TaLB: Tensor-aware Load Balancing for Distributed DNN Training Acceleration

Jinbin Hu[1], Yi He[1], Wangqing Luo[1], Jiawei Huang[2], Jianxin Wang[2], Jin Wang[1,3]
[1]Changsha University of Science and Technology, [2]Central South University
[3]Hunan University of Science and Technology

***Abstract*—Increasingly large-scale models and rich data sets make communication overhead a key bottleneck for distributed Deep Neural Network (DNN) training, constantly attracting the attention of academia and industry. Despite continuous efforts, prior solutions such as pipelining computation/communication and in-network gradient compression/scheduling do not focus on how to accelerate DNN training through load balancing in datacenter networks (DCNs). However, the existing load balancing mechanisms are unaware of tensor integrity and priority for gradient parameter synchronization during the DNN training iterations, resulting in severe tensor tail latency and slow model convergence speed. In this paper, we present a Tensor-aware Load Balancing (TaLB) scheme to accelerate DNN training. Specifically, TaLB identifies the different priority tensors and makes (re)routing decisions based on the tensor-level granularity to cut the high-priority tensors tail delay. The testbed implementation and large-scale NS-3 simulation results show that TaLB effectively accelerates DNN training speed. For example, TaLB significantly reduces the average flow completion time (FCT) by up to 55%, and accelerates the model training speed up to 2.37× on VGG19, ResNet50 and AlexNet models.**

***Index Terms*—Deep Neural Networks, Load Balancing, Multi-path, Distributed**

I. Introduction

In recent years, deep learning (DL) has played a key role in modern artificial intelligence applications such as computer vision and natural language processing [1], [2]. However, due to the increasing model complexity and large amount of data communication processes, communication time currently accounts for the majority of model training time in distributed training. The long communication time makes model training increasingly time-consuming [3]–[6]. How to effectively accelerate distributed deep neural networks (DNN) training becomes a crucial and important issue in production DCNs.

To alleviate the communication bottleneck problem in DNN training, a large number of works have been proposed recently. For example, some works reduce communication overhead by careful layer scheduling, such as ByteScheduler [7], TicTac [8], and Poseidon [9], which use tensor partitioning and priority scheduling techniques to explicitly schedule the order of parameter transmission from workers, in order to better parallelize communication and computation tasks. Other works reduce communication overhead by reducing the amount of data transmitted during communication, with the most commonly used methods of gradient quantization [10] and gradient sparsification [11].

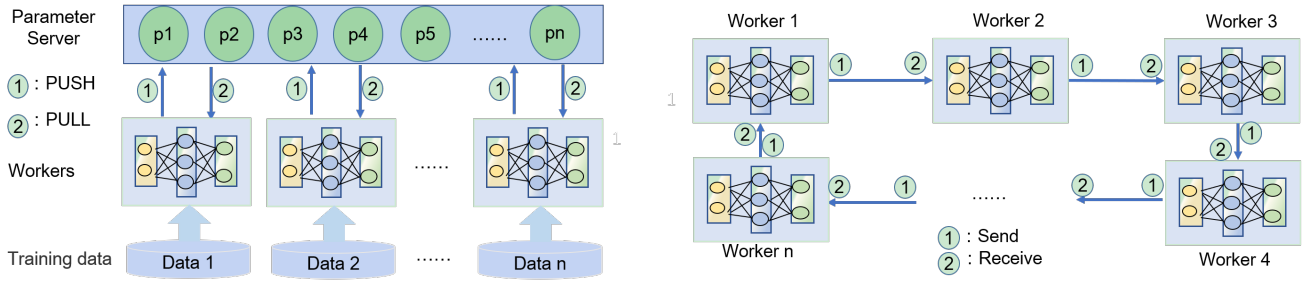
While these solutions are optimized specifically for DNN training, they run at the application layer and can cause large tail delays or even inefficient communication scheduling under heavy DNN training traffic load. Even Geryon [12], the recently proposed network-side scheduling scheme, divides only two priority parameter flows, but does not combine the priority of parameter flow with actual link load in-network. Furthermore, all of these solutions ignore an important and fundamental aspect of load balancing to reduce communication time. An effective load balancing mechanism can coordinate the transmission of parameter flows of different priorities on multiple parallel paths to avoid heavily congested links, reduce the tail delay of parameter flows, and ultimately reduce the communication time.

Unfortunately, existing load balancing mechanisms deployed in DCNs, such as CONGA [13], Hermes [14], LetFlow [15] and DRILL [16], perform poorly in the DNN training scenario. Their designs are based on meeting the needs of long and short flows or various granularity divisions in order to maximize link utilization, while ignoring the characteristic of tensor priority and integrity in DNN training. As a result, these mechanisms may lead to large tail delays for parameter flow. Specifically, the existing load balancing mechanisms cannot sense the priority and integrity of tensor packets in DNN training scenarios. Consequently, when network is under heavy load, high-priority tensor packets may be (re)routed to a highly congested link and suffer from significant tail delays. Even if the back tensor packet reaches the receiver first, the next round of forward propagation (FP) [17] must wait for the tensor packets of previous layer to begin. Additionally, because the current load balancing mechanism unaware of the tensor integrity, it also results in tailing problems of the packets within tensor when gradient aggregation, which severely reduces the convergence speed of DNN training.

Therefore, we propose a new load balancing scheme, TaLB, to accelerate the distributed DNNs training. The key idea behind TaLB is to add a unique TaLB header (hierarchical and size information) to each tensor packet in the network, distinguishing tensor packets of different hierarchical levels. By monitoring the average queueing delay and link utilization of each path and normalizing weighted calculation of port congestion metric (PCM) values, TaLB balances the load of each tensor packet in the network using the hierarchical independence of tensor packets without causing any reordering

Corresponding Author: Jin Wang

979-8-3503-5012-8/24/$31.00 ©2024 IEEE



(a) Parameter server (b) All-reduce

Fig. 1: Communication architectures

issues within the tensor packets. TaLB makes load balancing decisions at the tensor granularity, prioritizing routing high-priority tensor packets to the light load links to maximize the efficient transmission for tensors.

In summary, our major contributions are:

- We conduct an extensive simulation-based study to analyze the problems of existing load balancing mechanisms in DNN training scenario, including the unawareness of tensor priority and integrity, resulting in large tail delay of high-priority tensor and the internal packets.
- We propose a tensor-aware load balancing mechanism called TaLB, which matches the tensors priority with link congestion, and preferentially routes high-priority tensor packets to light traffic links to solve the high-priority tensor packets tailing problem and improve link utilization.
- We demonstrate that TaLB performs significantly better than the state-of-the-art load balancing mechanisms. Especially under heavy traffic load, TaLB greatly reduces the average FCT by about 31%-55%, and in terms of model training speed, TaLB achieves up to 2.37× speedup.

The rest of this paper is organized as follows. We analyze the problems of prior load balancing mechanisms in Section II. In Section III, we describe the design overview and design details. We evaluate the performance of TaLB through testbed and large-scale NS-3 simulations in Section IV and Section V, respectively. We discuss the related work in Section VI and summarize this paper in Section VII.

II. Background and Motivation

A. DNN Training

Deep neural network (DNN) training involves minimizing a loss function value that represents the accuracy of the DNN model. The process involves updating the model's parameters through a parallelized stochastic gradient descent (SGD) algorithm, which is typically done over several epochs. The training data is divided into multiple mini-batches, which are distributed across multiple computing nodes or workers. Each worker maintains a copy of the entire model and independently executes forward and backward propagation to update its local parameters.

