Traffic Management for Distributed Machine Learning in RDMA-enabled Data Center Networks

Weihong Yang, Yang Qin*, and Zukai Jiang
Department of Computer Science,
Harbin Institute of Technology (Shenzhen),
Shenzhen, China
*corresponding author: csyqin@hit.edu.cn

Xiaowen Chu
Department of Computer Science,
Hong Kong Baptist University,
Kowloon Tong, Kowloon, Hong Kong, China
chxw@comp.hkbu.edu.hk.

Abstract—**It has become a common practice to train large machine learning (ML) models across a cluster of computing nodes connected by RDMA-enabled networks. However, the communication overhead caused by parameter synchronization deteriorates the performance of such distributed ML (DML), especially in a large-scale setting. This paper tackles this issue by developing a traffic management scheme to support DML traffic, called TMDML (Traffic Management for DML), which needs only a minor modification to the existing RDMA congestion control scheme DCQCN. We assume that there is only one instance of DML workload running in a network. Existing literature has shown that Fat-Tree, a predominant topology in data center, poorly supports DML compared with BCube. With our proposed TMDML, training DML in Fat-Tree can achieve better performance than that in BCube. We first study the impact of multi-bottlenecks on DML via NS-3-based simulations. The results show that DCQCN is inefficient for DML traffic in the multi-bottlenecks scenario. To mitigate the impact of multi-bottlenecks, we propose an optimization model to minimize the maximum flow completion time (FCT) while stabilizing the queues, and then apply the Lyapunov optimization technique to solve the problem. For all the practical purposes, we present two heuristic implementations of TMDML for different deployment requirements. We evaluate the performance of our proposals by simulation, comparing with DCQCN. We use All-Reduce parameter synchronization in Fat-Tree and BCube with traffic trace of modern deep neural network models, including AlexNet, ResNet50, and VGG-16. Our proposals can achieve up to 59% of the time reduction.**

Keywords—distributed machine learning (DML); multi-bottlenecks; RDMA; transport protocol

I. Introduction

The training of machine learning (ML) model is time-consuming due to its extensive data set and complicated model structure. Distributing the training task across a computer cluster can speed up the training. The distributed solution relieves the pressure of computing node; however, the communication traffic becomes a new bottleneck of distributed ML (DML). During each iteration of stochastic gradient descent (SGD) based training, each computing node calculates the gradients locally, then the local gradients are aggregated and updated according to the parameter synchronization scheme. This process of parameter synchronization deteriorates the performance by introducing a mass of communication. Recent works reveal that the communication overhead makes it challenging to achieve linear scale-up while training DML [1]. Therefore, reducing the communication overhead is vital to deploying efficient DML in the distributed environment.

There are several current works proposed to mitigate the communication bottleneck in DML [2]–[12]. R^2SP [2] adopts the round-robin scheme to minimize network contention under parameter server (PS); however, R^2SP cannot be directly used in other parameter synchronization schemes without modification. Some works adopt the idea of overlapping the communication with computation [3], [5], [6]; moreover, MG-WFBP [5] merges gradients from small layers into a large tensor to reduce the communication startup time while ByteScheduler [6] partitions and rearrange the transmission to obtain good performance in scheduling. Overlapping-based solution needs an elaborate control between computation and communication, making the implementation complicated. Another thread of research attempts to compress and/or quantize the gradients to reduce the volume of communication [7]–[10]. E.g., a recent global Top-*k* (*g*Top-*k*) mechanism [8] chooses the global *k* most significant gradients to control the overall communication complexity. These proposals may cause a loss of model accuracy under the same training budget. This paper provides a transport layer solution to reduce the communication overhead. Our proposed solution needs no modification to the existing DML models and algorithms and is orthogonal to the previous solutions.

Remote Direct Memory Access (RDMA) technique has been applied to improve the performance of DML. To support RDMA, the protocols such as RoCEv2 (RDMA over Converged Ethernet version 2) [13] and DCQCN (Data Center Quantized Congestion Notification) [14] are proposed. DCQCN improves throuhput and fairness of RoCEv2 traffic. In DCQCN, switch marks packet with Explicit Congestion Notification (ECN) based on the probability when the egress queue length exceeds a threshold. On the arrival of ECN-marked packet, receiver sends a Congestion Notification Packet (CNP) to sender. Sender adjusts the transmission rate based on whether it has received a CNP or not. However, DCQCN cannot well support DML traffic in multi-bottlenecks scenario. Switch overwhelmed by multiple flows becomes a bottleneck. When a flow travels through several bottlenecks, its transmission rate is significantly reduced. Bulk Synchronous

