-pod traffic is transmitted through this connection from the source port to the destination port. Each connection in the OCS layer operates in half-duplex mode, meaning that at any given time, a connection can provide bandwidth in only one direction. Due to technical constraints, the reconfiguration latency of the OXC connection topology is on the order of hundreds of microseconds, which cannot be overlooked during model training. The ToR layer consists of switches for each pod, with each top-of-rack switch connected to several OXC ports to facilitate communication for both inter-pod and intra-pod traffic. The server layer comprises GPUs across multiple pods, which are utilized to execute training tasks.

In this network model, we have the known traffic demand for each stage in the network, denoted as $D_{i,j}^{k,t}$, the bandwidth between the ToR layer and the server layer, represented as B_i, the number of OXC ports allocated to each ToR, denoted as N_{oxc}, and the bandwidth of each port, represented as B_{oxc}. Based on this information, we calculate the number of OXC connections allocated between each pod and the corresponding communication duration for that stage.

B. Job model

In this paper, we consider serving three types of jobs within the aforementioned network model, each utilizing different classical collective communication schemes: Parameter Server (PS) class, Ring-AllReduce (RAR) class, and AllToAll class. The first two types of jobs employ data parallelism, while the AllToAll job utilizes expert parallelism.

- Data parallelism: As shown in the upper half of *Figure* 3, the dataset is evenly divided, and each GPU storing all parameters of the model and training its respective subset of the dataset. After completing a training process, it is necessary to synchronize the parameters across all GPUs using the Stochastic Gradient Descent (SGD) algorithm [6], after which the synchronized parameters are used for the next training process. During the parameter synchronization process, two commonly used collective communication schemes among GPUs are the PS scheme and the AllReduce scheme.
  - PS: The left side of *Figure* 4 illustrates how the PS architecture facilitates collective communication.

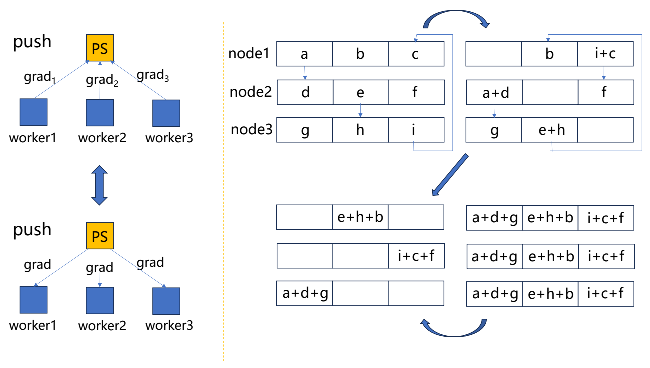

Fig. 4: PS architecture and RAR architecture

The parameter server is employed to synchronize parameters among GPUs. After each worker completes its training process, the parameters are transmitted to the GPU designated as the parameter server. Following the completion of this transmission, parameter aggregation and updating occur on the parameter server. Subsequently, the updated data is multicast to all workers.

- AllReduce: The aggregation and update operations of parameters essentially constitute a type of allreduce. Various algorithms can achieve AllReduce operations without the use of a parameter server. Among these, the RAR algorithm is currently the most commonly used scheme, which requires the nodes to form a ring topology, as illustrated in right part of *Figure* 4. Additionally, hierarchical RAR algorithm [7] and double binary tree algorithm [8] are also being implemented and widely used. However, it is important to note that RAR is the most classic and commonly used algorithm among this class of collective communication algorithms.

- EP [9]: Due to the explosive growth of model parameter capacity, single GPU can no longer accommodate all parameters of large models. As shown on the right side of *Figure* 4, expert parallelism addresses this challenge by partitioning the model into distinct experts, with each GPU storing the parameters of a specific expert. Training data is routed to different experts according to its characteristics. This approach not only significantly enhances training efficiency but also considerably reduces the parameter storage requirements for individual GPU. However, it introduces complex AllToAll collective communication demands, as tokens must be exchanged between every pair of experts to access to the according expert.

