Published in IEEE Transactions on Network Science and Engineering (Volume: 11, Issue: 5, Sept.-Oct. 2024)

Efficient Inter-Datacenter AllReduce With Multiple Trees

Shouxi Luo, Renyi Wang, Huanlai Xing

***Abstract*—In this paper, we look into the problem of achieving efficient inter-datacenter AllReduce operations for geo-distributed machine learning (Geo-DML). Compared with intra-datacenter distributed training, the heterogeneous wide-area network (WAN) connections among Geo-DML workers are scarce, expensive, and unstable, making existing proposals designed for homogeneous networks fall short. Despite that some recent optimizations have been proposed for Geo-DML, they break the consistency semantics of bulk synchronous parallel (BSP), thus bringing no benefit to the widely existing BSP-based applications.**

To address these issues, we propose MTREE, a topology management suite for Geo-DML. With the global view of the heterogeneous WAN connections, MTREE builds multiple optimized spanning trees along with suggested workload distribution proportions, respecting the constraints of both the number of trees and their maximum height specified by the training. Based on these results, geo-distributed workers could launch concurrent tree-based pipelined AllReduce operations to make efficient use of the heterogeneous network. Detailed performance studies on real-world network topologies imply that MTREE achieves efficient AllReduce, significantly outperforming existing solutions.

***Index Terms*—AllReduce, Inter-DC WAN, Communication Optimization**

I. Introduction

Nowadays, an increasing number of geo-distributed machine learning (Geo-DML) systems have been employed to train large sophisticated models for applications like *image and video classification*, *speech processing*, *machine translation*, and *topic modeling* over massive data around the globe [1], [2]. In these systems, involved training workers are hosted on different datacenters which are networked with scarce, expensive, and unstable wide-area network (WAN) connections [2]. To guarantee convergence, workers participating in data-parallel training must synchronize their local training results with AllReduce operations periodically. As confirmed by numerous recent studies, the time cost of performing parameter synchronization over these cross-datacenter connections has dominated the efficiency of geo-distributed training [1]–[3]. Accordingly, improving the efficiency of inter-datacenter (inter-DC) AllReduce operations over WAN connections becomes the key to optimizing the performance of large-scale Geo-DML. Then, a fundamental question follows: *How to make maximum usage of heterogeneous inter-datacenter WAN connections to achieve efficient AllReduce operations for geo-distributed training?*

This work was supported in part by NSFC under Projects 62002300, and in part by the Fundamental Research Funds for the Central Universities under Project 2682024ZTPY050. (Corresponding author: Huanlai Xing.)

The authors are with the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China, and also with the Engineering Research Center of Sustainable Urban Intelligent Transportation, Ministry of Education, Chengdu 611756, China (e-mail: sxluo@swjtu.edu.cn; renyiwang@my.swjtu.edu.cn; hxx@swjtu.edu.cn).

Despite that many proposals have been proposed for the optimization of parameter synchronization for distributed training, they fail to provide efficient AllReduce ability over heterogeneous WAN connections in various aspects [1], [4]–[7]. More specifically, some of them break the consistency semantics of bulk synchronous parallel (BSP) thus bringing no benefit to the widely existing BSP-based applications [1], [2], [7]—Indeed, BSP is still the most-widely used consistency model in production. Some others do provide BSP-based AllReduce ability—They work well in homogeneous network environments but fall short in heterogeneous cross-datacenter network environments [4], [5], or employ inefficient communication designs [6], [8], [9], as detailed in Section II-C.

For example, solutions based on *peer-to-peer* messaging [10] and *parameter server* (PS) [4] are unaware of both the application-level semantics of AllReduce and the heterogeneous topology of the underlying network. As nodes in Geo-DML are generally not fully connected, these schemes would generate duplicated traffic on the shared links according to the routing. Likewise, solutions like *Recursive Halving & Doubling* [11] and *Butterfly* [12] let a worker communicate with only another worker for data delivery or exchange at each step. Besides the problem of duplicated traffic, they could not efficiently use the heterogeneous abundant links among workers. Since there might not exist a ring to cover all nodes for some networks, ring-based solutions would also fail to make efficient usage of all available links and their capacities [13]. Several recent works like TopoAdopt [6], BLINK [8], and PackingTrees [9] explore the idea of finding multiple spanning trees for workers and then distributing workloads among them to make efficient usage of their heterogeneous yet abundant interconnections. However, they either do not explore the possibility of splitting tensors into fine-grain chunks for pipelined transmission [6], or ignore the tree structures when constructing [8], [9], thus are far from optimal.

In this paper, we propose MTREE (i.e., height-bounded *multiple trees*) to address all these issues and provide efficient AllReduce communication ability to geo-distributed training over inter-datacenter WANs. Basically, MTREE is a topology management suite that could construct multiple spanning trees and a vector of workload distribution proportions, to efficiently use all the available WAN connection bandwidth for pipelined AllReduce operations. Based on the trees and their workload proportions given by MTREE, a logical central controller consistently splits the data on all workers proportionally to conduct concurrent pipelined AllReduce operations. To control the system complexity, there would be a limit on the maximum number of concurrent AllReduce trees. Also, to control the structure of the generated tree and reduce the network latency of pipelined communication, MTREE allows training tasks to specify their limits on the trees' weighted heights. All these constraints and design objectives make the problem faced by MTREE hard to solve. Thus, it employs heuristic algorithm designs. Detailed evaluations using real-world inter-datacenter network topologies show that, compared with baselines including BLINK [8] and TOPOADOPT [6], MTREE not only achieves high throughput for concurrent pipelined AllReduce operations but also further reduces the height and maximum fanout of AllReduce trees.

In summary, our contributions are three-fold:

1) A thorough analysis that identifies the drawbacks of existing schemes along with the challenges of performing AllReduce operations over heterogeneous WANs (§II).
2) MTREE, a suite of flexible and efficient topology management and chunk size optimization algorithms that can construct multiple height-optimized trees to make efficient use of the heterogeneous WAN connections for concurrent pipelined AllReduce operation (§III).
3) Evaluations based on real-world inter-DC WAN topologies showing that MTREE is effective and efficient, outperforming existing solutions significantly (§IV).

Finally, we conclude this paper in Section V.

II. Background and Motivation

Before looking into the design details of MTREE, in this section, we briefly introduce the background of Geo-DML (§II-A) and the requirement of AllReduce operation (§II-B), overview the drawbacks of existing solutions through examples (§II-C), and highlight our design goals and challenges (§II-D).

A. Geo-Distributed Machine Learning

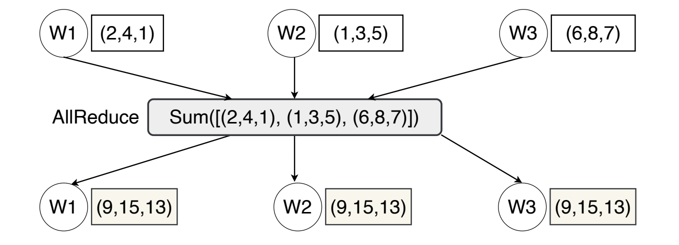
Currently, machine learning techniques (e.g., deep learning) are widely employed to train models for various applications using big data geographically distributed worldwide. For example, banks and financial institutions could jointly train shared models for *credit risk control* and *anti-money laundering*; hospitals and medical centers would use their medical data to train models for smart healthcare applications like *auxiliary diagnosis*; and a large cross-regional enterprise may use its geo-distributed datasets to train models for *item recommendation* [14]–[16]. In practice, multiple factors prevent the massive amount of training data from being gathered into a single data center for training. First, transferring such a huge amount of raw data over the bandwidth-scarce wide-aware network is quite slow and thus time-consuming [1], [16], [17]. Second, to meet the huge computing capacity demands in the era of large models, companies might prefer to make joint use of their legacy data center infrastructure worldwide to train new models efficiently [18]. Third and most critical, to satisfy the laws of data sovereignty and/or meet the requirement of privacy protection, the cross-regional gathering of data is impossible thus geographically distributed machine learning (i.e., Geo-DML) is a must [1]. With the rapid development of AI and its increasing use in production, we believe that Geo-DML will become more and more popular in the future. Regarding the model size, it might range from the magnitude of 10^7 (e.g., ResNet50) to 10^8 (e.g., Bert), to 10^{12} (e.g., GPT3), and continues to increase explosively in the era of large language models [19].

In distributed training, a model training job is cooperatively performed by a group of networked worker nodes. To guarantee and accelerate the coverage of the iterative distributed training, workers conducting training computation on their local data would perform global communication to synchronize the trained models over the network periodically. Compared with intra-datacenter DML, there are three main differences faced by Geo-DML. Firstly, in scenarios like federated learning, the training data might be inherently distributed, i.e., not *independently identically distributed* (i.e., non-IID) [20]—-Hence, the training worker nodes have to communicate more frequently for convergence. Secondly, to achieve privacy-protected Geo-DML over untrustworthy environments, technologies like *homomorphic encryption* might be employed and the traffic volumes would increase greatly since encrypted rather than raw tensors are transmitted over the network [21]. Last but not least, the connections between geo-distributed workers generally with scarce, expensive, and unstable available bandwidth [2], [21]. All these factors make the efficiency of model synchronization more critical and hard to improve in the context of Geo-DML.

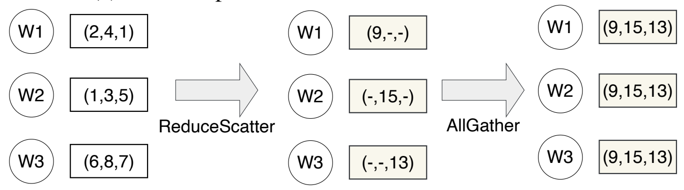
B. Requirements of AllReduce Operation

In Geo-DML, a model training job is cooperatively performed by a group of worker nodes interconnected with wide-area networks. To guarantee and accelerate the coverage of the iterative distributed training, workers conducting training computation on their local data would perform global communication to synchronize the trained models periodically; and Bulk Synchronous Parallel (BSP) is the most popular synchronization mode used by data-parallel distributed training in production today [22]. In each round of BSP-based model synchronization, workers would move to the next round of training at the same step until all of them obtain the newly global aggregated model. Thus, optimizing the execution efficiency of BSP model synchronization is critical to improve the performance of data-parallel Geo-DML training.