Parallel (BSP) is one of the commonly used synchronization mdoel where workers start the next iteration only when parameter servers finish updating all the parameters received from workers [15]. In BSP, the training process of DML can be stalled since the server has to wait for the parameters carried by the flow with long FCT. DCQCN tends to suppress the transmission rate of flow with several bottlenecks. Recent work also reports that compared with BCube, Fat-Tree fails to support DML traffic well because of the imbalanced load caused by uncertain hashing result of ECMP (Equal Cost Multi-Path) and PFC (Priority-based Flow Control) pause frames [16]. However, authors do not provide any solution for supporting DML traffic in [16]. There are state-of-the-art works about traffic management in the data center such as pFabric [17], and AuTO [18]. Those traditional TCP/IP-based transport protocols cannot meet the requirement of deploying DML in the distributed large-scale network due to the high CPU overhead. Other traffic control schemes for RDMA-enabled DCNs such as Multipath-RDMA (MP-RDMA) and Improved RoCE NIC (IRN) are implemented based on FPGA (Field-Programmable Gate Array), which increase the hardaware cost and cannot be employed data center easily [19].

In this paper, we develop a traffic management scheme TMDML (Traffic Management for DML) to improve the performance of DML. We assume that there is only one DML training job in a network, and we leave the case where multiple DML workloads are training in a network in the futrue work. The contributions of our work are summarized as follows:

1. We develop TMDML for data center networks in order to better support DML traffic. To the best of our knowledge, this work is the first attempt to improve the performance of DML through traffic management.

2. We study the impact of multi-bottlenecks on DML training. We use NS-3-based simulator to model RoCEv2 NIC and simulate PS and All-Reduce parameter synchronization schemes in Fat-Tree with VGG-16 traffic. We conclude that DCQCN cannot handle DML traffic well in the multi-bottlenecks scenario.

3. We present an optimization model to minimize the worst-case FCT with the constraint of keeping the queue stable and solve it by Lyapunov optimization. For all the practical purposes, we propose a heuristic implementation of TMDML denoted by TMDML-NIC, where the main functionality of TMDML is implemented at NICs and only needs minor modifications to the existing protocol DCQCN.

4. We study the performance of our proposal with All-Reduce scheme in both Fat-Tree and BCube, and simulate the distributed training of three representative DML models: AlexNet, ResNet50, and VGG-16. The simulation results show that TMDML-NIC makes Fat-Tree more RDMA-friendly: training DML in Fat-Tree has less communication time than that in BCube, and TMDML-NIC can reduce up to 59% of the time in Fat-Tree compared with DCQCN.

The rest of the paper is organized as follows. We study the impact of multi-bottlenecks on DML in Section II. Section III presents the design of TMDML. We present the performance evaluation in Section IV. Section V concludes this paper.

II. The Impact of Multi-bottlenecks on DML

Considering the popularity of Fat-Tree topology in commercial data centers, we adopt Fat-Tree in our case study since it has been shown RDMA-inefficient [13], [16]. We consider a multi-bottlenecks scenario in Fig. 1. We simulate PS and All-Reduce in Fat-Tree. In PS, nodes are organized as parameter server node(s) and worker nodes. Workers train the model locally, and send the calculated gradients to the server that is responsible for aggregating the gradients from all workers and then updating the model parameters. The updated parameters will be pulled by workers. Servers can be the bottleneck due to the frequent communications, and All-Reduce tackles this issue by leveraging direct communication between workers. Without central servers, nodes have to maintain partial global parameters. The synchronization in All-Reduce consists of two stages: the scatter and gather. During the scatter stage, all the nodes send the corresponding gradients to each other, and then nodes aggregate the received gradients to update the parameters. In gather stage, each node sends the updated parameters to other nodes. The size of synchronized parameter is 527.8 MB, which is the size of VGG-16, a famous DNN model. The bandwidth of each link is 10Gbps. We simulate the gather stage in PS, i.e., each worker sends its parameter to PS, and the scatter stage in All-Reduce, where each node sends a partial set of parameters to each other. We present the throughput of Server 8 of All-Reduce in Fig. 2.