Based on the training characteristics of the three types of jobs, we can derive the relationships between the size of traffic and the complexity of topology requirements for each iteration. Jobs utilizing DP necessitate updating the entire model parameters, whereas jobs employing EP only require the transmission of data used for an iteration. Consequently, the ranking of collective communication traffic is as follows: PS $>$ RAR $>$ AllToAll. Conversely, regarding the complexity of topology requirements, AllToAll services exhibit the highest complexity, followed by RAR, and finally PS. The reason RAR jobs are more complex than PS jobs is that, after determining the deployment scheme of GPUs, the PS jobs have only one topology, while the ring formation for RAR jobs remains uncertain.

C. Time division multiplexing scheduling

For RAR and AllToAll jobs, the iterative process can be straightforwardly divided into two stages: the training process and the collective communication process. However, in the case of PS, we cannot treat it in this manner. In the upper part of *Figure* 5, we explain the reasoning. We desire that the traffic demands within each stage do not experience abrupt changes, which would facilitate the scheduling of our OXC topology. It is evident that after the PS service completes aggregation, the traffic demand shifts from push to pull, which may not be accommodated by the OXC topology. On the contrary, although the collective communication process of RAR also involves aggregation, the traffic demand remains unchanged. Consequently, we divide each iteration of the PS service into four stages: local training,push,aggregation and pull. Since the iterations for twice of RAR and AllToAll jobs also consist of four stages, these four stages can adequately represent the iterative processes of all three jobs, with fixed traffic demands in each stage.

To overlap the communication and training processes of different jobs, we divide the jobs into two groups and schedule these groups using time division multiplexing. The example in the lower part of *Figure* 5 illustrates our four-phase scheduling scheme. As shown, each round is divided into four phases. We incorporate the topology reconfiguration time, which occurs after communication, into the communication time. Taking the first stage, consisting of P11 and P21, as an example, we simultaneously train one group of jobs while conducting communication for the second group. After the training and communication tasks in this phase are completed, we start the tasks of the second phase. This scheduling scheme significantly enhances training efficiency in theory compared to a collective scheduling scheme that does not utilize time division multiplexing. Ultimately, the entire scheduling problem can be described as follows: given the known job deployment scheme and job information, we aim to compute a rational grouping method and the corresponding OXC topology within our network to enhance training efficiency.

D. ILP

To describe the previously presented problem rigorously and quantitatively, we have developed an integer linear programming (ILP) model. It should be noted that this model contains nonlinear components, such as the multiplication of parameters and maximization; however, these nonlinear terms

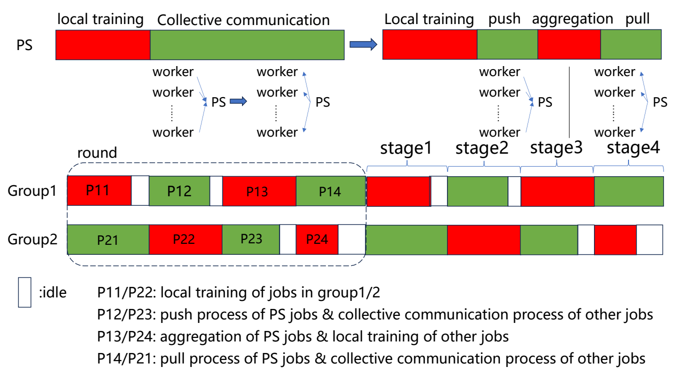
local training
Collective communication
Local training
push
aggregation
pull
PS
worker
round
stage1
stage2
stage3
stage4
Group1
Group2
P11
P12
P13
P14
P21
P22
P23
P24
:idle
P11/P22: local training of jobs in group1/2
P12/P23: push process of PS jobs & collective communication process of other jobs
P13/P24: aggregation of PS jobs & local training of other jobs
P14/P21: pull process of PS jobs & collective communication process of other jobs

Fig. 5

TABLE I: Definition of parameters

G	The set of jobs.
V	The set of pods.
B	The bandwidth capacity of each OXC port.
B_{tor}	The bandwidth capacity between ToR and server pool of each pod
N_link	The amount of OXC ports in each pod.
t^n_{train}	The training time of job n.
F_n	The output traffic size of each GPU belong to job n.
$G_{i,n}$	The amount of GPU in pod i of job n.
E_n	The EP parallel degree of job n.
I_n	The index of pod where the parameter server of PS job n is deployed.
PN^n_i/AN^n_i	The amount of master node of each parallel unit of PS/AllToAll job n in pod i.
U_n	The GPU number of each parallel unit of job n.
RM^n_i	The boolean parameter that equals 1 if there is a master node of RAR job n in pod i and 0 otherwise.
$FI^n_{i,j}$	The size of inter-unit traffic from pod i to pod j of PS job n in push process/other job in collective communication process.
RN_n	The set of pod index that master nodes of job n are deployed.

can be expressed in a linear form. Due to space constraints, we omit the detailed linearization process.