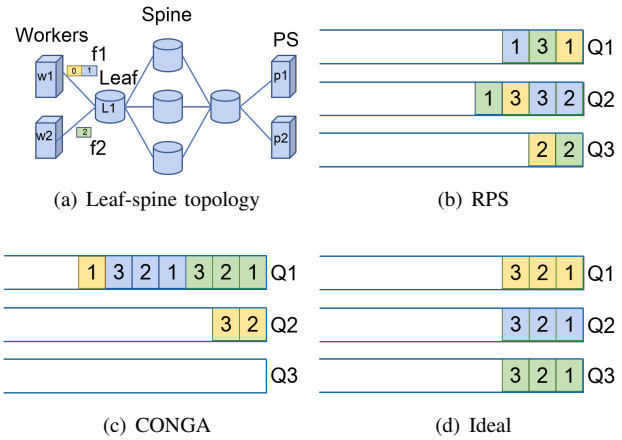
Forward and backward propagation. In large-scale DNN training, training dataset is usually divided into multiple small batches and assigned to multiple workers for processing. Each worker has a complete model and performs forward propagation (FP) on each small batch to generate a loss value. Then, workers perform backward propagation (BP) by backtracking from the output layer to the input layer to calculate the error and gradient, and update the model parameters using stochastic gradient descent. During this process, different workers need to share gradients, which involve network communication issues. Currently, parameter server (PS) and all-reduce are commonly used communication architectures.

Parameter server. In the data parallelism mode, the PS architecture [18] decomposes the neural network into multiple parts, each of which is computed by different workers. As shown in Fig. 1 (a), each worker computes the gradient locally and then uses the PUSH step to transmit it to the PS. Then PS aggregates the gradients from different workers using a specific aggregation strategy and updates the global model parameters using the aggregation result. Then, each worker uses the PULL step to retrieve the updated parameters from the parameter server and continues with the next round of computation. We can approximate the PS parameter synchronization flow pattern as many-to-one model of bandwidth-sensitive flows.

All-reduce. In contrast to PS, all-reduce [19] is a decentralized distributed DNN training architecture. As shown in Fig. 1 (b), in the ring-based All-reduce, each worker receives gradients from its left neighbor worker and sends the collected gradients to its right neighbor worker. Then, they broadcast the gradients around the entire ring to enable all workers to update their parameters. The ring-based All-reduce architecture can alleviate communication bottlenecks and has bandwidth optimality properties. However, All-reduce requires a large amount of communication between workers, resulting in high communication overhead and introducing too many inter-node dependencies, which can easily cause network congestion or failure.

B. Drawbacks of Prior Load Balancing in DNN Training

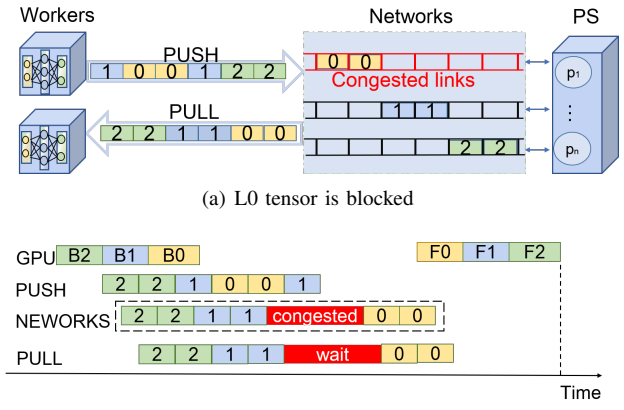
1) Unaware of Tensor Integrity: Existing load balancing designs make routing decisions at different switching granularities. There are three main types: packet, flowlet, and flow granularity. Specifically, packet-level load balancing mechanisms such as RPS [20] simply sprays each packet onto



(a) Leaf-spine topology (b) RPS

(c) CONGA (d) Ideal

Fig. 2: Queueing under different granularities



(a) L0 tensor is blocked

(b) DNN training stalls significantly

Fig. 3: Unawareness of tensor priority in network transmission

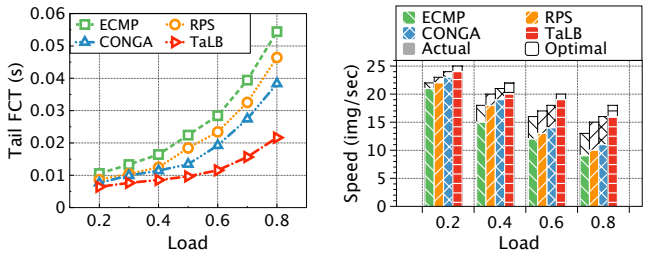
all available shortest paths. ECMP [21] randomly chooses forwarding path in a flow granularity based on the hash value. The switching granularity of CONGA [13] is flowlet, which uses time intervals formed naturally between packets and is randomly sent to other paths to avoid out-of-order problems. However, the above three granularities overlook a characteristic of tensor packets in DNN training: even though tensors may vary in size across different layers, after being sliced into fixed sizes, the tensor packets have the same size in the network. These load balancing mechanisms do not consider the integrity of tensor packets, thus making it challenging to achieve effective load balancing in DNN training.

We use a simple example to illustrate the problem with typical load balancing mechanisms. As shown in Fig. 2 (a), two senders (W1, W2) and two receivers (P1, P2) are connected to their corresponding leaf switches. The output ports on switch $L1$ correspond to three queues (Q1, Q2, Q3), assuming that each tensor size is three packet units after tensor partition. In the subsequent discussion, we denote the tensor of the x-th layer as Lx. W1 and W2 continuously send long flow f1 and short flow f2 to P1 and P2 at time T1, where f1 transfers the L0 and L1 tensor packets, and f2 transfers the L2 tensor packet. We use four different granularities to (re)route at the leaf switch $L1$. Fig. 2 (b) shows that packets are randomly scattered into output port queues in RPS. Although RPS fully utilizes multipath resources, it results in serious out-of-order problems, As shown in Fig. 2 (c), at the flowlet granularity, f1 will only be re-routed when its path is very congested. Due to insufficient inactive intervals, the utilization of other paths is very low, even idle (Q3). Therefore, it can be seen that existing load balancing mechanisms in DNN training cannot achieve optimal load balancing effects because they do not consider the integrity of tensor packets. The ideal load balancing scheme, as shown in Fig. 2 (d), uses the characteristic of the same tensor sizes in DNN to achieve switching tensor-level granularity, which can effectively ensure link utilization and reduce packet disorder, while considering the traffic scenario of DNN to achieve fine-grained priority control.

2) Unaware of Tensor Priority: The existing load balancing mechanisms do not aware of the tensor priority, and they do not combine the tensor priority with the link load to (re)route. This may cause the tensor packets in the previous layer to experience severe link congestion and further delay the calculation of latter layer.

We use the above example to further illustrate the problem due to the unaware tensor priority in existing load balancing mechanisms. As shown in Fig. 3 (a) and Fig. 3 (b), assuming that the model adopts PS architecture synchronization parameters, the tensor packets generated by workers are sent from the host to the network through communication scheduling in a sequence of 2-2-1-0-0-1 for parameter aggregation update. It is assumed that there are three links in the network, one of which is a heavy congested link. Because the existing load balancing mechanisms do not perceive the tensor priority, when the 0-layer (L0) tensor packets are rerouted to the heavy congested link as a trailing parameter flow, a new round of model iteration will be stalled. Even though the tensor packets of the latter layer have been pulled to workers, training still cannot be continued, seriously reducing the training speed of DNN model. Considering that recent deep learning models consist of tens or hundreds of layers, this parameter trailing problem poses a serious challenge for high DNN training performance [22]–[24].