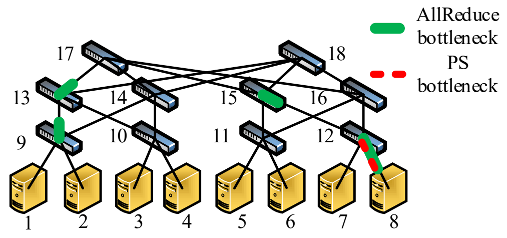
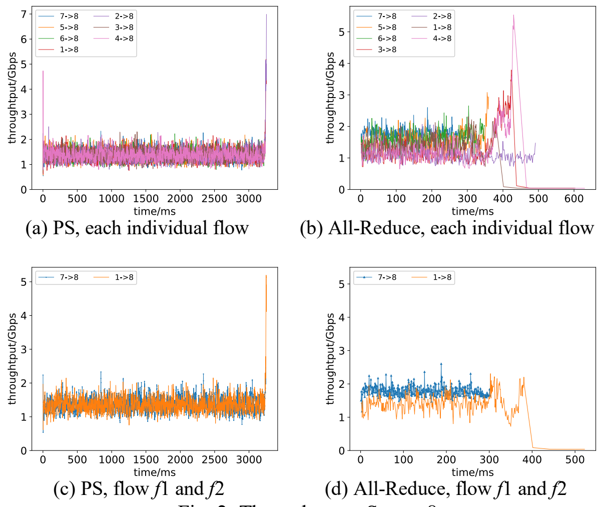


Fig. 1. Fat-Tree (Server 8 is the parameter server under PS).

Server 8 receives seven flows from other servers in both PS and All-Reduce scenarios. The transmission rate of each flow is almost the same in PS (see Fig. 2(a)); however, in All-Reduce, each flow is allocated with different bandwidth (see Fig. 2(b)). It can be observed that the flows from different pods (i.e., Server 1, 2, 3, and 4) are allocated with less bandwidth. We consider two flows, *f*1:Server 1 → Server 8 and *f*2: Server 7 → Server 8. By ECMP, flow *f*1 takes the routing path 1-9-13-17-15-12-8, and *f*2 takes the routing path 7-12-8. In All-Reduce, *f*1 has lower transmission rate than *f*2 (see in Fig. 2(d)) while they have the same bandwidth in PS (see Fig. 2(c)). To explain the rate deviation, we summarize the number of ECN generated by the switches and CNP received by each server in Table I and II, respectively.We also highlight the bottleneck switches of flow *f*1 in PS and All-Reduce in Fig. 2.

DCQCN probabilistically marks the packet with ECN based on the egress queue length. In PS, workers send the local gradients to PS (i.e., Server 8). Then, all the flows are aggregated at Switch 12, and compete for the egress bandwidth, resulting in a substantial buildup of queue. Comparing with other switches, Switch 12 marks packet with ECN at higher probability (see Table I). Although Switch 9, 13, and 15 generate a certain number of ECNs, they have a marginal contribution to the rate reduction: Server 8 generates almost the same number of CNPs for both $f1$ and $f2$ even though it will receive more ECNs for flow $f1$ than that for $f2$ (see Table II). We call Switch 12 the *bottleneck* switch. Both flow $f1$ and $f2$ have only one bottleneck; thus, they have almost the same bandwidth allocation. The results in Table II also reveal that the path length has little impact on the bandwidth allocation in the multi-bottlenecks scenario. In All-Reduce, nodes have to communicate with each other simultaneously, and the competition for the egress bandwidth becomes much more intense. The packets are ECN-marked with higher probability (see Table I), and more switches become bottlenecks. Flow $f1$ encounters more bottlenecks, and it receives about twice as many CNPs as $f2$ from Server 8 (see Table II). Therefore, $f1$ has a higher reduction in its rate. We call the flow with multi-bottlenecks the *slow* flow.



(a) PS, each individual flow (b) All-Reduce, each individual flow

(c) PS, flow $f1$ and $f2$ (d) All-Reduce, flow $f1$ and $f2$

Fig. 2. Throughput at Server 8.

TABLE I. THE NUMBER OF ECN GENERATED BY SWITCHES AT THE ROUTING PATH OF FLOW $f1$.

Switch	9	13	17	15	12
PS	126	230	0	286	118058
All-reduce	42348	49949	0	32726	38091

TABLE II. THE NUMBER OF RECEIVED CNP GENERATED BY SERVER 8 FOR FLOW $f1$.

Server	1	2	3	4	5	6	7
PS	12477	12525	12487	12489	12368	12410	12464
All-reduce	1966	1931	2116	1991	1083	1103	1043

DCQCN is unfair to the flow with multi-bottlenecks. The *slow* flow is allocated with less bandwidth, and its long FCT stalls the process of parameter synchronization. The impact of multi-bottlenecks on PS seems to be limited in our simulation; however, PS can suffer from the multi-bottlenecks in multi-tasks scenarios. One of the keys to improving DML via communication is to make the *slow* flow faster at each iteration. Thus, we propose TMDML in order to reduce the difference in the completion time of flows.

III. DESIGN OF TMDML

We first present an optimization model for traffic management, then propose the heuristic implementation of TMDML.

