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 [12], 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 effectively and efficiently schedule multiple AI training jobs within the network, we propose a time-division multiplexing-based multi-stage algorithm for the joint scheduling of the training and communication processes of each job.

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.

II. Problem Description

A. Network model

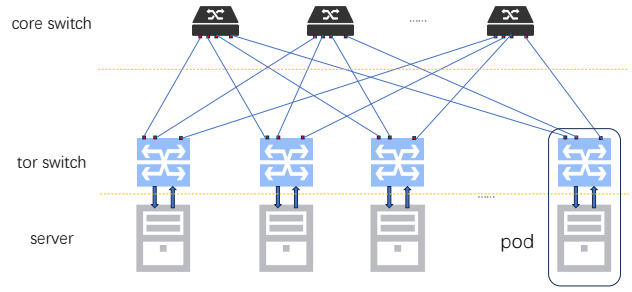

Fig. 1: Traditional data center network

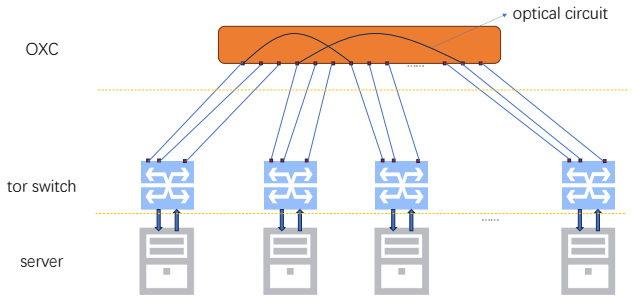

Fig. 2: Optical data center network

Figure 2 illustrates the three-layer Clos architecture under consideration, which primarily consists of the OXC layer, top-of-rack(ToR) layer, and server layer. The OXC layer is utilized to fulfill optical interconnect communication between ToRs. Unlike electrical packet switching, the OXC layer necessitates the establishment of a one-to-one connection between ports prior to communication, allowing traffic to flow from the source port to the destination port through this connection. With the introduction of circulators, each connection within the OXC layer enables full-duplex communication, meaning that a single connection can simultaneously provide bandwidth in both directions. Due to technical constraints, the reconfiguration delay of the OXC topology is on the order of hundreds

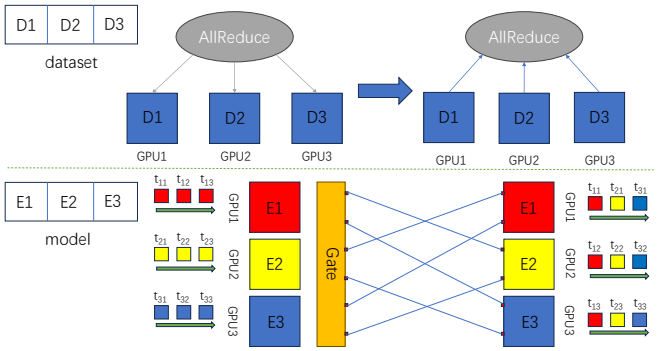

Fig. 3: Training process of DP and EP

of microseconds, which is a significant factor that cannot be overlooked during model training.The ToR layer consists of programmable ToR switches for each pod, with each switch connecting to several OXC ports for communication with other ToR switches. These switches are also capable of performing parameter aggregation for distributed training in place of parameter server. The server layer comprises GPUs in multiple pods for executing training tasks. Upon the generation of communication demands, traffic enters the ToR corresponding to the source pod and is then routed through the OXC to the ToR corresponding to the destination pod.

We utilize two planes to characterize the traffic demand within the network. Due to the presence of intra-pod traffic and the utilization of INC, the traffic entering the ToR from the server layer does not necessarily flow into the OXC; similarly, the traffic transmitted to the ToR via the OXC connection may not enter the pod corresponding to this ToR. In this context, the OXC traffic matrix alone cannot comprehensively describe the traffic state. To address this issue, we employ the OXC plane and the intra-pod plane to represent the traffic demand in the network. The OXC plane is modeled as a bidirectional acyclic graph, with $B_{oxc}^{i,j}$ denoting the traffic demand from pod i to pod j within the OXC layer. The intra-pod plane describes the traffic from the ToR to the corresponding servers within the pod, represented by B_{pod}^{i} for the traffic in pod i.Due to the symmetry of collective communication, the size of traffic from the ToR to the servers is equal to the size of traffic in the reverse direction.