In order to illustrate the influence of in-network tensor unaware on the synchronization process of DNN training, we use PS communication architecture in DNN training to simulate the traffic pattern of VGG19 model synchronization parameters in NS-3. We adopt the leaf-spine structure topology of Fig. 3 (a) with a bottleneck bandwidth of 100Gbps and the default delay of each link is 2 μs. We set 16 workers to cooperate with one PS, and the per-port switch buffer size is 128KB. The size of the tensor packets communicated in each training iteration is 32KB, and the traffic pattern mixed with both front layer tensor flows and back layer tensor flows. We test the impact of load balancing mechanisms with different switching granularities (CONGA, RPS, ECMP) on the traffic in the DNN traffic pattern during one iteration, including the



(a) Tail flow completed time (b) Training speed

Fig. 4: Numeric and simulation comparison

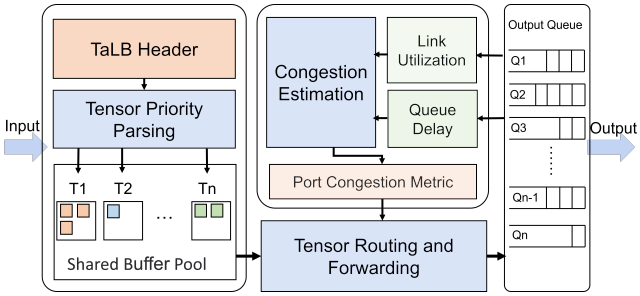


Fig. 5: TaLB overview

trailing tensor flows completion time, the difference between the actual VGG19 model training speed and the optimal value. It's worth noting that we simulated the degree of link load by sending background flows unrelated to training from workers, rather than increasing the training data rate to raise link load.

The experimental results show that the tail delay of parameter flow increases with the increase of link load in the traffic mode of DNN training due to the unaware of tensor packets. As shown in Fig. 4 (a), the tail delay reaches up to 54.4 ms, 46.3 ms and 38.3 ms under 0.8 load for ECMP, RPS and CONGA, respectively.

As shown in Fig. 4 (b), the actual VGG19 model training speed cannot reach the theoretical optimal. In the case of high link load, the model training speed decreases by 26.2% -41.3%, and link congestion will lead to serious parameter tailing problem and prolong the overall training time of DNN training. The fundamental reason for the above problems is that the existing load balancing mechanism is unaware of the tensors priority and integrity, resulting in high priority tensor packets experiencing large tail delay on congested links, seriously decreasing the overall DNN training speed.

Through analysis of existing load balancing and communication scheduling mechanisms, we conclude that: 1) Existing load balancing mechanisms, such as ECMP, RPS and CONGA using flow, packet and flowlet granularity respectively for (re)routing, are unaware of tensor integrity, resulting in low link utilization in DNN training scenarios. 2) Existing load balancing mechanisms are unaware of tensor priority, they do not combine tensor priority with link conditions for rerouting, leading to serious tailing problems for high-priority tensor packets. To solve these problems, we are motivated to explore a new tensor-aware load balancing scheme.

III. TaLB Design

A. TaLB Overview

Our goal is to design a tensor-aware load balancing mechanism, TaLB, to address the blocking of high-priority tensor packets, achieve high link utilization, and faster DNN training speed. Specifically, TaLB partitions the parameter flow into sets of tensor packets to provide finer-grained load balancing and alleviate packet reordering problems. Furthermore, TaLB carefully selects output ports for each arriving tensor packet and (re)routing based on hierarchical priority order for each tensor packet, routing high-priority tensor packets to lightly congested links to prevent them from encountering link congestion. The overview of TaLB is shown in Fig. 5, and TaLB is implemented at switches, consisting of three modules:

- **Tensor Priority Parsing:** When a new packet arrives, the switch parses the packet header (e.g., TCP, IP, and Ethernet) based on the order of arrival. TaLB aims to perceive the priority of tensor packets and send previous layer tensor packets first. To achieve this goal, TaLB requires switches to parse new packet header (TaLB header) and identify unique priority of tensor packets based on new metadata header.
- **Congestion Estimation:** To prevent high-priority tensor packets from being (re)routed to heavy congested links, TaLB needs to establish a congestion awareness mechanism based on switches to real-time perceive link congestion. To achieve this goal, TaLB monitors the port utilization and the average queue delay of each port as indicators of link congestion, and normalizes them to convert them into relative weights as port congestion metric.
- **Tensor Routing and Forwarding:** TaLB prioritizes routing high-priority tensor packets to light congested links by monitoring the parsed and cached tensor packets in the switch to accelerate the transmission of high-priority tensor packets. TaLB also applies port-weighting based on the remaining link utilization to ensure high link utilization.

B. Parsing Tensor Priority

Considering the recent trend of increasing model depth with the emergence of deep learning models such as Transformer, BERT, GPT-3, and others, relying solely on packet priority to determine the priority of multi-layer tensor packets during transmission is not feasible, and it cannot provide fine-grained control over tensors. Therefore, we have chosen a tensor-aware load balancing scheme based on P4 programmable switches. P4 [25] is an advanced programming language designed for programmable switches and data plane processors. We divide the priority-aware module into two parts: the worker sender and the switch processor.

Specifically, as depicted in the Fig. 6, at the worker sender, TaLB header needs to be added to each tensor packet, (a unique identifier, layers, length) , to enable the switch to identify different tensor packets. At the switch processor, the header information needs to be parsed to determine the priority of tensor packets, and then sent to the appropriate output port based on the predefined routing strategy to achieve priority management and load balancing functions. We will provide detailed design details for these two parts.

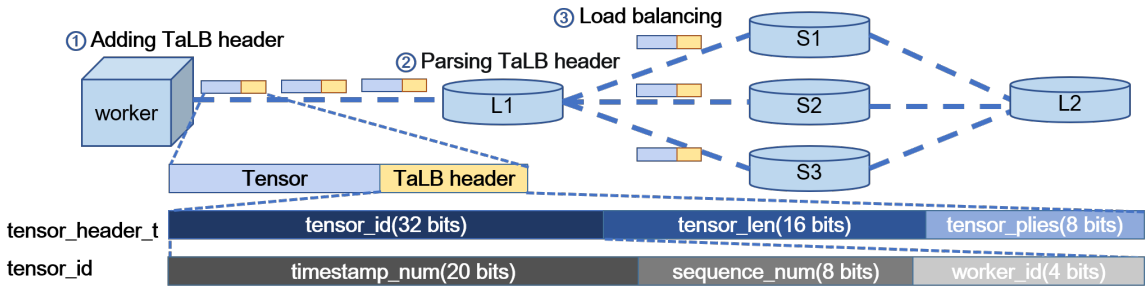


Fig. 6: Parsing TaLB header and sensing tensor priorities

Adding TaLB header in workers. In DNN training, large tensors are divided into multiple small tensors adopting tensor partitioning techniques to improve the parallelism of computation and communication. To identify these small tensors, TaLB adds a P4 metadata header (TaLB header) to each tensor packets on the workers. The TaLB header defines a data structure named `tensor_header_t` that includes three fields: `tensor_id` (32-bits), `tensor_layer` (8-bits), and `tensor_len` (16-bits), used to identify and distinguish different tensor packets. To generate a unique `tensor_id` for each tensor packet, as shown in algorithm 1,TaLB designs an improved distributed Snowflake algorithm, including three parts: `timestamp_num` (20-bits), `sequence_num` (8-bits), and `worker_id` (4-bits). The `timestamp_num` is the current timestamp, `sequence_num` is used to resolve conflicts in `tensor_id` generated at the same timestamp, and `worker_id` is used to resolve ID conflicts between workers in distributed DNN training scenarios. In large-scale DNN training scenarios, the sequence number and node count can be adjusted according to specific requirements.

