Impact of Network Topology on the Performance of DML: Theoretical Analysis and Practical Factors

Abstract—To deal with the increasingly larger input data and model sizes, it has become necessary to scale the training of machine learning models to multiple nodes, even a server cluster, which we call distributed machine learning, or DML. However, DML utilizes more computation power at the cost of high communication overhead, which may limit the overall performance in turn. In this paper, we study the impact of network topology on the DML performance both in theory and in practice. We compare two representative network topologies, namely, Fat-Tree which is widely-used in modern data centers, and BCube, which is a low-cost and server-centric network topology, both running on top of RDMA. The results show that Fat-Tree not only has theoretically higher global synchronization time (GST) than BCube, but its practical GST (by NS-3 based simulation) is also considerably larger than the theoretical one. By analyzing the large-scale simulation traces, we find that the root cause for the gap in Fat-Tree comes from the load imbalance among the multiple parallel paths as well as the inevitable PFC frames, both of which do not appear in BCube. For a cluster of around 250 servers, BCube achieves 53% ~ 70% lower GST than Fat-Tree from the simulation. As a result, we suggest using server-centric network topology such as BCube, instead of the common Fat-Tree network, to build a special-purpose DML cluster, due to its parallel synchronization, RDMA friendliness, natural load balance, as well as low economical cost.

I. INTRODUCTION

In recent years, machine learning (ML) has made exciting progress and improved its functionality in a wide range of applications, including image recognition, speech recognition and machine translation. Those ML models, especially deep learning (DL) models, consist of multiple layers and massive parameters. They are both data- and computation-hungry, usually taking days or even weeks to train on a standalone machine. This unacceptable time cost makes it necessary to train these large models with big data on a distributed cluster [1]–[3], which we call distributed machine learning (DML).

A ML model is trained iteratively with batch gradient descent or one of its variants, e.g., stochastic gradient descent and the widely-used minibatch gradient descent, until the parameters become (sub-)optimal. During each iteration, a minibatch of training data is chosen as the input of the ML model to compute a set of gradients of the current parameters. In DML, each compute node trains the ML model with a different portion of the minibatch, called sub-minibatch, and then the gradients calculated by each node are aggregated and applied to the parameters before starting the next iteration. Ideally, the training time will decrease as the number of compute nodes increases. However, the high volume of communication caused by synchronizing parameters can astonishingly overwhelm this naive assumption [1]. For example, [2] finds that the time of training a VGG-19 model with 229 million parameters on 32 machines can be even longer than that of a standalone machine. Therefore, communication is really a key point to efficiently run DML, especially large-scale one.

The performance of parameter synchronization is closely related to the following three factors: (1) parameter synchronization algorithm. PS-based synchronization is well supported by the major ML frameworks, e.g., TensorFlow, MXNet and PMLS-Caffe. Additionally, ring-based synchronization as well as mesh-based synchronization, a generalized form of PS-based synchronization, are also used in industry. (2) transport protocol. With the increasing availability of faster Ethernet networking technologies such as RDMA, implementation of DML framework with new transport protocol can achieve much better performance acceleration than the one with TCP [4]. (3) network topology. Network topology plays a fundamental role among these three factors, because parameter synchronization algorithm is based on the physical network topology and a high-performance transport protocol cannot maximize its performance without a suitable network topology. Fat-Tree [5] is the predominant topology in today’s data centers, but it has been shown RDMA-inefficient due to Priority-based Flow Control (PFC) congestion-spreading [6], even though patches such as Data Center QCN (DCQCN) [7] are used to mitigate this problem.

In this paper, we focus on studying the impact of network topology on the performance of DML, by using the most advanced parameter synchronization algorithms and transport protocol (RDMA), both in theory and in NS-3 simulation. Specifically, we want to answer a question: as the default network topology in today’s data centers, is Fat-Tree the ideal network fabric for building a special-purpose DML cluster? As a comparison, we also consider an alternative server-centric network topology, BCube [8]. Although constructed with much less switches to connect a specific number of servers, BCube theoretically has much less global synchronization time (GST) compared with Fat-Tree, due to its multi-NIC connection in each server as well as the server-switch interleaving layout. Moreover, the packet-level simulation in NS-3 shows that the practical gap between the GSTs under the two network topologies is even larger. For instance, to synchronize a model with about 57.5 MB parameters, the theoretical GST in a Fat-Tree with 250 servers is 22,908 ms but the simulation result is 25.140 ∼ 39.608 ms, with a 10% ∼ 73% times expansion; on the contrary, BCube with 256 servers achieves almost the same theoretical and practical GSTs, with 11.454 ms and 11.695 ms.
respectively.

To better understand this problem, we carry out a detailed analysis of the NS-3 trace. We find that the root cause of the deviation between theoretical and practical results in Fat-Tree, primarily comes from the load imbalance among the multiple parallel paths as well as its inefficient RDMA support. First, though ECMP is used to distribute the massive synchronization flows into multiple paths, ideal load balance cannot be achieved in practice due to uncertain hashing results. The trace shows that the traffic loads on different links may differ by up to 19 times. The load imbalance greatly reduces the overall network utilization. Second, although we enable DCQCN in the simulation to mitigate the PFC problem, PFC pause frames still occur in practice. For example, we find 86, 230 and 1,227 PFC pause frames when using mesh-based synchronization and hierarchical parameter synchronization (HiPS) in Fat-Tree, respectively. These PFC frames will hurt the overall network throughput during parameter synchronization [7].

