Serverless-DFS: Serverless Federated Learning with Dynamic Forest Strategy

Bobin DENG*
Department of Computer Science,
Kennesaw State University

Xinyue Zhang
Department of Computer Science,
Kennesaw State University

Dan Chia-Tien Lo
Department of Computer Science,
Kennesaw State University

ABSTRACT

Training large-scale machine learning models typically requires support from High-performance Computers (HPCs) or cloud servers. One strategy to reduce the computational burden and alleviate the communication bottleneck of servers or HPCs is to move a subset of training computation to the edge or end-user devices. Federated learning is a distributed training approach that aims to protect the privacy of end-user data. However, most federated learning systems utilize centralized server architecture, which slows down the model training and limits the system's scalability. Many machine learning trainings require high performance while maintaining end-user data security. The state-of-the-art serverless approaches have performance limitations and can be further improved. This paper proposes Serverless-DFS, a serverless federated learning with a dynamic forest strategy, to improve the performance and scalability of federated learning while keeping its privacy protection capability. From the experimental results, our Serverless-DFS approach is observed to have a 63.9X speedup compared to the default server-client method in a large-scale system with 256 clients.

CCS CONCEPTS

• **Network architectures**; • **Network performance evaluation**; • **Systems security**;

KEYWORDS

Federated Learning, Serverless, All-reduce, Dynamic Forest

ACM Reference Format:
Bobin DENG, Xinyue Zhang, and Dan Chia-Tien Lo. 2023. Serverless-DFS: Serverless Federated Learning with Dynamic Forest Strategy. In *2023 the 12th International Conference on Networks, Communication and Computing (ICNCC) (ICNCC 2023), December 15–17, 2023, Osaka, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3638837.3638865

1 INTRODUCTION AND RELATED WORK

Federated learning [1–3] is a distributed Artificial Intelligence (AI) training technology that does not require clients to share their private data. The AI model that needs training will be forwarded to all participating clients, and clients can compute their own versions of gradients locally by using the exclusive data. Each client only provides its own version of gradients to the server or another client for gathering, and it is unnecessary to share other data. Besides the security and privacy benefits, applying federated learning to edge or mobile systems may also reduce the computational burden on the server side by fully utilizing clients' processing resources. A typical network topology for federated learning on edge or mobile systems is one-to-many server-client architecture. Each client computes a mini-batch of inputs and forwards updates to the server for accumulation. Later, the server broadcasts the overall updates to all clients. This operation that includes an accumulation and a broadcast is called all-reduce and is considered the current bottleneck of parallel machine learning training. The one-to-many server-client architecture strategy is straightforward and good for small-scale systems. The server is a centralized component in this architecture, which will become a bottleneck if the number of clients grows. So, the scalability of one-to-many server-client architecture is problematic.

λ-FL [6] is a serverless method for the update aggregation of federated learning. However, λ-FL is not a systematic solution for alleviating the entire all-reduce bottleneck, which consists of a gather operation followed by a broadcast. Moreover, λ-FL utilizes a queue to track and schedule the aggregation process, which is similar to a single tree topology that has constraints, including low network utilization and high communication latency. FedLess [7] is another serverless federated learning methodology that is based on FaaS (function-as-a-Service) platforms. Similar to λ-FL, FedLess mainly pays attention to aggregation instead of the entire all-reduce bottleneck. Multitree [4] is a parallel training method for gradient all-reduce operations with high performance and excellent network load balance. Multitree is designed based on data parallelism, where each processing node computes a mini-batch of inputs within a particular timeframe. However, Multitree is a static strategy where all nodes and data links are always available. In the mobile or edge systems that support federated learning, clients usually dynamically leave the wireless network. Moreover, Multitree requires each router to be connected to one private node within the system, and all connected routers must have functional component augments (e.g., Lookup Table, counter, comparator, etc.). So, we must explore a new methodology for serverless federated learning that can dynamically adjust the forest topology.

1) The main contributions of this paper are summarized below:

We explore a methodology on how to effectively construct a forest topology for all-reduce operations in the edge or mobile network.