A. The Optimization Problem

We can make the *slow* flow faster by allocating more bandwidth, i.e., the bandwidth allocated to each flow varies proportionally as its number of bottlenecks. Let $M_i(t)$ be the remaining data size of flow $i \in F$ at time t. Let $r_i(t)$ be the bandwidth allocated to flow i at time t, n_i denotes the number of bottlenecks on the path of flow i, $\Omega(i)$ denotes the destination server of flow i, $\sum_{j\in\Omega(i)} n_j$ denotes the total number of bottlenecks of flow j that arrived at $\Omega(i)$. We use the following function to evaluate the time it takes to accomplish the transmission of flow i:

$$f(r_i(t)) = \frac{n_i}{\sum_{j\in\Omega(i)} n_j} \times \frac{M_i(t)}{r_i(t)}, \tag{1}$$

We assume that the BSP synchronization model is adopted. BSP can guarantee the training accuracy, and it converges faster than SSP and ASP when training DNN model [1]. In BSP, the next iteration cannot be carried out until all the parameters are collected and updated. Therefore, an intuitive idea is to appropriately increase the transmission rate of the *slow* flow to make all the flows arrive at the same time. Our goal is to minimize the worst-case FCT, i.e., the completion time of the *slowest* flow. We formulate the traffic management problem as a min-max robust optimization constrained by link capacity while maintaining stable queues. The optimization problem is presented as following:

$$\begin{aligned} \min \max_{i\in F} \quad & f(r_i(t)) \\ \text{s.t.} \quad & \sum_{i=1}^{F} r_i(t) a_{il} \le C_l, \forall l \in L \\ & \lim_{T\to\infty} \tfrac{1}{T} \sum_{t=0}^{T-1} \mathrm{E}\{|Q_s(t)|\} < \infty, \forall s \in N \\ & r_i(t) \ge 0, \forall i \in F \end{aligned} \tag{2}$$

where C_l is the capacity of link l, $Q_s(t)$ is the queue length of switch s at time t. $a_{il} = 1$ indicates that flow i uses link l, and 0 otherwise. The first constraints are the link capacity constraint. The second constraint guarantees the stable queues. The third constraint is the non-negative constraint of $r_i(t)$.

B. Solving Problem Using Lyapunov optimization

By transforming the min-max robust optimization problem into the minimization problem, we rewrite (2) as follow:

$$\begin{aligned} \min \quad & \phi \\ \text{s.t.} \quad & f(r_i(t)) \le \phi \\ & \sum_{i=1}^{F} r_i(t) a_{il} \le C_l, \forall j \in L \\ & \lim_{T\to\infty} \tfrac{1}{T} \sum_{t=0}^{T-1} \mathrm{E}\{|Q_s(t)|\} < \infty, \forall s \in N \\ & r_i(t) \ge 0, \forall i \in F. \end{aligned} \tag{3}$$

We adopt the Lyapunov optimization to derive the solution since it is a powerful technique to maintain stability in an online manner. We define $Q_s(0) = 0, \forall s \in N$, and update the queue by $Q_s(t+1) = \max[Q_s(t) + \sum_{i\in F(s)} r_i(t) - \mu_s, 0]$, where μ_s is the service rate of switch s, $F(s)$ is the set of flows passing through switch s. The Lyapunov function is defined as $L(\mathbf{Q}(t)) = \frac{1}{2}\sum_{s=1}^{N} Q_s^2(t)$, which represents the congestion level. To keep the queue stable, we introduce $\Delta(\mathbf{Q}(t))$ as the Lyapunov drift: $\Delta(\mathbf{Q}(t)) = \mathrm{E}\{L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t)) \mid \mathbf{Q}(t)\}$. By the Lyapunov optimization, at each time slot, the objective is to minimize the supremum bound on *drift-plus-penalty* $\Delta(\mathbf{Q}(t)) + V\mathbb{E} \quad \mathbf{Q} \quad \}$, where V is the control parameter that affects the balance between penalty optimization and drift minimization. By the similar technique used in [20], [21], we find the upper bound of *drift-plus-penalty*

$$\begin{aligned}\Delta(\mathbf{Q}(t)) + V\mathbb{E} \quad \mathbf{Q} \quad \} &\le \tfrac{1}{2}\sum_{s=1}^{N}(|F(s)|^2 C^2 + \mu_s^2) \\ +V\mathbb{E} \quad \mathbf{Q} \qquad\qquad \mathbb{E} &\qquad\qquad |\mathbf{Q}(t)\}.\end{aligned} \tag{4}$$