However, we do not observe the two problems above in similar-sized BCube network. First, BCube’s dual NIC ports enable parallel synchronization, and each synchronization flow has a fixed forwarding path based on the BCube parameter synchronization algorithm, hence the traffic loads are ideally balanced among the links. Second, in BCube switches are not directly connected. Thus, even when a switch generates PFC pause frames, the neighboring server has enough memory to absorb them, which will not cause further spreading. Based on the advantages of BCube over Fat-Tree in supporting large-scale DML as well as its lower cost, we strongly suggest using BCube as the network topology to build a special-purpose DML cluster. One may argue about the wiring cost of BCube. But it is an affordable cost for a DML cluster with hundreds of servers, which is the typical DML cluster size.

The rest of the paper is organized as follows. Section II presents the background and motivation of this work. Section III compares the GSTs in DML under Fat-Tree and BCube networks, in both theory and practice. Section IV describes the related works. Finally, Section V concludes the whole paper.

II. BACKGROUND AND MOTIVATION

A. DML Synchronization

There are two different approaches to distribute a ML job, model parallelism and data parallelism. In model parallelism, different compute nodes are responsible for the computations in different parts of the same neural network (for DL). In data parallelism, different compute nodes have a complete copy of the model but a different portion of the data is trained on each node. While model parallelism does have benefits such as scalability for large models, data parallelism is easy to implement and widely used in the industry. In this paper, we focus on data parallelism.

In data parallelism, two stages, the computation stage and the synchronization stage, are executed successively during each iteration. In the computation stage, each node calculates the gradients of all parameters based on the predefined loss function and the current values of parameters. In the synchronization stage, gradients from different nodes are somehow combined and then used to update parameters. The new parameters will be used to calculate the new gradients in the next iteration. To accelerate the computation stage, GPUs are widely used in ML, because the most computational operations, e.g., matrix-matrix multiplications, can be efficiently performed by GPUs.

However, the acceleration GPUs bring to computation makes communication become the bottleneck of DML. Parameter synchronization is a costly operation that can significantly reduce the benefits of parallelism. To trade off between synchronization overhead and fresher updates, three synchronization models are proposed: (1) Bulk Synchronous Parallel (BSP) [9], which will not begin the next iteration until the parameters are updated using the gradients from all compute nodes; (2) Asynchronous Parallel (ASP) [10], which allows the compute nodes to keep running and only best-effort parameter synchronization is executed; (3) Stale Synchronous Parallel (SSP) [11], which allows the slowest node to fall behind the fastest node by a bounded number of iterations, otherwise the fast node has to stop calculating and wait for the slowest node. There is no guarantee for ASP to reach a convergence, while it is computationally efficient compared with BSP and SSP. Though SSP outperforms BSP in convergence speed when solving a convex problem [11], it has been shown in [11] that BSP is faster to get the same accuracy than SSP and has a much faster convergence performance when training a DNN model, a non-convex problem. Therefore, BSP is the most widely-used synchronization model in today’s DML practice. Due to the above reasons, we focus on BSP for data parallelism in this paper.

Fig. 1 shows different parameter synchronization architectures for DML. Parameter server (PS) architecture [3], the most influential one, divides the nodes into server group and worker group. Each node in server group maintains a partition of global parameters. Gradients are calculated independently on each node in worker group and then are pushed to the server group to update the parameters. Worker nodes do not communicate with each other. The new parameters are directly pulled from the server group by worker nodes. However, too many workers can easily cause network bottleneck at the server group. Mesh architecture can be used to solve this problem, in which each node is both a server and a worker, i.e., it not only calculates the gradients but also maintains a partition of the global parameters. Ring architecture is
proposed with a new parameter synchronization algorithm, Ring-Allreduce [12]. Similar to mesh architecture, each node in ring architecture also acts as both a server and a worker. All the nodes are arranged as a logical ring, and the gradients/parameters are divided and passed sequentially along the ring. Therefore, each node communicates only with its neighbors. In addition, hierarchical parameter synchronization (HiPS) [13], which advocates that gradients/parameters are synchronized in a hierarchical way to better accommodate the network topologies, has also been proposed recently.

B. DML Transport Protocol

Transport protocol has a great impact on the communication efficiency in DML. Though TCP gets great success in wide area network (WAN), it achieves poor performance in data centers. Some variants of TCP have been proposed to overcome the deficiency of TCP in the data center environment. For example, Data Center TCP (DCTCP) [14] uses Explicit Congestion Notification (ECN) to mark congestion degree at the switch and adjusts the sending rate according to the fraction of the packets marked by ECN at the sender side. Considering the burst traffic caused by parameter synchronization, DML can indeed benefit from DCTCP, which handles well with TCP incast.

RDMA technology, employed in High Performance Computing (HPC) at first, is also introduced in DML recently [4]. RDMA allows the application running on one host to directly access the memory of the remote host without involving the remote CPU. Compared with TCP, RDMA has the following advantages: (1) zero-copy. Data is directly transferred from NIC to the application’s buffer, avoiding the memory copy and system call overhead; (2) kernel-bypass. By offloading the networking protocol stack into hardware, RDMA significantly reduces overall latency without kernel involvement; (3) no CPU involvement. Remote memory can be accessed without consuming any CPU time, saving more CPU resources for computing. Thus, RDMA can achieve much higher throughput and lower latency than any existing TCP scheme.