B. Job model

In this paper, we consider three types of collective communication workloads: PS type, RAR type, and AllToAll type. The first two types employ data parallelism, while the AllToAll type utilizes expert parallelism.

- DP [5]: As shown in the upper part of *Figure* 3, the dataset is evenly partitioned, with each GPU storing all parameters of the model and training its assigned subset of the dataset. After completing a training iteration, it is necessary to synchronize the parameters across the GPUs using the Stochastic Gradient Descent (SGD) [6] algorithm, after which the synchronized parameters are

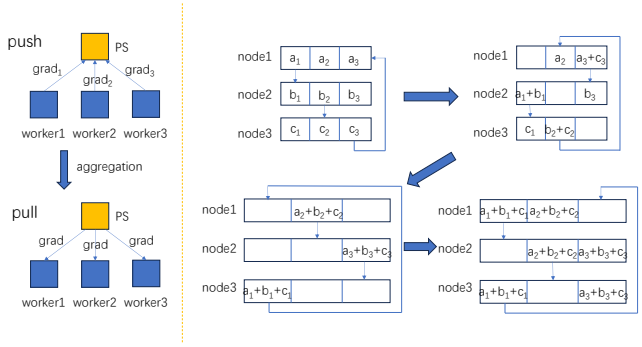

Fig. 4: PS architecture and RAR architecture

used for the subsequent training iteration. During the parameter synchronization process, two commonly utilized collective communication schemes between GPUs are the Parameter Server (PS) [7] scheme and the AllReduce scheme. PS and Ring-AllReduce (RAR) schemes are compatible with in-network computing (INC) [8], [9]. INC enables the ability of switches within the network to synchronize parameters, allowing computation in switches and data transmission to occur simultaneously. In other words, with the implementation of INC, the aggregation and communication processes during each iteration are synchronized. In data center networks, programmable top-of-rack (ToR) switches such as Tofino can be employed to enable the In-network computing ability of network.

- PS: The left side of *Figure* 4 illustrates how the PS architecture facilitates collective communication. 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:Parameter aggregation and updating essentially represent an AllReduce operation, which can be realized through various algorithms without PS. Among these algorithms, the RAR algorithm is currently the most prevalent approach, as depicted in the right side of *Figure* 4. In addition, the double binary tree [10] algorithm and others are also implemented within the collective communication library NCCL.

- EP [11]: 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.

C. Time division multiplexing scheduling

Regardless of the type of job, each iteration can be divided into a training phase and a communication phase, during which the GPUs are idle. We propose a time-division multiplexing scheduling approach, wherein one portion of the jobs undergoes communicating while another portion utilizes the idle GPUs for training. This effectively employs the idle GPUs during communication, significantly enhancing GPU utilization and indirectly improving training efficiency.To simplify the scheduling process, we classify the job set J into two groups, A and B. While the A group engages in collective training, the B group conducts collective communication. We refer to this process as Process 1. Once both groups complete, the B group utilizes the GPUs from the A group for training, while the A group engages in communication; this is termed Process 2. These two processes utilize different OXC topology to accommodate varying traffic demands. The combination of these two processes constitutes a single round, and the entire scheduling process consists of several identical rounds. To enhance training speed, we aim to minimize the duration of a single round, denoted as t_{round} , thus setting our optimization goal to $\min\{t_{round}\}$.We address this issue by scheduling the group and GPU allocation of jobs, while employing a more efficient OXC topology.

D. ILP

$$\min\{\sum_{t=1}^{2}\sum_{i=1}^{2}(t_{i,t})\} \tag{1}$$

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

$$t_{k,t} = max\{t^t_{k,comm}, t^i_{train}\}, \forall i \in J/group_k, \forall t, k \in \{1,2\} \tag{3}$$

$$\begin{cases} P_i^{p,t} = G_{i,p} * F_p, \forall p \in group_{k,ps} \\ R_i^r = 2 * G_{i,r} * F_r, \forall r \in group_{k,ring} \\ A_i^a = 2 * G_{i,a} * F_a * (U_a - 1), \forall a \in group_{k,alltoall} \end{cases} , \forall i \in P, \forall t \in \{1, 2\} \tag{4}$$