The functional requirements of BSP-based model synchronization can be captured by the collective of AllReduce. Consider that there are n workers holding tensors $t_1, t_2 \cdots, t_n$, respectively. Then, using an AllReduce operation, all workers could obtain the same result of $\mathcal{R}([t_1, \cdots, t_n])$, where $\mathcal{R}$ is the applied reduction operation on each item of these tensors. sum, max, min, and $multiply$ are examples of such operations [23]. In practice, tensor t_i might denote the locally trained model parameters or the gradients, depending on the design of the distributed training algorithms [13]; and sum is widely used to average a group of model parameters or gradients for model synchronization. As an example, Figure 1 showcases the tony example of performing the sum-based AllReduce collective operation for three workers whose local tensors are $(2, 4, 1)$, $(1, 3, 5)$, and $(6, 8, 7)$, respectively. Once the AllReduce is completed, all workers would hold the same summarized tensor $(9, 15, 13)$.



(a) An example shows the definition of AllReduce.



(b) AllReduce can be split into ReduceScatter + AllGather.

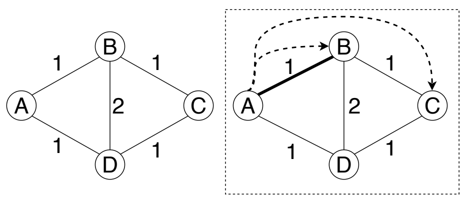
Fig. 1: An example showcases the operation of AllReduce.

C. *Drawbacks of Existing AllReduce Implementations*

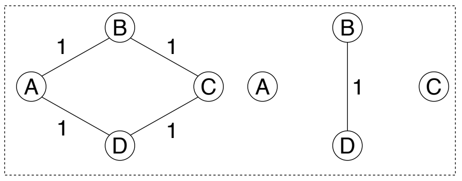
Despite there are abundant AllReduce implementation designs triggering different communication patterns, they could not efficiently use the scarce and heterogeneous bandwidth.

1) Peer-to-peer and PS: In *peer-to-peer* based solutions, each worker directly sends its tensors to all other workers via either unicast or multicast at the network layer, forming a logical broadcast on the overlay among all workers; on receiving the data from all workers (including itself), each worker could compute the aggregated tensors, independently [10]. Different from peer-to-peer based solutions that directly "broadcast" the input data among workers, the "star"-based solutions configure some nodes to work as "aggregators", a.k.a., *parameter servers or PS for short* [4]; workers first push the local tensors to these aggregators and then pull the results. To achieve BSP-based model synchronization, a PS would reply to pull requests until it generates the aggregated result based on all the input tensors. Logically, a parameter server can share the same physical server with a training worker, respecting the configurations set by developers and/or operators. If multiple parameter servers are available, training workers could consistently distribute their aggregation workloads into them for load balance [24].

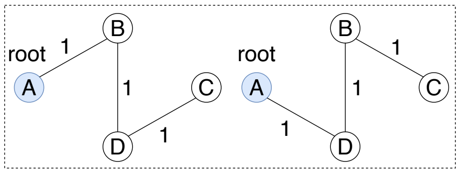
Even though super simple, solutions based on both *peer-to-peer* [10] and PS [4] (including its variants like PSLD [24] and PLINK [25]) are far from optimal for Geo-DML. Fundamentally, they are unaware of the application-level semantics of AllReduce and the heterogeneous topology of the underlying network. Due to the variability of WAN topology, some node sites in Geo-DML are networked via others rather than directly connected; thus peer-to-peer messaging and PS-based pushing & pulling generically introduce abundant duplicated traffic on the wire. Consider the cases shown in Figure 2 as examples. The system has four nodes, labeled A, B, C, and D, respectively. Their inter-node links have a bidirectional capacity of either 1 or 2. When peer-to-peer messaging is employed, A might send the data to B via their direct connection; meanwhile, A also needs to send the same data to node C via B. Thus, there are duplicated data transmissions on the link of (B, C), leading to a waste of bandwidth. In the case



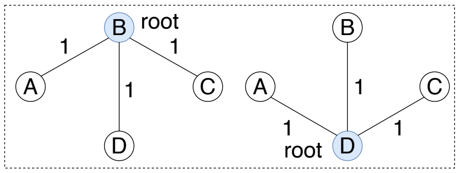
(a) Network (b) Peer-to-peer or PS



(c) Ring



(d) Two spanning trees with the height of 3 hops



(e) Two spanning trees with the height of 1 hop

Fig. 2: Motivation Examples: (a) A heterogenous network; (b) the data sent by A with the destination of B and C might introduce duplicated traffic on the link of (A, B), when peer-to-peer or PS-based schemes are employed; (c) it is impossible to make full use of the link capacities of this heterogeneous network with rings; (d) these two trees rooted at A could consume all the network capacities, however, their structures are unbalanced; (e) these two trees with the height of 1 hop, rooted at B and D, yield the best solution.

of PS-based model synchronization, as a parameter server, A would disseminate the updated model parameters to both B and C; then, a similar situation occurs.

2) Recursive Halving & Doubling and Butterfly: In schemes like *Recursive Halving & Doubling* [11] and *Butterfly* [12], at each step, a worker communicates with only another worker to deliver or exchange data; and the endpoint changes in different steps. Such a type of design suffers from two issues in the context of Geo-DML, thus far from optimal. Firstly, as some workers do not have direct connections, like in the cases of peer-to-peer and PS, there might be duplicated transmission on some links. Secondly, when there are abundant links with heterogeneous capacities in the network (e.g., Figure 2), they cannot fully use all the network bandwidth.

3) Ring and Tree: Inherently, the order of conducting reduction operation for a group of tensors would not impact the final results. Thus, instead of directly exchanging the raw input data, solutions based on *ring* and *tree* employ the design of arranging the logical communication relationships between workers following a specifically directed topology, and performing the reduction operation along the planned way, such that the last worker would exactly hold the final fully reduced tensor, e.g., $(9, 15, 13)$ in the example shown in Figure 1. Following a similar design, this reduced result would be disseminated to all workers over a predefined topology, achieving the goal of AllReduce. By sharding all the input tensors into multiple groups of chunks and conducting their *reduce* then *broadcast* operations concurrently, training workers could make more efficient use of their interconnections. For instance, by letting different chunks be reduced to different workers and then broadcasted to all other workers over the ring at the same time, the implementation of the well-known ring-based AllReduce is internally split into two stages, i.e., ReduceScatter then AllGather, as Figure 1b shows.

Despite being very efficient in homogeneous environments, ring-based AllReduce implementation might perform poorly in Geo-DML. As the example in Figure 2c shows, after establishing a bidirectional ring for all the nodes, there is remaining bandwidth on the link between B and D, implying that ring-based solutions might be unable to make full use of all the heterogeneous link capacities. Indeed, as pointed out by [8], it is even theoretically impossible to find a ring to cover all the nodes exactly once in some cases. Alternatively, a promising solution to this instance is to generate two spanning trees, e.g., Figures 2d and 2e. However, existing solutions [6], [8], [9] have performance issues in executing the communication and/or generating trees.

TopoAdopt has formulated the joint optimization problem of generating trees to assign workloads as a comprehensive mixed-integer nonlinear programming model and designed a heuristic algorithm based on *Simulated Annealing* to solve. However, it does not explore the possibility of splitting tensors into fine-grain chunks for pipelined transmission—Such a design not only suffers from the problem of inefficient bandwidth utilization due to the lack of pipelining [8], [9] but also prevents training workers from overlapping their communication with computation [26], [27]. For example, consider that a generated rooted tree has a height of l, meaning that it takes l hops for a message sent by the farthest worker to reach the root and vice versa. Assume that all links have the latency of α and can transmit $\frac{1}{\beta}$ bytes per second. Without pipelined communication, a tensor with the size of v bytes would be encapsulated as a single large message. Then, it takes about $l(\alpha + \beta v)$ and $2l(\alpha + \beta v)$ seconds, for these workers to complete a reduction/broadcast and AllReduce operation, respectively, even ignoring the time cost of reduction computation. However, by splitting the tensor into fine-gain chunks with the size of μ bytes, using pipelined chunk communication, the time cost of reduction/broadcast and AllReduce could decrease to about $(l+\lambda-1)(\alpha+\beta\mu)$ and $(2l + \lambda - 1)(\alpha + \beta\mu)$, respectively, where $\lambda = \frac{v}{\mu}$.[1]

Both BLINK [8] and PackingTrees [9] support pipelined communication; however they do not limit the height of the generated trees—As a result, trees with uncontrolled structures might be generated. For example, they might find and use the two trees rooted at a with the height of 3 hops for AllReduce as shown in Figure 2d. In comparison, as Figure 2e shows, the two spanning trees with a height of 1 hop, rooted at b and d, respectively, yield a better solution. As Section IV will show, the uncontrolled structures of trees might lead to a non-trivial loss of performance. Moreover, regarding the design of pipelined communication, PackingTrees [9] would split the tensors into very small chunks and distribute their "reduce" and "broadcast" traffic over many diverse trees. This not only makes the management of communication more complex but also suffers from performance loss as using a too-small chunk size also leads to performance loss [8].

[1]We assume there would only be at most one chunk on a link at any time.

D. Design Challenges

Motivated by the above observations, constructing multiple height-optimized spanning trees in a bandwidth-aware way to conduct pipelined AllReduce operations concurrently yields a promising insight to achieve efficient AllReduce communication operations for Geo-DML. However, it is hard to do so, as the following challenges should be addressed properly.

- **Heterogeneous resources**. Firstly, to efficiently use the abundant WAN connections and their skewed bandwidth, the generated trees should be effective and have optimized structures. Due to the computational complexity of the original optimization problem, as reported by recent works [6], [8], novel algorithm designs are needed.
- **Increased complexity**. Secondly, distributing the workloads among multiple trees for pipelined reduction and/or broadcast is definitely more complex than the case of only using a single tree, yielding non-trivial overheads for the implementation and management. To be practical, the proposed execution plan should be easy to implement and manage, with controllable complexity.
- **Pipelining optimization**. Thirdly, pipelined communication could make efficient usage of the bandwidth thus accelerating the execution; however, as pointed out by [8], performance loss occurs when the chunk size is unbefitting. A guideline on the best chunk size for pipelined communication on heterogeneous networks is still missing and theoretical studies are needed.