2) We propose a Serverless Dynamic Forest Strategy (*Serverless-DFS*) for all-reduce operations of federated learning. This DFS does not require the server's participation, thus alleviating the potential congestion and ensuring network load balance, leading to better system performance and scalability.

*bdeng2@kennesaw.edu

ICNCC 2023, December 15–17, 2023, Osaka, Japan

ACM ISBN 979-8-4007-0926-5/23/12.
https://doi.org/10.1145/3638837.3638865

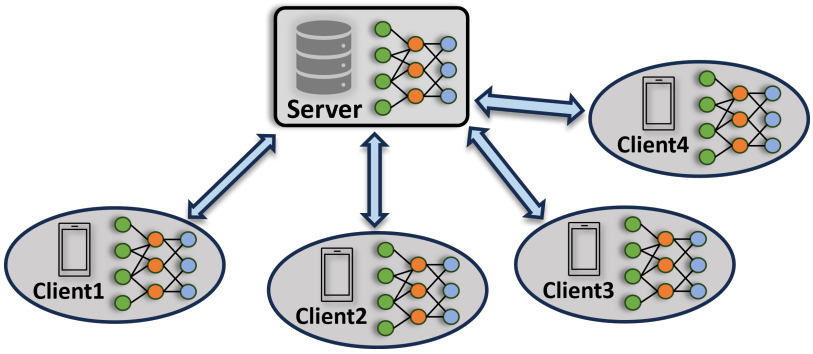

Figure 1: The Server-Client One-to-Many Architecture for Federated Learning

3) Perform empirical experiments to demonstrate the benefits of this *Serverless-DFS* approach.

The remaining sections of this paper are organized as follows: Section 2 will briefly introduce the federated learning background. Section 3 will discuss the design and implementation details of *Serverless-DFS*. Our experimental results are demonstrated in Section 4, and finally, we will conclude in Section 5.

2 FEDERATED LEARNING BACKGROUND

The primary motivation of federated learning is to provide a parallel training solution for a distributed system that does not require clients to provide their private data. The strategy is to move the machine learning model close to the data side and collect the gradients after completing the input batch computation. Figure 1 demonstrates the Sever-Client one-to-many architecture of federated learning. Each client has private data that is unwilling to share but is critical for the specific machine learning model training. The federated learning process is typically summarized as the following steps: *(1)* The server forwards the original version of the model to all participating clients. *(2)* After receiving the model, every client can start training using its private data set and obtain a version of the gradients. From the client's perspective, the gradients generated from the training process are not considered raw private data and can be shared with others via network communications. *(3)* Each active client sends its version of gradients to the server for gathering. One general gathering methodology is to sum up all the newly received versions and compute their average to update the model. *(4)* The server broadcasts the updated model to all clients. *(5)* Repeat *Step (1)* to *Step (4)* until the model convergence. The combination of one gathering and one broadcast is also known as the all-reduce operation.

The clients of edge or mobile systems may leave for various reasons. E.g., the client runs out of power. Essentially, this scenario will not block the training process because other clients will continuously forward the gradients and receive the updated model. However, an important problem of conventional server-client federate learning is that the server may become a bottleneck if the client number grows, thus lacking good scalability.

3 SERVERLESS FEDERATED LEARNING WITH DYNAMIC FOREST STRATEGY

This section will introduce our serverless federated learning with dynamic forest strategy (*Serverless-DFS*). We aim to remove the centralized bottleneck server from the federated learning framework to ensure better scalability while lowering the gradient gathering latency. According to Section 2, federated learning consists of multiple rounds of all-reduce operations until the model convergence. Ring-all-reduce [5] is a prevalent method for gathering and broadcasting gradients from parallel machine learning training. The benefits of ring-all-reduce include better scalability from its decentralized topology and ensure fairness. However, the communication latency of both gathering and broadcast is *O(N)*, where *N* is the total number of participating clients. One solution to reduce the communication delay is to change the ring topology into a tree topology, where the latency may be lower from *O(N)* to *O(log N)*. Unfortunately, the single tree strategy has serious load imbalance problems, potentially leading to congestion and long waiting delays if the size of the gradients is enormous.