RoCEv2 (RDMA over Converged Ethernet v2) [6], an RDMA technology based on IP/Ethernet, is scalable, high-performance and cost-effective. A lossless network is better for RoCEv2 to guarantee reliable data transfer and achieve high performance. PFC [15] is used for this purpose. PFC pauses the upstream sending entity when the downstream switch port’s ingress buffer occupancy exceeds a certain threshold to avoid buffer overflow. Nevertheless, some additional problems, e.g., PFC deadlock and PFC pause frame storm, are introduced by PFC. Besides, due to the limited buffer, only two lossless priorities are used in practice [6]. Therefore, half or more RoCEv2 flows, regardless of elephant flows or mice flows, on the same link are paused when PFC is triggered at one of the priorities, resulting in poor performance for individual flows. In order to trigger PFC as little as possible, DCQCN has been proposed. Built on QCN and DCTCP, DCQCN [7] leverages ECN to adjust the sending rate before the buffer occupancy reaches a specified threshold. Furthermore, DCQCN improves the fairness of RoCEv2 traffic by operating on a per-flow basis. In the following, we refer RDMA to RoCEv2 unless explicitly stated, and DCQCN is regarded as its default congestion control mechanism.

To better understand the merits of RDMA over TCP in supporting DML, we carry out a simple experiment as follows. We run TensorFlow to train a Convolutional Neural Network (CNN) model, which has 1 million parameters, for the well known MNIST digit recognition task. Two hosts are used to build the minimum DML environment, with one host as worker and the other host as parameter server. We run the training for 10000 iterations, using TCP and RDMA as the transport protocol, respectively. As shown in Table I, the total training time under TCP and RDMA is 4278 seconds and 2420 seconds, respectively. The experiment result clearly demonstrates that the communication cost in DML is extremely high and RDMA-based DML can greatly reduce the training time compared with TCP-based one. Due to the substantial performance gap, we mainly focus on RDMA as the transport protocol in DML in this paper.

<table>
<thead>
<tr>
<th>Transport protocol</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>4278</td>
</tr>
<tr>
<td>RDMA</td>
<td>2420</td>
</tr>
</tbody>
</table>

C. DML Network Topology

Network topology, which plays an important role in large-scale DML, needs to meet the following requirements: (1) communication-efficient. The topologies must gracefully support the synchronization algorithms, i.e., the performance of the synchronization algorithms cannot be limited by the underlying network; (2) fault-tolerant. Fault is inevitable in large-scale network, hence, it is expected that network performance does not drop significantly when small-scale failures occur in the network; (3) scalable based on commodity devices. In view of the cost, commodity Ethernet switches and routers are used in large-scale DML clusters. Thus, the topology needs to be scalable on commodity devices.

Fat-Tree [5], built with n-port switches, is the most used topology in modern data centers. In Fat-Tree, \( n^{3/4} \) servers are divided into \( n \) pods, in which each of \( n/2 \) edge switches is directly connected to \( n/2 \) servers and \( n/2 \) aggregation switches. The rest \( n/2 \) ports of each aggregation switch are connected to \( n/2 \) of \( (n/2)^2 \) core switches. Fat-Tree leverages enough commodity switches to make the aggregate bandwidth between arbitrary adjacent two layers equal, and there are \( (n/2)^2 \) equivalent paths between servers in any two different pods. Thus, it is rearrangeably non-blocking. Fig. 2(a) shows an instance of Fat-Tree topology with \( n = 4 \).

BCube [8], designed for container-sized data centers, is a representative server-centric network topology. BCube(\( n,k \)) takes a recursive way to build a large-scale network consisting of \( n^k \) servers with \( k \cdot n^{k-1} \) n-port commodity switches.
Each server, which participates in packet forwarding, has $k$ NICs. These switches are organized into $k$ levels, identified from 0 to $k-1$, and a switch in level-$i$ (0 ≤ $i$ ≤ $k-1$) is identified by an address array $[i, a_{k-2}, a_{k-3}, \ldots, a_0]$ (0 ≤ $a_i$ ≤ $n-1$). Also, the servers are identified by an address array $[a_{k-1}, a_{k-2}, \ldots, a_0]$ (0 ≤ $a_i$ ≤ $n-1$). The server $[a_{k-1}, \ldots, a_i, \ldots, a_0]$ is connected to the switch $[i, a_{k-2}, \ldots, a_{i+1}, a_{i-1}, \ldots, a_0]$ using one of its NICs. It is worth noting that the servers are only connected to different levels of switches, and there is no direct connection between any two switches or any two servers. There are $k$ parallel paths between arbitrary two servers in a BCube($n, k$). Fig. 2(b) shows an instance of BCube topology with $n = 4$ and $k = 2$.

To build a BCube($n, k$) network, $k \times n^{k-1}$ switches are needed. But $5 \times n^{k-1}$ switches are required for a Fat-Tree network to support the same number of servers. In practice, a server machine can be equipped with eight or even more GPU cards to carry out ML training [16]. Hence, a DML cluster can be equipped with eight or even more GPU cards to carry out ML training [16]. Consequently, the composed of tens to hundreds of machines should be enough to support reasonable large-scale training. Hence, a DML cluster contains tens of to hundreds of servers and each server is equipped with 2 NIC cards. Dual-port servers are commonly deployed in today’s data centers [20]. The wiring cost is also affordable for building such a special-purpose cluster. Second, the traffic pattern in DML training can lead us to design synchronization algorithms [13] in which flows only need to participate in packet forwarding. Moreover, servers can contribute to aggregating synchronization traffic and absorbing PFC pause frames if any.