Problem (2) is now transformed to the following optimization problem at each time slot t

$$\begin{aligned}\min \ & V\phi - \sum_{s=1}^{N} Q_s(t)\left[\mu_s - \sum_{i\in F(s)} r_i(t)\right] \\ \text{s.t. } & f(r_i(t)) \le \phi \\ & \sum_{i=1}^{F} r_i(t) a_{il} \le C_l, \forall l \in L \\ & r_i(t) \ge 0, \forall i \in F.\end{aligned} \tag{5}$$

By introducing the Lagrangian multiplier ω_1, ω_2 and ω_3 associated with the constraints, we can derive the Lagrangian duality problem of (5). Using the gradient descent method [22], we obtain the following updating rules for $r_i(t)$:

$$r_i^{\tau+1}(t) = \left[r_i^{\tau}(t) - \varepsilon_1 \left(\begin{aligned} \sum_{s\in S} Q_s(t) - \omega_1 \frac{n_i M_i(t)}{(r_i^{\tau}(t))^2 \sum_{j\in\Omega(i)} n_j} \\ + \omega_2 \sum_{i\in F} a_{il} - \omega_3 \end{aligned} \right) \right]^{+} \tag{6}$$

The following algorithm can solve the problem (5).

```
Algorithm 1 The gradient-based algorithm.
1: At the beginning of each time slot $t$, get the current queue backlog $Q_s(t)$ and other information $\mu_s$ at switch;
2: Determine Lagrangian multiplier $\omega_1$, $\omega_2$ and $\omega_3$ to minimize the duality problem;
3: Determine control decision $r_i(t)$, $\phi$ to minimize Lagrangian function;
4: Update the queues $\mathbf{Q}(t+1)$ and the newly determined control decision.
```

C. Heuristic Algorithm

In the dynamic network environment, solving the global optimization problem can introduce high overhead and latency in both centralized and distributed manner. For all practical purposes, we design a heuristic implementations of TMDML (denoted by TMDML-NIC), where implements the main functionality of TMDML at NICs. TMDML-NIC is fully distributed and only need a little modification to the existing protocol. Recall the updating rule of rate in (6), the change of $r_i(t)$ is mainly caused by the first two terms, i.e., $\sum_{s\in S} Q_s(t)$ and $n_i M_i(t) / r_i^2(t) \sum_{j\in\Omega(i)} n_j$. We develop the heuristics for rate control based on these two terms. Third term $\sum_{i\in K} a_{il}$ is the hop count of path that flow i takes, and we have already shown in Section III that the path length has a marginal contribution to the change of rate.

Rate decrement: The reduction in rate $r_i(t)$ is caused by the substantial backlog of queues $\sum_{s\in S} Q_s(t)$. Therefore, the ECN-based congestion control [23], which marks the packet with ECN based on the queue length at the switch, can be used to slow down the flow when congestion occurs.

Rate increment: In $n_i M_i(t) / r_i^2(t) \sum_{j\in\Omega(i)} n_j$, the number of bottlenecks n_i has a significant contribution to the rate increase. As we analyze in the last section, the flow with more bottlenecks is allocated with less bandwidth due to the massive CNPs it introduces. Therefore, an intuitive idea for our heuristic design is to reduce the number of CNP for those flows that have multi-bottlenecks.

The TMDML-NIC algorithm has three components: (i) the sender server, or Reaction Point (RP); (ii) the switch, or Congestion Point (CP); and (iii) the receiver server, or the Notification Point (NP).

CP Algorithm (Switch): The CP algorithm is similar to DCTCP and DCQCN: the switch marks a packet with ECN based on the probability [14], [24].

NP Algorithm (Receiver): The receiver needs to maintain the *progress* of flow. We define the *progress* of a flow as the number of packets received for this flow. The global information of flow progress can help to achieve better performacne; however, maintaining global progress needs the central entity, and accessing such an entity can cause extra overhead. Therefore, in NP algorithm, each receiver maintain its own local progress. The NP algorithm follows the procedure in Fig. 3. Let $pro_j.i$ be the progress of flow i arrives at receiver j, and we assume that packet P belongs to flow i. Different from DCQCN, the CNP mechanism is triggered not only by the arrival of the ECN-marked packet but also when flow i does not fall behind the average progress $pro_j.avg$. In this case, the slow flow will not become slower even if network congestion occurs. The average progress $pro_j.avg$ is updated using a packet counter. The packet counter updates $pro_j.avg$ for every B packets. The selection of parameter B depends on the requirement of the accuracy and overhead. For example, using the large value of B can reduce computing overhead at NIC due to the infrequent update. We will study the impact of parameter B on the accuracy of average progress in our future work.