Parsing TaLB header in switchs. The switch needs to identify and classify the received different tensor packets. Upon receiving a packet, the switch first parses the packet header information to determine if it belongs to a tensor packet. If it does, the switch continues to parse the TaLB header. Based on the `tensor_id`, the switch determines which tensor the packet belongs to and stores it in buffer. It is worth mentioning that the switch utilizes a Weighted Fair Queuing (WFQ) algorithm for buffer management. Tensor packets are assigned different weights based on their priority, and the switch schedules packet transmission based on these weights. Higher priority tensor packets are given more significant shares of buffer resources, ensuring that important packets are processed with minimal delay. TaLB assigns priority to tensor packets based on the size of their layers (for BP, tensor packets of front layers have higher priority than those of back layers), and forwards different tensor packets to the TaLB routing forwarding module for processing and sending to the corresponding port.

C. Calculating Port Congestion Metrics

TaLB faces another challenge in how to perceive link congestion, which involves calculating the quantified value of port congestion. Due to the large number of tensor packets that need to be transmitted in the parameter synchronization communication of DNN training, the burstiness and scale of the traffic are significant, and congestion occurs frequently and obviously [7]–[9]. Therefore, it is necessary to monitor the network congestion situation in real-time and quickly to adapt to the strong burstiness of DNN training traffic, and make timely routing decisions. Based on the above observations of the flow characteristics of DNN training, we use two indicators based on link utilization and average queueing delay to reflect the congestion in the network. The link utilization can reflect the degree of busyness in the network, while the average queueing delay can reflect the time that a packet waits in the queue. Therefore, these two indicators can accurately and quickly reflect the congestion in the network and take timely measures to avoid congestion.

To calculate the port congestion metric (PCM), TaLB uses Min-Max normalization and weighted normalization of the two indicators: link utilization and average queue delay. TaLB makes load balancing decisions based on the tensor priority and PCM values. Specifically, TaLB obtains the present throughput and bandwidth of each port via the switch's port monitoring module, such as "Bits/Second" and "Packets/Second" in the port statistics information of the switch. If a port is in a particular time window T, the number of data packets sending data is N, the size of data packets is M, and the port i's bandwidth is C_i, then the bandwidth utilization rate U_i of the port in this time window can be expressed as follows:

$$U_i = \frac{N \cdot M}{C_i \cdot T}. \tag{1}$$

Additionally, we analyze the average queueing delay per port using the M/G/1-FCFS (first-come first-served) queueing model. $E[\mathrm{W}]$ is the expected average waiting time in each port's queue. For the M/G/1-FCFS queue, we use the well-known Pollaczek-Chintchinb formula to determine the expected average waiting time W as

$$E[w]_i = \frac{\lambda^2 \cdot Var[V] + \rho^2}{2(1-\rho)\lambda} + E[V], \tag{2}$$

where ρ is the load intensity, $E[V]$ is the mean value of the service time $\frac{1}{C}$, and $Var[V]$ is the variance of the processing time for each packet. The value of $Var[V]$ is 0, because the service rate of each output port on the switch is a constant value C. The second equation can be simplified as

$$E[w]_i = \frac{\rho^2}{2(1-\rho)\lambda} + \frac{1}{C}. \tag{3}$$

We define the load intensity ρ as $\rho = \frac{\lambda}{C}$, where the average packet arrival rate is λ and port service rate is C. When $\rho =$

$\frac{\lambda}{C} < 1$, the system can reach a stable state, Equation (3) can be simplified as

$$E[w]_i = \frac{1}{2C^2} \cdot \frac{\lambda}{C - \lambda}. \tag{4}$$

Finally, as shown in Equation (5), we use Min-Max normalization and weighted normalization on the port utilization U_i and average queuing time $E[W_i]$ of each port, and calculate the quantitative value of congestion degree P_i of each port.

$$P_i = \frac{W_1 \cdot \frac{|U_i - \underset{max}{U}|}{\underset{max}{U} - \underset{min}{U}} + W_2 \cdot \frac{|E[w]_i - \underset{max}{E[w]}|}{\underset{max}{E[w]} - \underset{min}{E[w]}}}{W_1 + W_2}. \tag{5}$$

The relative weights of port utilization and average queueing time are denoted by W_1 and W_2, respectively. The weights can be adjusted dynamically based on actual application scenarios. For instance, in the scenario where traffic delay sensitivity is high, the weights of the average queueing delay can be increased to improve the congestion awareness module's perception of queue length. The congestion quantization value p_i of each port is calculated according to Equation (5), which intuitively represents the status of ports or links with varying degrees of congestion. The higher p_i, the greater the likelihood of congestion, the lower p_i, the lesser the likelihood of congestion. Based on the PCM value p_i, TaLB further directs the routing and forwarding of tensor packets.

D. Tensor Routing and Forwarding

After identifying the tensor priority and calculating PCM values of each reachable port, the routing and forwarding modules should accomplish three objectives.1) TaLB should make routing decisions at the tensor-level granularity. 2) TaLB should ensure that the high-priority tensor packets match the lightly congested link in order to prevent the high-priority tensor packets from encountering link congestion, which would impair the DNN training performance. 3) Under the assumption of rapid transmission of high-priority tensor packets, the link utilization is guaranteed as much as possible to improve the actual throughput.

Specifically, for the first target, TaLB maintains a routing table in the switch, as shown in Table I, which records the information of particular tensor packets (the unique identifier, length, received packets, previous port). When a new packet arrives, the switch looks up the corresponding tensor packet information in the Table I according to the unique identifier `tensor_id` of the packet, and determines whether the packet is the last packet of the tensor packet according to the information of received packets. If not, it updates the number of received packets in the table and the port information of the last packet, and forwards the packet to the destination port. If so,the packet is forwarded to the specified port based on the destination port information in the table, and the information of this tensor packet is removed from the table. For the second and third objectives, we provide two specific illustrations

CASE 1: Assuming there are multiple levels of tensors in the shared buffer pool, and L1 has higher priority than L2. As described in Fig. 7, TaLB routes the tensor packets of different levels to different ports with different PCM values, and the high-priority L1 packets are sent to the lowest PCM port (P1). Low-priority tensor packets L2 is routed to the lesser PCM port (P2) to prevent link congestion for L1 tensor packets. Similarly, we discover that, because TaLB routes L2 tensor packets to P2 with a high PCM value to avoid congestion in L1, L2 link congestion overload may be induced, but L1 link utilization is still relatively low. If the PCM value p_i is about to greater than the specified threshold S, (S defaults to the maximum value of PCM among all ports), we select the port with the smallest PCM value (i.e., P1) for the L2 layer (num.4 tensor packet) in order to balance the link load and prevent new congestion hotspots.