Due to the topological features, the propagation of PFC pause frames in BCube and Fat-Tree also differs. In BCube, since the switches are not directly connected to each other, PFC pause frames can be limited within 1-hop: when a switch sends PFC pause frames, only the server connected to it is paused. The large memory buffer in the server makes it possible for the server to continue receiving packets from its upstream sending entity, without triggering PFC pause frames further. However, if a switch is paused in Fat-Tree, the limited buffer of the switch makes it have to further pause its upstream sending entity to avoid packet dropping. As a result, we call networks such as BCube RDMA-friendly.

In Fat-Tree, a large number of equivalent parallel paths exist between two servers. Therefore, we need to use load balance techniques to make good utilization of the rich link resources. ECMP is the most widely-used method in today’s data center networks. Since ECMP selects the next hop using the hash value of the flow’s 5-tuple, it guarantees that a flow’s packets are forwarded along the same path [17]. Existing works show that the uncertain hashing results in ECMP may cause considerable load imbalance on the parallel paths, thus per-packet scheduling such as round-robin are proposed to achieve better balance among multiple paths [18]. To solve the packet reorder problem, a large buffer is required at the receiver to absorb out-of-order packets. Although this method works in TCP transport, it does not fit RDMA scenario, where the NIC buffer is quite limited [19]. One may argue that we can use the main memory on the receiver to absorb the packets, but recent work demonstrates that the frequent swap between the RDMA NIC buffer and the main memory can significantly degrades the flow throughput [19]. Therefore, in this work, we still assume that ECMP is used for load balance in Fat-Tree. On the contrary, in BCube network we do not have this problem. Although $k$ equivalent paths exist in a BCube($n, k$) network, the sender of a flow can determine the exact path by choosing the specific outgoing NIC port. Therefore, ideal load balance can be naturally achieved in BCube by the applications.

**D. Motivation of This Work**

Fat-Tree and BCube are two representative data center network topologies proposed about ten years ago [5], [8]. Since then, Fat-Tree is gradually becoming the de facto network topology in modern data centers. BCube is usually criticized for two aspects, namely, the high wiring cost and the low packet forwarding efficiency on servers. However, we argue that the two issues do not exist for building a DML cluster. First, as aforementioned, a reasonable-sized modern DML cluster contains tens of to hundreds of servers and each server is equipped with 2 NIC cards. Dual-port servers are commonly deployed in today’s data centers [20]. The wiring cost is also affordable for building such a special-purpose cluster. Second, the traffic pattern in DML training can lead us to design synchronization algorithms [13] in which flows only cross 1-hop switch in BCube. It means that BCube servers do not need to participate in packet forwarding. Moreover, servers can contribute to aggregating synchronization traffic and absorbing PFC pause frames if any.

By considering that the deficiencies of BCube in traditional data centers are significantly mitigated when building a special-purpose DML cluster, as well as the advantages of BCube over Fat-Tree with regard to low switch cost, RDMA-friendliness and natural load balance, in this work we want to revisit the comparison between Fat-Tree and BCube networks in supporting DML. In what follows, we make the comparison by considering both theoretical results and practical factors. We expect that our findings can be helpful to those who plan to
build a server cluster specifically for running high-performance ML training.

III. FAT-TREE VS. BCUBE: THEORETICAL ANALYSIS AND SIMULATION EVALUATION

In this section, we first theoretically analyze the DML performance in Fat-Tree and BCube networks, with GST in each training iteration as the overall performance metric. To investigate the impact of load imbalance and PFC on the real DML performance, we then use NS-3 simulator to carry out large-scale simulations in the two networks respectively, which will investigate the packet-level behaviors and demonstrate the impact of network fabric on the practical GST\(^1\). For convenience of expression, we summarize the notations in Table II.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>the number of ports of a switch in Fat-Tree or BCube</td>
</tr>
<tr>
<td>(k)</td>
<td>the number of switch levels in BCube</td>
</tr>
<tr>
<td>(N)</td>
<td>the total number of servers in Fat-Tree or BCube</td>
</tr>
<tr>
<td>(B)</td>
<td>the bandwidth of the server NIC or the switch port</td>
</tr>
<tr>
<td>(P)</td>
<td>the total size of all the gradients/parameters of a ML model</td>
</tr>
</tbody>
</table>

A. Theoretical Analysis

For analyzing the theoretical GST, we choose three representative parameter synchronization algorithms, i.e., mesh-based synchronization, ring-based synchronization and hierarchical synchronization. The first is actually the P2P implementation of the widely-used PS architecture, which avoids the bottleneck of a single parameter server. The second is the state-of-the-art implementation for scale-efficient DML in industry. The third is a recently proposed synchronization algorithm for networks with hierarchical structures.