Objective:

Our goal is to enhance training efficiency; thus, the optimization objective is set to minimize the duration of a round.

$$\text{Minimize} \quad \sum_{t=0}^{1}\sum_{k=0}^{1} t_{k,t} \tag{1}$$

Constraints

TABLE II: Definition of variables

G_1/G_2	The set of jobs in group 1 /group 2.
$G_{i,ps}/G_{i,RAR}$ $/G_{i,alltoall}$	The set of ps/RAR/AllToAll jobs in group i.
$t_{k,t}$	The duration of each stage. Stage index in a round is equal to 2*t + k + 1 and k,t $\in \{0,1\}$.
$D^{k,t}_{i,j}/D^{k,t}_i$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand of pod i in each stage.
$P^{n,t}_{i,j}/P^{n,t}_i$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand in pod i of PS job n. If t = 0, job n is in push process and pull process otherwise.
$R^n_{i,j}/R^n_i$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand in pod i of RAR job n.
$A^n_{i,j}/A^n_i$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand in pod i of AllToAll job n.
$L^{k,t}_{i,j}$	The amount of OXC links between pod i and pod j in each stage.
$T^{k,t}_{i,j}$	The completion time of OXC traffic between pod i and pod j in each stage.
$t^{k,t}_{comm}$	The communication time of each stage.
$Ring^n_{i,j}$	The boolean variable that equals 1 if a OXC link is desirable from pod i to pod j and 0 otherwise.

$$\begin{cases} D^{k,t}_{i,j} = \sum\limits_{p\in G_{k,ps}} P^{p,t}_{i,j} + \sum\limits_{r\in G_{k,RAR}} R^r_{i,j} + \sum\limits_{a\in G_{k,alltoall}} A^a_{i,j} \\ D^{k,t}_{i} = \sum\limits_{p\in G_{k,ps}} P^{p,t}_{i} + \sum\limits_{r\in G_{k,RAR}} R^r_{i} + \sum\limits_{a\in G_{k,alltoall}} A^a_{i} \\ D^{k,t}_{i,j} = B * L^{k,t}_{i,j} * T^{k,t}_{i,j} \\ t^{k,t}_{comm} = \max\{T^{k,t}_{i,j}, \dfrac{D^{k,t}_i}{B_{tor}}\} \end{cases}$$
$$, \forall i,j \in V, i \neq j, \forall k,t \in \{0,1\} \tag{2}$$

Eq. (2) describes the composition of the inter-pod traffic and intra-pod traffic within the network during each phase, specifically comprising three types of traffic. Given that the RAR traffic is relatively high in the current scenario, we neglect the aggregation time of the RAR traffic during the collective communication process, as it has minimal impact on our problem. By identifying the communication time at the bottleneck, we obtain the total duration of the collective communication for that phase.

$$t_{k,t} = max\{t^{k,t}_{comm}, t^n_{train}\}, \forall n \in J/G_k, \forall t,k \in \{0,1\} \tag{3}$$

Eq. (3) illustrate that the duration of each stage is the maximum of communication time and training time.

$$\begin{cases} P^{n,0}_{i,I_p} = P^{n,1}_{I_p,i} = PN^n_i * F_n * U_n + FI^n_{i,I_p}, \forall n \in G_{1,ps} \cup G_{2,ps} \\ P^{n,0}_{i,m} = P^{n,1}_{m,i} = FI^n_{i,m}, \forall m \in V, m \neq i, I_p \\ R^n_{i,j} = Ring^n_{i,j} * F_n * U_n + FI^n_{i,j}, \forall n \in G_{1,RAR} \cup G_{2,RAR} \\ A^n_{i,j} = AN^n_i * AN^n_j * F_n * U_n + FI^n_{i,j}, \forall n \in G_{1,alltoall} \cup G_{2,alltoall} \end{cases} \quad ,\forall i,j \in V, i \neq j \tag{4}$$

$$\begin{cases} P^{n,t}_i = G_{i,n} * F_n, \forall n \in G_{k,ps} \\ R^n_i = 2 * G_{i,n} * F_n, \forall n \in G_{k,RAR} \\ A^n_i = 2 * G_{i,n} * F_n * (U_n - 1), \forall n \in G_{k,alltoall} \end{cases} \quad ,\forall i \in V, \forall t \in \{0,1\} \tag{5}$$