TABLE I: TaLB routing table

tensor_id	tensor_len	received packets	previous port
tensor_1	64	53	2
tensor_2	256	153	1
tensor_3	32	30	3
tensor_4	128	109	5

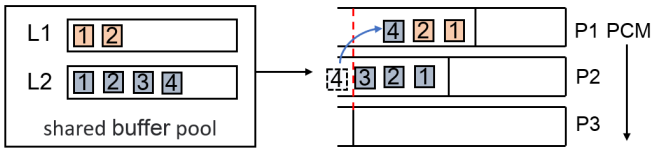


Fig. 7: Multilayer tensor routing and forwarding

CASE 2: When there are only multiple tensor packets of the same layer in the shared buffer pool, as shown in Fig. 8, there are only tensor packets of L1 layer in the buffer pool, TaLB will perform weighted round-robin allocation from high to low according to the PCM residual value (1-p_i) of each reachable receiver port. The higher the PCM residual, the higher the residual link utilization, the lower the delay, and the greater the weight. Then, the switch polls and routes all tensor packets according to the port weight in queuing order to maximize link utilization, balance the load of each link, and avoid congestion hotspot links. By implementing the weighted round-robin allocation strategy based on PCM residual values, TaLB leverages its congestion awareness capabilities to ensure an equitable distribution of tensor packets among the available ports.

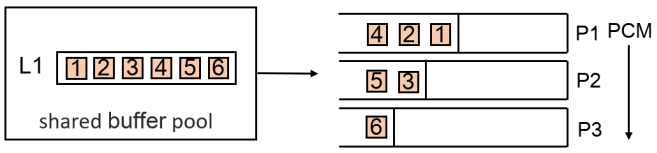
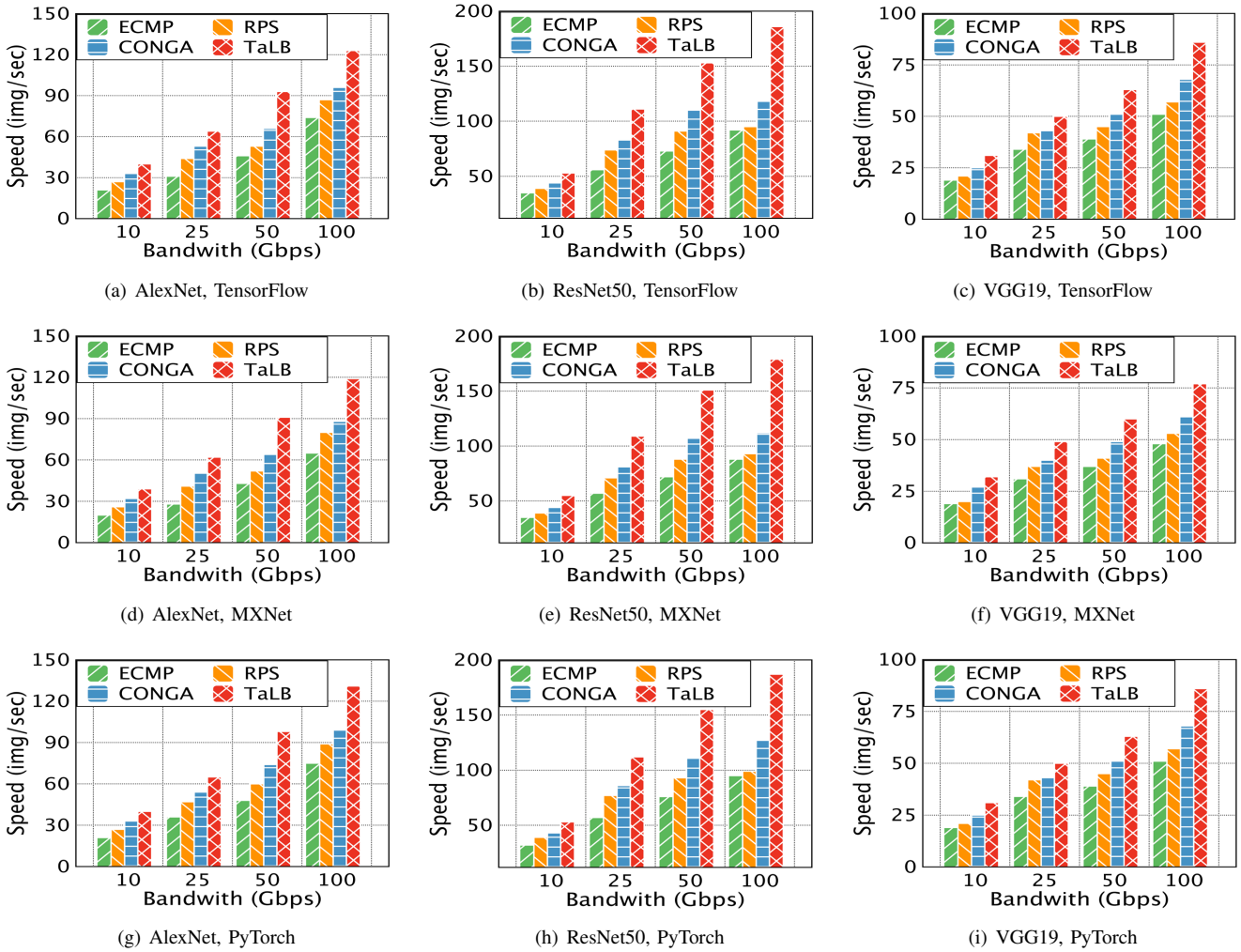


Fig. 8: Single layer tensor routing and forwarding

IV. TESTBED EVALUATION

A. Testbed Setup

- **Topology:** Our test platform consists of 16 nodes, each with 20 CPU cores, 128GB of memory, 16 GeForce RTX 3090, and a Mellanox CX-6 single-port network card. We



(a) AlexNet, TensorFlow (b) ResNet50, TensorFlow (c) VGG19, TensorFlow

(d) AlexNet, MXNet (e) ResNet50, MXNet (f) VGG19, MXNet

(g) AlexNet, PyTorch (h) ResNet50, PyTorch (i) VGG19, PyTorch

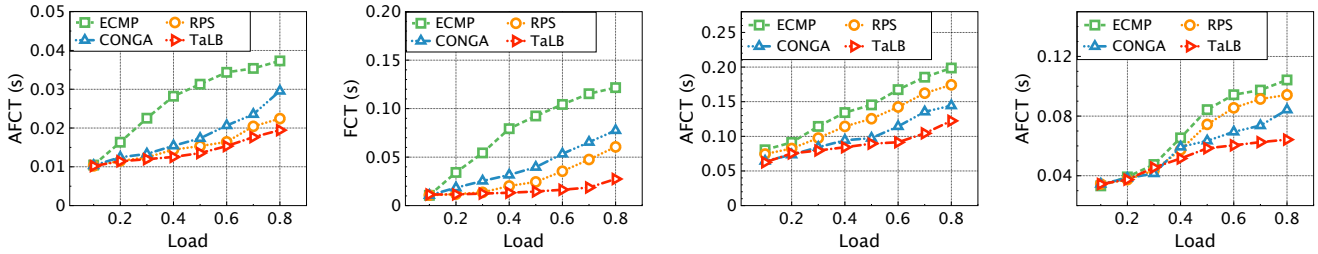
Fig. 9: The training speed of AlexNet,ResNet50,VGG19 models under different bandwidth conditions in PS.

use a leaf-spine topology with 4 racks, each containing 4 nodes. The switches use Mellanox SN2700, the operating system is Ubuntu 20.04, and the Linux kernel version is 5.4.0-74-generic. The link bandwidth is 100Gbps, and the oversubscription ratio is 1:2. The Mellanox driver version is 5.1-0.6.6.0.