$$\begin{cases} P_{i,n_p}^{p,1} = P_{n_p,i}^{p,2} = PN_i * F_i, \forall p \in group_{1,ps} \cup group_{2,ps} \\ R_{i,j}^r = Ring_{i,j} * F_i, \forall r \in group_{1,ring} \cup group_{2,ring} \\ A_{i,j}^a = AN_i * AN_j * F_i, \forall a \in group_{1,alltoall} \cup group_{2,alltoall} \end{cases} , \forall i, j \in P, i \neq j \tag{5}$$

$$\begin{cases} group_1 \cap group_2 = \emptyset \\ group_1 \cup group_2 = J \\ group_1 = group_{1,ps} \cup group_{1,ring} \cup group_{1,alltoall} \\ group_2 = group_{2,ps} \cup group_{2,ring} \cup group_{2,alltoall} \end{cases} \tag{6}$$

III. Algorithm

A. Overall procedure

Algorithm 1 illustrates the iterative approach we employed using the hill-climbing method to determine the optimal solution. *Line* 1 employs formula (1) to estimate the training time per GPU for each job. Here, K is a constant; specifically, K=8 when the model employs recomputation, and K=6 otherwise.Φ represents the quantity of model parameters.F_{GPU} denotes the computational speed of the GPU.We can utilize the size of parameters to estimate the traffic size of data parallelism job, while expert parallelism job estimate their traffic size based on the number of tokens. *Line* 2 initializes two groups, A and B, along with the neighborhood solution set NS. *Line* 3 - 13 detail the iterative process of the hill-climbing method, where *Line* 4-8 indicate the replacement of the iterative solution with the optimal solution from the neighborhood. Given that GPU resources are fully utilized in *Algorithm* 2, if communication times are prolonged during each process, the GPU utilization for each pod can be proportionally reduced without affecting the results. *Line* 9 identifies all neighborhood solutions using the iterative solution. *Line* 10 calculates the OXC topology for both processes using *Algorithm* 2. *Line* 11 computes the training time for the jobs in both groups. Finally, *Line* 12 adds the durations of each process to obtain t_{round}.

$$t_{train} = \frac{K \times tokens \times \Phi}{F_{GPU} \times efficient} \tag{7}$$

B. Count OXC topology of each group

Algorithm 2 is utilized to compute the OXC topology for each group. *Line* 1 predicts the traffic size of each job and puts them in descending order within set G. *Line* 2 determines the GPU usage of each job by utilizing all GPUs in the network while averaging the training time of each job. *Line* 3 initializes the traffic between ToR and servers, as well as the traffic between ToR and OXC within each pod. *Line* 4 - 9 detail the process of deploying and connecting topology. Specifically, *Line* 5 - 8 address the scenario where the job is of the PS type, requiring the identification of a pod with the lowest OXC traffic to serve as the PS. *Line* 9 initializes a root node set for this service, which will be used to form a ring topology in subsequent steps. *Line* 10 - 15 deploy the parallel units of the job sequentially, adhering to the principle of deploying on the pod with the minimum traffic. If a complete unit cannot be deployed on this pod, search for the next available pod until the entire unit is successfully deployed; here, exc_rate represents the ratio of the OXC plane bandwidth to the intra-pod plane bandwidth for each pod. *Line* 16 - 18 account for jobs of the ring-AllReduce type, apply *Algorithm* 3 to determine the ring formation scheme. Finally, *Line* 20 - 21 indicate that upon completion of the deployment for all jobs, the traffic matrix can be obtained, which will subsequently guide the acquisition of the OXC topology.