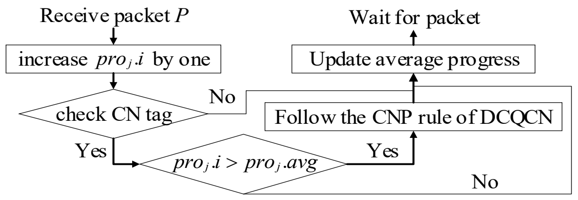


Fig. 3. Procedure of NP algorithm.

RP Algorithm (Sender): As in DCQCN, the sender reduces the current rate upon the arrival of CNP and increases the sending rate using a timer and a byte counter. The detail about rate updating rules can be found in [14].

IV. Performance Evaluation

A. Simulation Setup

We choose two commonly used network topologies in data center networks: Fat-Tree and BCube. For Fat-Tree, we use an small-scale 8-server topology (as shown in Fig. 1) and a large-scale 128-server topology. For BCube, we use a 9-server (see

Fig. 4) and 121-server topology. We use All-Reduce for parameter synchronization because All-Reduce suffers severely from multi-bottlenecks. The bandwidth of each link is 10Gbps. We consider three famous DNN models: AlexNet, ResNet50, and VGG-16. The model details are shown in Table III. We train AlexNet and ResNet50 on a workstation equipped with one GPU (NVIDIA GeForce GTX 1080Ti, GPU memory 11GB) and 32GB memory, while VGG-16 is trained on a server with one GPU (NVIDIA TitanXp, GPU memory 12GB) and 96GB memory. We use CIFAR-10 to train these three models in TensorFlow. The training converges if the loss value less than 0.5 in 10 consecutive iterations. Based on the total training time and number of iterations, we calculate the average time for each iteration and use the average iteration time as the computing time of each node at each iteration. Other traffic control schemes such as MP-RDMA and IRN are implemented based on FPGA, and they cannot be easily employed in DCNs. Since DCQCN is a practical solution to controlling traffic in RDMA-enabled DCNs, we compare our proposal with DCQCN for performance evaluation.

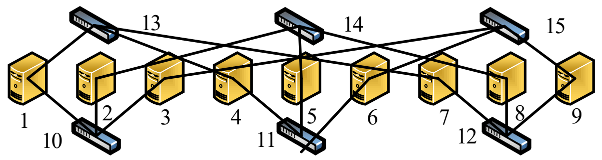


Fig. 4. BCube topology.

TABLE III. PARAMETER SIZE AND ITERATION TIME OF MODELS.

	AlexNet	ResNet50	VGG-16
Parameter size	240 MB	97.7 MB	527.8 MB
Iteration number	3804	9053	4132
Total time	2148 s	7321 s	4678 s
Average iteration time	0.565 s	0.809 s	1.132 s

B. *Simulation Result*

1) Comparison of Heuristic Algorithm and Numerical Results

We study the gap between the heuristic algorithm and the gradient-based solution (Algorithm 1). We solve problem (5) in Matlab by the build-in function `fmincon`. We use the 8-server Fat-Tree topology in this comparison. Fig. 5 presents the total throughput received at each server. It can be observed that both TMDML-NIC and the numerical result have a high utilization at the receivers' link.

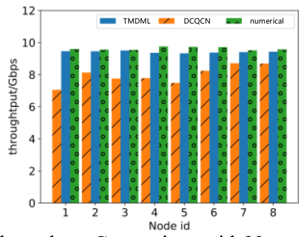
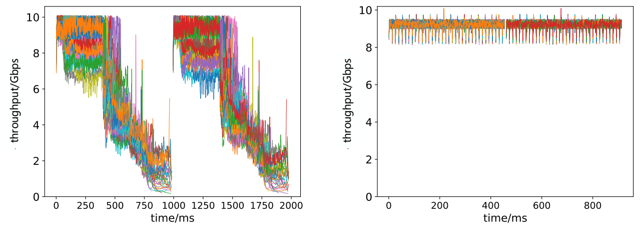


Fig. 5. Throughput Comparison with Numerical Results.