Eq. (4) and (5) show that how we can specify the traffic of each type of job according to the deployment scheme.

$$\sum_{i=1}^{|V|}(L^{k,t}_{i,j} + L^{k,t}_{j,i}) \leq N\_link \tag{6}$$

Eq. (6) ensures that the number of OXC ports of each pod is limited.

$$\begin{cases} \sum_{i=1}^{|V|} Ring^n_{i,j} = \sum_{i=1}^{|V|} Ring^n_{j,i} = RM^n_i \\ \sum_{i=1,i\neq m}^{|V|} Ring^n_{i,j} = \sum_{i=1,i\neq m}^{|V|} Ring^n_{j,i} = 0 \end{cases} \tag{7}$$
$$\forall n \in G_{1,RAR} \cup G_{2,RAR}, \forall m \in RN_n, \forall i,j \in |V|, i \neq j$$

Eq. 7 ensures that topology we form for RAR job should be a single ring.

III. Algorithm

A. Overall procedure

Algorithm 1 illustrates our approach to iteratively solving for the optimal grouping using hill-climbing method. In *Line* 1, the grouping scheme is initialized by placing all jobs into set A while leaving set B empty. *Lines* 2-4 compute the OXC topology for each stage under the initial grouping, along with the corresponding durations, such that the sum of the solutions from the four stages yields our initial solution for optimization. *Line* 3 determines whether reconfiguration is needed between adjacent stages and calculates the total reconfiguration time. *Lines* 5-12 detail the iterative process of the hill-climbing method to optimize $t_{feasible}$. Specifically, *Line* 7 updates the grouping based on the best neighborhood solution from the previous iteration. *Lines* 8-10 traverse all jobs, with *Line* 9 involving the swapping of the group of each job to obtain the corresponding neighborhood groupings A_n and B_n. *Line* 10 employs *Algorithm* 2 to compute the respective duration for each stage. When $t_{feasible}$ is less than all neighborhood solutions, we conclude that the optimal solution has been found, at which point the optimal grouping and corresponding OXC topology are output.

Algorithm 1: Overall Procedure of Hill Climbing

```
1  Initial group A = J,B = $\emptyset$;
2  Calculate corresponding OXC topology $L^{k,t}_{i,j}$ and
     completion time $t_{k,t}$ of each stage by applying
     Algorithm 2;
3  Determine whether reconfiguration existing or not and
     calculate reconfiguration time $\varepsilon$ in single round;
4  $t_{feasible} = \sum_{t=0}^{1}\sum_{k=0}^{1} t_{k,t} + \varepsilon$;
5  while $t_{feasible} < min\{t_{neighbor}\}$ do
6      $t_{neighbor} = \emptyset$;
7      update A,B based on best neighbor solution;
8      for n in set G do
9          Exchange the group of job n and we update
             $A_n, B_n$;
10         Apply Algorithm 2 to new group and calculate
             its $L^{k,t}_{i,j}$ and $t_{k,t}$;
11         Put $\sum_{t=0}^{1}\sum_{k=0}^{1} t_{k,t}$ into $t_{neighbor}$
12     end
13 end
14 Return A, B, $t_{feasible}$ and corresponding OXC
     topology;
```

B. Count OXC topology of each group

Algorithm 2 is designed to compute the OXC topology for each group. *Line* 1 initializes the traffic demand in the network and the boolean value of connection demand, denoted as LB, which equals 1 when there is traffic from pod i to pod j, and 0 otherwise. *Lines* 2-4 traverse all non-RAR jobs, as the traffic for these jobs is fixed and can be inferred; during this process, the traffic demand and connection demand boolean values are updated. *Lines* 5-8 handle all non-RAR jobs, for which we need to determine a topology configuration for the constituent ring topology. We aim to use *Algorithm* 3 to obtain a schedule that minimizes the increase in connection demand, which also aids in reducing transmission time. *Line* 9 initializes the OXC topology by allocating an OXC connection for all existing traffic demand connections. *Line* 10 calculates the completion time of traffic within each pod. *Lines* 11-17 detail the process of allocating OXC connections based on completion time. In *Line* 12, the completion time for each pair of pods is calculated to identify the maximum completion time and the corresponding pod pair. *Lines* 13-16 describe the conditions under which OXC allocation concludes: either when adding OXC connections no longer enhances transmission efficiency or when no additional connections can be made at the bottleneck, then the allocation should be finished. Otherwise, an additional OXC connection is allocated at the bottleneck. Ultimately, we obtain the OXC topology configuration along with the corresponding transmission duration for this configuration.