Algorithm 1: Overall Procedure of Hill Climbing

```
1  Predict the communication traffic F^{i,j}_{oxc,k}, F^i_{pod,k} of
     each workload and the iteration duration per GPU ;
2  Initial group A = J,B = ∅, NS = ∅;
3  while max{NS} < t_round or NS = ∅ do
4      if NS ≠ ∅ then
5          Modify the group A,B by the best
             neighborhood solution.;
6          t_round = max{NS};
7          Adjust GPU resource allocation based on
             communication duration.;
8      end
9      Obtain a set of neighborhood solutions by
         exchanging the group of each workload.;
10     Apply Algorithm 2 to get OXC bandwidth
         B^{i,j}_{oxc1}, B^{i,j}_{oxc2} of the two processes;
11     Calculate training time of each group by
         F^{i,j}_{oxc,k}, F^i_{pod,k}, B^{i,j}_{oxc1}, B^{i,j}_{oxc2}, B^i_{pod};
12     Obtain t_round of each neighborhood solutions and
         store them in NS;
13 end
14 Return A, B, t_round and corresponding OXC
     topology;
```

C. How to form a ring

Algorithm 3 addresses the formation of a ring topology when this job needs a ring topology. Our objective is to achieve a ring topology that exhibits a higher degree of redundancy with the current topology, thereby significantly reducing the complexity of the topology and enhancing the compatibility between OXC topology and traffic matrix.In *Line* 1 - 2, we obtain the existing connections between node pairs within the root_node and calculate the degrees of connectivity for all nodes in this topology. To connect these nodes into a ring, it is essential to ensure that each node has a degree of exactly 2

Algorithm 2: OXC topology algorithm of each group

Predict the traffic size of each job and Sorting them in descending order in the set G.;
Utilizing all GPUs within the network and allocating GPU resources based on the average training duration.;
Initial tor_traffic,oxc_traffic = $[0]_{1\times|P|}$;
for *each job k in set G* **do**
  if *job k is PS type* **then**
    ps_node = index of minimum oxc_traffic;
    update oxc_traffic;
  end
  root_node = $\emptyset$;
  for *each parallelism unit of job k* **do**
    put $argmin_p \max\{\text{oxc\_traffic}_p, \frac{\text{tor\_traffic}_p}{\text{exc\_rate}}\}$ into root_node;
    if *The unit cannot be fully placed in root pod* **then**
      Identify the next pod with the most abundant bandwidth and put remaining gpu until the unit is fully deployed;
    end
  end
  if *job k is RAR type* **then**
    Utilizing *Algorithm* 3 to form a ring connection among the root nodes;
  end
end
Obtain traffic size $F_{i,j}$ of this group;
Determine the OXC topology $L_{i,j}$ by $F_{i,j}$;
Return OXC topology;

Algorithm 3: Ring algorithm

Initial Degree = $[0]_{0\times|P|}$;
Calculate Degree and link boolean value $L_{sub}^{i,j}$ in the sub-topology comprising only the root nodes.;
for *each node n in root_node* **do**
  while *Degree_n ¿ 2* **do**
    Find the neighboring node n' with the highest degree.;
    $L_{sub}^{n,n'} = L_{sub}^{n',n} = 0$;
  end
end
Identify all disjoint sub-topologies and severing the sub-rings;
Connect all sub-topologies sequentially and obtain a ring;
Return ring topology;

without forming any sub-rings. *Line* 3 - 8 regulate the degrees of all nodes to be less than or equal to 2. If any node possesses a degree greater than 2, we identify the neighborhood with the highest degree and remove the connections between this node and this neighborhood in the topology, continuing this process until all nodes's degree in the root_node are not greater than 2.In *Line* 9, we identify all mutually disjoint sub-topologies and sever any sub-rings present. Finally, in *Line* 10, we sequentially connect these sub-topologies to obtain a unified ring topology, which we consider to have the highest degree of redundancy with the current topology.

IV. Performance evaluations

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, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "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, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba HPN: A data center network for large language model training," in *Prof.of SIGCOMM 2024*, Aug. 2024, pp. 691—706.

[5] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using megatron-LM," in *Prof.of SC 2021*, Nov. 2021, pp. 1–15.

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

[7] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. of OSDI 2014*, Oct. 2014, pp. 583–598.

[8] N. Zilberman, "In-network computing," Apr. 2019. [Online]. Available: https://www.sigarch.org/in-network-computing-draft/.

[9] Z. Chen, X. Liu, M. Li, Y. Hu, H. Mei, H. Xing, H. Wang, W. Shi, S. Liu, and Y. Xu, "Rina: Enhancing ring-allreduce with in-network aggregation in distributed model training," in *Prof. of ICNP 2024*, Jul. 2024.

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

[11] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, "A survey on mixture of experts," *ACM Comput. Surv*, Oct. 2024.

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