- **Models and Workloads:** In the experiment, we use three models and one dataset. The three image classification models include VGG19, ResNet50, and AlexNet, and the training sets use the classic image classification dataset CIFAR-100. We compare network bandwidth from 10Gbps to 100Gbps (10Gbps, 25Gbps, 50Gbps, 100Gbps) and PS parameter server communication schemes implemented using three deep learning frameworks: Tensorflow [22], MXNet [23], and PyTorch [24].
- **Baseline and Metrics:** We use the PS architecture as the parameter synchronization scheme and use ByteScheduler [7] as communication scheduler. ByteScheduler combines priority scheduling and tensor partitioning and uses default Bayesian Optimization (BO) settings for partition sizes and credit sizes. For the mini-batch sizes, we set 64 for AlexNet, 128 for ResNet-50, and 32 for VGG-19. Note that TaLB neither decreases the accuracy of the parameters nor changes model structure, training data sets and mini-batch sizes, it has no impact on the convergence of the model accuracy [12]. We compare TaLB with CONGA [13], RPS [20], and ECMP [21], and use the number of images processed per second as the metric for training speed to show TaLB's performance on improving model training.

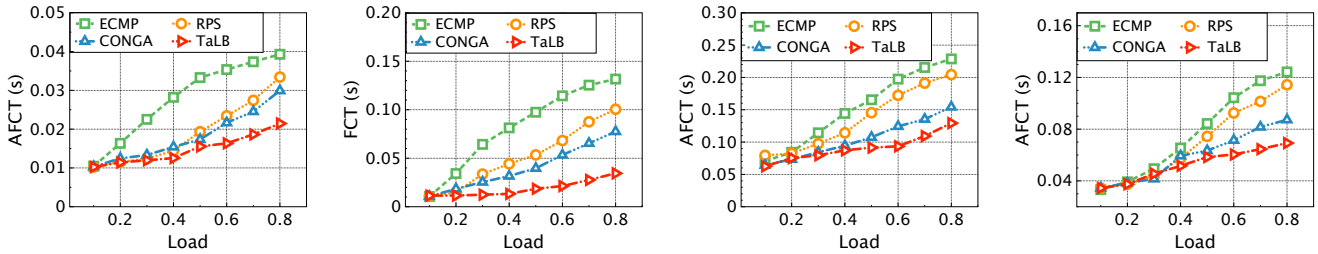
B. Testbed Results

We compare the impact of network bandwidth from 10Gbps to 100Gbps (10Gbps, 25Gbps, 50Gbps, 100Gbps),as shown in Fig. 9, for Fig. 9 (a)- Fig. 9 (c), which show the results based on TensorFlow, we observed that TaLB improved training speed by 106.5%/ 75.4%/ 40.9%, 109.5%/ 95.7%/ 57.6%, and 68.6%/ 50.8%/ 39.6% compare to ECMP/ RPS/ CONGA on AlexNet, ResNet50, and VGG19, respectively. For Fig. 9 (d)-(f), which show the results based on PyTorch, we observe that TaLB improved training speed by 121.4%/ 75%/ 42.1%, 109.7%/ 95.7%/ 57.6% and 68.4%/ 46.3%/ 26.2% compare to ECMP/ RPS/ CONGA on AlexNet, ResNet50, and VGG19, respectively. For Fig. 9 (g)-(i), which show the results based on MXNet, we observe that TaLB improved training speed by 104.1%/ 63.3%/ 32.4%, 96.8%/ 88.8%/ 59.8% and 68.6%/



(a) AFCT of short flows ($\leq$ 100KB) (b) 99^{th}-ile FCT of short flows (c) AFCT of long flows ($\geq$ 1MB) (d) Overall AFCT

Fig. 10: Performance with varying load in symmetric scenario



(a) AFCT of short flows ($\leq$ 100KB) (b) 99^{th}-ile FCT of short flows (c) AFCT of long flows ($\geq$ 1MB) (d) Overall AFCT

Fig. 11: Performance with varying load in asymmetric scenario

50.8%/ 26.4% compare to ECMP/ RPS/ CONGA on AlexNet, ResNet50, and VGG19, respectively.

Based on the above results, we have made the following three observations:

1) TaLB performs the best in various settings. TaLB achieves the best performance in all bandwidth scales and model frameworks. This is because when multiple workers send tensor flows to PS, it leads to incast and periodic congestion on the network. TaLB can maximize the avoidance of congestion hotspots for high-priority tensor packets through a tensor-aware load balancing scheme, which alleviates network congestion and accelerates DNN model training.

2) With the increase of bandwidth, TaLB accelerates model training more significantly. This is because existing load balancing mechanisms such as ECMP and CONGA passively (re)route, resulting in low link utilization and load imbalance. TaLB proactively reroutes based on tensor-level granularity, greatly improving link utilization and accelerating DNN training speed by perceiving the remaining link utilization.

3) TaLB achieves the most acceleration in ResNet50. This is because ResNet50 (50 layers) has more layers than VGG19 (19 layers) and AlexNet (10 layers), which means that the priority based on layers is larger. This gives TaLB more opportunities to make load balancing decisions based on tensor-awareness at the network, thus obtaining greater acceleration and more benefits in multi-layer models.

Furthermore, an in-depth analysis of the results depicted in Fig. 9 reveals an intriguing aspect of TaLB's performance advantage. It becomes evident that TaLB outperforms CONGA, RPS, and ECMP not solely due to the utilization of more advanced load balancing algorithms, but rather because of its enhanced capabilities in congestion identification and tensor priority awareness.

While other load balancing mechanisms may employ sophisticated algorithms to distribute traffic across network paths, TaLB focuses on two crucial aspects that significantly contribute to its superior performance. First, TaLB excels in accurately identifying congestion within the network. By monitoring various metrics such as link utilization, queue delay time, and buffer occupancy, TaLB can promptly detect congested paths and dynamically adjust its routing decisions accordingly. This congestion identification capability enables TaLB to proactively avoid congested paths, thus effectively mitigating performance degradation and reducing the overall packet delay.

Second, TaLB demonstrates a deep understanding of the importance and prioritization of tensor packets. As opposed to treating all packets equally, TaLB possesses the intelligence to differentiate between various types of tensor packets based on their priority levels. This tensor priority awareness enables TaLB to assign higher priority to critical tensor packets, such as those associated with front layers in BP algorithms, and allocate network resources accordingly. By prioritizing the transmission of important tensor packets, TaLB ensures timely delivery and minimizes potential delays in critical computational tasks.

V. Large-scale Performance Evaluation

A. Simulation Setup

- **Topology:** We set up a leaf-spine topology with 100Gbps links, 12 spine switches and 24 Leaf switches, and 48 hosts under each leaf switch and the default delay of each link is 2 μs. The switch queue buffer size is 512KB per port. The transport layer adopts DCTCP as the transport protocol, referring to the default parameter settings in the recommendation in [26]. We compare TaLB with three typical

existing load balancing mechanisms, i.e., CONGA, RPS, ECMP, because they cover all load balancing granularities, i.e., flowlet, packet, flow and tensor.