We explore the solutions to utilize dynamic forest topology in federated learning to achieve lower latency, better scalability, and load balance. The '*dynamic*' indicates that one or more clients may leave the system for various reasons, but the training process must continually move forward.

3.1 Forest Construction Algorithm

Before starting the all-reduce operations of federated learning, the first step is to construct the forest topology with N binary trees, where N is the number of participating clients within the edge or mobile system. Every client will be the root node of one particular tree. All trees are constructed in parallel. But for simplicity, 2 only demonstrates the messages of a single tree (Root Node: *C0*; Client 0) construction and assumes that this system only has four clients. In 2*(a)*, the root client *C0* broadcasts the *RT0_T_L1_Req* messages to all other clients. The *RT0_T_L1_Req* indicates this is a request to construct *Layer-1* of the *Root-0-Tree* (Client 0 is the tree root). Once a client receives the *RT0_T_L1_Req*, it will check if it has been joined to the *Root-0-Tree*. If not, it will reply with an *RT0_T_L1_Avail* message to the sender to notify the join intention. Otherwise, the client already joined the *Root-0-Tree* just ignores the *RT0_T_L1_Req*

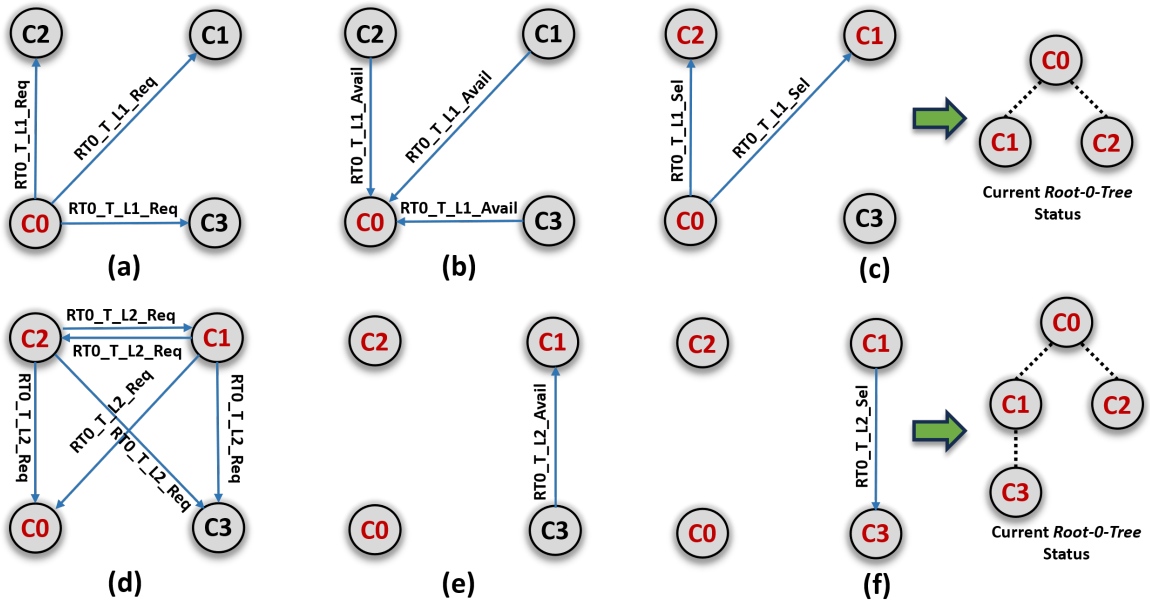

Figure 2: An example of the forest construction algorithm in a four-client edge network. For simplicity, we only illustrate the messages for *Root-0-Tree* construction. The messages of the remaining tree constructions are ignored, but all trees in the forest are structured in parallel.

