

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 (EP), 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 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.

II. PROBLEM DESCRIPTION

A. Network model

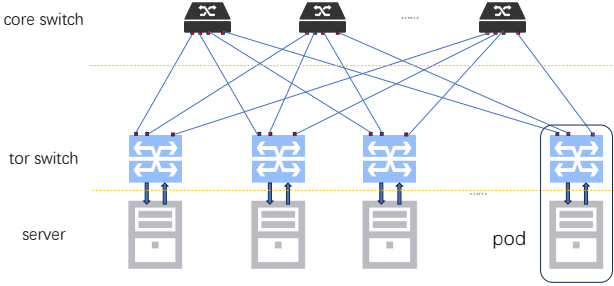


Fig. 1: Traditional data center network

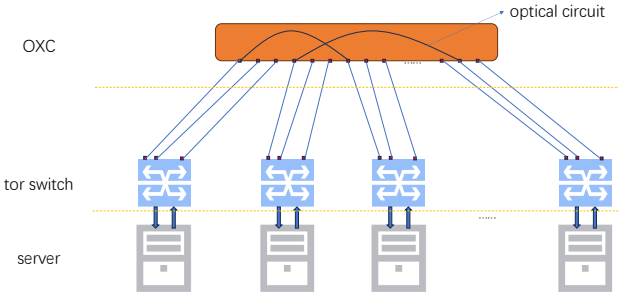


Fig. 2: Optical data center network

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

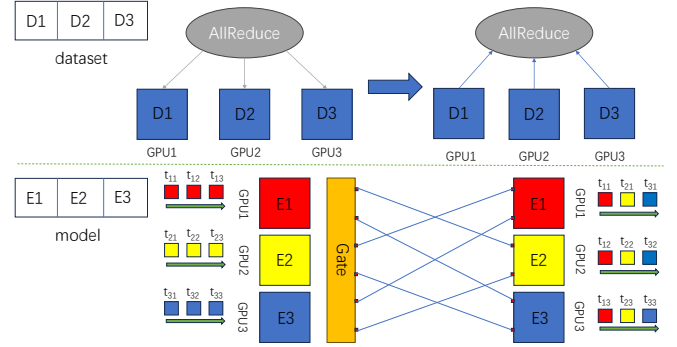


Fig. 3: Training process of DP and EP

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.

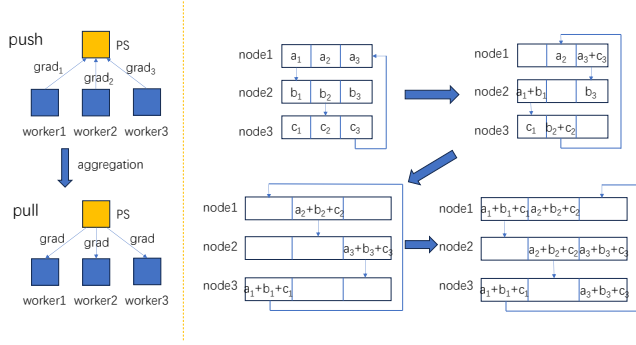


Fig. 4: PS architecture and RAR architecture

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 and double binary tree algorithm [10] 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 [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 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

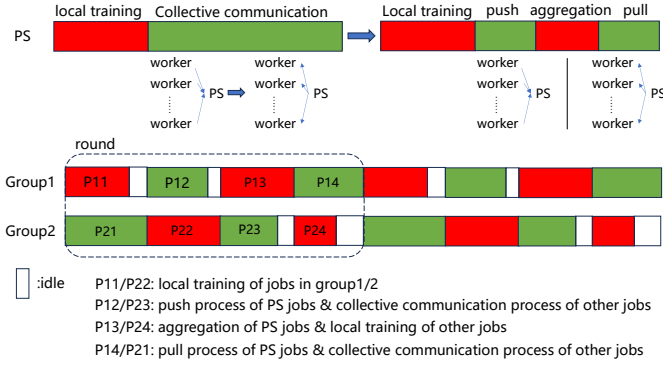


Fig. 5

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_{train}^n	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_i^n / AN_i^n	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_i^n	The boolean parameter that equals 1 if there is a master node of RAR job n in pod i and 0 otherwise.
$FI_{i,j}^n$	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.