- **Simulation Model:** For simplicity, we assume that DNN model has only three layers, which are divided into L0, L1 and L2, and the tensor partition size of each layer is same. The ratio of tensor packet size of L0:L1:L2 is 1:2:3, and the L0 tensor packets is transmitted by short flow, while L1 and L2 tensor packets is transmitted by long flow. In order to simulate tensor packets priority scheduling at the transport layer, we assume that the L2 tensor packets and a portion of the L1 tensor packets are transmitted first, followed by the L0 tensor packets, and finally the remaining L1 tensor packets. When a worker receives all the aggregation results of a certain layer, it can directly start the calculation of this layer. The calculation of the latter layer must wait for the completion of the calculation of the former layer and the arrival of the aggregation results.
- **Workloads:** We simulate and train the traffic patterns of VGG19 and ResNet50 under PS architecture, and the ratio of workers to PS is 4 : 1. Since the parameter size of each training iteration communication is the same, so in order to simulate the availability, we simulate the communication of one iteration and obtain the FCT of the parameter flow. We vary the overall workload from 0.2 to 0.8 to fully evaluate the performance of TaLB.
- **Baseline and Metrics:** We use ECMP as the baseline because it is the most widely deployed load balancing mechanism in many production data centers. Flow completion time is an important indicator in our test. For short flows ($\leq$ 100KB), i.e. the L0 tensor flows, we compare the mean value and the 99^{th} percentile FCT; for long flows ($\geq$ 1MB), i.e. the L1 and L2 tensor flows, we compare the throughput of long flows. We compare the average FCT of all parameter flows to comprehensively evaluate TaLB performance.

B. Simulation Results

We test the average FCT (AFCT) and 99^{th} percentile FCT of short flows under symmetric and asymmetric topologies for different load balancing mechanisms. As shown in Fig. 10 (a) and Fig. 10 (b) and Fig. 11 (a) and Fig. 11 (b), we observe that TaLB significantly reduced the AFCT of short flows and 99^{th} percentile FCT of short flows compare to ECMP, RPS, and CONGA. Specifically, TaLB reduces AFCT of short flows by 68%, 55%, and 25% at 0.8 workloads compare to ECMP, RPS, and CONGA, respectively. And 99^{th} percentile FCTS of short flows by 57%, 42% and 19%. TaLB and RPS perform better than CONGA and ECMP for short flows. The reason is that, due to a large number of hash collisions in the network, the front layer tensor packets is easy to encounter link blocking, resulting in a very large FCT of short flows in the ECMP. CONGA performs slightly better than ECMP, but its long control loop does not respond to frequent congestion under heavy congestion loads. TaLB detects congestion in real time, and routes short flows of the front layer parameters reasonably quickly, thus achieving better performance.

We also test the AFCT of long flows and the AFCT of the overall parameter flow under the symmetric and asymmetric topologies for different load balancing mechanisms. As shown in Fig. 10 (c) and Fig. 10 (d) and Fig. 11 (c) and Fig. 11 (d), we observe that TaLB still performs quite well compared to the other three mechanisms. Specifically, TaLB reduces long flows AFCT by 61%, 51% and 29%, and overall parameter flows AFCT by 52%, 34% and 13%, respectively, compared to ECMP, RPS, and CONGA at 0.8 workloads. ECMP reroutes traffic based on flow granularity and does not change the routing of traffic even if it encounters severe congestion. CONGA passively reroutes packets according to the timeout interval between packets clusters. As rerouting is inflexible in CONGA, link utilization is easily reduced, resulting in load imbalance.

Although RPS takes full advantage of multipathing with packets granularity, it results in many reordering packets and large AFCT, and RPS performs worse in asymmetric topology scenarios. TaLB scheme always achieves better performance in both symmetric and asymmetric topologies. The reason is that TaLB, on the premise of ensuring the priority routing of short flows, considers the throughput of long flows and takes the fixed tensor-level granularity to carry out reasonable multipath transmission of long flows, so as to achieve higher link utilization. In addition, TaLB flexibly adjusts routing and forwarding policies based on real-time link loads and is resilient to asymmetric topology scenarios.

These test results prove that TaLB has the advantage of ensuring the transmission of high-priority tensor flows and achieving high link utilization. Under the granularity of flow, flowlet and packet, when the load becomes heavy, more mixed flows are queued on the same output port of the switch. ECMP, RPS and CONGA are unaware of the priority and integrity of tensor, which leads to more high-priority tensor flows suffering from link congestion and tailing problems. TaLB can adaptively make (re)routing decisions according to the tensor priority, and consider the overall link utilization based on the tensor packets routing granularity, so as to obtain sufficient gain. Therefore, TaLB alleviates the effect of the former layer tensor packets being blocked in the heavily congested link case in the DNN training scenario.

VI. Related Works

Optimizing communication in DNN training. Many existing works optimize the communication of distributed DNN training from different perspectives. These methods include but are not limited to 1) adjusting the minibatch size to reduce communication rounds and speedup convergence for DNN training [17], [27]; 2) compressing gradients by gradient quantization, gradient sparsification, and low-rank decomposition to reduce communication overhead [10], [11]; 3) adopting tensor partitioning and priority scheduling technology to achieve a high overlap between calculation and and communication [3], [4], [7], [28]; 4) adopting in-network switch data plane to aggregate model parameters of multiple rack switches to effectively reduce in-network traffic [29], [30]; 5) overlapping communication with computation in model parallelism context [6], [9], [31], [32]; 6) leveraging traditional coflows scheduling to optimize communication by minimizing the FCT of coflows [33], [34]; 7) developing and optimizing network transport

protocols [35], [36]. The above works do not consider the tensor priority combined with the link congestion, note that these works are orthogonal to TaLB.

Load balancing mechanisms. Various load-balancing mechanisms have been proposed in recent years to optimize network performance.There are four main categories based on different granularities: 1) Hermes [14], ECMP [21], MPTCP [37], etc. perform load balancing at the flow or subflow granularity; 2) DRILL [16], RPS [20], $ECN^{\#}$ [39], DeTail [40], etc. perform load balancing at the packet granularity; 3) CONGA [13], LetFlow [15], etc. are typical flowlet-level load balancing mechanisms; 4) TLB [38] and LSS [41] adopt adaptive granularity. TaLB exploits the domain-specific knowledge of distributed DNN training to perform tensor-level load balancing, which greatly accelerates DNN training.

VII. Conclusion

This paper proposes a tensor-aware load balancing scheme called TaLB, which is suitable for distributed DNN training scenarios. TaLB identifies tensor packets with different priorities at the switch and calculates port congestion metrics. Then, it routes high-priority tensor packets to light load links to avoid high-priority tensor packets from link congestion, and speed up the forward propagation of the DNN training. The evaluation results show that TaLB effectively avoids the tailing problem of high-priority tensor packets, and TaLB outperforms the state-of-the-art solutions in terms of flow completion time and practical training speed under DNN training scenarios.

VIII. ACKNOWLEDGEMENT

This work was sponsored in part by National Natural Science Foundation of China under Grant No. 62072056, 62102046, 62132022, and the Natural Science Foundation of Hunan Province under Grant No.2024JJ3017, 2022JJ30618, 2021JJ30867, and the Hunan Provincial Key Research and Development Program under Grant No.2022GK2019, and the Scientific Research Fund of Hunan Provincial Education Department under Grant No.22B0300.