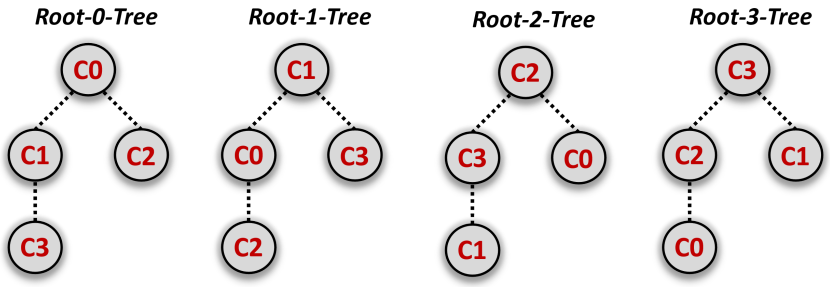

Figure 3: An example of the final forest with four trees.

message. In 2*(b)*, because all other clients have not joined the *Root-0-Tree*, they reply *RT0_T_L1_Avail* to Client 0 (*C0*) to express the intention to join *Layer-1* of *Root-0-Tree*. Because we will construct a binary tree to ensure the *O(log N)* latency, the senders of the first two *RT0_T_L1_Avail* messages will be selected as the nodes of *Layer-1* of *Root-0-Tree*. In this example, we assume *C1* and *C2* are selected, and *C0* will ignore the last *RT0_T_L1_Avail* message from *C3*. 2*(c)* shows that *C0* sends the *RT0_T_L1_Sel* messages to *C1* and *C2* to notify them that they have been selected for *Layer-1* of *Root-0-Tree*. The status of the *Root-0-Tree* is illustrated on the right of 2*(c)*, where *C1* and *C2* are the two children of the root *C0*. In 2*(d)*, the newly joined clients *C1* and *C2* broadcast the *RT0_T_L2_Req*. Because *C0*, *C1*, and *C2* are already within the *Root-0-Tree*, they directly ignore all *Root-0-Tree* joining requests. *C3* does still not join the *Root-0-Tree*, so it will reply *RT0_T_L2_Avail* to the sender of the first *RT0_T_L2_Req* message it received. In 2 *(e)*, we assume the first *RT0_T_L2_Req* received by *C3* is from *C1*. So, in 2*(f)*, *C1* replies *RT0_T_L2_Sel* to *C3* to notify that *C3* has been selected to join the *Layer-2* of *Root-0-Tree*. The final *Root-0-Tree* is shown on the right of 2*(f)*

Every tree construction is independent, so all trees within the forest are created in parallel. 3 demonstrates an example of four trees generated via our forest construction algorithm. Because the order of receiving request and response messages is uncertain, we may create many different versions of forests. We are only required to execute this Forest Construction Algorithm once right before the first all-reduce operation of parallel training.

3.2 Serverless Federated Learning with Logical Forest Topology

After completing the static forest construction for all participating clients, we can start the federated learning process. The training

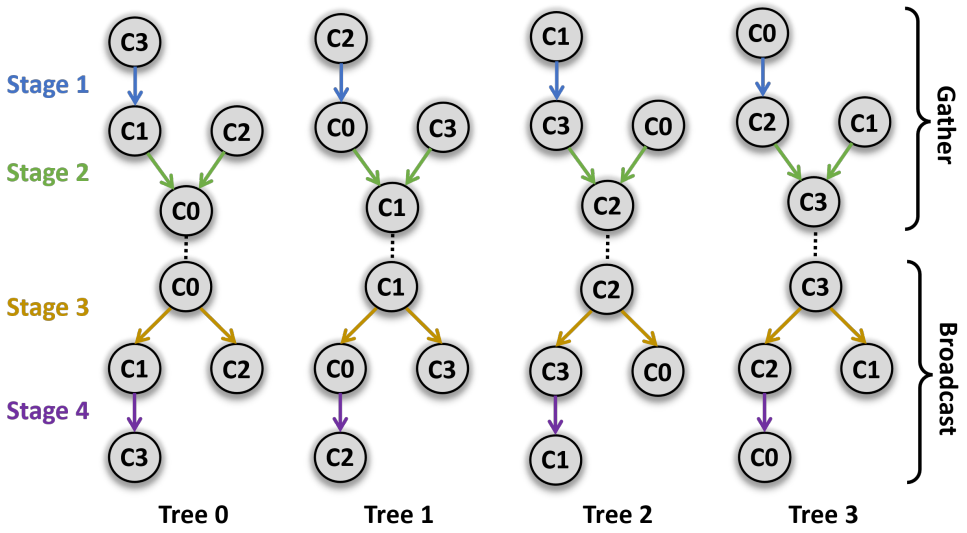

Figure 4: The process of an all-reduce operation in a four-client system via a logical forest topology.