1) Mesh-based Synchronization: In mesh-based synchronization, all the gradients/parameters are equally divided into \(N\) segments. Each server is responsible to update only one parameter segment using the corresponding gradient segments aggregated from all the servers. There are two stages, called the scatter stage and the gather stage, respectively, in the synchronization process. To be specific, in the scatter stage, when one server finishes computing, all gradient segments except the one corresponding to the parameter segment maintained by itself, i.e., \(N - 1\) gradient segments, are sent to other servers. After receiving \(N - 1\) gradient segments from other \(N - 1\) servers, each server combines them together with its own one in some way, e.g., computing the average of them. Then, the aggregated gradient segment is used to update the corresponding parameter segment. Finally, in the gather stage, the server sends its updated parameter segment to other \(N - 1\) servers in a similar way as the scatter stage. As a result, all the parameters on each server are updated with all the gradients computed by all the servers. In Fat-Tree, each link has the same bandwidth of \(B\). Thus, for mesh-based synchronization in Fat-Tree, the GST can be calculated as follows:

\[
GST(F, m) = \frac{2(N - 1)^2}{B} = \frac{2(N - 1) P}{N} \frac{1}{B} \quad (1)
\]

Mesh-based synchronization can also be employed in BCube. Since the synchronization is the same as in Fat-Tree, we omit the detailed description. According to Theorem 6 in [8], the aggregate bottleneck throughput of BCube under all-to-all traffic is \(\frac{\sqrt{N} - 1}{\sqrt{N}} (N - 1)B\). Therefore, we directly give the result of mesh-based synchronization in BCube as follows:

\[
GST(B, m) = \frac{2(\sqrt{N} - 1) P}{\sqrt{N}} \quad (2)
\]

Considering that \(\sqrt{N}\) is much smaller than \(N\), BCube can achieve marginal shorter, or at least equal, synchronization time when using mesh-based synchronization, compared with Fat-Tree.

2) Ring-based Synchronization: In ring-based synchronization, all \(N\) servers are organized as a logical ring. The gradients/parameters are divided and maintained like in mesh-based synchronization. We take one of these segments as an example to describe the synchronization process. At the beginning of the scatter stage, server 0 sends a gradient segment to server 1. Then, server 1 aggregates this gradient segment with its local portion, and sends the aggregated gradient to server 2. Finally, this gradient segment is aggregated with all the servers’ local portions when it reaches server \(N - 1\), and used to update its corresponding parameter segment. At the same time as the above process, the rest servers send different gradient segments to their neighbors in this way. In the gather stage, these updated parameter segments scattered across different servers are transferred along the logical ring in a similar way to the scatter stage, but they are used to directly replace the old parameter segments.

Obviously, both Fat-Tree and BCube satisfy that all the servers can be organized as a logical ring (The simplest way is to form a ring in the order of the servers’ addresses, and it is also the most effective way in Fat-Tree). However, according to Euler’s circuit theorem, if a BCube satisfies that both \(n\) and \(k\) are even, there must exist a path which passes by each link exactly once while starts and stops at the same server. In this case, two logical rings can be formed based on bidirectional links. Thus, two parallel processes can be executed simultaneously to share the synchronization workload. But there is no such path in Fat-Tree, because the degree of each server is odd. Hence, the GSTs of ring-based synchronization in Fat-Tree and BCube are as follows:

\[
GST(F, r) = \frac{2(N - 1)^2}{B} = \frac{2(N - 1) P}{N} \frac{1}{B} \quad (3)
\]

\[
GST(B, r) = \begin{cases} \frac{(N - 1) P}{\sqrt{N}} & , \text{ otherwise} \\
\frac{2(N - 1) P}{\sqrt{N} \sqrt{N}} & , \text{ otherwise} 
\end{cases} \quad (4)
\]

\(1\)Since the total costs of RDMA devices for large-scale testbed experiment is really high, we cannot afford it for real deployment. Fortunately, the NS-3 simulator for RDMA is available at [7] and can be used for our evaluation.
Considering the typical value of \( k \) when building a BCube DML cluster is 2, it only requires that \( n \) is even to synchronize in both directions, which is exactly the case for most switches. Therefore, ring-based synchronization is able to achieve better performance in BCube than in Fat-Tree.

3) Hierarchical Synchronization: Hierarchical parameter synchronization (HiPS) [13] can speed up the parameter synchronization in hierarchical networks such as BCube. There are also the scatter stage and the gather stage in HiPS. However, each stage consists of \( k \) phases in a BCube\((n, k)\). In each phase, \( n^k \) servers are divided into \( n^{k-1} \) groups according to the level of switch, i.e., the servers under the same switch are located in the same group. In each group, the \( n \) servers can choose mesh-based synchronization or ring-based synchronization. Here we take mesh-based synchronization as an example.

In the first phase of the scatter stage, the servers are divided according to the level-0 switches. Then mesh-based synchronization is executed in each group simultaneously. It is worth noting that the gradients are divided into \( n \) (rather than \( N \)) segments in each group and each synchronization processes occurs only among \( n \) servers. At the end of this phase, the server \([s, \ldots, s, i] (0 \leq i \leq n-1, * \text{ is a wild-card})\) aggregates the \( i \)th segments from all the servers in this group. In the next phase, the servers are divided according to the level-1 switches. And each gradient segment aggregated in the last phase is further divided into \( n \) segments. After this phase, each server has a gradient segment aggregated from \( n^2 \) servers. The remaining phases are similar, except the divided gradient segments are changed to \( 1/n \) of the previous phase. At the end of the scatter stage, the aggregated gradient segments (The size is \( P/n^k \)) are used to update the corresponding parameter segments, which will be synchronized in a similar way but in reverse order in the gather stage.