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

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 $k, t \in \{0, 1\}$.
$D_{i,j}^{k,t} / D_i^{k,t}$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand of pod i in each stage.
$P_{i,j}^{n,t} / P_i^{n,t}$	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.
$R_{i,j}^n / R_i^n$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand in pod i of RAR job n.
$A_{i,j}^n / A_i^n$	The intra-pod traffic demand from pod i to pod j/The inter-pod traffic demand in pod i of AllToAll job n.
$L_{i,j}^{k,t}$	The amount of OXC links between pod i and pod j in each stage.
$T_{i,j}^{k,t}$	The completion time of OXC traffic between pod i and pod j in each stage.
$t_{comm}^{k,t}$	The communication time of each stage.
$Ring_{i,j}^n$	The boolean variable that equals 1 if a OXC link is desirable from pod i to pod j and 0 otherwise.

$$\begin{cases} D_{i,j}^{k,t} = \sum_{p \in G_{k,ps}} P_{i,j}^{p,t} + \sum_{r \in G_{k,RAR}} R_{i,j}^r + \sum_{a \in G_{k,alltoall}} A_{i,j}^a \\ D_i^{k,t} = \sum_{p \in G_{k,ps}} P_i^{p,t} + \sum_{r \in G_{k,RAR}} R_i^r + \sum_{a \in G_{k,alltoall}} A_i^a \\ D_{i,j}^{k,t} = B * L_{i,j}^{k,t} * T_{i,j}^{k,t} \\ t_{comm}^{k,t} = \max\{T_{i,j}^{k,t}, \frac{D_{i,j}^{k,t}}{B_{tor}}\} \end{cases}, \forall i, j \in V, i \neq j, \forall k, t \in \{0, 1\} \quad (2)$$

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.

$$t_{k,t} = \max\{t_{comm}^{k,t}, t_{train}^n\}, \forall n \in J/G_k, \forall t, k \in \{0, 1\} \quad (3)$$

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

$$\begin{cases}
P_{i,I_p}^{n,0} = P_{I_p,i}^{n,1} = PN_i^n * F_n * U_n + FI_{i,I_p}^n, \forall n \in G_{1,ps} \cup G_{2,ps} \\
P_{i,m}^{n,0} = P_{m,i}^{n,1} = FI_{i,m}^n, \forall m \in V, m \neq i, I_p \\
R_{i,j}^n = Ring_{i,j}^n * F_n * U_n + FI_{i,j}^n, \forall n \in G_{1,RAR} \cup G_{2,RAR} \\
A_{i,j}^n = AN_i^n * AN_j^n * F_n * U_n + FI_{i,j}^n, \forall n \in G_{1,alltoall} \cup G_{2,alltoall}
\end{cases}
\quad (4)$$

$$\begin{cases}
P_i^{n,t} = G_{i,n} * F_n, \forall n \in G_{k,ps} \\
R_i^n = 2 * G_{i,n} * F_n, \forall n \in G_{k,RAR} \\
A_i^n = 2 * G_{i,n} * F_n * (U_n - 1), \forall n \in G_{k,alltoall}
\end{cases}
\quad (5)$$

$, \forall i \in V, \forall t \in \{0, 1\}$

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

$$\sum_{i=1}^{|V|} (L_{i,j}^{k,t} + L_{j,i}^{k,t}) \leq N_link \quad (6)$$

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

$$\begin{cases}
\sum_{i=1}^{|V|} Ring_{i,j}^n = \sum_{i=1}^{|V|} Ring_{j,i}^n = RM_i^n \\
\sum_{i=1, i \neq m}^{|V|} Ring_{i,j}^n = \sum_{i=1, i \neq m}^{|V|} Ring_{j,i}^n = 0
\end{cases} \quad (7)$$

$\forall n \in G_{1,RAR} \cup G_{2,RAR}, \forall m \in RN_n, \forall i, j \in |V|, i \neq j$

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 = ∅;
2 Calculate corresponding OXC topology  $L_{i,j}^{k,t}$  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  $L_{i,j}^{k,t}$  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.

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

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  $L_{i,j}^{k,t}, \max\{T_{tor}, T_{OXC}\}$ 
```

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 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.

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 for each node  $n$  in root_node do
4   while  $Degree\_n \geq 2$  do
5     Find the neighboring node  $n'$  with the highest
     degree.;
6      $L_{sub}^{n,n'} = L_{sub}^{n',n} = 0$ ;
7   end
8 end
9 Identify all disjoint sub-topologies and severing the
  sub-rings;
10 Connect all sub-topologies sequentially and obtain a
  ring;
11 Return ring topology;
```

- [2] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeier, "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.

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>.