consists of multiple rounds of all-reduce operations. Each all-reduce is a gather operation followed by a broadcast operation. To support all-reduce operations via the forest topology from 3, the same as the Multitree method [4], we flip the entire forest to get the topology for gathering. The broadcast step can be directly supported by the forest generated in *Section 3.1* (3). The main reason to utilize a forest instead of a single tree is that it can improve the bandwidth utilization rate and further eliminate potential bottlenecks. Because all clients train the same model, the size of total gradients from all clients should be the same. So, we can divide the gradients into N subsets, and each subset of gradients would transmit via one tree (with its flipped tree) within the forest. According to 4, the all-reduce is divided into four stages in this four-client example, where the first two stages are gathering via the flipped forest, and the last two stages are broadcasting via the default forest. The number of stages equals two times the tree height: *2log(N)*. After an all-reduce operation is complete, a new round of mini-batch computation is continuous until the model convergence. So, this *Serverless-DFS* scheme can effectively complete multiple rounds of all-reduce operations for federated learning without the server's participation.

3.3 All-reduce Routing Lookup Table in Clients

During the all-reduce process, once a client receives a subset of gradients, this client should know where it should forward this gradient to or if it is the gather/broadcast end node of this gradient subset. So, each client should have its private Lookup Tabel (LUT) for this routing information. The routing table is similar to that of Multitree [1], but our routing tables need to include parent and children information about the client loss problem, which will be further discussed in *Section 3.4*. Moreover, the LUT of the Multitree method is stored in each client's directly connected router, where each router within the network only has one corresponding client. The relationship between the client and router in the edge or mobile systems typically is not bijective. Considering the heterogeneity of the client devices and enhancing the practicality of *Serverless-DFS*, we store a routing LUT in the RAM of each client. The routing LUT in each client should be different because they are only required to save the routing information related to themselves.

5 shows the all-reduce routing LUTs in all four clients. Each client has a private LUT, which has six columns: *Tree ID*, *Parent ID*, *Child ID 1*, *Height 1*, *Child ID 2 and Height 2*. The *Tree ID* is to identify the current subset of gradients using which tree for all-reduce. The *Parent ID* and two *Child IDs* are determined by default forest (4). All the trees in the forest are balanced binary trees, and we require two columns for child IDs and two columns for subtree heights in each LUT. When a subset of gradients traverses a specific tree and its flipped tree, this subset of gradients should also include a 1-bit value to specify whether it is in gather or broadcast operation. Besides this 1-bit operation value, the subset of gradients should also include the Tree ID value, which requires *O(log N)* bits. E.g., in the *Gather* part of *Tree 0* (4), *C1* receives a subset of gradients from *C3*. Now, the *C1* should check its LUT to find out what actions it should take and where this subset of gradients should be forwarded. Because this subset of gradients contains the *Tree ID* (Tree 0) and the 1-bit operation value (*Gather*), C1 can use these two values to determine the next step destination. In *Client 1*'s LUT, the row with *Tree ID '0'* has been selected via the *Tree ID* value. The '*Gather*' value indicates that the next destination should be its *Parent*, where we can get the value *0*. This means *Client 1* should forward this subset of gradients to *Client 0* to continue the *Gather* operation.