It is worth noting that \( k \) processes can be executed in parallel, each of which begins from different level of switches, to fully utilize the bandwidth resources. In each process, each link has a bandwidth of \( B \). Therefore, for HiPS in BCube, the GST is as follow:

\[
GST(B, H) = \sum_{i=1}^{k} \frac{(n-1)P}{kn^i B} = \frac{2(N-1)P}{knB} (5)
\]

In fact, we find that HiPS can also be deployed in Fat-Tree. Like in BCube, the servers are divided into many groups. There are three phases in both stages. In the first phase, \( n/2 \) servers under the same edge switch, i.e., the address of each of these servers is \([p, e, s] (0 \leq p < n, 0 \leq e < \frac{n}{2}, * \text{ is a wild-card})\), form a group. The gradients are divided into \( n/2 \) segments, and are synchronized in each group. In the next phase, \( n/2 \) servers in the same pod but not under the same edge switch, i.e., the address of each of these servers is \([p, * , s] (0 \leq p < n, 0 \leq s < \frac{n}{2}, * \text{ is a wild-card})\), form a group. Each aggregated gradient segment is further divided into \( n/2 \) segments, and continues being synchronized in each group. Like this, in the third phase, \( n \) servers in different pods, i.e., the address of each of these servers is \([s, e, s] (0 \leq e < \frac{n}{2}, 0 \leq s < \frac{n}{2}, * \text{ is a wild-card})\), form a group. The gather stage is processed in reverse order. Thus, for HiPS in Fat-Tree, the GST is as follow:

\[
GST(F, H) = 2 \left[ \frac{(\frac{n}{2} - 1)P}{\frac{n}{2} B} + \frac{(\frac{n}{2} - 1)P}{(\frac{n}{2})^2 B} + \frac{(n-1)P}{(\frac{n}{2})^n B} \right] = \frac{2(N-1)P}{N-B} \tag{6}
\]

The comparative result is summarized in Table III. We can draw a conclusion that the same parameter synchronization algorithm, whether mesh-based synchronization, ring-based synchronization or HiPS, spends less time to finish a round of parameter synchronization in BCube than in Fat-Tree.

<table>
<thead>
<tr>
<th>TABLE III</th>
<th>Theoretical GSTs in Fat-Tree and BCube</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fat-Tree</td>
</tr>
<tr>
<td>mesh-based synchronization</td>
<td>( \frac{2(N-1)P}{n/N} )</td>
</tr>
<tr>
<td>ring-based synchronization</td>
<td>( \frac{2(N-1)P}{n/N} )</td>
</tr>
<tr>
<td>HiPS</td>
<td>( \frac{2(N-1)P}{n/N} )</td>
</tr>
</tbody>
</table>

Why do these synchronization algorithms have the same theoretical GST in fat-tree, longer than the ones in BCube? More importantly, we are surprised to find that the three synchronization algorithms achieve the same theoretical GST in Fat-Tree. That is, neither ring-based synchronization nor HiPS can speed up the synchronization in Fat-Tree. The root cause of this is that each server in Fat-Tree has only one NIC to connect to the network. For HiPS in Fat-Tree, though multiple communication processes are executed simultaneously on each server, it cannot achieve better performance. Because these communication processes have to share the only link resource. For ring-based synchronization in Fat-Tree, only one logical ring can be used because a server has to use one of the two directions of the link between it and the edge switch to send gradients/parameters to its next neighbor, and use the other direction to receive from its previous neighbor. This not only fails to speed up the synchronization, but also wastes lots of network resources. Especially, at most \( 4/n^2 \) of core network resources can be utilized, since the flow passes by the core switch only when two servers in different pods communicate with each other. Compared with Fat-Tree, BCube allows these synchronization algorithms to be executed efficiently in parallel, due to the multiple NICs on each server. And the network resources of BCube are fully utilized with any synchronization algorithm (Even if ring-based synchronization is used when \( n \) is odd, BCube can still achieve higher network utilization than Fat-Tree).

Knowledgeable readers may ask whether Fat-Tree can achieve the same, or even better, performance than BCube, by equipping each server with the same number of NICs as in
BCube. However, to support the same number of servers by this kind of interconnection in Fat-Tree, the cost of switches has to be doubled. It means that, for a BCube(n,2) network, its switch cost is only 20% of its correspondent one in Fat-Tree. Moreover, the gap between practical performance and theoretical value in Fat-Tree will be larger than that in BCube, as what we will describe as follows.

B. Simulation Study on NS-3

Although these three synchronization algorithms have the same theoretical GST in Fat-Tree, they are interfered by other factors in actual use. We use NS-3 to evaluate their performance by taking the packet-level practical factors into account. Also, we simulate ring-based synchronization and HiPS in BCube. For convenience of expression, we denote these five scenarios as follows: 1) scenario 1: mesh-based synchronization in Fat-Tree; 2) scenario 2: ring-based synchronization in Fat-Tree; 3) scenario 3: ring-based synchronization in BCube; 4) scenario 4: HiPS in Fat-Tree; 5) scenario 5: HiPS in BCube.