References

[1] C. Zeng, X. Liao, X. Cheng, H. Tian, X. Wan, H. Wang, K. Chen. Accelerating Neural Recommendation Training with Embedding Scheduling. In Proc. USENIX NSDI 2024.

[2] J. Hu, C. Zeng, Z. Wang, J. Zhang, k. Guo, H. Xu, J. Huang, k. chen. Enabling Load Balancing for Lossless Datacenters. In Proc. IEEE ICNP, 2023.

[3] C. Chen, W. Wang, B. Li. Round-robin synchronization: Mitigating communication bottlenecks in parameter servers. In Proc. IEEE INFOCOM, 2019.

[4] J. Xu, S. Huang, L. Song, et al. Live gradient compensation for evading stragglers in distributed learning. In Proc. IEEE INFOCOM, 2021.

[5] L. Luo, J. Nelson, L. Ceze, et al. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In Proc. ACM SOCC, 2018.

[6] A. Jayarajan, J. Wei, G. Gibson, et al. Priority-based parameter propagation for distributed DNN training. In Proc. MLSys, 2019.

[7] Y. Peng, Y. Zhu, Y. Chen, et al. A generic communication scheduler for distributed dnn training acceleration. In Proc. ACM SOSP, 2019.

[8] S. Hashemi, S. AbduJyothi, R. Campbell. Tictac: Accelerating distributed deep learning with communication scheduling. In Proc. MLSys, 2019.

[9] H. Zhang, Z. Zheng, S. Xu, et al. Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters. In Proc. USENIX ATC, 2017.

[10] D. Alistarh, D. Grubic, J. Li, et al. QSGD: Communication-efficient SGD via gradient quantization and encoding. In Proc. Advances in neural information processing systems, 2017.

[11] X. Wan, K. Xu, X. Liao, Y. Jin, K. Chen, X. Jin. Scalable and Efficient Full-Graph GNN Training for Large Graphs. In Proc. ACM SIGMOD, 2023.

[12] S. Wang, D. Li, J. Geng. Geryon: Accelerating distributed cnn training by network-level flow scheduling. In Proc. IEEE INFOCOM, 2020.

[13] M. Alizadeh, T. Edsall, S. Dharmapurikar, et al. CONGA: Distributed congestion-aware load balancing for datacenters. In Proc. ACM SIGCOMM, 2014.

[14] H. Zhang, J. Zhang, W. Bai, et al. Resilient datacenter load balancing in the wild. In Proc. ACM SIGCOMM, 2017.

[15] E. Vanini, R. Pan, M. Alizadeh, et al. Let it flow: Resilient asymmetric load balancing with flowlet switching. In Proc. USENIX NSDI, 2017.

[16] S. Ghorbani, Z. Yang, P. Godfrey, et al. DRILL: Micro load balancing for low-latency data center networks. In Proc. ACM SIGCOMM, 2017.

[17] Y. Li, J. Huang, Z. Li, J. Liu, S. Zhou, W. Jiang, J. Wang. Reducing Staleness and Communication Waiting via Grouping-based Synchronization for Distributed Deep Learning. In Proc. IEEE INFOCOM, 2024.

[18] M. Li, D. Andersen, J. Park, et al. Scaling distributed machine learning with the parameter server. In Proc. USENIX OSDI, 2014.

[19] J. Hu, Y. He, J. Wang, W. Luo, J. Huang. RLB: Reordering-Robust Load Balancing in Lossless Datacenter Networks. In Proc. ACM ICPP, 2023.

[20] A. Dixit, P. Prakash, Y. Hu, et al. On the impact of packet spraying in data center networks. In Proc. IEEE INFOCOM, 2013.

[21] C. Hopps, Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, Internet Engineering Task Force, 2000.

[22] M. Abadi, P. Barham, J. Chen, et al. Tensorflow: a system for large-scale machine learning. In Proc. USENIX OSDI, 2016.

[23] J. Hu, C. Zeng, Z. Wang, J. Zhang, K. Guo, H. Xu, J. Huang, K. Chen. Load Balancing with Multi-level Signals for Lossless Datacenter Networks. IEEE/ACM Transactions on Networking, 2024, DOI: 10.1109/TNET.2024.3366336

[24] A. Paszke, S. Gross, F. Massa, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 2019.

[25] P. Bosshart, D. Daly, G. Gibb, et al. P4: Programming Protocol-independent Packet Processors. In Proc. ACM SIGCOMM Computer Communication Review, 44(3):87-95, 2014.

[26] K. Qian, W. Cheng, T. Zhang, et al. Gentle flow control: avoiding deadlock in lossless networks. In Proc. ACM SIGCOMM, 2019.

[27] Y. Wang, D. Sun, K. Chen, F. Lai, Mosharaf Chowdhury. Egeria: Efficient DNN Training with Knowledge-Guided Layer Freezing. In Proc. ACM EuroSys 2023.

[28] K. Hsieh, A. Harlap, N. Vijaykumar, et al. Gaia: Geo-distributed machine learning approaching LAN speeds. In Proc. USENIX NSDI. 2017.

[29] C. Lao, Y. Le, K. Mahajan, et al. ATP: In-network Aggregation for Multi-tenant Learning. In Proc. USENIX NSDI. 2021.

[30] H. Wang, Y. Qin, C. Lao, Y. Le, W. Wu, K. Chen. Preemptive Switch Memory Usage to Accelerate Training Jobs with Shared In-Network Aggregation. In Proc. IEEE ICNP, 2023.

[31] Y. Huang, Y. Cheng, A. Bapna, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in neural information processing systems, 2019.

[32] D. Narayanan, A. Harlap, A. Phanishayee, et al. PipeDream: Generalized pipeline parallelism for DNN training. In Proc. ACM SOSP, 2019.

[33] W. Bai, L. Chen, K. Chen, et al. Information-agnostic flow scheduling for commodity data centers. In Proc. USENIX NSDI, 2015.

[34] M. Chowdhury, Y. Zhong, I. Stoica. Efficient coflow scheduling with varys. In Proc. ACM SIGCOMM, 2014.

[35] C. Guo, H. Wu, Z. Deng, et al. RDMA over commodity ethernet at scale. In Proc. ACM SIGCOMM, 2016.

[36] Z. Wang, L. Luo, Q. Ning, et al. SRNIC: A scalable architecture for RDMA NICs. In Proc. USENIX NSDI, 2023.

[37] C. Raiciu, S. Barre, C. Pluntke, et al. Improving datacenter performance and robustness with multipath TCP. In Proc. ACM SIGCOMM, 2011.

[38] J. Hu, J. Huang, W. Lv, W. Li, Z. Li, W. Jiang, J. Wang and T. He. Adjusting Switching Granularity of Load Balancing for Heterogeneous Datacenter Traffic. IEEE/ACM Transactions on Networking, 2021, 29(5): 2367-2384.

[39] J. Zhang, W. Bai, K. Chen. Enabling ECN for datacenter networks with RTT variations. In Proc. ACM CoNEXT 2019.

[40] D. Zats, T. Das, P. Mohan, et al. DeTail: Reducing the flow completion time tail in datacenter networks. In Proc. ACM SIGCOMM, 2012.

[41] Y. Fu, D. Li, S. Shen, Y. Zhang, K. Chen. Clustering-preserving network flow sketching. In Proc. IEEE INFOCOM 2020.