2) Throughtput and Complete Time

In this subsection, we compare TMDML-NIC with DCQCN in terms of total throughput and transmission completion time in both Fat-tree and BCube for three DNN models. Due to the space limit, we here only present the total throughput results of VGG-16 in large scale 128-server Fat-Tree and BCube. Fig. 6 shows the total throughput of each server. We simulate the data transmission in one iteration, consisting of the scatter and gather stage. In DCQCN, the imbalanced distribution of flows is caused by ECMP collisions. Multiple flows arriving at a switch compete with each other for egress bandwidth and suppress each other's transmission. DCQCN slows down the flows that encounter multi-bottlenecks, resulting in a long tail when others flows have completed their transmission (see Fig. 6 (a)). However, the next iteration cannot start until the server receives all the parameters. TMDML allocates the bandwidth to each flow based on the number of bottlenecks; thus it can achieve a balanced completion time among all the flows. Moreover, the fluctuations of the total throughput are limited within 8-10 Gbps in TMDML, as shown in Fig. 6 (b). We compare the transmission completion time between DCQCN and TMDML with both small-scale and large-scale topology in Table IV. As shown in Table IV, TMDML can achieve a larger reduction of FCT when the number of servers increases, which means TMDML has better performance in a larger-scale network.



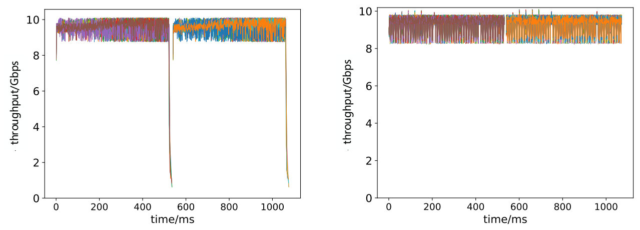
(a) VGG-16, DCQCN, 128-Fat-Tree (b) VGG-16, TMDML, 128-Fat-Tree

Fig. 6. Total throughput of each server in Fat-Tree.

TABLE IV. TRANSMISSION COMPLETION TIME OF DCQCN AND TMDML IN FAT-TREE.

Model	Topology	DCQCN	TMDML	Time reduction
ResNet50	8-Fat-Tree	206.04 ms	139.89 ms	32.10 %
	128-Fat-Tree	392.72 ms	160.32 ms	59.18 %
AlexNet	8-Fat-Tree	592.57 ms	349.69 ms	40.99 %
	128-Fat-Tree	934.16 ms	394.48 ms	57.77 %
VGG-16	8-Fat-Tree	1236.63 ms	769.01 ms	37.81 %
	128-Fat-Tree	1972.64 ms	907.92 ms	53.97 %

Fig. 7 shows the total throughput of each server at one iteration while training VGG-16 in BCube. BCube can better fit the DML traffic as compared with Fat-Tree, which is consistent with the conclusion in [16]. In Fig. 7 (a), although BCube can achieve better load balance, there still exist some flows that have longer FCT. Due to the balanced completion time, TMDML can achieve up to 24% reduction of transmission completion time, as shown in Table V. Moreover, comparing Table V with IV, Fat-Tree has less transmission completion time, which means that TMDML can accelerate the training process of DML in Fat-Tree.



(a) VGG-16, DCQCN, 121-BCube (b) VGG-16, TMDML, 121-BCube

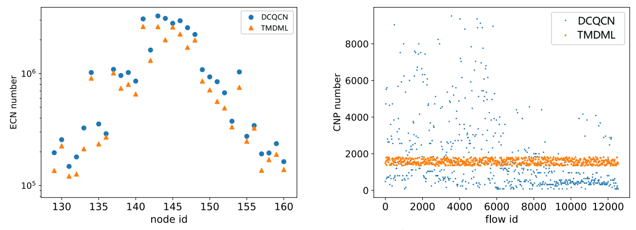
Fig. 7. Total throughput of each server in BCube.

TABLE V. TRANSMISSION COMPLETION TIME OF DCQCN AND TMDML IN BCUBE.

Model	Topology	DCQCN	TMDML	Time reduction
ResNet50	9-BCube	187.42 ms	162.68 ms	13.20 %
	121- BCube	197.28 ms	182.56 ms	7.46 %
AlexNet	9- BCube	467.43 ms	353.82 ms	24.30 %
	121-BCube	487.6 ms	460.4 ms	5.58 %
VGG-16	9- BCube	1063.69 ms	900.88 ms	15.31%
	121- BCube	1134.72 ms	1060.4 ms	6.55 %

3) *Mitigation of Multi-bottlenecks*

We then show the results of mitigating bottlenecks by TMDML in terms of ECN and CNP. Fig. 8 shows the number of ECN generated by each switch and CNP received for each flow. In Fig. 8 (a), TMDML reduces the number of ECN at each switch. The less ECNs implies a less occupied queue, indicating that TMDML can reach a lower congestion state. The receiver issues a CNP on the arrival of an ECN-marked packet if no CNP has been sent for this flow over a period of time. On receiving CNP, the sender server reduces the rate of the corresponding flow. In Fig. 8 (b), the flows with more CNPs are identified as the *slow* flows, and TMDML makes these flows faster by reducing the number of CNP. On the other hand, the *fast* flow will be dragged for the purposed of congestion control.