1) Simulation Setup: Our simulator, obtained from [21], can be used to model RoCEv2 NIC. We modify the number of flows each NIC (or switch port) supports to simulate large-scale mesh-based synchronization and add support for collecting PFC pause frame information. As aforementioned, DCQCN is enabled to mitigate PFC pause frames. Dynamic PFC threshold is used to automatically adjust the threshold based on the free shared buffer. The parameters of DCQCN are set according to the recommendations in [7], with $K_{in}, P_{in}, g, m, K_{max}, K_{max}, P_{m}, m_{max}, g, T_{ime}, B_{yte}$ Counter) = (1000KB, 40KB, 1%, 1/256, 60μs, 10MB).

The size of the synchronized parameter is set to 57.5 MB, which is one-tenth of the size of VGG-19, a famous CNN model that consists of 19 layers. RDMA MTU is 1 KB. For the default Fat-Tree topology, there are 250 servers connected by 125 10-port switches; For the default BCube topology, there are 256 servers connected by 32 16-port switches in two levels. We consider 40 Gbps links, each with a propagation delay of 1 μs. ECMP is used for load balance.

2) GST Comparison: Distribution of the actual GSTs of all the servers with different scenarios are shown in Fig. 3(a). The results show that BCube achieves 53% ~ 70% lower GST than Fat-Tree. We find that the actual GSTs of the scenarios which have the same theoretical GST may be quite different. For example, scenario 1 and scenario 2 have the same theoretical GST of 22.908 ms, but their actual GSTs are 28.086 ms and 25.140 ms, respectively, with a difference of 12%. This is because these scenarios generate different traffic patterns, resulting in performance differences even in the same network topology. Another observation is that GSTs of different servers are almost the same in any of these scenarios, except scenario 4. Therefore, additional straggler will not be introduced by parameter synchronization in these scenarios, if the servers have homogeneous computation capability.

3We omit mesh-based synchronization in BCube due to its much higher theoretical GST than the other two synchronization algorithms.

Fig. 3(b) shows the difference between actual GSTs and theoretical GSTs in different scenarios. Compared with the theoretical GST, the actual GST of HiPS in BCube is only expanded by 2%, while the actual GSTs of the three synchronization algorithms in Fat-Tree are expanded by 10% ~ 73%. To our surprise, HiPS, which speeds up the GST in BCube, achieves the worst performance in Fat-Tree among the three algorithms. In addition, the actual GST of ring-based synchronization in BCube is also 16% larger than the theoretical value.

Theoretical GST (ms)

50 45 40 35 30 25 20 15 10 5 0

Scenario 1 Scenario 2 Scenario 3 Scenario 4 Scenario 5

(a) CDF of actual GSTs of all the servers with different scenarios.

(b) Comparison of actual GST and theoretical GST in different scenarios.

Fig. 3. GST comparison between different scenarios.

To explore how the differences between the actual GSTs and the theoretical GSTs come, we randomly choose a server in each scenario to observe its throughput. The results are shown in Fig. 4. Illustrated by Fig.4(j), in scenario 5, the total throughput is alway equal to the total bandwidth of the two NICs on the server, i.e., 80 Gbps. This is because BCube achieves good load balance by the application when running HiPS. Therefore, HiPS can achieve almost the same GST as the theoretical value in BCube.

As shown in Fig. 4(i), in scenario 4, the total throughput is quite low in both the last two phases of the scatter stage and the first two phases of the gather stage. This is caused by ECMP collisions. In these four phases, multiple equivalent paths exist between two servers that need to communicate to each other. However, these paths are joint between different communication pairs. Therefore, the distribution of these flows on different links is likely to be imbalanced [18], [19], [22]. Fig. 5 further shows the distribution of all flows in the third phase of the scatter stage on uplinks from aggregation switches to core switches, the maximum being 19 times the minimum. While some links are very congested, other links are under light traffic load, which not only reduces the overall network utilization, but also slows down the process of synchronization.

ECMP collisions also cause long tail in both the scatter stage and the gather stage shown in Fig. 4(f). The straggler flows, i.e., the flows with large FCT among those sent to the same server, occupy less bandwidth resources, due to more competitors. Therefore, though some flows have been completed ahead of time, it is not possible for the server to start the gather stage or the next iteration, because the gradients/parameters carried by the straggler flows have not yet been received.

Whether in BCube or in Fat-Tree, ring-based synchronization is not affected by ECMP collisions. In BCube, the path
The total throughput in scenario 3 is reduced to about negligible compared to the transmission time. For example, important role in synchronization efficiency when it is not its previous neighbor, the propagation delay will play an its next neighbor until it finishes receiving the segment from server cannot send the next gradients/parameters segment to ring-based synchronization. Nonetheless, considering that a the set of paths between these pairs is disjoint when using the shortest path between a server and its neighbor, thus ideal load balance is always guaranteed. In Fat-Tree, while there are of a flow is determined by its sender and there is only one shortest path between a server and its neighbor, thus ideal load balance is always guaranteed. In Fat-Tree, while there are multiple equivalent paths between a server and its neighbor, the set of paths between these pairs is disjoint when using ring-based synchronization. Nonetheless, considering that a server cannot send the next gradients/parameters segment to its next neighbor until it finishes receiving the segment from its previous neighbor, the propagation delay will play an important role in synchronization efficiency when it is not negligible compared to the transmission time. For example, the total throughput in scenario 3 is reduced to about 70 Gbps due to the non-negligible propagation delay. The same case is also shown in Fig. 4(g). Therefore, ring-based synchronization is not recommended when the transmission time of one round is small, whether in BCube or in Fat-Tree.