3.4 Dynamic Forest Adjustment for Client Loss

For federated learning in the edge or mobile systems, the client may quit the training group for various reasons, such as running out of power or moving outside the valid range. To improve the robustness of *Serverless-DFS*, we should explore the mechanism to dynamically prune the forest when the client leaves. This section

Client 0

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	N/A	1	2	2	1
1	1	2	1	N/A	N/A
2	2	N/A	N/A	N/A	N/A
3	2	N/A	N/A	N/A	N/A

Client 1

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	0	3	1	N/A	N/A
1	N/A	0	2	3	1
2	3	N/A	N/A	N/A	N/A
3	3	N/A	N/A	N/A	N/A

Client 2

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	0	N/A	N/A	N/A	N/A
1	0	N/A	N/A	N/A	N/A
2	N/A	0	1	3	2
3	3	0	1	N/A	N/A

Client 3

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	1	N/A	N/A	N/A	N/A
1	1	N/A	N/A	N/A	N/A
2	2	1	1	N/A	N/A
3	N/A	1	1	2	2

Figure 5: All-reduce Routing Lookup Tables (LUTs) of the four-client example; The LUT in every client is different from others and only stores the routing information related to itself.

Client 0

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	N/A	1	~~2~~ 1	3 ~~2~~ N/A	1 ~~1~~ N/A
1	1	~~2~~ N/A	~~1~~ N/A	N/A	N/A
~~2~~	~~2~~	~~N/A~~	~~N/A~~	~~N/A~~	~~N/A~~
3	~~2~~ 3	N/A	N/A	N/A	N/A

Client 1

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	0	~~3~~ N/A	~~1~~ N/A	N/A	N/A
1	N/A	0	~~2~~ 1	3	1
~~2~~	~~3~~	~~N/A~~	~~N/A~~	~~N/A~~	~~N/A~~
3	3	N/A	N/A	N/A	N/A

Client 2

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	0	N/A	N/A	N/A	N/A
1	0	N/A	N/A	N/A	N/A
2	N/A	0	1	3	2
3	3	0	1	N/A	N/A

Client 3

Tree ID	Parent ID	Child ID 1	Height 1	Child ID 2	Height 2
0	1	N/A	N/A	N/A	N/A
1	1	N/A	N/A	N/A	N/A
~~2~~	~~2~~	~~1~~	~~1~~	~~N/A~~	~~N/A~~
3	N/A	1	1	~~2~~ 0	~~2~~ 1

Figure 6: Updates of all-reduce routing LUTs when Client 2 quits. The updates in Step (2) are marked in red, and those in Step (3) are marked in green.

continues to use the four-client system as an example. Without loss of generality, we assume Client 2 now leaves the group for parallel training. The procedure to update the forest information is summarized as three steps: (1) Fetch the *Client 2* routing LUT from a backup storage. (2) Combining the information from *Client 2* LUT, every client scans its private LUT and modifies all cells related to *Client 2*. (3) Unbalanced binary tree detection and adjustment.

6 demonstrates how to dynamically update the LUTs to refresh the forest topology. Once the leave of *Client 2* has been detected, all other clients should fetch the *Client 2* LUT (*blue* background) from the backup storage to support the updates of LUTs in other clients. Then, in *Step (2),* each client scans all its LUT entries and modifies the cells that are associated with *Client 2*. The updates from *Step (2)* are marked as red text in 6. The forest status after *Step (2)* is shown on the left part of 7. To maintain the *O(log N)* traverse latency, we require an extra step (*Step 3*) to detect the unbalanced binary trees and make the corresponding adjustments. From the left side of 7, we know that *Tree 0* is unbalanced after *Step (2)*, which could be detected by computing the difference between *Height 1* and *Height 2* values in the same LUT row. The updates of LUT from *Step (3)* are marked as green text in 6. Finally, all binary trees in the all-reduce routing forest are balanced and shown on the right side of Figure 7.

4 EVALUATIONS

We systematically model three federated learning frameworks: one-to-many Server-Client topology, Single Tree serverless topology, and dynamic Forest serverless topology (*Serverless-DFS*). This section aims to evaluate the speedup and scalability of our *Serverless-DFS* for federated learning.