C. How to form a ring

Algorithm 3 considers how to form a ring topology in RAR scenarios. We aim for this ring topology to exhibit a higher degree of redundancy with the current topology, which

Algorithm 2: OXC topology algorithm of each group

```
1  Initial traffic demand $D_{i,j}^{k,t}/D_{i}^{k,t}$ and $LB_{i,j}^{k,t}$;
2  for each job n in $G/G_{RAR}$ do
3      Specify traffic demand of job n and update
         $D_{i,j}^{k,t}/D_{i}^{k,t}$ and $LB_{i,j}^{k,t}$;
4  end
5  for each job in $G_{RAR}$ do
6      Apply Algorithm 3 to decide how to form a ring;
7      update $D_{i,j}^{k,t}/D_{i}^{k,t}$ and $LB_{i,j}^{k,t}$;
8  end
9  Initial OXC topology $L_{i,j}^{k,t} = LB_{i,j}^{k,t}$;
10 Calculate $T_{tor}$;
11 while 1 do
12     Calculate the completion time of each pair of pods
         and find the maximum time $T_{OXC}$ and
         corresponding pod pair $(i_m, j_m)$;
13     if $T_{tor} \geq T_{OXC}$ or there is no available port in $i_m$
         or $j_m$ then
14         break
15     end
16     Add a OXC link from pod $i_m$ to pod $j_m$ and
         update $L_{i,j}^{k,t}$;
17 end
18 Return $L_{i,j}^{k,t}$, $\max\{T_{tor}, T_{OXC}\}$
```

Algorithm 3: Ring algorithm

```
1  Initial $In\_degree_i = Out\_degree_i$ = 0;
2  Calculate Degree and link boolean value $L_{sub}^{i,j}$ in the
     sub-topology comprising only the root nodes.;
3  Initial Ring = $L_{sub}^{i,j}$;
4  for each node n in root_node do
5      while $In\_degree\_n > 1$ do
6          Find the neighboring node $n'$ with the highest
             $Out\_degree$;
7          Delete the link from $n'$ to n in Ring and update
             $In\_degree$ and $Out\_degree$;
8      end
9      while $Out\_degree\_n > 1$ do
10         Find the neighboring node $n'$ with the highest
             $In\_degree$;
11         Delete the link from n to $n'$ in Ring and update
             $In\_degree$ and $Out\_degree$;
12     end
13 end
14 Identify all disjoint sub-topologies of Ring and
     severing the sub-rings;
15 Connect all sub-topologies sequentially and update
     Ring;
16 Return Ring;
```

can significantly reduce the complexity of the topology and improve the degree of matching between the topology and the traffic matrix. To connect these nodes into a ring, we need to ensure that each node has an in-degree and out-degree of 1 in the topology, and that no sub-rings exist. In *Line* 2, we identify all relevant nodes associated with this job and form a sub-topology $L_{sub}^{i,j}$ consisting of edges between these nodes, subsequently calculating their in-degrees and out-degrees. *Line* 3 initializes our ring topology and, through appropriate transformations, forms a genuine ring with the minimum number of increasing connections. *Lines* 4-12 iterate through all nodes, ensuring that their in-degrees and out-degrees in the ring topology do not exceed 1. For example, in *Lines* 5-8, we control the in-degree by locating the neighboring node with the highest out-degree and removing the connection between these two nodes in the ring topology; a similar method is employed for controlling the out-degree. In *Line* 14, we identify all mutually independent sub-topologies within this ring topology and locate any sub-rings among them, cutting these sub-rings and updating the ring topology. Finally, in line 15, we sequentially connect these sub-topologies to obtain a ring topology.