We can see that ECMP collisions in Fat-Tree greatly reduce the DML performance in practice. Compared with Fat-Tree, BCube achieves good load balance by the applications, especially, when running HiPS in BCube. Because synchronization occurs only under the same switch, the ideal load balance is achieved within each level, and the levels do not interfere with each other. While the impact of ECMP collisions on all-to-all communication is gradually reduced with the scale of Fat-Tree network increasing [22], all-to-all communication consumes lots of queue pairs in RDMA NICs. Given the limited cache in RDMA NICs, the flow throughput will be significantly degraded due to frequent cache misses in practice [19].

3) PFC Analysis: We collect PFC pause frame information in the simulation to study the support of RDMA in Fat-Tree and BCube when running DML synchronization. We find that no PFC pause frame is triggered in scenario 2, scenario 3 and scenario 5. The distributions of PFC pause frames received by different nodes in the other two scenarios are shown in Fig. 6. Obviously, ring-based synchronization triggers no PFC pause frame, because there is only one flow in each direction of each link. As a result, DCQCN can trigger ECN in time to tell the sender to adjust its sending rate before PFC is triggered. There is also no PFC pause frame when running HiPS in BCube. This is for two reasons: 1) The hierarchical synchronization greatly reduces the number of flows simultaneously processed by a switch’s port. That is, HiPS does not generate a large burst. For example, if we use mesh-based synchronization in BCube with 256 servers, there are 255 flows overwhelming a port’s buffer simultaneously. However, using HiPS can reduce this number to 15. Given the small number of concurrent flows, DCQCN can efficiently adjust the sending rate before PFC is triggered. 2) The number of flows processed by each ingress queue is equal to the one processed by each egress queue, due to the symmetrical traffic pattern. It is worth noting that PFC threshold is set based on ingress queue while ECN marking is based on egress queue [7]. The symmetrical traffic pattern ensures that the ingress queue length and the egress queue length are not significantly different. Therefore, ECN can always be triggered before PFC with proper parameters.

As a contrast, though HiPS in Fat-Tree can reduce the number of servers in each synchronization group, ECMP collisions break the symmetry of the traffic pattern, as shown by the scenario 4 curve in Fig. 6. The flows processed in the egress queue may be much more than the ones processed in the ingress queue. Thus, the packets are quickly built up in the ingress queue because the egress queue does not have time to
forward these packets. Finally, PFC is triggered though ECN has been triggered. The result also shows that while some PFC pause frames are found in the last two phases of the scatter stage and the first two phases of the gather stage, there is no PFC pause frame when synchronization is executed under edge switches, due to the only one shortest path.

There are the most PFC pause frames when running mesh-based synchronization in Fat-Tree, which is the current practice in the industry, as shown by the scenario 1 curve in Fig. 6. This is because it does not satisfy any of the above two conditions during the whole synchronization process. Furthermore, we also note that PFC pause frames are frequently received by some nodes, but never received by other nodes. This is also due to load imbalance caused by ECMP collisions.

IV. RELATED WORKS

Datacenter Network Topology: Besides Fat-Tree and BCube, many other topologies have been proposed in data centers. In particular, server-centric topologies, e.g., DCell [23] and FiConn [20], leverage servers with multiple NICs and forwarding capabilities to provide better support for various traffic patterns, including many-to-many. Also, these hierarchical designs are naturally adapted to communication-efficient hierarchical parameter synchronization methods [13].

Transport Protocol: In order to accelerate GPU communication, NVIDIA provides the NVIDIA Collective Communications Library (NCCL) [24] optimized for its GPUs to achieve high bandwidth in multi-GPU and multi-node collective communication. To get out of PFC for RoCEv2, Improved RoCE NIC (IRN) [25] has been designed to achieve the same, even better, performance without PFC. And resilient RoCE [26], based on RoCEv2 implementation, provides the ability to run over a lossy network with ECN, not PFC.

Load Balance: There are many literatures proposed to balance traffic in data centers [18], [19], [22], [27]. Unfortunately, most of these works only consider TCP traffic. To handle out-of-order packets, some of them require an extra reordering buffer [18]. However, due to the limited NIC hardware resources, they are hard to apply to RDMA. Other works, e.g., [19], [27], attempt to control out-of-order degree by proactive response or out-of-order aware load balance. However, they cannot guarantee out-of-order free, thus the performance of network will still be affected.

Other Approaches to Minimize DML Communication Cost: MLNET [28] proposes a host-based communication layer, which reduces network traffic using tree-based overlays and prioritizes traffic of different models to improve average training time, for DML. Additionally, some methods attempt to reduce ML model size, thus further reducing the communication traffic, such as removing insignificant parameters [29].

V. CONCLUSION

In this paper, we analyze the impact of network topology on the performance of DML from theoretical and practical aspects. Our analysis shows that DML achieves much worse performance in Fat-Tree than in BCube, even though Fat-Tree has 2.5X more switches. It is primarily caused by three reasons. First, the unique NIC of each server in Fat-Tree does not allow for highly-efficient parameter synchronization algorithms to execute in parallel. Second, hash collision of ECMP leads to traffic load imbalance, further causing congestion links to limit overall performance. Third, BCube is more RDMA-friendly than Fat-Tree. Based on these advantages of BCube over Fat-Tree, we strongly suggest using BCube as the network topology for special-purpose DML clusters.

REFERENCES

[15] "IEEE DCB 802.1Qbb," https://ieee802.org/dcb/802-1qbb-