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 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

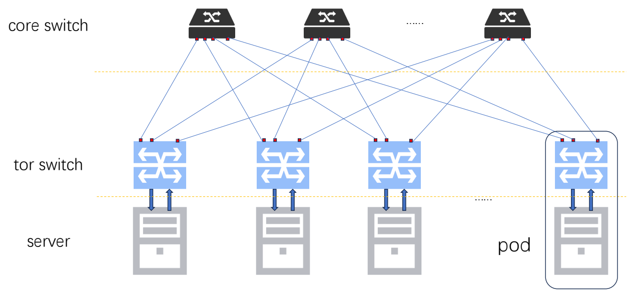

Fig. 1: Traditional data center network

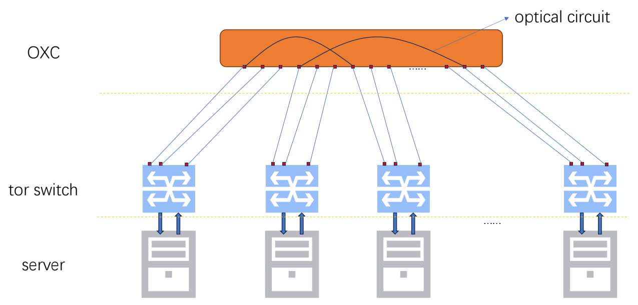

Fig. 2: Optical data center network

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

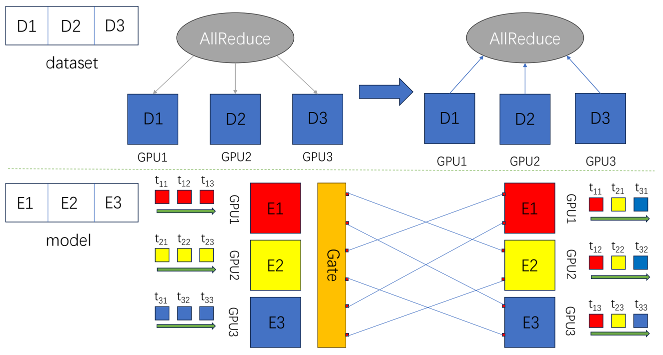
D1
D2
D3
dataset
AllReduce
D1
D2
D3
GPU1
GPU2
GPU3
AllReduce
D1
D2
D3
GPU1
GPU2
GPU3
E1
E2
E3
model
t_{11} t_{12} t_{13}
t_{21} t_{22} t_{23}
t_{31} t_{32} t_{33}
GPU1
GPU2
GPU3
E1
E2
E3
Gate
E1
E2
E3
GPU1
GPU2
GPU3
t_{11} t_{21} t_{31}
t_{12} t_{22} t_{32}
t_{13} t_{23} t_{33}

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.

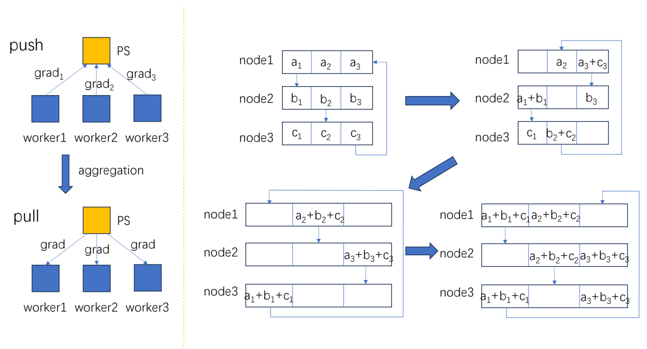

Fig. 4: PS architecture and RAR architecture

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 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 phase, 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, we decompose the scheduling problem into two subproblems: 1. How to group the jobs; 2. How to calculate the OXC topology for the four phases.

D. *ILP*

To rigorously and quantitatively describe the previously presented problem, 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 can be expressed in a linear form. Due to space constraints, we omit the detailed linearization process.

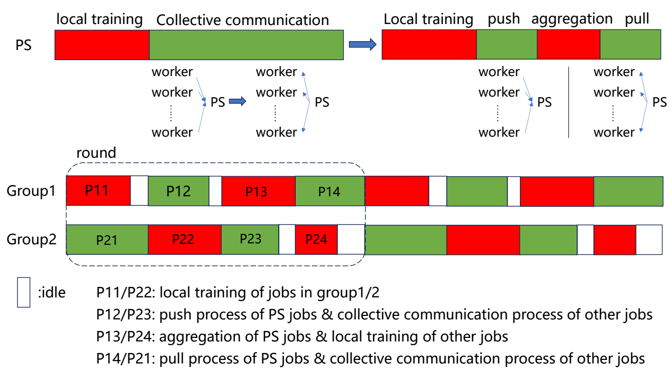
local training
Collective communication
Local training
push
aggregation
pull
PS
worker
PS
round
Group1
P11
P12
P13
P14
Group2
P21
P22
P23
P24
:idle
P11/P22: local training of jobs in group1/2
P12/P23: push process of PS jobs & collective communication process of other jobs
P13/P24: aggregation of PS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

Fig. 5

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

$$\begin{cases} t^{t}_{k,comm} = \max\{\frac{D^{k,t}_{i,j}}{B * L^{k}_{i,j}}, \frac{D^{k,t}_{i}}{B_{tor}}\} \\ D^{k,t}_{i,j} = \sum_{p\in group_{k,ps}} P^{p,t}_{i,j} + \sum_{r\in group_{k,ring}} R^{r}_{i,j} + \sum_{a\in group_{k,alltoall}} A^{a}_{i,j} \\ D^{k,t}_{i} = \sum_{p\in group_{k,ps}} P^{p,t}_{i} + \sum_{r\in group_{k,ring}} R^{r}_{i} + \sum_{a\in group_{k,alltoall}} A^{a}_{i} \end{cases}$$
$$, \forall i,j \in V, i \neq j, \forall k \in \{1,2\} \tag{2}$$

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

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

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

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

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}.

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

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 = $\emptyset$, NS = $\emptyset$;
3  while $\max\{NS\} < t_{round}$ or NS = $\emptyset$ do
4      if NS $\neq \emptyset$ 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;
```

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.

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 2: OXC topology algorithm of each group

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

Algorithm 3: Ring algorithm

```
1  Initial Degree = $[0]_{0\times|P|}$;
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 ¿ 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;
```

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.