(a) ECN, 128-Fat-Tree (b) CNP, 128-Fat-Tree

Fig. 8. The number of ECN generated by switches.

V. CONCLUSION & FUTURE WORK

In this paper, we study the impact of multi-bottlenecks on DML via a packet-level simulation. Simulation results reveal that the exiting RDMA-based protocol (i.e., DCQCN) cannot perform well in terms of DML traffic in the multi-bottlenecks scenario. To mitigate the impact of multi-bottleneck, we propose a transport layer solution call TMDML. We present an optimization model with an objective function for minimizing the maximum FCT, constrained by the stability of queues. Based on the heuristics, we design two implementations of TMDML to solve the optimization problem. Simulation results show that our proposals can reduce up to 59% FCT in Fat-Tree and up to 24% in BCube. Moreover, with the proposed TMDML, DML can achieve better performance in Fat-Tree that BCube. The performance in Fat-Tree is significant since Fat-Tree (or Clos) is one of the most widely used topologies in today's data center. In the future, we will first extend the current work to the scenario where multiple instances of DML are training with in a network. Then, we will present the detail of parameters tuning for TMDML. The parameter of packet counter B can affect the trade-off between accuracy and overhead; therefore, it needs to be tuned elaborately.

ACKNOWLEDGMENT

This work was supported by the Science and Technology Fundament Research Fund of Shenzhen under grant JCYJ20170307151807788.

REFERENCES

[1] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "GeePS: Scalable Deep Learning on Distributed GPUs with a GPU-specialized Parameter Server," in *EuroSys '16*, 2016, pp. 1–16.
[2] C. Chen, W. Wang, and B. Li, "Round-Robin Synchronization: Mitigating Communication Bottlenecks in Parameter Servers," in *IEEE INFOCOM 2019*, Apr. 2019, pp. 532–540.
[3] H. Zhang *et al.*, "Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters," in *USENIX ATC*, 2017, pp. 181–193.
[4] Q. Ho *et al.*, "More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server," in *NeurIPS*, 2013, pp. 1223–1231.
[5] S. Shi, X. Chu, and B. Li, "MG-WFBP: Efficient Data Communication for Distributed Synchronous SGD Algorithms," in *IEEE INFOCOM 2019*, 2019, pp. 172–180.
[6] Y. Peng *et al.*, "A generic communication scheduler for distributed DNN training acceleration," in *SOSP '19*, 2019, pp. 16–29.
[7] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," *arXiv preprint arXiv:1712.01887v2*, 2018.
[8] S. Shi *et al.*, "A Distributed Synchronous SGD Algorithm with Global Top-k Sparsification for Low Bandwidth Networks," in *IEEE ICDCS*, 2019, pp. 2238–2247.
[9] W. Wen *et al.*, "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning," in *NeurIPS*, 2017, pp. 1509–1519.
[10] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," in *NeurIPS*, 2017, pp. 1709–1720.
[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.
[12] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-Efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training," in *ICPP*, 2019, pp. 1–11.
[13] C. Guo *et al.*, "RDMA over Commodity Ethernet at Scale," in *ACM SIGCOMM*, 2016, pp. 202–215.
[14] Y. Zhu *et al.*, "Congestion Control for Large-Scale RDMA Deployments," in *ACM SIGCOMM*, 2015, pp. 523–536.
[15] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.
[16] S. Wang, D. Li, J. Geng, Y. Gu, and Y. Cheng, "Impact of Network Topology on the Performance of DML: Theoretical Analysis and Practical Factors," in *IEEE INFOCOM 2019*, Apr. 2019, pp. 1729–1737.
[17] M. Alizadeh *et al.*, "pFabric: minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.
[18] L. Chen, J. Lingys, K. Chen, and F. Liu, "AuTO: scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of ACM SIGCOMM*, 2018, pp. 191–205.
[19] Z. Guo, S. Liu, and Z.-L. Zhang, "Traffic Control for RDMA-Enabled Data Center Networks: A Survey," *IEEE Systems Journal*, vol. 14, no. 1, pp. 677–688, Mar. 2020.
[20] Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, and H. Jin, "Carbon-Aware Online Control of Geo-Distributed Cloud Services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2506–2519, Sep. 2016, doi: 10.1109/TPDS.2015.2504978.
[21] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. San Mateo, CA, USA: Morgan Kaufmann, 2010.
[22] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
[23] M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar, "Stability Analysis of QCN: The Averaging Principle," in *Proceedings of the ACM SIGMETRICS*, 2011, pp. 49–60.
[24] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397-413, 1993.