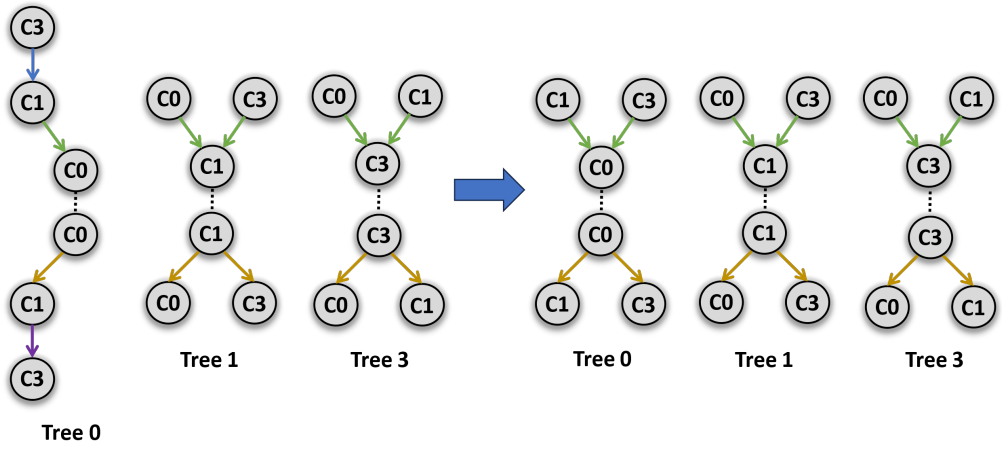

Figure 7: Adjust the unbalanced binary tree(s) to balanced binary tree(s).

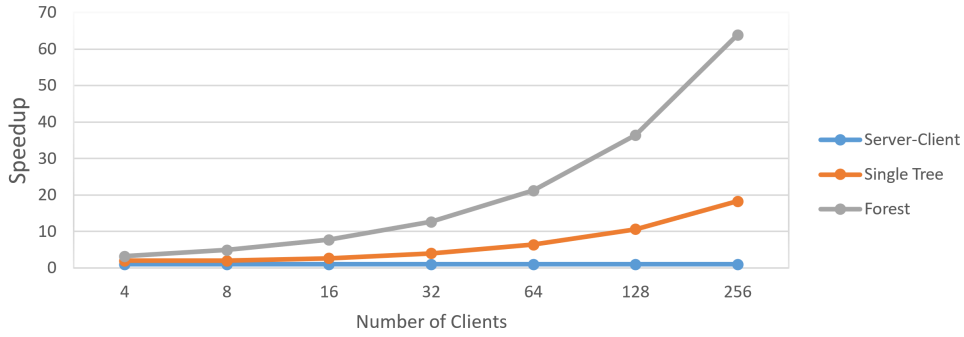

Figure 8: Speedup Comparisons for Client-Server, Single Tree, and Forest Topologies with Numbers of Clients Scaling

4.1 Speedup Comparisons with Different Client Numbers

Currently, over 15 billion IoT (Internet of Things) devices are connected worldwide, and the number is expected to double by 2030 [8]. So, we should evaluate the performance scalability of our *Serverless-DFS* approach by gradually increasing the client number in the system. 8 exhibits the speedup of three approaches with client numbers from 4 to 256. The speedup of one-to-many Server-Client topology is normalized as 1. The benefits of *Serverless-DFS* become more significant if the client number grows. For example, in a system with 256 clients, our *Serverless-DFS* demonstrates a 63.9X speedup compared to the default Server-Client topology, where the Single Tree approach's speedup is 18.3X. The main reason for observation is the forest method optimizes the load balance of the entire network, which is critical for alleviating congestion in large-scale systems.

4.2 Speedup Comparisons with Different Client Capabilities

The capabilities of IoT or edge devices are expected to continue growing due to the development of semiconductor technology. The devices may have more processing elements and memory resources. For example, the NVIDIA Jetson AGX Orin contains a 12-core CPU and a GPU with 2048 CUDA cores and 64 Tensor Cores [9]. So, the edge or IoT devices are expected to have more parallel capability, and their Instructions Per Cycle (IPC) should gradually increase. 9 shows the speedup comparisons of three all-reduce topologies with the clients' IPC from 0.1 to 10. The experimental results demonstrate that *Serverless-DFS* (forest) gets better speedup if the clients' IPC is higher, indicating that *Serverless-DFS* should have more significant potential for future powerful edge devices. For example, if the IPC of edge devices is 10, the Forest approach gets 25X, and the Single Tree only obtains 4X. Similar to *Section 4.1*, the speedup of the Server-Client approach is normalized as 1. The main reason for this observation is that the devices with higher IPC would reduce the computational time, and the communication delay would dominate the overall latency. *Serverless-DFS* is a forest approach to lower the communication delay and ensure network load balance, so we can see the huge speedup of the forest method with high client IPC.