IV. Performance evaluations

A. Simulation setup

Our large-scale simulation is conducted in a three-layer Clos network as illustrated in *Figure* 2. This network comprises 256 pods, each deploying 2048 A100 GPUs, whose utilization rate is equal to 0.4. To ensure that *Algorithm* 1 satisfies the constraints on the number of OXC ports in initial scenarios, thereby preventing optimization error through iteration, we assume that the OCS layer consists of 32 OXCs, each can provide 128 ports. All pods are divided into 32 groups, with each group containing 8 pods. Each group of pods utilizes one OXC and is equipped with 16 OXC ports, where each port has a bandwidth of 150G and an oversubscription ratio of 1:1.

TABLE III: Parameters size of each job

	1B	3B	8B	90B
parameter size (GB)	2.46	6.42	21.2	177.6

In the default scenario, the type od job in this network is Llama 3, with computational precision set to FP64. The parallel degree of each model is randomly selected from 2 to 16. The model size is randomly selected from existing options of 1B, 3B, 8B, and 90B, corresponding to the parameter quantities presented in *Table* III. The communication strategies for the workload are randomly chosen from PS, RAR, and AllToAll. For the first two strategies, we assume this job utilizes DP, while for AllToAll, we assume EP is employed. The training time for each workload is given by *Eq.* (8). Under the default scenario with recomputation enabled, $K = 8$, the context length is set to 8k, Φ represents the parameter size provided in the table, F_{GPU} denotes the peak FLOPs of the GPU, and *efficient* indicates the GPU utilization rate.By default, $F_{GPU} = 26TFLOPs, efficient = 0.4$.

$$t_{train} = \frac{K \times tokens \times \Phi}{F_{GPU} \times efficient} \tag{8}$$

B. Deployment scheme

Before our scheduling process, it is essential to determine which GPUs in the network will be utilized for each parallel unit of each job. We require that each job is deployed exclusively within a single pod group to prevent the issue of insufficient OXC ports caused by an excessive number of related pods for PS and AllToAll jobs. We deploy the parallel units of the jobs by averaging the traffic across the pods. First, we identify the pod group with the minimum total traffic and sufficient remaining server resources, and we allocate all GPUs for that job within this pod group. Subsequently, the parallel units are deployed within the pod group based on the averaged traffic across the pods.

C. Large-scale simulation

Our large-scale simulations observed the comparison of single-round training times with varying numbers of jobs and different context lengths. The number of jobs is closely related to the communication time, while the context length is associated with the training time, as shown in the figure...

V. Conclusion

In this work, we consider scenarios in the ODCN where multiple jobs are trained simultaneously using different parallel schemes. We propose to group the jobs for time-division multiplexing to overlap the training and communication processes, thereby enhancing training efficiency. To achieve this, we designed an ILP along with relevant grouping algorithms and topology computation algorithms. Our large-scale simulations also demonstrate that our approach can significantly improve training efficiency.

References

[1] D. Patel and G. Wong, "GPT-4 architecture, infrastructure, training dataset, costs, vision, MoE," Jul. 2023. [Online]. Available: https://www.semianalysis.com/p/gpt-4-architecture-infrastructure.

[2] J. Verbraeken *et al.*, "A survey on distributed machine learning," *CSUR*, vol. 53, pp. 1–33, Mar. 2020.

[3] L. Kevin, G. Adi, and O. Mathew, "Building Meta's GenAI infrastructure," Mar. 2024. [Online]. Available: https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/

[4] K. Qian *et al.*, "Alibaba HPN: A data center network for large language model training," in *Prof.of SIGCOMM 2024*, pp. 691—706, Aug. 2024.

[5] J. Zerwas, W. Kellerer, and A. Blenk, "What you need to know about optical circuit reconfigurations in datacenter networks," in *Proc. of ITC 2021*, pp. 1–9, Aug./Sept. 2021.

[6] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," in *Proc. of NeurIPS 2010*, pp. 1–9, Dec. 2010.

[7] X. Jia *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," 2018. [Online]. Available: https://arxiv.org/abs/1807.11205

[8] P. Sanders, J. Speck, and J. L. Träff, "Full bandwidth broadcast, reduction and scan with only two trees," in *Prof. of PVM/MPI 2007*, p. 17–26, Sep. 2007.

[9] W. Cai *et al.*, "A survey on mixture of experts," *ACM Comput. Surv*, Oct. 2024.