III. mTree Designs

As Figure 3 shows, mTree acts as an AllReduce synthesizer that could generate AllReduce execution plans for Geo-DML, to efficiently use the skewed heterogeneous WAN connections. With the monitored global view of the inter-datacenter WANs, it abstracts workers and their WAN connections as an undirected graph G and computes the pipelined AllReduce communication scheme at the application layer (i.e., L7). Then, workers would implement the scheme using the supported underlay high-performance transport protocols or communication libraries. When the network condition changes significantly, mTree updates G to reconstruct new schemes for the next round of synchronization.

At the core, the algorithms of mTree need to accomplish three coupled goals: *i*) constructing spanning trees, *ii*) deciding their workload distributions, and *iii*) optimizing the chunk size for each tree's pipelined communication. A straightforward

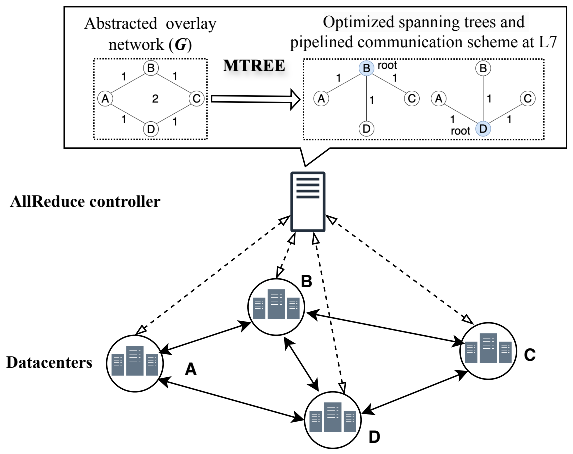

Abstracted overlay network (G)
Optimized spanning trees and pipelined communication scheme at L7
MTREE
root
AllReduce controller
Datacenters
A
B
C
D

Fig. 3: MTREE Architecture.

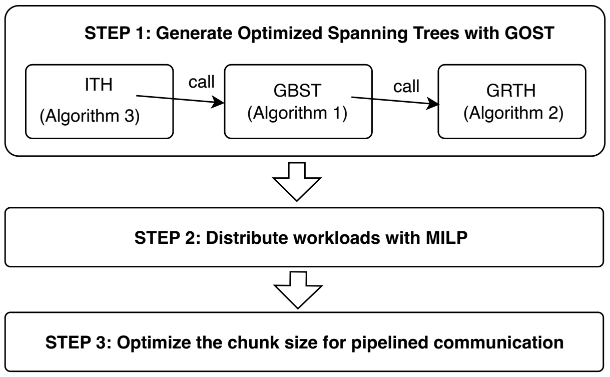

STEP 1: Generate Optimized Spanning Trees with GOST
ITH (Algorithm 3)
call
GBST (Algorithm 1)
call
GRTH (Algorithm 2)
STEP 2: Distribute workloads with MILP
STEP 3: Optimize the chunk size for pipelined communication

Fig. 4: Algorithm Framework of MTREE.

solution is formulating the above tasks as a joint optimization problem to solve. However, due to the huge and complex solution space, a joint optimization method would require a lot of attempts. Indeed, as the work of [28] shows, for a given undirected graph G, it is NP-hard in theory even to find k edge-disjoint spanning trees rooted at the given vertex with the optimal tree heights. Since the task faced by MTREE is more complex, we conjecture it to be NP-hard and design decoupled heuristic solutions. As Figures 4 shows, MTREE decouples these tasks into three stages: *i*) constructing optimized, rooted spanning trees with GOST (§III-A), *ii*) determining their optimal workload allocation scheme with MILP (§III-B), and finally *iii*) computing the optimal chunk size for each tree's pipelined communication (§III-C). During the procedure, we would care about the theoretical throughput these generated trees could obtain. Hereafter, we use the term **AllReduce rate** to refer to the rate of *reduce* and/or *broadcast* workers would conduct over a rooted tree. Accordingly, for an AllReduce operation using multiple trees, its throughput is defined by the total AllReduce rate of all the involved trees.

To address the challenge of *heterogeneous resources*, MTREE employs a novel height-optimized algorithm to generate and select spanning trees in good structures for pipelined communication, ensuring that their maximum height would not exceed the given limit of H (see §III-A). To address the challenge of *increased complexity*, MTREE not only lets the reduce traffic of each tensor go exactly the same but reversed rooted tree with its reduce process, but also allows applications to specify the upper limit of the number of allowed spanning trees (i.e., K) with each request, which would be used for selecting active trees (see Section III-B). And to address the challenge of *pipelining optimization*, MTREE develops a math model to analyze the impacts of the chunk size on the completion time of pipelined communication over trees made up of latency-skewed connections, based on prior study [29], and thus obtain the theoretical optimal value (see §III-C).

A. Generating Optimized Spanning Trees with GOST

Now, we explain the design details of the *Generate Optimized Spanning Trees* (GOST) algorithm, which is made up of GBST (Algorithm 1), GRTH (Algorithm 2), and ITH (Algorithm 3), as Figure 4 sketches. Especially, we find that, for a given network, although there is a trade-off between the achieved total AllReduce rate and the value of allowed weighted height, once the height is larger than a certain value, the increase in tree height only leads to tiny improvements in the achieved total AllReduce rate. Empirically (see Section IV), optimizing the maximum tree heights is likely to bring benefits to network utilizations. This observation motivates us to further improve the actual height of each AllReduce tree rather than directly using the upper bound specified by the application. Taking all the above considerations into account, MTREE first involves a function named GBST (Generate Basic Spanning Trees, Algorithm 1), with the assistance of GRTH (Get Rooted Tree Height, Algorithm 2), to find multiple optimized basic spanning trees under the given constraints of H and $\bar{r}$. Then, it tries to improve the maximum weighted height of constructed trees, with binary search alike techniques, i.e., ITH (Improve Tree Heights, Algorithm 3).

1) GBST: The procedure of GBST is as Algorithm 1 shows. Given a network G along with the constraints of H and $\bar{r}$, it repeats to find trees for concurrent AllReduce, until no more tree that meets the maximum height limit of H and the minimum bandwidth requirement of $\bar{r}$ could be found (Line 14), or the network is disconnected (Line 19). For the construction of a tree, like the design of *Prim's Algorithm* [30], GBST starts from a randomly selected vertex (Line 3); then, at each step, it adds the edge with the maximum possible bandwidth from the tree to another vertex, provided that the addition of this vertex would not violate the constraint of H (Lines 9-13). Here, b_j and b_κ represent the available capacity of the j-th and κ-th edge respectively. In case there are multiple candidate edges with the same available bandwidth, GBST selects the one that would increase the tree height as few as possible (Lines 12-13). Note that, selecting different nodes as the root generally leads to various tree heights; we do not determine the root of the target tree during the construction. Instead, for the already selected vertexes V_s, we use a vector D_j (calculated by *Get Rooted Tree Height*, i.e., GRTH, as Algorithm 2 shows) to record the heights of all possible trees, in which $D_j[i]$, the i-th item, denotes the height of tree constructed from $E_s \cup \{e_j\}$ rooted at $v_i \in V_s$ (Line 11). In the worst case, the maximum value in a D_j would equal

Algorithm 1 GBST: Generate Basic Spanning Trees

```
Input: network $G$, maximum allowed tree height $H$, minimum allowed non-zero rate $\bar{r}$
Output: the set of constructed spanning trees $\mathbb{T}$, workload allocations of spanning trees $\mathbb{R}$
1:  $\mathbb{T} \leftarrow []$; $\mathbb{R} \leftarrow []$; $d \leftarrow 2H$
2:  while True do                                              ▷ repeat to find trees
3:      $V_s \leftarrow \{RandomSelect(V(G))\}$
4:      $E_s \leftarrow \emptyset$                              ▷ edges selected for tree construction
5:      $E_c \leftarrow \{(u,v) \in E(G) : u \in V_s \wedge v \notin V_s \wedge b_{(u,v)} \geq \bar{r}\}$
6:      $r_* \leftarrow +\inf$; $D_* \leftarrow [0]_n$; $M \leftarrow [0]_{(n \times n)}$
7:      while $E_c \neq \emptyset$ do
8:          $\kappa \leftarrow null$; $D \leftarrow null$; $d_\kappa \leftarrow d$
9:          for $e_j \in E_c$ in non-increasing order of $b_j$ do
10:             if $\kappa \neq null \wedge b_j < b_\kappa$ then break
11:             $D_j \leftarrow$ GRTH$(e_j, V_s, M, D_*, \mathcal{L}(G))$
12:             if $\max_i D_j[i] \leq d_\kappa$ then          ▷ select this edge
13:                 $\kappa \leftarrow j$; $D \leftarrow D_j$; $d_\kappa \leftarrow \max_i D_j[i]$
14:         if $\kappa = null$ then return $\mathbb{T}, \mathbb{R}$        ▷ fail
15:         $(u,v) \leftarrow e_\kappa$ where $u \in V_s$ and $v \notin V_s$
16:         $V_s \leftarrow V_s \cup \{v\}$; $M[u][v] \leftarrow 1$; $M[v][u] \leftarrow 1$
17:         $E_s \leftarrow E_s \cup \{e_\kappa\}$; $D_* \leftarrow D$
18:         $E_c \leftarrow \{(u,v) \in E(G) : u \in V_s \wedge v \notin V_s \wedge b_{(u,v)} \geq \bar{r}\}$
19:     if $len(V_s) \neq len(V(G))$ then return $\mathbb{T}, \mathbb{R}$      ▷ fail
20:     $\tau \leftarrow \arg\min_{i \in V_s} D_*[i]$
21:     $T_* \leftarrow$ construct the tree rooted at $\tau$, using $E_s$
22:     $r_* \leftarrow \min\{b_j > 0 : e_j \in E(G)\}$          ▷ globally smallest
23:     $\mathbb{T} \leftarrow \mathbb{T} + [T_*]$; $\mathbb{R} \leftarrow \mathbb{R} + [r_*]$       ▷ append
24:     for $e_j \in E(T_*)$ do
25:         $b_j \leftarrow b_j - r_*$
26: return $\mathbb{T}, \mathbb{R}$
```

the diameter of the tree, while selecting the vertex closest to the center of the diameter path would yield the minimum tree height. Thus, when selecting an edge to extend the spanning tree, MTREE guarantees that the values in D_j would not exceed $d = 2H$ (Line 12). Once all vertexes are appended, with these selected edges, GBST builds the AllReduce tree T_* by selecting the vertex yielding the minimum tree height as the root (Line 20). Finally, GBST appends the generated tree T_* along with its allowed r_* to $\mathbb{T}$ and $\mathbb{R}$, respectively (Line 23), and updates the available capacity of each link (Line 25). Notice that r_* is the global minimum bandwidth, which would be further explained in III-B.

2) GRTH: The sub-procedure of GRTH is described in Algorithm 2. Using it, GBST tries to find the link that makes the height (quantized latency) of the spanning tree increase the least as much as possible when constructing the spanning tree (i.e., finding the e_j in Algorithm 1). Algorithm 2 is based on Breadth First Search (BFS). Algorithm 2 first puts the vertex u and the length/distance (i.e., quantized latency) of e_j into the queue (Line 4), and then iteratively pops out the vertex (p) and the t from the front of the queue, where t represents the quantized distance between vertex p and vertex v. Calculate the quantized distance from vertex *next* which is adjacent to

Algorithm 2 GRTH: Get Rooted Tree Height

```
Input: candidate edge $e_j$, processed nodes $V_s$, adjacency matrix of spanning tree $M$, vector $D_*$, link latency $L$
Output: $D_*$ updated by $e_j$
1:  $(u,v) \leftarrow e_j$ where $u \in V_s$ and $v \notin V_s$
2:  $D_j \leftarrow D_*.copy()$
3:  $q \leftarrow Queue()$; $visited \leftarrow [0]_n$; $\ell \leftarrow -\inf$
4:  $q.push(\langle u, L_{v,u} \rangle)$; $visited[u] \leftarrow 1$
5:  while $q$ is not empty do
6:      $\langle p, t \rangle \leftarrow q.pop()$; $\ell \leftarrow \max(\ell, t)$
7:      for $next \in \{i \in V_s : visited[i] = 0 \wedge M[p][i] = 1\}$ do
8:          $D_j[next] \leftarrow \max(D_j[next], t + L_{p,next})$
9:          $q.push([next, t + L_{p,next}])$
10:         $visited[next] \leftarrow 1$
11: $D_j[v] \leftarrow \ell$
12: return $D_j$
```

p to vertex v, and if it is larger at this time, update $D_j[next]$ (Line 8), then put the vertex *next* and the corresponding distance into the end of the queue (Line 9). Throughout the whole iteration, ℓ records the maximum distance generated, and ℓ is assigned to $D_j[v]$ at the end of the iteration (Line 11).

Algorithm 3 ITH: Improve Tree Heights

```
Input: network $G$, maximum allowed (weighted) tree height $H$, minimum allowed non-zero rate $\bar{r}$, allowed loss rate $\rho$
Output: optimized maximum allowed weighted tree height $H^U$, the set of corresponding spanning trees $\mathbb{T}^*$, and workload allocation of spanning trees $\mathbb{R}^*$
1:  $(\mathbb{T}^*, \mathbb{R}^*) \leftarrow$ GBST$(G, H, \bar{r})$
2:  $r_\rho \leftarrow \rho \sum \mathbb{R}^*$            ▷ the threshold of acceptable total rate
3:  $H^L \leftarrow 1$; $H^U \leftarrow H$
4:  while $H^L < H^U$ do                                ▷ binary search
5:      $H^M \leftarrow \left\lfloor \frac{H^L + H^U}{2} \right\rfloor$
6:      $(\mathbb{T}, \mathbb{R}) \leftarrow$ GBST$(G, H^M, \bar{r})$
7:      if $\sum \mathbb{R} \geq r_\rho$ then
8:          $H^U \leftarrow H^M$; $\mathbb{T}^* \leftarrow \mathbb{T}$; $\mathbb{R}^* \leftarrow \mathbb{R}$
9:      else
10:         $H^L \leftarrow H^M + 1$
11: return $H^U, \mathbb{T}^*, \mathbb{R}^*$
```

3) ITH: Despite that the heights of generated spanning trees are always smaller than or equal to the limit of H, there is still room for optimization. By using GBST as the building block, MTREE achieves this goal via a binary search alike design as Algorithm 3 shows. Basically, ITH first computes the maximum total AllReduce rate that MTREE could achieve under the original limit of H (Line 1). This gives a baseline for the improvement of H. Then, it tries to find a minimum $H^M \leq H$ that would not reduce the total AllReduce rate too much, i.e., a H^M is selected if and only if its achieved total AllReduce rate is within the ρ of the baseline (Line 7). Here, $\rho \in (0, 1]$ is a tunable parameter controlling the loss of aggregated AllReduce rate. Finally, the best H^M is obtained.

4) Complexity: Regarding the algorithm complexity, both the maintenance of the order of b_j in E_c and the calculation of the value of D_j involved in Algorithm 1 can be processed incrementally thus their completion times can be amortized. Here, we conduct a loose worst-case analysis. It would take no more than $O(m + m^2 + mn^2)) = O(\max(m^2, n^2 m))$ times for GBST to build a tree upon a network made up of n vertexes and m edges. As GBST would find at most K trees and ITH would obtain the results within $\log_2^H$ attempts, the worst-case time complexity of MTREE is no more than $O(\ln H \cdot K \cdot \max(m^2, n^2 m))$.

B. Distributing Workloads with MILP

With GOST, MTREE would obtain a set of optimized spanning trees $\mathbb{T}^*$ along with a corresponding bandwidth allocation plan $\mathbb{R}^*$. We find that, instead of directly employing all trees in $\mathbb{T}^*$ and the sending rates given by $\mathbb{R}^*$ for pipelined current AllReduce, there is room for improvement; thus we explore the best workload allocation ratios by formulating the problem as a mixed-integer linear programming (MILP) and solving it using off-the-shelf solvers like Gurobi.

Consider that there are γ trees in $\mathbb{T}^*$, i.e., $T_1 \cdots, T_\gamma$. Let binary constant $a_{i,j}$ indicate whether the spanning tree T_i involves the edge of e_j or not; use binary variable y_i to imply whether T_i is selected for AllReduce or not; and use variable r_i to denote the AllReduce rate on it. Then the problem of selecting at most K trees from $\mathbb{T}^*$ and balancing the workload among them to maximize the aggregated AllReduce rate/throughput), can be expressed as the MILP of (1). Here, M is a large positive constant number used to guarantee that, if the tree T_i is selected (i.e., $y_i = 1$), then we have $r_i \geq \bar{r}$ (refer to (1c)), otherwise (i.e., $y_i = 0$), $r_i = 0$ (refer to (1d)).

$$\textbf{MILP} \qquad \textbf{Maximize} \sum_{i=1}^{\gamma} r_i \tag{1a}$$

$$\textbf{s.t.,} \qquad \forall e_j \in E : \sum_{i=1}^{\gamma} a_{ij} r_i \leq b_j \tag{1b}$$

$$\forall i : M(1 - y_i) + r_i \geq \bar{r} \tag{1c}$$

$$\forall i : r_i \leq M y_i \tag{1d}$$

$$\sum_{i=1}^{\gamma} y_i \leq K \tag{1e}$$

$$\forall i : y_i \in \{0, 1\}; r_i \geq 0 \tag{1f}$$

Following the results of MILP (1), MTREE distributes the workload to selected trees (whose $y_i = 1$) in proportion to their r_i values for concurrent AllReduce. Suppose that MTREE has constructed k trees $T_1, T_2, \cdots, T_k$, whose scheduled sending rates are $r_1, r_2, \cdots, r_k$, respectively. Then, for a model with the size of S, the i-th tree T_i could be responsible for the AllReduce operation of $\frac{r_i}{\sum_{j=1}^{k} r_j}$ of the tensors.

Despite that MILP models are theoretically time-consuming to solve, MTREE directly uses off-the-shelf commercial available solvers (e.g., Gurobi [31]) to obtain the optimal selections, due to the following reasons. As it generally takes hours and even days for workers to complete a training task, thus, once the execution plan of AllReduce has been determined, it could be used for a long time. Accordingly, the time cost of solving

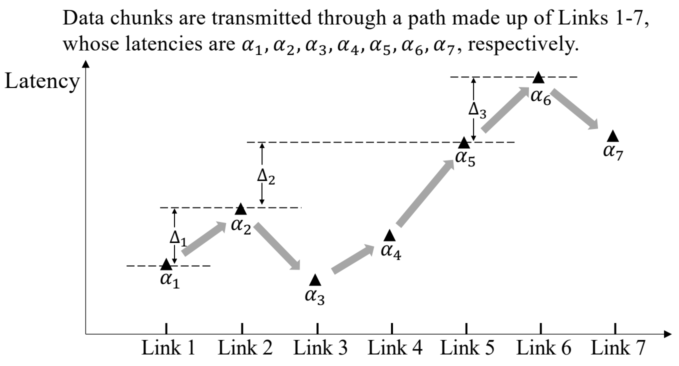


Fig. 5: An example shows the impacts of various link latencies.

MILPs is still acceptable, if it is not too much. Indeed, our tests show that, by using the Gurob as the solver, it takes only a few seconds for MTREE to select AllReduce trees and determine their sending rates for the well-known WAN topologies like GScale [32], Equinix [33], and IDN [34]. When the scale of the MILP model is quite huge, an efficient approximation or heuristic algorithm is needed and we leave the design of such algorithms as future work.

C. Optimizing Chunk Sizes for Pipelining

So far, we have described how MTREE generates and selects spanning trees to maximize the total AllReduce throughput. Now, we further explain the principle that MTREE uses to determine the optimal chunk size for each AllReduce tree.

Given an AllReduce tree used by MTREE, all its links have the same amount of bandwidth but various latencies. As a result, the communication time that it takes to complete the parameter aggregation over a tree is dominated by the longest path from all leaf nodes to the root. So does the broadcast of results. For pipelined AllReduce whose broadcast stage overlaps with the reduce stage, a similar conclusion is still held, except that the longest path is a round trip. Mainly, we extend the optimal chunk size model explained in [29] to support various network latency values thus determining the optimal chunk size for MTREE.

Support that the longest path involved in the communication is made up of l links, whose latencies are $\alpha_1, \alpha_2, \cdots, \alpha_l$, respectively. All these links would send data belonging to the task at the same rate, saying b for instance. Then, following the well-known $\alpha-\beta$ model [35], the time cost of delivering a chunk with the size of μ over the i-th link can be estimated as $\alpha_i + \beta\mu$, where $\beta = \frac{1}{b}$. That is to say, the endpoints of this link can either send or receive a chunk every $\alpha_i + \beta\mu$ seconds. Note that, links might have various latencies. When two successive data chunks are transferred from a low-latency link to a high-latency link, their inter-chunk arrival latency would increase. However, when they are transferred from a high-latency link to a low-latency link, the inter-chunk arrival latency remains unchanged. For instance, Figure 5 shows a path made up of 7 links, with latencies $\alpha_1, \alpha_2, \ldots, \alpha_7$, respectively. Here, the X-axis represents the link indexes, the Y-axis denotes the latency of each link, and the arrows demonstrate the order of links that data chunks would go through. In the case of pipelined delivery, when the data chunks pass through a link with a latency of α_1, the inter-chunk arrival latency is $\alpha_1 + \beta\mu$. Then, when the data chunks pass through the link with the latency of α_2 ($\alpha_2 > \alpha_1$), the inter-chunk arrival latency increases to $\alpha_2 + \beta\mu$, with an increment of $\Delta_1 = \alpha_2 - \alpha_1$. When these data chunks pass through the 3rd and 4th links, their inter-chunk arrival latency is still $\alpha_2 + \beta\mu$, but the transmission times of each data chunk on these two links are $\alpha_3 + \beta\mu$ and $\alpha_4 + \beta\mu$, respectively. Following this, let $\Delta_2 = \alpha_5 - \alpha_2$, and $\Delta_3 = \alpha_6 - \alpha_5$; then, it can be concluded that, after passing through all links, the accumulated inter-chunk arrival latency would increase from α_1 to $\alpha_1 + \Delta_1 + \Delta_2 + \Delta_3 = \max_{i=1}^{l} \alpha_i$, due to the change of link latency.

Assume that there are λ data chunks for pipelined communication in total. Obviously, workers can deliver a chunk at most every $(\max_{i=1}^{l} \alpha_i + \beta\mu)$ seconds on each inter-worker connection; and the first data chunk would take about $\sum_{i=1}^{l}(\alpha_i + \beta\mu)$ seconds to reach the final destination. Based on the above analysis, the time cost of delivering λ data chunks with the size of μ over the path can be estimated as follows:

$$\begin{aligned} t(\lambda, \mu) &= \sum_{i=1}^{l} (\alpha_i + \beta\mu) + (\lambda - 1)(\max_{i=1}^{l} \alpha_i + \beta\mu) \\ &= \sum_{i=1}^{l} \alpha_i + (\lambda - 1) \max_{i=1}^{l} \alpha_i + (l-1)\beta\mu + \beta\lambda\mu \end{aligned} \tag{2}$$

Given a communication task with the data size of v on each node, we would have $\lambda = \frac{v}{\mu}$ and the following formulation:

$$\begin{aligned} t(\mu) &= t(\frac{v}{\mu}, \mu) \\ &= \sum_{i=1}^{l} \alpha_i + (\frac{v}{\mu} - 1) \max_{i=1}^{l} \alpha_i + (l-1)\beta\mu + \beta v \end{aligned} \tag{3}$$

Or alternatively,

$$\begin{aligned} t(\lambda) &= t(\lambda, \frac{v}{\lambda}) \\ &= \sum_{i=1}^{l} \alpha_i + (\lambda - 1) \max_{i=1}^{l} \alpha_i + (l-1)\beta\frac{v}{\lambda} + \beta v \end{aligned} \tag{4}$$

And its derivative is

$$\frac{\mathrm{d}t(\mu)}{\mathrm{d}\mu} = -\frac{v}{u^2} \max_{i=1}^{l} \alpha_i + (l-1)\beta \tag{5}$$

Obviously, the chunk size would not be larger than v; and with the chunk size (μ) decreasing, the value of $t(\mu)$ would decrease first and reaches the minimum value when

$$\frac{\mathrm{d}t(\mu)}{\mathrm{d}\mu} = 0 \tag{6}$$

which yields

$$\mu^* = \arg\min t(\mu) = \sqrt{\frac{\max_{i=1}^{l} \alpha_i}{(l-1)\beta} v} \tag{7}$$

In other words, MTREE could obtain the minimum pipelined communication time by splitting the data on each worker into $\lambda^* = \frac{v}{\mu^*}$ chunks.

$$\lambda^* = \sqrt{\frac{(l-1)\beta}{\max_{i=1}^{l} \alpha_i} v} \tag{8}$$

Once μ is smaller than μ^* (i.e., $\lambda > \lambda^*$), $t(\mu)$ turns to increase. By solving $t(\mu) > t(v)$, we get $\mu < \mu^\# = \frac{\max_{i=1}^{l} \alpha_i}{(l-1)\beta v}$. That is to say when the chunk size is too small, e.g., smaller than $\mu^\#$ (i.e., $\lambda > \lambda^\# = \frac{v}{\mu^\#} = \frac{(l-1)\beta v}{\max_{i=1}^{l} \alpha_i}$), the completion time might even be larger than the case without using pipelined communication.

IV. Performance Evaluation

In this section, we quantify the performance of MTREE by employing it to construct concurrent pipelined AllReduce trees for three real-world inter-DC network topologies GScale [32], Equinix [33], and IDN [34]. Results show that compared with TOPOADOPT [6] and BLINK [8], MTREE can obtain much higher throughput (i.e., normalized total AllReduce rate) for all topologies under various network link capacities, respecting the requirements of both the allowed number of trees and their maximum height. Meanwhile, the spanning trees constructed by MTREE achieve a good balance in the average height and the average maximum fanout; and MTREE can significantly reduce the *communication completion time*.

A. Methodology

1) Baselines: To the best of our knowledge, currently, there are few direct studies on model synchronization between data centers under heterogeneous WAN. From the perspective of network heterogeneity, parameter division, and multi-aggregators, we select four parameter aggregation algorithms, namely BLINK [8], PLINK [25], PSLD [24], and TOPOADOPT [6], as baselines. Basically, BLINK, TOPOADOPT, and MTREE belong to the spanning tree algorithm,[2] PSLD belongs to the multi-aggregator algorithm, and PLINK belongs to the 2-level hierarchical aggregation algorithm (intra-group aggregation and inter-group aggregation). In PSLD and PLINK, it is necessary to measure the communication performance between nodes. Since the experiment scenario is a wide-area network of arbitrary topology, many nodes are not directly connected, so the experiment calculates the shortest path between nodes with the weight of latency, and the average bandwidth on the path is used as the communication bandwidth between nodes. Some algorithms, such as BLINK, TOPOADOPT, and MTREE, involve the solving of (mixed-integer) linear programming models. We employ the off-the-shelf solver of Gurobi [31] to solve.

2) Workloads: Regarding the network between workers, we use the three well-known real-world inter-DC topologies, namely GScale [32], Equinix [33], and IDN [34], involving 12, 20, and 40 nodes, together with 38, 282, and 462 edges, respectively, for performance studies. Bandwidth and latency between data centers are often affected by geographic distance [36], [37]. In our experiments, the available link bandwidth of a link is an integer multiple of ten randomly taken from $[200 - 100sk, 200 + 100sk]$, with the unit of Mbps.

[2]In the scenario of this paper, the work of PackingTrees [9] shares a very similar high-level design with BLINK. However, given an AllReduce task, it might use a large number of asymmetric trees for the involved reduce and broadcast, making the execution of AllReduce hard to manage and introducing non-trivial overheads. For a fair comparison, we do not use it as a baseline.

Here, sk ($skewness$) $\in [0, 1]$ is a configurable parameter controlling the degree of link bandwidth fluctuation. By modifying the value of sk, we can test the stability of the algorithms under different bandwidth fluctuations. Regarding the network latency, following the measured data reported in [37], each inter-DC WAN connection's latency is chosen from the range of $[50, 400]$, inversely proportional to its available bandwidth, with the unit of ms. To highlight the impact of H on the generation of trees, we directly set it with a determined value.

3) Metrics: Given a network, for the trees generated by BLINK, TopoAdopt, and our proposed scheme mTree, we mainly use the following metrics to assess their performances.

- **Throughput**, i.e., the total AllReduce rate they can achieve, normalized by the value of $\frac{\sum_{e_j \in E(G)} b_j}{|V(G)|-1}$. Here, b_j is the bidirectional bandwidth of link e_j and $|V(G)|$ is the total number of vertexes. Such a value also represents the average network utilization achieved by these trees.
- **AvgMaxFanout**, the average maximum fanout of generated trees.
- **AvgTreeHeight**, the average height of generated trees.
- **Number of Trees**, the number of generated trees.

Besides, in the comparison of all schemes beyond BLINK and TopoAdopt, we also look into the *communication completion time (CCT)* of the AllReduce operation of a large vector, in seconds, under the schedule of each algorithm. For each parameter setting, we conduct at least 20 trials to calculate their average values.

4) Tools: To measure their detailed performances, we have implemented all the above schemes in Python 3 and further developed a discrete event simulator, which can accurately simulate data transmission, forwarding, and other functions under the schedule of various algorithms. In data transmission on an inter-worker link, if multiple tasks compete for the bandwidth, the default max-min fairness (i.e., FS) allocation mode is adopted, i.e., concurrent tasks share the link capacities equally. Finally, we carry out experiments to evaluate the comprehensive performance of mTree from the perspective of network bandwidth fluctuation, the height and the number of spanning trees, and algorithm running time, to verify the practicality of mTree. All experiments were conducted on a 64-bit Ubuntu 20.04 server equipped with one Intel(R) Core(TM) i7-8700 CPU and 2×16GB DDR4 RAM cards.

B. Throughput of Generated Trees

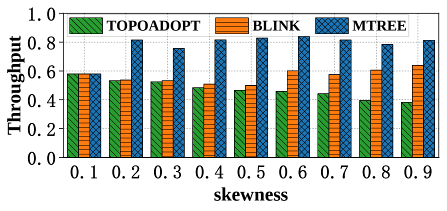
First, we compare TopoAdopt, BLINK, and mTree in terms of generated trees' throughput (i.e., bandwidth utilization), AvgMaxFanout, and AvgTreeHeight. Figure 6 shows that mTree achieves efficient bandwidth utilization in all topologies. The highest normalized throughput is larger than 0.8, indicating that mTree could use nearly 80% of the bandwidth in heterogeneous networks. Moreover, with the increase of network heterogeneity, the normalized throughput achieved by mTree does not decrease significantly, implying that it can effectively deal with heterogeneous networks. In contrast, with the increase of network heterogeneity, TopoAdopt is unable to utilize heterogeneous bandwidth effectively, thus its achieved throughput decreases significantly. The normalized throughput achieved by TopoAdopt in the topology of GScale (Figure 6a) is almost 2-3× of that in the topology of Equinix (Figure 6b) and IDN (Figure 6c). Such results imply that the performance of TopoAdopt is unstable and would be heavily impacted by the structure of the inter-worker network, due to its simulated annealing [6] based random algorithm designs. Both Equinix and IDN involve many nodes and links, greatly decreasing the probability that TopoAdopt could successfully construct optimized spanning trees. As a result, available solutions are scarce, implying that TopoAdopt cannot deal with very complex network scenarios. As shown in Figures 6a and 6b, despite that BLINK achieves higher throughput than TopoAdopt, it still significantly underperforms mTree. As the degree of network heterogeneity increases, there is a certain increase in the achieved throughput of BLINK. However, as shown in Figure 6c, due to the excessive complexity of IDN, the improvement of BLINK is slight, indicating that it is ineffective in dealing with complex network heterogeneity. In short, results show that mTree has obvious advantages in bandwidth resource utilization and the ability to deal with network heterogeneity.

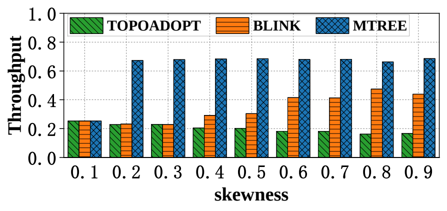
C. Structure of Generated Trees

Figure 7 shows the average heights and maximum fanouts of the spanning trees constructed by BLINK, TopoAdopt, and mTree, respectively. In tests, both TopoAdopt and BLINK do not limit the maximum height of the generated tree. Instead, mTree limits the maximum height of the tree with H; without breaking this bound, it searches for as many available edges as possible, and ensures that the overall height of the tree changes least, to make trees balanced. Results show that the structure of the generated tree varies with the tested network topology. Generally, the average tree heights generated by mTree are smaller than those of TopoAdopt, and smaller than (IDN) or almost equal to (GScale, Equinix) those of BLINK. Regarding the average value of these trees' maximum fanouts, for the smallest topology of GScale, the gaps between their achieved average maximum fanouts generally do not exceed one, except when the skewness of link capacities is small ($sk = 0.1$). And for the larger topologies of Equinix and IDN, when $sk \geq 0.2$, mTree achieves the smallest average maximum fanout. Such a result implies that compared with baselines, the trees generated by mTree are generally more balanced, with relatively small average heights and average maximum fanouts.

D. Communication Completion Time

To explore the benefits of pipeline transmissions, we have developed a message-level simulator with Python 3 that could simulate the instances of conducting Reduce, Broadcast, and AllReduce operations for a group of 1 GB vectors with different designs in detail. Considering that the aggregation and broadcast are symmetric processes and could be pipelined as well, we just show the results of the process of data aggregation. For AllReduce operation, we would obtain similar results, except that the longest path is with the length of $2H$ rather than H. For each inter-worker connection, we use the well-known Hockney (a.k.a., α–β) messaging model [35] to


(a) GScale


(b) Equinix

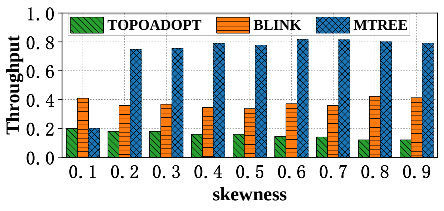
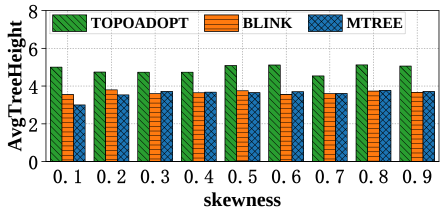
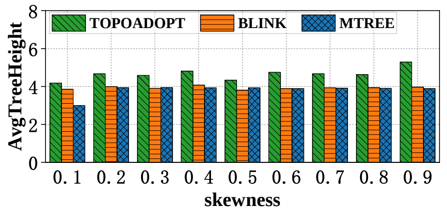
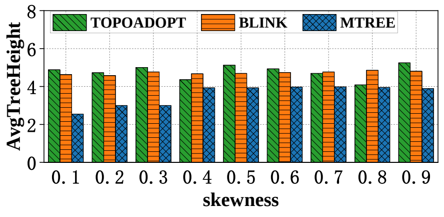
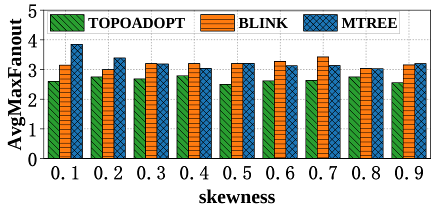

(c) IDN

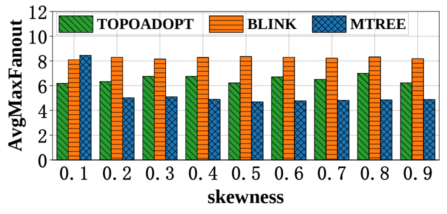
Fig. 6: MTREE generally achieves higher throughput (i.e., normalized average AllReduce rate) than BLINK and TOPOADOPT.


(a) AvgTreeHeight, GScale


(b) AvgTreeHeight, Equinix


(c) AvgTreeHeight, IDN


(d) AvgMaxFanout, GScale


(e) AvgMaxFanout, Equinix

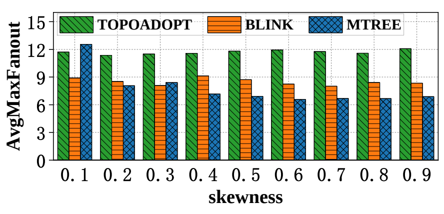
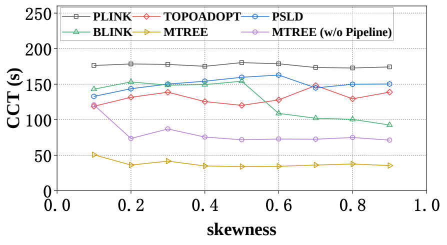
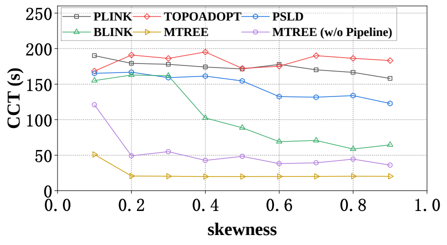

(f) AvgMaxFanout, IDN

Fig. 7: Results show that the structure of the generated tree varies with the tested network topology.


(a) GScale


(b) Equinix

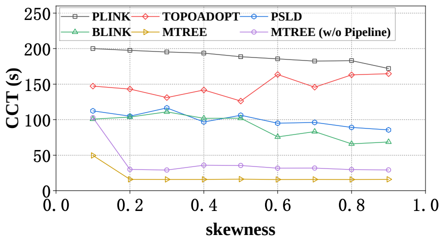

(c) IDN

Fig. 8: MTREE efficiently utilizes the heterogeneous network resources to accelerate AllReduce operations with multiple trees.

formulate the time cost of message delivery. Regarding the latency of each connection, we set it inversely proportional to its link bandwidth following [36] and re-scale them into the range of $[50, 400]$ according to the measurement of [37]. For all baseline methods including MTREE (w/o Pipeline), which is the variant of MTREE without pipelining, training workers encapsulate all the data into a single message. While for MTREE, the data is split into fine-grained chunks to support pipelined transmissions following Eq. (2).

As Figure 8 shows, using height-limited spanning trees and pipelined communication, MTREE achieves the best performance, significantly outperforming all baseline schemes that do not enable pipelined communication, i.e., BLINK, PLINK, TOPOADOPT, and PSLD, about $2.0 - 12.4\times$ speedups on executing AllReduce. The speedup of scheme A over scheme B is defined as $\frac{t_B}{t_A}$, where t_A and t_A are the completion of executing an AllReduce under the schedule of schemes A and B, respectively. Moreover, consistent with the results of achieved throughput shown in Figure 6, when pipelined communication is not employed, MTREE (w/o pipeline) still generally achieves smaller AllReduce communication completion times than TOPOADOPT and BLINK, yielding up to about $6.0\times$ and $3.8\times$ speedups, respectively. Such a result implies that even enabling the same pipelined communication settings for the trees generated by BLINK and TOPOADOPT, they are still likely to underperform MTREE. Overall, all the above results confirm the benefits of conducting pipelined communication over multiple balanced spanning trees for the acceleration of AllReduce.

E. Impacts of Chunk Size

Given a tree and data size v, the best chunk size (μ^*) and the best chunk amount (λ^*) for pipelined communication can

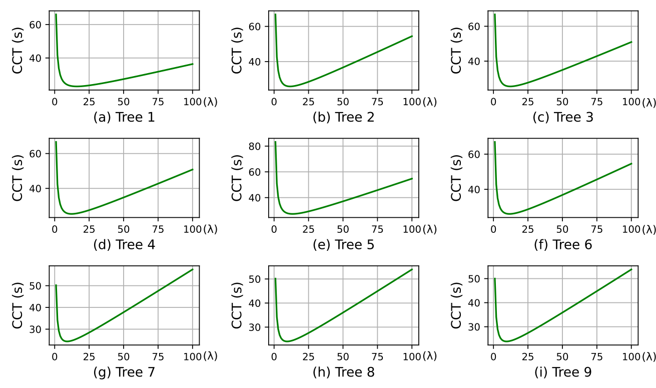


Fig. 9: The changes of CCT with the increase of λ for the 9 trees that MTREE generates to transmit data in pipeline over an instance of Equinix ($sk = 0.7$).

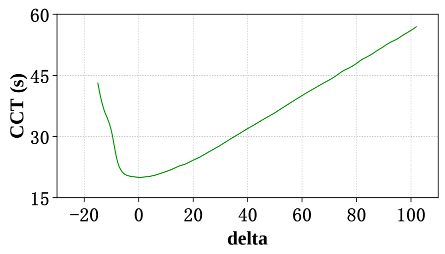


Fig. 10: The changes of CCT with the increase of $delta$ for pipelined communication of an instance of Equinix ($sk = 0.7$).

be computed via Eq. (7) and (8), respectively. Now we look into the impact of chunk size on communication time. Here, we use the instance of performing pipelined reduce operations over the Equinix network topology, by using trees generated by MTREE as examples. By default, the skewness parameter sk is set to 0.7. As shown in Figure 9, MTREE would construct 9 spanning trees in total, and the workloads are distributed among these trees respecting their allocated sending rates. These trees have various λ^* values to achieve the minimum communication time.

To verify the accuracy of the theoretical analysis in §III-C, we attempt to add an offset, denoted as $delta \in [-50, 100]$, to the λ^* of each tree and thus obtain $\lambda^*(delta) = \max(\lambda^* + delta, 1)$. As Figure 10 shows, when $delta = 0$, the communication time reaches a minimum of about 20s, consistent with the communication time of MTREE (with Pipeline), i.e., yellow line, in the same scenario in Figure 8. When $delta$ (based on a value of 0) increases or decreases, the communication computation time increases, and the magnitude of the increase is consistent with the theoretical magnitude in Figure 9. When $delta$ is a large negative value, then $\lambda_i^*(delta) = 1$, and all spanning trees do not divide its parameters—At this time, the pipelined communication degrades into non-pipelined communication, which takes about 43s, consistent with the communication time of MTREE, i.e., purple line, in the same scenario in Figure 8. The above results corroborate the theoretical analysis of communication time in §III-C and show that reasonable use of pipelined design in MTREE can further accelerate the completion of operations like reduce; so does broadcast and AllReduce operations.

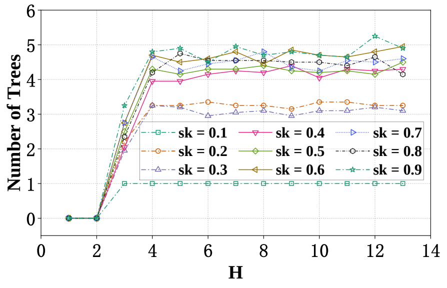
F. Impacts of H

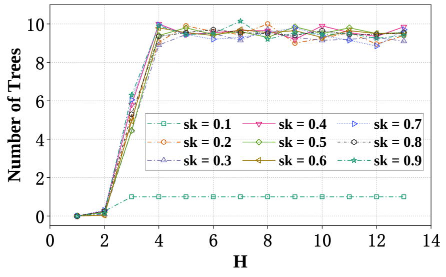
Now, we study the effect of the parameter H on the achieved throughput. In this case, to eliminate the effects of K, we set the value of K to infinity, such that MTREE is allowed to generate as many spanning trees as possible. As shown in Figure 11, at first, GOST (i.e., MTREE) can construct more trees as the maximum weighted height restriction H is relaxed, but the number of trees stops increasing after a certain value. This is because GOST's idea that prioritizes high bandwidth allows a smaller number of spanning trees to take up most of the bandwidth, and after the first 10 trees (or less, e.g., GScale has only 5-6 trees) are constructed, the network does not have any spare links left for the construction of new trees (i.e., the network becomes no longer connected). Thus, overall, even without the restriction of K, MTREE generally ensures that fewer trees are obtained, simplifying the management complexity of AllReduce trees.

Figure 12 further shows the variation of the throughput achieved by MTREE with the growth of H on the three topologies when the value of K is set to infinity. Basically, the throughput increases rapidly with increasing H at first, which can be seen in conjunction with Figure 11. This is mainly because increasing H allows GOST to construct more spanning trees. However, once H reaches a certain value, e.g., 4, 5, and 6 for GScale, Equinix, and IDN (Figure 12), the increase in the realized throughput is small after that. This is mainly because there is almost no increase in the number of spanning trees (corresponds to Figure 11). The above results imply that MTREE usually has an optimal H in each topology, which is small; and at the same time, under the constraints of this optimal H, MTREE can utilize as much bandwidth resources as possible by constructing a smaller number of spanning trees, resulting in higher AllReduce performance. Such observation shows the significance of designing ITH (Algorithm 3). Using ITH, an optimal H can be found quickly to achieve efficient AllReduce performance.

G. Impacts of K

We next explore the effects of the parameter K on the achieved throughput. Note that, both a topology's structure and state of link capacities might impact the best value of H for MTREE. Thus, when looking into the impacts of K, it is unsuitable to set H with a fixed value for all instances. To deal with this issue, when investigating the impact of K on the performance of AllReduce, given a test instance (e.g., Equinix with sk = 0.7), its H is set to the optimal H outputted by ITH by setting K to infinity. As shown in Figure 13, as K increases, MTREE can get more spanning trees, achieving higher throughput. In practical applications, a smaller K can simplify the management of parallel pipelined AllReduce operations. Figure 13 shows that MTREE has utilized 70% to 85% of the bandwidth resources in the network with K not exceeding 10. Thus, it implies that MTREE can achieve efficient AllReduce performance with a smaller number of


(a) GScale


(b) Equinix

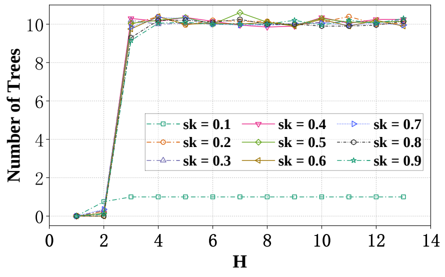
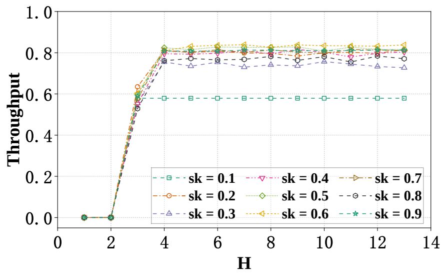
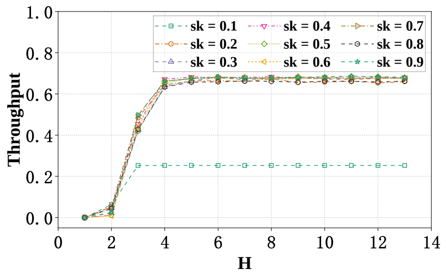

(c) IDN

Fig. 11: mTree generally ensures that a small number of trees are used even without the restriction of K.


(a) GScale


(b) Equinix

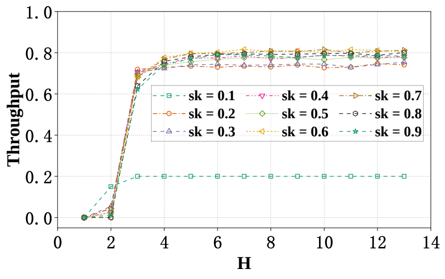
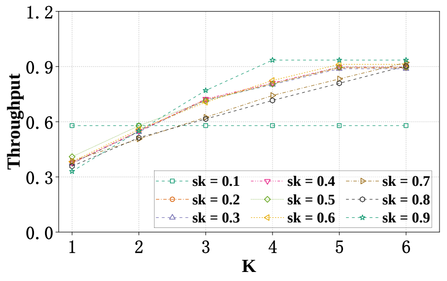
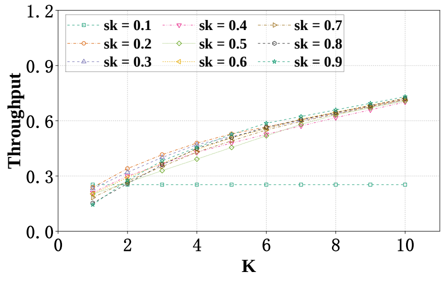

(c) IDN

Fig. 12: There is generally an optimal H for mTree to achieve high throughput while keeping the number of trees small.


(a) GScale


(b) Equinix

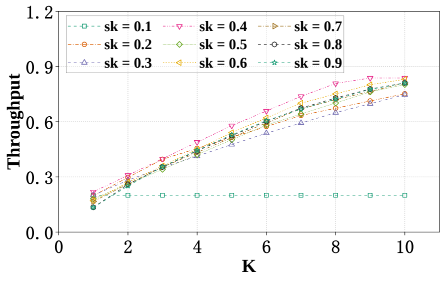

(c) IDN

Fig. 13: mTree can utilize 70% to 85% of the link capacities with K not exceeding 10 in the test instances.

spanning trees. Together with the results of Figures 11 and 12, it is illustrated that there does exist a small optimal value for H—Using this H, mTree could construct a smaller number of spanning trees to achieve efficient AllReduce performance.

H. *Efficiency of* mTree *Algorithms*

Given that GOST is a polynomial time algorithm and the optimal chunk size is also easy to compute, the most time-consuming part involved in mTree is solving the MILP model. We now further study the running time of solving MILP (1) with commercial off-the-shelf Gurobi optimization solver [31]. Results show that in the cases of GScale and Equinix, the involved MILP model of (1) can be solved in a very short time; even in the case of IDN, it takes less than 1s for the solver to select trees and determine their sending rates. Distributed training generally needs to iterate thousands of rounds, and even more, to converge; thus, we can pre-compute the spanning trees and schedule their transmissions in advance and such a time cost is acceptable.

V. Conclusion and Future Work

This paper explores the idea of achieving efficient inter-datacenter AllReduce operations for synchronous Geo-DML via multiple trees. To do so, we design a topology management suite named mTree. With novel heuristic designs, mTree could generate multiple pipelined AllReduce trees together with the suggested workload distribution proportions and chunk size settings to make efficient usage of the heterogeneous WAN connections, respecting the limits of both the maximum number of trees and their maximum weighted height specified by training tasks. Performance studies on real-world inter-datacenter topologies GScale, Equinix, and IDN imply that mTree outperforms existing solutions significantly.

Nowadays, beyond model training, geo-distributed datacenters have also been widely employed to provide a variety of cloud services like data storage and virtual network functions under various operational goals [38]–[42]. When multiple services coexist, they might dynamically compete for the available capacities of the shared inter-DC wan connections.

Accordingly, designing solutions to achieve efficient AllReduce operations when coexisting with other geo-distributed applications is interesting. Moreover, besides powerful cloud datacenters, resource-limited edge nodes also become available as popular AI infrastructures for model training [43]–[45] and inference [46], [47]. For training workloads, besides the nodes holding the training data, we can also select some other available edge nodes or cloud servers to act as in-network processing nodes to accelerate the involved AllReduce operations [9], [44]. Another possible future work is designing schemes to achieve efficient AllReduce in such cases.

References

[1] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proceedings of the 14th NSDI*. USENIX Association, Mar. 2017, pp. 629–647.

[2] P. Zhou, Q. Lin, D. Loghin, B. C. Ooi, Y. Wu, and H. Yu, "Communication-efficient decentralized machine learning over heterogeneous networks," in *Proceedings of the 37th ICDE*, 2021, pp. 384–395.

[3] L. Luo, Y. Zhang, Q. Jin, H. Yu, G. Sun, and S. Luo, "Fast synchronization of model updates for collaborative learning in micro-clouds," in *Proceedings of the IEEE HPCC*, 2021, pp. 831–836.

[4] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th OSDI*, Oct. 2014, pp. 583–598.

[5] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," *Journal of Parallel and Distributed Computing*, vol. 69, no. 2, pp. 117–124, 2009.

[6] L. Luo, S. Yang, W. Feng, H. Yu, G. Sun, and B. Lei, "Optimizing communication topology for collaborative learning across datacenters," in *Proceedings of the 1st International Conference on Emerging Networking Architecture and Technologies, ICENAT*, 2023, pp. 184–197, , November 15–17, 2022.

[7] S. Li, Y. Qin, Z. Jiang, and W. Yang, "Efficient communication scheduling for parameter synchronization of dml in data center networks," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 1970–1985, 2022.

[8] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ml," *Proceedings of MLSys*, vol. 2, pp. 172–186, 2020.

[9] Z. Zhang, C. Wu, and Z. Li, "Near-optimal topology-adaptive parameter synchronization in distributed dnn training," in *Proceedings of the IEEE INFOCOM*, 2021, pp. 1–10.

[10] S. Luo, P. Fan, K. Li, H. Xing, L. Luo, and H. Yu, "Fast parameter synchronization for distributed learning with selective multicast," in *Proceedings of the IEEE ICC*, 2022, pp. 4775–4780.

[11] R. Rabenseifner, "Optimization of collective reduction operations," in *Proceedings of the Computational Science - ICCS 2004*. Springer Berlin Heidelberg, 2004, pp. 1–9.

[12] J. Kim, W. J. Dally, and D. Abts, "Flattened butterfly: A cost-efficient topology for high-radix networks," in *34th ISCA*, 2007, pp. 126–137.

[13] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, mar 2020.

[14] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, "A review of applications in federated learning," *Computers & Industrial Engineering*, vol. 149, p. 106854, 2020.

[15] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, jan 2019.

[16] I. Cano, D. Mahajan, G. M. Fumarola, A. Krishnamurthy, M. Weimer, and C. Curino, "Towards geo-distributed machine learning," *IEEE Data(base) Engineering Bulletin*, vol. 40, pp. 41–59, December 2015.

[17] C. Fan, X. Zhang, Y. Zhao, Y. Liu, and S. Yu, "Self-adaptive gradient quantization for geo-distributed machine learning over heterogeneous and dynamic networks," *IEEE Transactions on Cloud Computing*, vol. 11, no. 4, pp. 3483–3496, 2023.

[18] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong, Y. Jia, S. He, H. Chen, Z. Bai, Q. Hou, S. Yan, D. Zhou, Y. Sheng, Z. Jiang, H. Xu, H. Wei, Z. Zhang, P. Nie, L. Zou, S. Zhao, L. Xiang, Z. Liu, Z. Li, X. Jia, J. Ye, X. Jin, and X. Liu, "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *Proceedings of the 21st NSDI*, Santa Clara, CA, Apr. 2024, pp. 745–760.

[19] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "Ai and memory wall," *IEEE Micro*, pp. 1–5, 2024.

[20] Z. Lu, H. Pan, Y. Dai, X. Si, and Y. Zhang, "Federated learning with non-iid data: A survey," *IEEE Internet of Things Journal*, vol. 11, no. 11, pp. 19 188–19 209, 2024.

[21] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning," in *Proceedings of the USENIX ATC*, Jul. 2020, pp. 493–506.

[22] X. Wan, H. Zhang, H. Wang, S. Hu, J. Zhang, and K. Chen, "Rat - resilient allreduce tree for distributed machine learning," in *Proceedings of the 4th Asia-Pacific Workshop on Networking (APNet)*. ACM, 2020, pp. 52–57.

[23] W. Kendall, "MPI Reduce and Allreduce," 2023, [Online; accessed 10-May-2024]. [Online]. Available: https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/

[24] Y. Chen, Y. Peng, Y. Bao, C. Wu, Y. Zhu, and C. Guo, "Elastic parameter server load distribution in deep learning clusters," in *Proceedings of the 11th ACM SoCC*, 2020, pp. 507–521.

[25] L. Luo, P. West, J. Nelson, A. Krishnamurthy, and L. Ceze, "Plink: Discovering and exploiting datacenter network locality for efficient cloud-based distributed training," in *Proceedings of MLSys*, 2020, pp. 82–97.

[26] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *Proceedings of the 27th SOSP*. ACM, 2019, pp. 16–29.

[27] X. Liu, S. Luo, K. Li, and H. Xing, "Approximate gradient synchronization with aqgb," in *Proceedings of the 6th Asia-Pacific Workshop on Networking (APNet)*. ACM, July 2022, pp. 101–102.

[28] T. Hasunuma, "On edge-disjoint spanning trees with small depths," *Information Processing Letters*, vol. 75, no. 1/2, pp. 71–74, 2000.

[29] E. Solomonik, "CS 598: Communication Cost Analysis of Algorithms," https://solomonik.cs.illinois.edu/, 2016, [Online; accessed 31-July-2023].

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[31] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[32] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM Conference*, 2013, pp. 3–14.

[33] L. Luo, Y. Kong, M. Noormohammadpour, Z. Ye, G. Sun, H. Yu, and B. Li, "Deadline-aware fast one-to-many bulk transfers over inter-datacenter networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 304–321, 2022.

[34] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven WAN," in *Proceedings of the ACM SIGCOMM Conference*, 2013, pp. 15–26.

[35] J. A. Rico-Gallego, J. C. Díaz-Martín, R. R. Manumachu, and A. L. Lastovetsky, "A survey of communication performance models for high-performance computing," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–36, Jan. 2019.

[36] A. C. Zhou, Y. Gong, B. He, and J. Zhai, "Efficient process mapping in geo-distributed cloud data centers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017, pp. 1–12.

[37] F. Lai, M. Chowdhury, and H. Madhyastha, "To relay or not to relay for inter-cloud transfers?" in *Proceedings of the 10th HotCloud*, 2018, pp. 1–7.

[38] H. Tang, F. Liu, G. Shen, Y. Jin, and C. Guo, "Unidrive: Synergize multiple consumer cloud storage services," in *Proceedings of the 16th Annual Middleware Conference*. New York, NY, USA: ACM, 2015, pp. 137–148.

[39] X. Fei, F. Liu, H. Xu, and H. Jin, "Towards load-balanced vnf assignment in geo-distributed nfv infrastructure," in *Proceeding of the IEEE/ACM 25th IWQoS*, 2017, pp. 1–10.

[40] Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, and H. Jin, "Carbon-aware online control of geo-distributed cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2506–2519, 2016.

[41] Z. Zhou, F. Liu, Z. Li, and H. Jin, "When smart grid meets geo-distributed cloud: An auction approach to datacenter demand response," in *Proceeding of the IEEE INFOCOM*, 2015, pp. 2650–2658.
[42] F. Liu, B. Luo, and Y. Niu, "Cost-effective service provisioning for hybrid cloud applications," *Mob. Netw. Appl.*, vol. 22, no. 2, pp. 153–160, apr 2017.
[43] K. Li, K. Chen, S. Luo, H. Zhang, and P. Fan, "Ubinn: A communication efficient framework for distributed machine learning in edge computing," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3368–3383, 2023.
[44] S. Luo, P. Fan, H. Xing, L. Luo, and H. Yu, "Eliminating communication bottlenecks in cross-device federated learning with in-network processing at the edge," in *Proceedings of the IEEE ICC*, 2022, pp. 4601–4606.
[45] L. Luo, C. Zhang, H. Yu, G. Sun, S. Luo, and S. Dustdar, "Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network," *IEEE Transactions on Services Computing (Early Access)*, pp. 1–14, 2024.
[46] K. Zhao, Z. Zhou, X. Chen, R. Zhou, X. Zhang, S. Yu, and D. Wu, "Edgeadaptor: Online configuration adaption, model selection and resource provisioning for edge dnn inference serving at scale," *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5870–5886, 2023.
[47] K. Zhao, Z. Zhou, L. Jiao, S. Cai, F. Xu, and X. Chen, "Taming serverless cold start of cloud model inference with edge computing," *IEEE Transactions on Mobile Computing (Early Access)*, pp. 1–16, 2023.

Shouxi Luo (Member, IEEE) received the bachelor's degree in communication engineering and the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, in 2011 and 2016, respectively. He is currently an Associate Professor with Southwest Jiaotong University. His research interests include data center networks, software-defined networking, and networked systems.

Renyi Wang received the master's degree in computer science and technology from Southwest Jiaotong University in 2023. His research interests include distributed deep learning and networked systems.

Huanlai Xing (Member, IEEE) received the B. Eng. degree in communications engineering from Southwest Jiaotong University, China, in 2006, the M. Eng. degree in electromagnetic fields and wavelength technology from the Beijing University of Posts and Telecommunications, China, in 2009, and his Ph.D. degree in computer science from the University of Nottingham, U.K., in 2013. Currently, he is an Associate Professor with Southwest Jiaotong University. His research interests include mobile edge computing, evolutionary computation, etc.