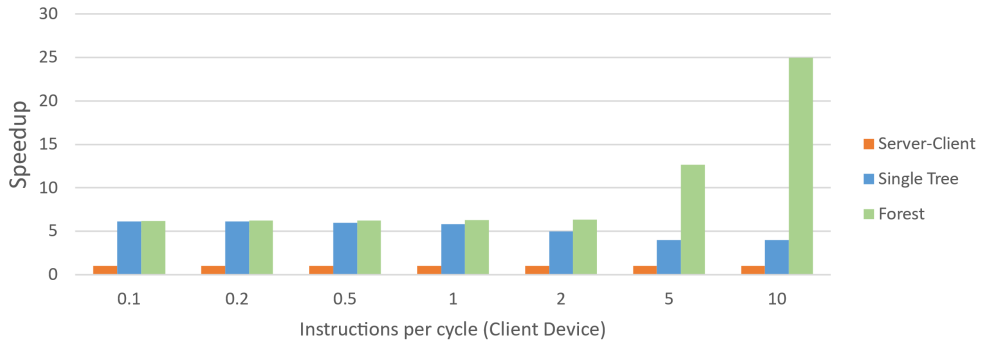

Figure 9: Speedup Comparisons for Client-Server, Single Tree, and Forest topologies with different computing capabilities (IPC: Instructions Per Cycle) of Clients

5 CONCLUSION

This paper introduces the implementation details of the *Serverless-DFS* approach, which aims to improve the performance and scalability of federated learning while maintaining end-user data security. Compared to the default Server-Client topology, *Serverless-DFS* gets 63.9X speedup in a large-scale system with 256 clients, whereas the Single Tree method is only observed 18.3X. In a 32-client system where each client IPC is 10, *Serverless-DFS* gets 25X speedup compared to Server-Client method, and the Single Tree approach only gets 4X in speedup. Data privacy is the primary concern of the federated learning framework. The future work of this paper will focus on identifying potential security issues of Serverless-DFS and propose corresponding protections.

REFERENCES

[1] Li, L., Fan, Y., Tse, M., & Lin, K. Y. (2020). A review of applications in federated learning. Computers & Industrial Engineering, 149, 106854.

[2] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. Knowledge-Based Systems, 216, 106775.

[3] Li, T., Sahu, A. K., Talwalkar, A., & Smith, V. (2020). Federated learning: Challenges, methods, and future directions. IEEE signal processing magazine, 37(3), 50-60.

[4] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum and E. J. Kim, "Communication Algorithm-Architecture Co-Design for Distributed Deep Learning," 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2021, pp. 181-194, doi: 10.1109/ISCA52012.2021.00023.

[5] M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan and J. Liu, "GADGET: Online Resource Optimization for Scheduling Ring-All-Reduce Learning Jobs," IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, London, United Kingdom, 2022, pp. 1569-1578, doi: 10.1109/INFOCOM48880.2022.9796785.

[6] Jayaram, K. R., Muthusamy, V., Thomas, G., Verma, A., & Purcell, M. (2022, February). Lambda FL: Serverless Aggregation For Federated Learning. In International Workshop on Trustable, Verifiable and Auditable Federated Learning (p. 9).

[7] Grafberger, A., Chadha, M., Jindal, A., Gu, J., & Gerndt, M. (2021, December). Fedless: Secure and scalable federated learning using serverless computing. In 2021 IEEE International Conference on Big Data (Big Data) (pp. 164-173). IEEE.

[8] Duarte, F., (2023, February). Number of IoT Devices (2023), https://explodingtopics.com/blog/number-of-iot-devices

[9] Leela S. Karumbunathan, NVIDIA Jetson AGX Orin Series, https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf