This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

Optimizing Aggregation Frequency for Hierarchical Model Training in Heterogeneous Edge Computing

Lei Yang, Yingqi Gan, Jiannong Cao, *Fellow, IEEE,* Zhenyu Wang

Abstract—Federated Learning (FL) has been widely used for distributed machine learning in edge computing. In FL, the model parameters are iteratively aggregated from the clients to a central server, which is inclined to be the communication bottleneck and single point of failure. To solve these drawbacks, hierarchical model training frameworks like Hierarchical Federated Learning (HFL) and E-Tree learning have been proposed. One of the most challenging problems in the hierarchical model training framework is optimizing the aggregation frequencies of the edge devices at various levels. Because, in an edge computing environment, heterogeneity in the resource can introduce synchronization delays caused by waiting for slow workers and significantly impact the training performance. This paper tackles the problem with weak synchronization where edge devices on the same level have different frequencies on local updates and/or model aggregations. Existing works based on weak synchronization lack solutions to quantitatively determine the aggregation frequencies of each edge device. Thus, we propose a resource-based aggregation frequency controlling method, termed RAF, which determines the optimal aggregation frequencies of edge devices to minimize the loss function according to heterogeneous resources. Our proposed method can alleviate the waiting time and fully utilize the resources of the edge devices. Besides, RAF dynamically adjusts the aggregation frequencies at different phases during the model training to achieve fast convergence speed and high accuracy. We evaluated the performance of RAF via extensive experiments with real datasets on our self-developed edge computing testbed. Evaluation results demonstrate that RAF outperforms the benchmark approaches in terms of learning accuracy and convergence speed.

Index Terms—Edge intelligence, distributed machine learning, aggregation frequency.

1 Introduction

An increasing number of mobile and Internet-of-Thing devices are connected to the Internet, generating high volumes of data at the network edge. Due to high communication costs and privacy leakage concerns, it's impractical to transmit a tremendous amount of data from the data sources to the cloud for building AI models. To address these challenges, edge intelligence [1] has been proposed to perform AI model training and inference in closer proximity to data sources. The major model training framework for edge intelligence is Federated Learning [2], and Local SGD [3], [4], where multiple workers run a few local gradient steps independently in parallel, and a central server computes the average periodically. These centralized approaches have drawbacks such as communication bottlenecks and single points of failure.

Thus, to solve the drawbacks, existing works designed hierarchical model training frameworks including E-Tree learning [5] proposed in our previous work, Hierarchical Federated Learning [6], [7] and so on. These hierarchical model training frameworks adopt a tree-based aggregation structure where the leaf nodes act as the training workers, and the non-leaf nodes act as model aggregation nodes. The structure with multiple levels of model aggregations is decided based on the network topology and data distribution.

One of the most challenging and important problems for hierarchical model training is to optimize the aggregation frequencies of the edge devices at various levels. The key characteristics of an edge computing environment are resource heterogeneity, including computation resource heterogeneity, data quantity heterogeneity, and communication resource heterogeneity. Heterogeneity in edge resources can significantly impact the training performance of hierarchical model training if edge devices have the same aggregation frequencies on local updates and model aggregations. Because edge devices with less computational capacity, more training data samples, and fewer bandwidth resources become stragglers. Synchronizing delay caused by waiting for the stragglers can affect the convergence speed and training accuracy.

Existing works on aggregation frequency controlling, such as parallel mini-batch SGD [8], Local SGD [3], [4], Federated Learning [2], and Hierarchical Federated Learning [6], [7], adopt strong synchronization method where edge devices have the same aggregation frequencies in the same level, and aggregation nodes compute the average periodically. In these works, faster nodes need to wait until all the other nodes finish their local updates or model aggregations, and this will lead to a waste of resources on faster nodes. The strong synchronization method is not suitable for a heterogeneous edge computing environment. To tackle the problem, E-Tree learning [5] and PR-SGD [9]

- *L. Yang, Y. Gan and Z. Wang are with the School of Software Engineering, South China University of Technology, Guangzhou, China, 510006. E-mail: {sely, wangzy}@scut.edu.cn, 201921043967@mail.scut.edu.cn*
- *J. Cao is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. E-mail: csjcao@comp.polyu.edu.hk*

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

adopt a weak synchronization method where edge devices with heterogeneous resources have different aggregation frequencies, i.e., faster nodes perform more local updates and/or model aggregations than slower nodes. However, these works using weak synchronization lack experiments and solutions on how to determine the aggregation frequencies of each edge device.

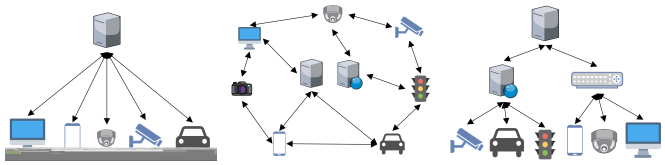
Therefore, we propose a resource-based aggregation frequency controlling algorithm, namely RAF, to optimize the frequencies of local update and model aggregation for each edge device in a hierarchical model training framework, aiming to fully utilize the resources of the heterogeneous edge devices and improve the convergence speed and model accuracy. RAF utilizes a heuristic algorithm to obtain the aggregation frequencies according to the computation and communication resources of each edge device. However, if the difference of the aggregation frequencies among each edge device is large, this method can result in an inferior error-convergence. To tackle this problem, RAF leverages an adaptive communication method, which starts with a given aggregation frequency and decreases it with respect to wall-clock time.

We develop a real-world testbed to provide the first empirical study of validating the performance of the weak synchronization as opposed to the strong synchronization in a heterogeneous environment. Our experimental results show that RAF accelerates the convergence speed and improves training accuracy while alleviates the waiting time and makes effective use of the available resources in comparison with hierarchical fully synchronous SGD and hierarchical federated learning. The main contributions of this paper are summarized as follows.

- To the best of our knowledge, this work is the first one to consider the model training with varying aggregation frequencies in a tree-based hierarchical model training.
- We propose an efficient aggregation frequency controlling algorithm, named RAF, which optimizes the aggregation frequencies for all edge devices in the hierarchical model training. Furthermore, to tackle the problem that large discrepancies between the aggregation frequencies on edge devices may result in inferior training performance, we adaptively adjust the aggregation frequencies at different phases during the model training.
- We verify RAF via extensive experiments on a real-world testbed. Results show that RAF obviously outperforms the state-of-arts approach like hierarchical fully synchronous SGD and hierarchical federated learning.

2 Background and Motivation

In this section, we firstly describe the existing distributed model training structures. We then introduce the existing synchronization approaches, including strong synchronization and asynchronous synchronization. Finally, we show how the heterogeneous resource poses challenges to existing hierarchical model training with strong synchronization and present an alternative method, namely weak synchronization, to tackle the problem.



(a) Centralized. (b) Decentralized. (c) Hierarchical.

Fig. 1: Distributed model training structure.

2.1 Distributed Model Training

Learning Problem. We consider a set of edge devices connected via a network. Each device i has a local dataset D_i. Distributed model training refers to that the edge devices cooperatively learn a model using the distributed datasets by exchanging important model features or parameters. Normally the devices do not exchange the original data directly due to privacy concerns. The purpose is to achieve high model accuracy and fast convergence. We assume the underlying network model is a mesh network, in which the edge devices can communicate with each other via multiple hops. This network model fits many practical edge computing applications such as the vehicular network, drone network, and multi-robots system. Most approaches for distributed model training use the idea of data parallelism since it is easy to implement. Considering the approach of data parallelism in this paper, we simply define this general learning problem as follows.

We assume there are N edge devices with local datasets D_i. We denote $D \triangleq \sum_{i=1}^{N} D_i$. For each dataset D_i at the edge device i, the loss function is

$$F_i(\boldsymbol{w}) \triangleq \frac{1}{|D_i|} \sum_{j \in D_i} f_j(\boldsymbol{w}). \tag{1}$$

The $f_j(\boldsymbol{w})$ is a short representation of $f(\boldsymbol{x}_j, y_j, \boldsymbol{w})$, denoting the loss fucntion of the j-th data sample. The $\boldsymbol{x}_j$ is regarded as the input sample, and y_j is the desired output of the model. We use $|\cdot|$ to denote the size of the dataset. The training process is to minimize the global loss function based on distributed datasets:

$$F(\boldsymbol{w}) \triangleq \frac{\sum_{i=1}^{N} |D_i| F_i(\boldsymbol{w})}{|D|}. \tag{2}$$

Distributed Model Training Structures. As shown in Fig. 1, there exist three structures for distributed model training, i.e., centralized structures, decentralized structures, and hierarchical structures. Representative centralized structures in Fig. 1a are Federated Learning (FL) [2], Local SGD [3], [4] and Parameter Servers (PS) [10], which perform local training in parallel on different workers and average the model parameters at the central node periodically. These centralized structures often suffer from long communication delays caused by the stragglers. Besides, the communication traffic jam on the central node degrades the training performance, especially when the network bandwidth is limited. The central node also faces a single point of failure. To address the limitations of the centralized structures, decentralized structures in Fig. 1b like Gossip Learning (GL) [11], [12], EdgeGossip [13], Decentralized Parallel Stochastic Gradient Descent (D-PSGD) [14], Cooperative SGD (Co-SGD) [15], and Ring-AllReduce [16], have been proposed,

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

TABLE 1: Comparisons of Various Distributed Model Training Frameworks

	Strong Synchronization	Asynchronization	Weak Synchronization
Centralized Structure	FL [2], PS [10], Local SGD [4], BSP [17], [18]	SSP [19], ASGD [20], [21]	PR-SGD [9]
Decentralized Structure	D-PSGD [14], Co-SGD [15], Ring-AllReduce [16]	GL [11], [12], EdgeGossip [13], RNA [22]	-
Hierarchical Structure	HFL [6], [7]	-	**E-Tree Learning [5] (Our work)**

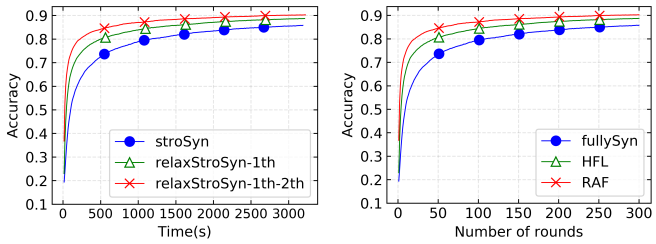
where each device performs not only local training but also aggregate models directly. Because the model aggregations are fully distributed on every device, this structure reduces the extremely high aggregation traffic onto the central node and avoids the single point of failure. However, the decentralized structures may result in an inferior convergence when the local training datasets in each device are unbalanced and non-IID.

To combine the advantages of centralized structures and decentralized structures, hierarchical structures in Fig. 1c have been proposed, including E-Tree learning [5] and hierarchical federated learning (HFL) [6], [7]. Hierarchical structures applies a tree-based model training structure, in which leaf nodes perform local updates and non-leaf nodes perform model aggregations. The experiment results in [5] have been shown that in a network with a relatively large number of edge devices, E-Tree learning obviously outperforms the federated learning and gossip learning in terms of convergence speed and final model accuracy. The reason is that hierarchical structures uses localized aggregations allowing edge devices to perform model aggregations in parallel. This can not only avoid high communication costs, but also can relieve the pressure on the central aggregation nodes. Besides, localized aggregations can also prevent the model deviation under non-IID data.

2.2 Existing Synchronization Approaches

Synchronization is an essential issue in distributed model training. Existing synchronization approaches include *strong synchronization* and *asynchronization*. The distributed model training structures like federated learning, hierarchical federated learning, BSP [17], [18], and D-PSGD [14] adopt strong synchronization, where the aggregation frequencies of edge devices in the same level are the same. The aggregation nodes compute the average only when all their child nodes have uploaded the model parameters. However, in a heterogeneous edge computing environment, edge devices have different rates of performing local updates or model aggregations. The faster devices need to wait for the stragglers until their models are aggregated simultaneously. This leads to slow convergence speed and low utilization of resources on the devices.

Another approach is *asynchronization*, which can be classified into two categories, i.e., *full asynchronization* and *constrained asynchronization*. Representative works like ASGD [20], [21] adopt *full asynchronization* in a centralized structure, where edge devices perform local iteration asynchronously in parallel without synchronization. Edge devices independently push the updates to the central aggregation server and do not need to aggregate simultaneously along with other devices. As a decentralized structure, Gossip Learning also adopts the *full asynchronization*. Each peer (device/worker) does the gradient descent and randomly sends the model updates to its neighbors. Upon receiving the model updates from neighbors, the peer averages them with its local model and then sends out the updated models. Though *full asynchronization* can avoid stragglers and efficiently use the resources of edge devices, it leads to a severe delay when the rate of iteration between edge devices varies greatly, especially in a heterogeneous environment. This will slow down the convergence due to the severe staleness problem and yield model inaccuracy. Thus, *constrained asynchronization* including RNA [22] and SSP [19], has been proposed, where edge devices conduct local iteration asynchronously within a given constraint like bounded staleness and update their local models with the order, stale versions of global parameters. These methods can prevent the number of iterations between the fastest node and the slowest node from being too far apart by giving some restrictions.



(a) Accuracy over time. (b) Accuracy over rounds.

Fig. 2: Comparison in accuracy of different methods on MNIST with heterogeneity resources.

We observe that *asynchronization* was mostly used in the centralized and decentralized model training structures, while in a hierarchical structure, it is not feasible to apply this approach because this structure requires the concurrent model aggregation from multiple clients level by level. The existing approach used for hierarchical structures was *strong synchronization*. As follows, we will conduct a measurement study to show the limitation of *strong synchronization* as it was used in a hierarchical structure.

2.3 Challenges and Problems

2.3.1 Case Study

We consider a heterogeneous environment, where edge devices have different computation capabilities, and the connected links between edge devices have different bandwidths. In this situation, strong synchronization for distributed model training is not necessarily the best choice. We conduct an experimental study on our self-developed testbed to demonstrate that strong synchronization exhibit a worse training performance in terms of convergence speed and final accuracy. We train a two-layer CNN model on the

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

TABLE 2: Comparison in Average Computation Time and Synchronization Time across Three Methods.

	stroSyn		relaxStroSyn-1th		relaxStroSyn-1th-2th	
	2-th Level	3-th Level	2-th Level	3-th Level	2-th Level	3-th Level
Comp. Time	1.19s	6.09s	2.42s	6.15s	2.44s	8.41s
Syn. Time	5.19s	4.53s	4.01s	4.56s	2.36s	2.33s
Ratio Value	0.23	1.34	0.60	1.34	1.03	3.60

MNIST dataset, which contains $60,000$ training samples and $10,000$ test samples of ten handwritten digits. We consider a three-level hierarchical model training structure with 10 devices connected through a physical mesh network. Each device has 5 classes of data samples randomly selected out of the 10 digits classes. The number of data samples in each edge device is 5000. Each device is assigned with various computation capabilities ranging from 0.2 CPU to 15 CPUs. The bandwidth of each link between devices ranges from 16 Mbps to 40 Mbps. Besides, the detailed configurations of the comparison methods are as follows: the *stroSyn* method adopts strong synchronization where the aggregation frequencies of edge devices in the tree-based hierarchical structure are set to 1. The *relaxStroSyn-nth* method means we relax the strong synchronization allowing faster nodes at the n-th level to perform more local updates and/or model aggregations and then aggregate together at the parent node. The *relaxStroSyn-1th* only relaxes the restriction at the 1^{st} level, while the *relaxStroSyn-1th-2th* relaxes the restriction at the 1^{st} level and the 2^{nd} level.

Fig. 2 shows that *stroSyn* has obviously worse performance than the other two methods in terms of convergence speed and convergence quality. To explain the reason for these observations, we measure the computation time and synchronization time of each device during a round of model training. Computation time refers to the time of model updating and aggregation, while the synchronization time represents the blocking time on the device, including the waiting time and transmission time of model parameters. By averaging over all the edge devices, the ratio of average computation time to the average synchronization time (named by ratio value in short) reflects the resource utilization of the devices. Table 2 compares the ratios of computation time to the synchronization time. The ratio value of *stroSyn* is much less than the other relaxed synchronization methods. It means that *stroSyn* spends more time waiting for the slow nodes and less time doing local model updating or aggregation. This is the reason why *stroSyn* leads to a waste of computation resources and slows down the convergence of model training.

For accuracy comparison, Fig. 2 shows that *relaxStroSyn-1th-2th* achieves much better accuracy compared to *stroSyn*, i.e., up to 4.63% better if training time is constrained to 3000s. If we look at the accuracy over rounds, *relaxStroSyn-1th-2th* also outperforms *stroSyn*, i.e., up to 4.43% after 300 rounds. Besides, the training performance of *relaxStroSyn-1th-2th* is better than *relaxStroSyn-1th*. This means both the leaf nodes performing local updates and non-leaf nodes performing model aggregations are required for relaxing the strict synchronization.

2.3.2 Weak Synchronization

From the above observations, we learn that strong synchronization does not fully exploit computation capacities because faster nodes wait for the stragglers to complete the training, leading to a waste of resources. Besides, stragglers, which fall behind the other nodes, can enlarge the time required for a round, resulting in worse convergence speed and lower model accuracy.

These observations motivate us to propose *weak synchronization* for hierarchical model training, where edge devices on the same level have different times of local updates and/or model aggregations before aggregating together at their parent node. It's also illustrated in the above experiment that *weak synchronization* should be adopted not only at the workers to perform local updates but also at the aggregation nodes to perform model aggregation. *Weak synchronization* at all the levels of hierarchical model training significantly maximizes the time spend in useful computation rather than in waiting for the stragglers. *Weak synchronization* can not only improve the utilization of resources, but also yields faster convergence than strong synchronization.

Research Problem and Scope. Learning the advantages of *weak synchronization*, we want to study the following key question: *how to quantitatively determine the aggregation frequencies of edge devices in weak synchronization?* Table 2 summarizes the existing works on distributed model training. Our paper focus on the hierarchical model training structure because this structure has been shown in our previous work [5] to have superior performance over the other structures. With a *weak synchronization* used in the hierarchical structure, we further study the problem of optimizing the aggregation frequency of edge devices at various levels.

Although the idea of *weak synchronization* has been reported in PR-SGD [9] for a centralized structure, our work has the following differences and innovations compared with PR-SGD. Firstly, we adopt *weak synchronization* in a hierarchical structure instead of a centralized structure. Our paper is the first work to formalize the process of model training with varied aggregation frequencies in hierarchical model training. It's significant for us to consider the key question: how to schedule the aggregation operations onto the edge devices. We need to decide where and when each model aggregations are performed based on the computation resources on the edge devices, the network topology, and communication resources. Also, PR-SGD does not provide solutions for quantitatively determining the aggregation frequency and does not experiment to verify the effectiveness of weak synchronization in the centralized structure.

3 Hierarchical Model Training

In this section, we first present the construction of the hierarchical aggregation structure and the process of model training and aggregation, namely E-Tree learning proposed in our previous work [5]. We then formulate the problem of aggregation frequency optimization. The main notations in this paper are summarized in Table 3.

3.1 E-Tree Construction and Model Aggregation

Building E-Tree Structure. Our previous work has proposed a method *KMA* [5] to organize edge devices into a

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

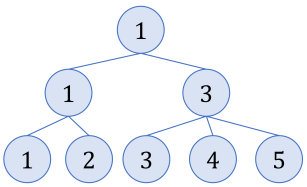
TABLE 3: Mathematical Notations in This Paper.

Notation	Description
H	The number of levels in a E-Tree structure;
N	The number of edge devices in the physical network;
$C^{p,h}$	Sets of child nodes connecting to parent node p in the h-th level;
D_i	Local datasets of edge device i;
$D^{p,h}$	Total number of aggregated dataset under parent node p in the h-th level;
$F(\boldsymbol{w})$	Global loss function;
$F_i(\boldsymbol{w})$	Local loss function for the edge device i;
k_i	The k-th local update performed by edge device i;
w^T	Final model parameters obtained at the end of learning process;
w^t	Global model parameters in round t;
w^*	True optimal model parameters obtained at the end of learning process;
$w_i(k_i)$	The local model parameters on the node i after k-th local updates;
$w^p_{i,h}$	A temporary variable that stores updated parameters of tree node i at the h-th level;
T	The total number of rounds performed in model training;
$t^{j,h}_{last}$	The time when node j at the h-th level receives the first parameter from its child node;
$t^{j,h}_{first}$	The time when node j at the h-th level receives the last parameter from its child nodes;
t^{min}	The minimum value of time required for one local update;
$\tau^p_{i,h}$	Aggregation frequency on the node i at the h-th level connecting to parent node p;
$t^{i,p}_{comp}$	The averaged time for one local update of edge device i connecting to edge device p in the pre-training process;
$t^{i,p}_{comm}$	The transmission time between edge device i and p;
$t^{i,h}_{res}$	The sum of time required for performing one local update or model aggregation of edge device i at the h level and transmission time between edge device i and its parent node;
$t^{j,p}_{h^{st}max}$	the time straggler j connecting to parent node p at the h level spent in local computations or model aggregations and model transmission;

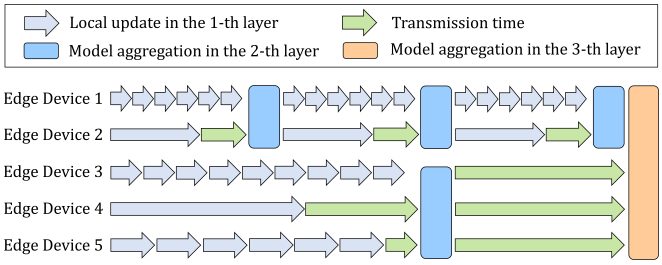
well-designed tree structure, where the leaf nodes represent the *workers* and the non-leaf nodes represent *aggregation nodes*. Assume that there is a physical network topology, including N edge devices. E-Tree structure is a logic aggregation topology built on top of the network topology. The process of *KMA* is as follows. We first use K-Means grouping algorithms based on network topology and data distributions to divide the edge devices in the same level onto different groups for model aggregation. It is noted that *KMA* follows the grouping principle [5], [23] that both the difference in data distributions between each group and global edge devices, and the communication delay of each edge device in the same group are simultaneously minimized. Then, we find the center node $n_{k_{m,h}}$ of each group $k_{m,h}$ as aggregation node by

$$n_{k_{m,h}} = \arg\min_{i \in k_{m,h}} \sum_{j \in k_{m,h}, j \neq i} d_{i,j}, \tag{3}$$

where $k_{m,h}$ denotes the index of the m-th group in the h-th level, $d_{i,j}$ denotes the communication delay between edge device i and j. Repeatedly, we conduct devices grouping and find the center node of each group recursively done layer by layer following a bottom-up approach until finding out the root node.



(a) A three-level E-Tree structure.



(b) A round of model training in E-Tree learning.

Fig. 3: An example of model training in E-Tree structure.

Performing Model Aggregation. Assume that our E-Tree was built by *KMA* with H levels and N edge devices at the 1-th level. Fig. 3a shows an example of a three-level E-Tree learning structure. We adopt distributed SGD algorithms for E-Tree learning framework. The learning process follows a bottom-up approach, including local update steps and model aggregation steps. We firstly denote $\tau^p_{i,h}$ as the *aggregation frequencies* on the tree node i at the h-th level connecting to its parent node p. In terms of local update steps, the leaf node i trains for $\tau^p_{i,1}$ local updates locally in parallel to minimize the local loss function. Then, the leaf nodes upload the model parameters to its parent node p at the 2-th level. For model aggregation steps, after the non-leaf node p receives the model parameters from all its child nodes, it computes the average and sends the results back to their descendant nodes. We name the above process by one model aggregation step. When the aggregation nodes (except for root node) have performed $\tau^p_{i,h}$ $(i < h < H)$ times of model aggregation, they send the updated parameter to their parent node. The aggregations are recursively done from the bottom to the top of the tree structure. Besides, when the root node in E-Tree finishes a global aggregation, the results are transmitted back to its descendants. We say *a round of model training* is completed.

We assume that each node spends a different time performing one local update or one model aggregation due to heterogeneous computation capabilities and the different sizes of local datasets for model training. Moreover, in the E-Tree structure, each child node connecting to the same parent node has different local updates or aggregation frequencies. Fig. 3 shows an example of model training in the E-Tree learning. The structure in Fig. 3a has five leaf nodes as workers and other non-leaf nodes as aggregation nodes. We show a round of model training in the E-Tree learning structure in Fig. 3b. Due to heterogeneous resources, leaf nodes 1 to 5 train for different times local updates before uploading the model parameters to their parent nodes 1 and 3 at the 2-th level. The aggregation frequency of the five edge devices are 6, 1, 9, 1, and 6 respectively in Fig. 3b. The transmission time of leaf nodes 1 and 3 is negligible because their parent nodes are themselves. For

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

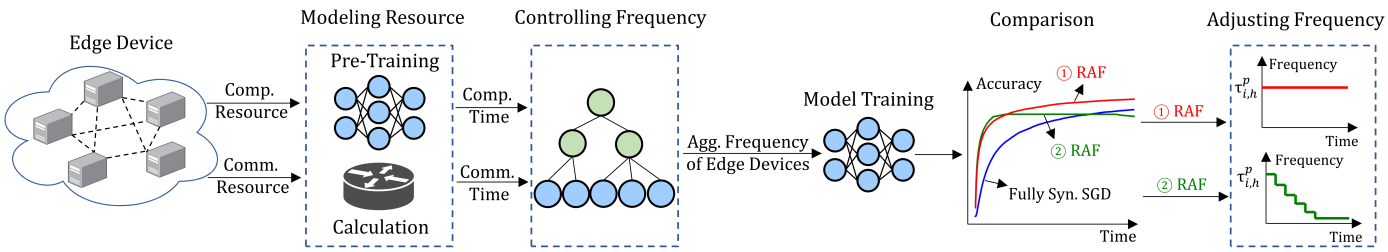


Fig. 4: The Overview Diagram of RAF.

model aggregation steps, aggregation nodes 1 and 3 at the 2-th level perform different times of model aggregation before sending their model parameters to the root node 1. The aggregation frequency of the nodes at the 2-th level is 3 and 1 respectively in Fig. 3b. Then, root node 1 performs global aggregation and sends the updated model parameters to all its descendant nodes. We name the above process by one round model training.

3.2 Problem Formulation

We denote $\boldsymbol{w}_i(k_i)$ as the local model parameters on the edge device i after k_i local updates. At $k_i = 0$, the local parameters for all edge devices are initialized to the same value. The evolution of local model parameters $\boldsymbol{w}_i(k_i)$ is as follows:

When $(k_i \bmod \tau^p_{i,1}) \neq 0$, the edge device i at the 1-th level performs local updates and its local model parameters are updated as:

$$\boldsymbol{w}_i(k_i) = \boldsymbol{w}_i(k_i - 1) - \eta_{k_i} \nabla F_i(\boldsymbol{w}_i(k_i - 1)). \tag{4}$$

When $(k_i \bmod \tau^p_{i,1} = 0) \cap (k_i \bmod (\tau^p_{i,1}\tau^{p_1}_{p,2}) \neq 0)$, the edge device p at the 2-th level performs model aggregation and the aggregation is done by:

$$\boldsymbol{w}^{p_1}_{p,2} = \frac{\sum_{i \in C^{p,2}} |D_i| [\boldsymbol{w}_i(k_i - 1) - \eta_{k_i} \nabla F_i(\boldsymbol{w}_i(k_i - 1))]}{|D^{p,2}|}, \tag{5}$$

where $C^{p,h}$ denotes the set of child nodes connecting to the parent node p at the h-th level and $|D^{p,h}|$ denotes the total size of the datasets on all the child nodes of the node p at the h-th level. We denote $\boldsymbol{w}^p_{i,h}$ as the updated model parameter of edge device i at the h-th level connecting to parent node p, which has performed model aggregation. Then the aggregation node p at the 2-th level transmit the results back to its child nodes and updates the model parameter $\boldsymbol{w}_i(k_i)$ of its child nodes by $\boldsymbol{w}^{p_1}_{p,2}$.

When $(k_i \bmod (\tau^p_{i,1}\tau^{p_1}_{p,2} \cdots \tau^{p_{h-3}}_{p_{h-3},h-1}) = 0) \cap (k_i \bmod (\tau^p_{i,1}\tau^{p_1}_{p,2} \cdots \tau^{p_{h-1}}_{p_{h-2},h}) \neq 0) \cap (h \neq H)$, the edge device p_{h-2} at the h-th level performs model aggregation and the model parameters are aggregated as follows:

$$\boldsymbol{w}^{p_{h-1}}_{p_{h-2},h} = \frac{\sum_{i \in C^{p_{h-2},h}} |D^{i,h-1}| \boldsymbol{w}^{p_{h-2}}_{i,h-1}}{|C^{p_{h-2},h}|}. \tag{6}$$

For each leaf node whose ancestor node is p_{h-2} at the h-th level, their model parameters were updated by $\boldsymbol{w}^{p_{h-1}}_{p_{h-2},h}$.

When $k_i \bmod (\tau^p_{i,1}\tau^{p_1}_{p,2} \cdots \tau^{p_{H-2}}_{p_{H-3},H-1}) = 0$, the root node p_{H-2} at the H-th level performs global aggregation and sends the aggregated model parameters to all its descendants. The model parameter after a complete round of aggregation is as follows:

$$\boldsymbol{w}^t = \frac{\sum_{i=1}^{N} |D_i| [\boldsymbol{w}_i(k_i - 1) - \eta_{k_i} \nabla F_i(\boldsymbol{w}_i(k_i - 1))]}{|D|} \tag{7}$$

where w^t denotes the global model parameters in round t. For each leaf node, their local model parameters were updated by $\boldsymbol{w}^t$.

We use T to denote the total number of rounds we perform in model training. We define the final model obtained after T rounds of training by

$$\boldsymbol{w}^T \triangleq \underset{\boldsymbol{w} \in \{\boldsymbol{w}^t : t = 0,1,2,\cdots,T\}}{\arg\min} F(\boldsymbol{w}). \tag{8}$$

Resource heterogeneity is a particularly significant feature in edge computing, which impacts the model training performance. The resources in consideration mainly include computation and network bandwidth. The resource capability is measured by the time that a local update or model aggregation takes. To shorten the waiting time and maximize the computation time, we constrain that the time when the parent node receives model parameters from all its child nodes in a model aggregation step is the same as possible. We define time constraint as

$$t^{j,h}_{last} - t^{j,h}_{first} < t^{min}, \forall j \in [1, N], \forall h \in [2, H], \tag{9}$$

where $t^{j,h}_{first}$ denotes the time when aggregation node j at the h-th level receives the first model parameters from its child nodes in one model aggregation step, $t^{j,h}_{last}$ denotes the time when aggregation node j at the h-th level receives the last model parameters from its child nodes in one model aggregation step, and t^{min} denotes the minimum value of time required for one local update among edge devices connecting to the same parent node.

We consider the learning problem under a fixed resource condition and find out the optimal values of $\tau^p_{i,h}$ in E-Tree learning so that the global loss function is minimized:

$$\begin{aligned} &\min_{\tau^p_{i,h}, \forall p, i \in [1,N], \forall h \in [1,H]} F(\boldsymbol{w}^T) \\ &\text{s.t. } t^{j,h}_{last} - t^{j,h}_{first} < t^{min}, \forall j \in [1, N], \forall h \in [2, H]. \end{aligned} \tag{10}$$

4 RAF: A Resource-based Aggregation Frequency Optimization

4.1 Method Overview

In this section, we present the aggregation frequency controlling algorithm termed as RAF for E-Tree Learning, which optimizes $\tau^p_{i,h}$ of each edge device to minimize the global loss function. The process of RAF is shown in Fig. 4. Suppose those edge devices are equipped with heterogeneous computation resources and are connected through a physical mesh network. The first step of RAF is resource modeling. We represent resources using computation and communication time, which are captured by pre-training

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

Algorithm 1 RAF Algorithm

Input: tree structure S, computation time of edge devices $t^{i,p}_{comp}$, transmission time of edge devices $t^{i,p}_{comm}$;
Output: the frequencies $\tau^p_{i,h}$ of all devices and aggregation nodes in S;

1: **for** each group of edge devices at the 1-th level **do**
2: Compute the time required for performing one local update and transmitting the model to parent node p for each edge device $t^{i,1}_{res} = t^{i,p}_{comp} + t^{i,p}_{comm}$;
3: Select the node k with maximum time value $t^{k,p}_{1^{st}max}$ from $t^{1,1}_{res}, t^{2,1}_{res}, \cdots, t^{i,1}_{res}$;
4: Set the frequency of local update for the selected node $\tau^p_{k,1} = 1$;
5: **for** edge devices other than node k **do**
6: Compute the frequency of local update $\tau^p_{i,1} = \lfloor \frac{t^{k,p}_{1^{st}max} - t^{i,p}_{comm}}{t^{i,p}_{comp}} \rfloor$;
7: **end for**
8: **end for**
9: **for** each group of aggregation nodes at the h-th level **do**
10: Compute the time required for performing one model aggregation and transmitting the model to parent node p_1 for each aggregation node $t^{j,h}_{res} = t^{i,j}_{h-1^{st}max} + t^{j,p_1}_{comm}$;
11: Select the node m with maximum time value $t^{m,p_1}_{h^{st}max}$ from $t^{1,h}_{res}, t^{2,h}_{res}, \cdots, t^{j,h}_{res}$;
12: Set the frequency of model aggregation for the selected node $\tau^{p_1}_{m,h} = 1$;
13: **for** aggregation nodes other than node m **do**
14: Compute the frequency of model aggregation $\tau^{p_1}_{j,h} = \lfloor \frac{t^{m,p_1}_{h^{st}max} - t^{j,p_1}_{comm}}{t^{i,j}_{h-1^{st}max}} \rfloor$;
15: **end for**
16: **end for**
17: Repeat Line 10-14 to update the frequency of aggregation nodes at the other levels;
18: Set the aggregation frequency for the root node $\tau^p_{i,H}$.

and calculation. Next, we compute the aggregation frequencies of each edge device recursively from the bottom up. Then, we train the model with the input aggregation frequencies. If the training performance of RAF is better than the benchmark method, i.e., fully synchronous SGD, we output the final aggregation frequency of each edge device and train the model with fixed aggregation frequencies. Otherwise, we adopt a method that adaptively adjusts aggregation frequencies during the model training.

4.2 Resource Modeling and Cost Profiling

In the RAF, the first step is to collect the resource information of all the edge devices. Because it is impractical to capture the accurate resource information directly in a real environment, we map the resources of edge devices into computation time or transmission time by pre-training and calculation. For computation resources, we consider CPU power and data sample size as the major factor that affects the local computation time. To measure the computation resource of edge devices, we obtain the averaged time required for one local update of each edge device in the pre-training process and suppose that the parameters required for pre-training, including batch size, learning rate, and so on, are all set in advance. Denote $t^{i,p}_{comp}$ as the averaged time for one local update, also called computation time, of edge device i connecting to edge device p. For communication resources, constrained access network bandwidth is the main factor affecting the transmission time. Assume that there is a physical network topology with N edge devices. The link in the topology represents the network connections between the edge devices, and it has a weight denoting the communication bandwidth. Besides, the topology is not necessarily a completely connected network, i.e., edge devices may need to communicate with other nodes in greater than or equal to one hop. Equation (11) shows how to model the transmission time between two edge device at a connected link:

$$t^{i,p}_{comm} = \frac{d}{B_{i,j}}, \tag{11}$$

where $B_{i,j}$ denotes as the network bandwidth between two connected edge devices i and j, and d denotes as the model size. We assume that the size of the model is the same among all the edge devices. Thus, we obtain the computation time of each edge device at the 1-th level and transmission time of any two devices according to the above methods.

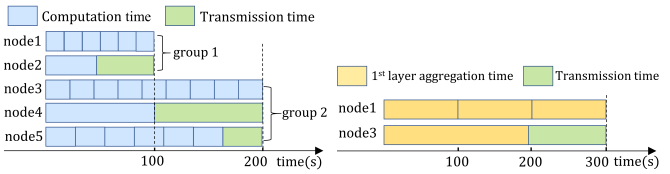
4.3 Controlling Aggregation Frequency for E-Tree

After measuring the resources among edge devices, we further optimize the frequencies of local update and model aggregation level by level from bottom to up in this subsection. The process is shown in Algorithm 1.

In E-Tree learning, edge devices connecting to the same parent node p are grouped for aggregation. Leaf nodes perform local updates, and non-leaf nodes perform model aggregations. The step of optimizing the frequency of local updates of leaf nodes include lines 2-6 of Algorithm 1, while lines 10-14 optimize the frequency of model aggregations of non-leaf nodes. For each level in the E-Tree structure, we first compare the available resource among the edge devices within the same group. Since we map the resource into time in Section 4.2, the resources of leaf node i refer to the time required for performing one local update of leaf node i and transmission time between leaf node i and its parent node. It is noted that when an edge device's parent node is itself, the value of transmission time is recorded as 0. For the aggregation node j at the h-th level $(1 < h < H)$, the resources refer to the time required for one $(h-1)^{st}$ level aggregation and transmission time between aggregation node j and its parent node. The time required for one $(h-1)^{st}$ level aggregation is the sum of aggregation time of aggregation node j and the time that the slowest child node of aggregation node j spends in performing responding local updates or model aggregations and transmitting the model parameters to aggregation node j. Because the aggregation nodes take a short time to aggregate, we set the aggregation time to 0. We calculate the sum of computation time (the time of performing one local update or model aggregation) and communication time of edge devices at each level and name the results as the time of resource mapping $t^{i,h}_{res}$ of edge device i at the h-th level in line 2 and line 10.

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing



(a) Compute the local updates. (b) Compute the model aggregation frequency.

Fig. 5: An example of controlling aggregation frequencies in E-Tree.

We then find out the stragglers in each group in line 3 and line 11. The stragglers are the nodes that spend the longest time on local computation and model transmission in comparison with other nodes in the same group. We denote $t^{i,p}_{h^{st}max}$ as the time straggler i connecting to parent node p at the h level spent in local computations and model transmission. To reduce the negative impact of these stragglers on the training performance of their group, we set the aggregation frequencies of these nodes as the value of 1 in line 4 and 12. Besides, the value of aggregation frequencies of other edge devices in the same group is calculated in line 6 and 14. To make full use of the available resource of edge devices, these devices in the same group can perform more computations than the stragglers. The calculation is recursively done level by level following a bottom-up approach. At last, the frequency of the root node is determined manually in line 18. We say a round of model training is completed when the root node finishes model aggregation and sends the results back to all its descendants.

Fig. 3a shows a three-level E-Tree structure, and we use this structure as an example for computing the aggregation frequencies of all the tree nodes. Fig. 5a shows the process of computing the local updates of edge devices at the 1-th level. As shown in Fig. 5a, the times of local update of the slowest node 2 in the group 1 are set to 1, and the faster nodes 1 can perform more local updates than node 2 but the total time $t^{1,1}_{comp} * 6$ cannot exceed the total time value of straggler 2. Besides, the total time for local computation and model transmission varies among different groups in Fig. 5a. Because not only do edge devices in the same group have different resources, but the resources owned by different groups are also different. Fig. 5b shows the process of calculating the model aggregation frequencies of edge devices at the 2-th level. Because node 2 at the 1-th level connecting to parent node 1 is the straggler in their group, the 1^{st} level aggregation time of aggregation node 1 at the 2-th level are set to $t^{2,1}_{1^{st}max}$, i.e., the time straggler node 2 at the 1-th level spent in performing local computation and transmitting model parameter to its parent node 1. In Fig. 5b, the times of model aggregation of the slowest node 3 is set to 1, and that of the faster node 1 is set to 3.

4.4 RAF with Adaptive Aggregation Frequency

When the available resources among edge devices differ significantly, the aggregation frequencies calculated by algorithm 1 on edge devices have large discrepancies. Therefore, it may result in an inferior convergence. We observe through extensive experiments that a large difference of aggregation

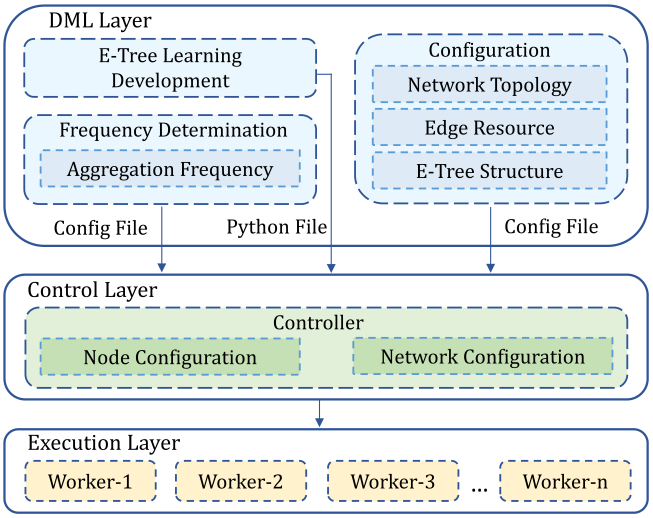


Fig. 6: The architecture of our self-developed testbed.

frequencies among edge devices have a faster accuracy increase at the early phase of training but results in an inferior model accuracy as the model reaches closer to convergence. This observation is demonstrate by the experiment result in Fig. 11. Because individual models of each edge device are trained at local datasets with different data distributions, faster nodes with larger aggregation frequencies introduce residual error as opposed to fully synchronous SGD and deviates the model too much.

Thus, RAF dynamically adjusts the aggregation frequencies at different time phases during the model training process, when the training performance with aggregation frequencies computed by algorithm 1 is worse than the benchmark method, i.e., fully synchronous SGD. The basic idea is to choose the best aggregation frequencies that maximize the accuracy at each fixed time interval. The process of our method is as follows. Firstly, we determine the initial time point to adjust the aggregation frequencies based on the observed training performance. We choose a specific point in time t_0 when the difference of model accuracy between RAF and fully synchronous SGD begins to be less than a certain threshold. Secondly, we determine the time interval T_0 that how often edge devices change the aggregation frequencies. The time interval T_0 can be adjusted manually. Thirdly, we adopt an update rule of aggregation frequencies from [24] to calculate the updated aggregation frequencies of each edge device. The update rule of each edge device is as follows:

$$\tau_n = \left\lceil \sqrt{\frac{\eta_{t_0}}{\eta_{t_0+nT_0}} \frac{F(\boldsymbol{w}_{t=t_0+nT_0})}{F(\boldsymbol{w}_{t=t_0})}} \tau_{t_0} \right\rceil, \tag{12}$$

where η_t represents the learning rate at the time point t, $F(\boldsymbol{w}_{t=t_0})$ represents the training loss at the time point t_0, τ_{t_0} represents the resulting aggregation frequency based on RAF of each edge device and τ_n represents the n-th updated aggregation frequency in the training process. When the learning rate is fixed, the above update rule tends to decrease the aggregation frequency of each edge device with respect to real time to achieve a trade-off between making full use of resources and introducing additional errors to

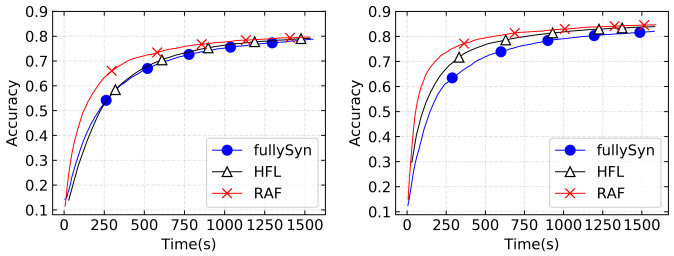
1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

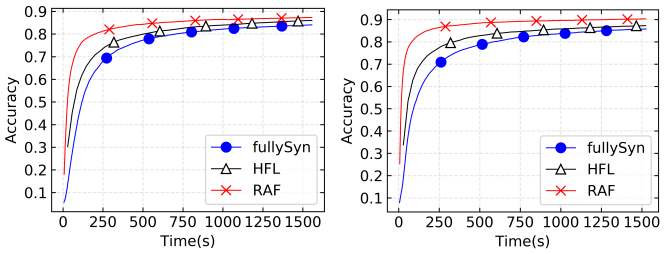
TABLE 4: Accuracy of Various Methods with Different Data Distributions after a Given Time.

	fullySyn	HFL	RAF
1-Class Non-IID	78.74%	79.14%	79.61%
2-Class Non-IID	82.76%	84.39%	85.07%
5-Class Non-IID	84.35%	86.15%	87.41%
IID	85.75%	87.25%	90.29%

(a) Average round time.

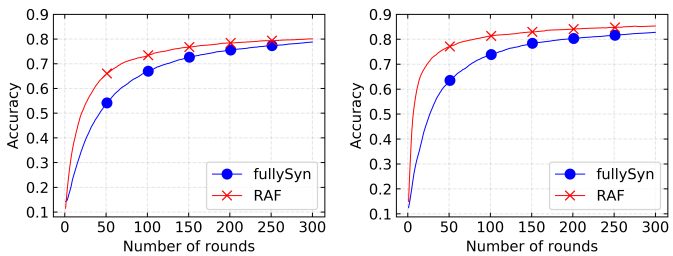


(a) 1-class non-IID data. (b) 2-class non-IID data.

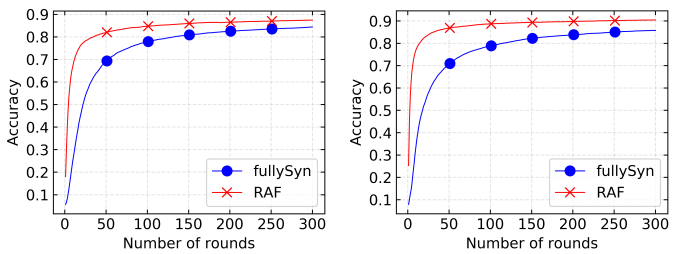


(c) 5-class non-IID data. (d) IID data.

Fig. 7: Comparisons of convergence speed with different data distributions for the LR model on MNIST.



(b) 1-class non-IID data. (c) 2-class non-IID data.



(d) 5-class non-IID data. (e) IID data.

Fig. 8: Comparisons of convergence quality with different data distributions for the LR model on MNIST.

the local model. Besides, it is noted that we only adjust the frequencies of local updates on edge devices at the 1-th level rather than the frequencies of model aggregations. Through extensive experiments, we found that adjusting the aggregation frequencies of model aggregations on aggregation nodes will not solve this problem.

5 Experimental Results

We have conducted experiments to evaluate our proposed algorithm RAF on our self-developed edge computing testbed with multiple datasets and models, different data distributions, a varying number of edge devices, and different model training structures.

5.1 Testbed

Fig. 6 depicts an overview of the architecture of our testbed. The implementation of the testbed includes 4000+ lines of codes. It is available on the https://github.com/Lin-1997/Edge-TB. The architecture is divided into three layers: Distributed Machine Learning (DML) layer, control layer, and execution layer. The DML layer provides libraries for us to develop DML. For example, the E-Tree learning needs the collaboration of edge devices to train a machine learning model, where the leaf nodes represent *workers* for model training and non-leaf nodes represent *aggregators* for model aggregation. The DML layer encapsulates specific function modules and execution environments, including package dependencies and Docker images for the roles of workers and aggregators. Next, we need to specify the test environment in the configuration files, including network topology, edge resource, and E-Tree structure. Besides, we use RAF to determine the optimal aggregation frequency of each edge device and generate a frequency configuration file. The configuration files serve as input to the control layer.

Our testbed adopts the Controller-Worker architecture, which allows for automating deployment and scaling. The architecture is implemented using Kubernetes (K8s) [25] and Docker [26]. The controller is responsible for receiving input from the DML layer and parsing them into specific deployment commands. The commands include node configuration and network configuration, which set up the execution environment on the workers in the execution layer. Then, the controller transmits these commands to the workers via the network. In the execution layer, the workers deploy the input commands to generate the target test environment. We use Docker containers to emulate multiple heterogeneous edge devices, which can support running python and commonly used machine learning libraries such as TensorFlow [27]. To emulate the resource heterogeneity, since the Docker engine uses the Linux Control Groups to manage the allocation of CPU resources of workers, we just need to specify how many CPUs are allocated to each worker in the configuration file at the DML layer. In addition, for the implementation of the network communication, we emulate the transmission delay by suspending execution of the calling thread for the given time through the python function. The experiments ran on two physical servers. One

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

TABLE 5: Accuracy of Various Methods with a Varying Number of Edge Devices after a Given Time.

	fullySyn	HFL	RAF
20 Edge Devices	83.77%	86.43%	87.76%
50 Edge Devices	78.14%	84.83%	84.81%
100 Edge Devices	73.65%	82.46%	84.32%

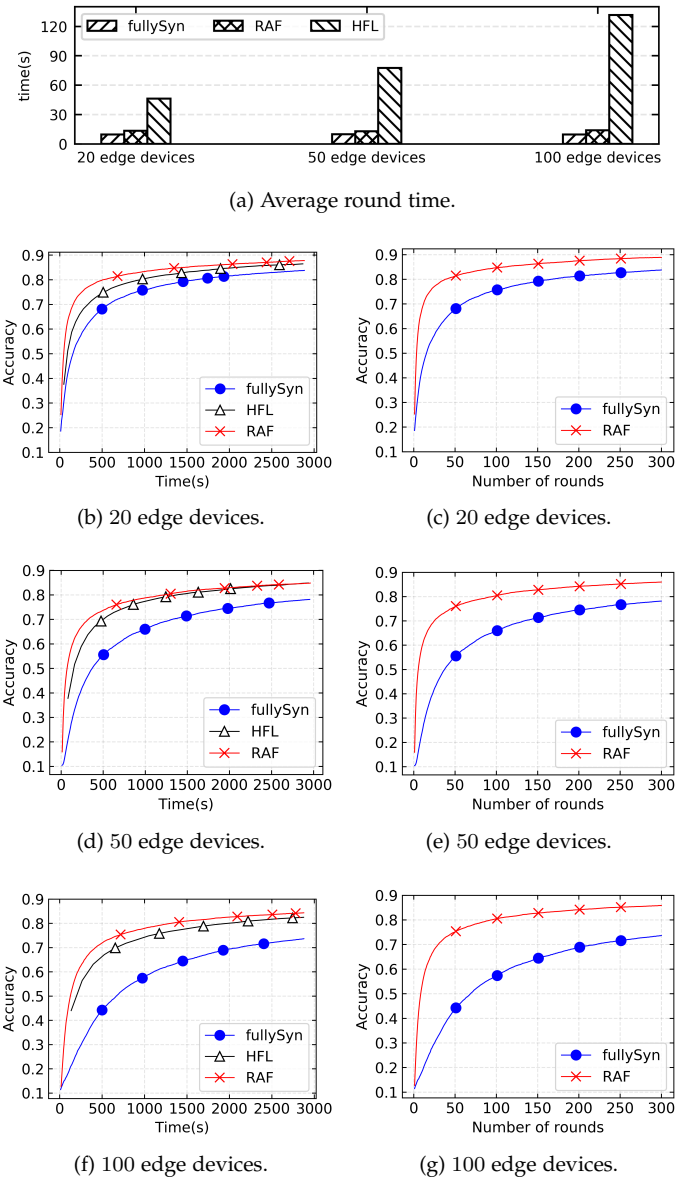
is a 6-core 12-thread computer with a frequency of 2.9 GHz and 8 GB RAM, and the other is a 64-core 128-thread computer with a frequency of 2.9 GHz and 256 GB RAM. The former computer acts as a controller in the control layer and the latter as workers in the execution layer.

5.2 Experiment Setup

Models and Datasets. We evaluate the training of three different models on three different datasets. The models include logistic regression (LR), multilayer perceptron (MLP), and convolutional neural networks (CNN). LR is trained on MNIST [28] dataset, which is a set of 10-class hand-written digits with $60,000$ training sets and $10,000$ test sets. MLP with a two-layer multilayer perceptron is also performed on MNIST. CNN is trained on two different datasets, including Fashion-MNIST [29] and EMNIST Balanced (EMNIST) [30]. The CNN is modeled with three convolution layers and two fully connected layers. Fashion-MNIST is a 10-class fashion product dataset with 28×28 grayscale images. The full dataset is partitioned into $60,000$ training images and $10,000$ test images. EMNIST is a set of handwritten character digits consisting of $112,800$ training sets and $11,800$ test sets. Each example is a 28×28 pixel image associated with a label from 47 balanced classes. We employ mini-batch Stochastic Gradient Descent (SGD) with batch size 32, and a learning rate 0.01 in each experiment.

Data Distribution and Data Quantity Setup. The setup of data distributions is similar to the paper [31], which sorts the data sample by classes. In terms of IID setting, each edge device is randomly assigned a uniform distribution over 10 classes in MNIST and Fashion-MNIST, 47 classes in EMNIST. For the n-class non-IID setting, we assign n different classes of datasets for each edge device. We consider different cases of non-IID settings in the experiment. Besides, we emulate data quantity heterogeneity by assigning varying data samples onto edge devices under a non-IID setting. The data in each edge device ranges from 550 to 4000 for MNIST, 600 to 3000 for Fashion-MNIST, and 750 to 900 for EMNIST.

Heterogeneous Resource Setup. The network topology is randomly generated, and the probability of direct connection between edge devices is 0.2. We set the upper and lower bandwidths for direct links between edge devices. The bandwidth of each link ranges from 0.1 Mbps to 0.3 Mbps for the LR model on MNIST, 2.4 Mbps to 5.6 Mbps for the MLP model on MNIST, and 1.6 Mbps to 4.0 Mbps for the CNN model on Fashion-MNIST and EMNIST. To emulate the resource heterogeneity, we allocate different CPU cores to different edge devices. Each edge device is assigned with CPU ranging from 0.15 CPU to 4 CPUs for the LR model on MNIST, 0.15 CPU to 3 CPUs for the MLP model on MNIST, and 0.3 CPU to 12 CPUs for the CNN model on Fashion-MNIST and EMNIST.



(a) Average round time.

(b) 20 edge devices.

(c) 20 edge devices.

(d) 50 edge devices.

(e) 50 edge devices.

(f) 100 edge devices.

(g) 100 edge devices.

Fig. 9: Performance comparisons with a varying number of edge devices for the MLP model on MNIST.

Comparison Methods and Metrics. We compare the performance of different synchronization methods in the hierarchical model training structure. For simplicity, we construct the tree-based model training structure according to network distance. The number of edge devices in the structure are 20, 50, and 100, respectively in the different experimental settings. The synchronization methods are adopted in the above tree-based structure. The process of model training and aggregation follows the definition of Section 3. We compare with the following approaches: fully synchronous SGD in E-Tree (*fullySyn*), HFL in E-Tree (*HFL*), RAF in E-Tree (*RAF*), RAF with adaptive frequency adjustment (*AdaptiveRAF*). For *fullySyn*, the value of aggregation frequencies of each edge device are set to 1. For *HFL*, the aggregation frequencies of edge devices in the same level are the same and greater than or equal to 1. The setups of aggregation frequencies are different in the different model training structures. The aggregation frequencies of edge devices are set to 5 at the 1-th level for two-level structure,

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

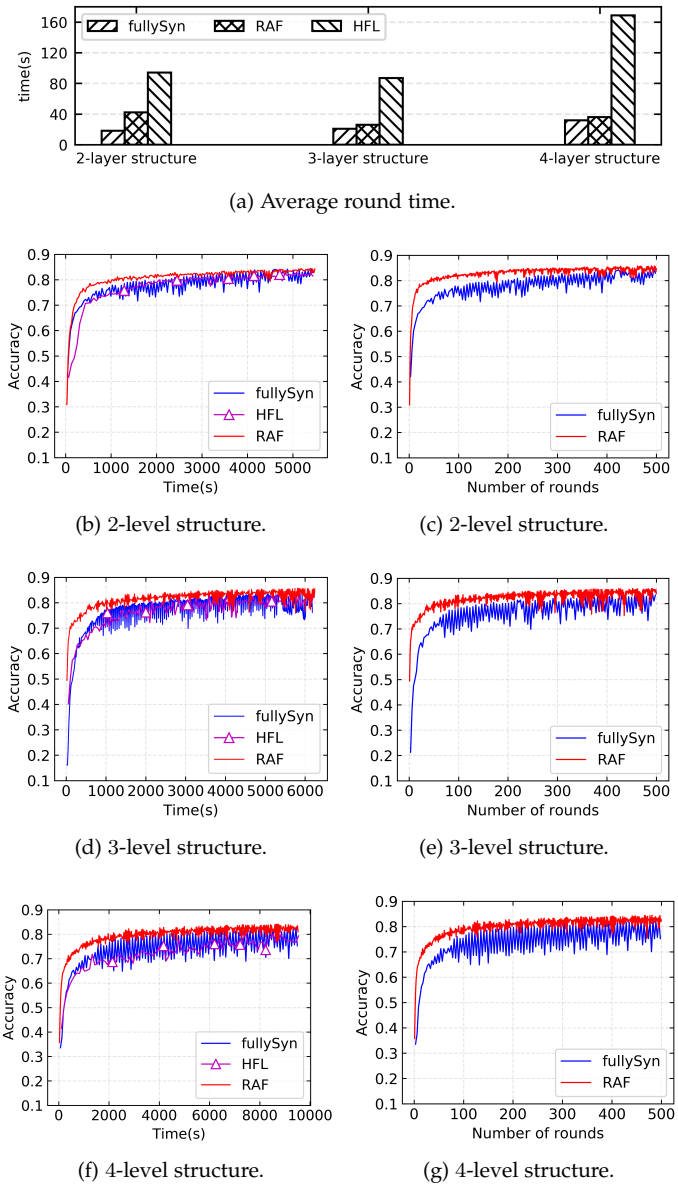
1 at the 1-th level and 5 at the 2-th level for three-level structure, 1 at the 1-th level and 5 at the 2-th level and 2 at the 3-th level for four-level structure. For *RAF*, we adopt our aggregation frequency controlling algorithm that each edge device on a different level has varying aggregation frequency. For *AdaptiveRAF*, the initial aggregation frequencies are the same as *RAF*. Then, the aggregation frequencies are adjusted dynamically according to the observed training loss during the training.

We define three performance metrics. Firstly, we measure the convergence speed by using the training time each method takes to reach a given training accuracy. Secondly, convergence quality is also another metric that measures the training rounds to reach the convergence. Thirdly, we measure the average round time of different synchronization methods in the model training.

5.3 Evaluation of RAF Method

Different Data Distributions. We evaluate the performance of RAF under different data distributions. We trained the LR model on MNIST with 20 edge devices. Fig. 7(a)-(d) show the comparisons of convergence speed under 1-class non-IID, 2-class non-IID, 5-class non-IID and IID setting. These results show that ***RAF* achieves much better accuracy compared to *fullySyn* and *HFL*** if training time is constraint under all the non-IID levels. As shown in Table 4, **0.91%, 2.31%, 3.06% and 4.54% accuracy increase are obtained by *RAF* over *fullySyn*** respectively under 1-class non-IID, 2-class non-IID, 5-class non-IID and IID setting. The reason why *RAF* and *HFL* have a faster convergence speed and higher final accuracy than *fullySyn* is that hierarchical model training structures use localized aggregations, i.e., edge devices can perform model aggregations in parallel. However, due to the features of strong synchronization, *HFL* has to spend much time waiting for the stragglers. On the contrary, *RAF* allows faster nodes in the same group to perform more local updates and/or model aggregations in a given time. This improves the utilization of resources. Thus, *RAF* results in faster convergence speed and higher final accuracy than *HFL*.

For the training time comparison in Fig. 8a, **the difference of average round time between *RAF* and *fullySyn* are very small, i.e., less than 0.6s after 300 rounds**. Compared with *fullySyn* and *RAF*, *HFL* spends almost 5 times longer in training time. This is because the stragglers affect the time required for a round. *RAF* and *fullySyn* limits the aggregation frequency of the stragglers to 1, which can limit the negative effect of the straggler on every round of training. However, in the *HFL*, the aggregation frequencies of stragglers are allowed to be greater than 1. Increasing the aggregation frequencies of the stragglers can enlarge the time required for a round. This leads to a long round time of *HFL*. Besides, we only compare the convergence quality between *RAF* and *HFL*, because their average round time is close. For accuracy comparison, Fig. 8(b)-(e) shows that ***RAF* has relatively obvious increase compared to *fullySyn* after 300 rounds** because *RAF* performs more local iterations than *fullySyn* in each round. Thus, RAF improves not only the convergence speed but also convergence quality under different data distributions.



(a) Average round time.

(b) 2-level structure.

(c) 2-level structure.

(d) 3-level structure.

(e) 3-level structure.

(f) 4-level structure.

(g) 4-level structure.

Fig. 10: Performance comparisons with varying model training structures for the CNN model on Fashion-MNIST.

A Varying Number of Edge Devices. We evaluate the performance of RAF using the MLP model on MNIST for the number of edge devices varying from 20 to 100 under 5-class non-IID setting. The results are demonstrated in Fig. 9. As expected, the average round time shown in Fig. 9a is similar to Fig. 8a, which *HFL* achieves up to 13 times longer than *fullySyn*. Note that the average round time varies between different numbers of edge devices due to different heterogeneous resource setups. **If we look at the convergence speed, *RAF* takes less time to reach a given training accuracy than *fullySyn* and *HFL***, see Fig. 9 (column 1). As shown in Table 5, **the accuracy of *RAF* increases up to 3.99%, 6.67%, and 10.67% respectively if training time is constraint**, under 20 edge device, 50 edge device, and 100 edge devices setting in comparison to *fullySyn*. These results show that as the number of edge devices increases, the difference of convergence accuracy between *RAF* and *fullySyn* also increases. This is because *fullySyn* incurs serious synchronization delay, especially when the

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

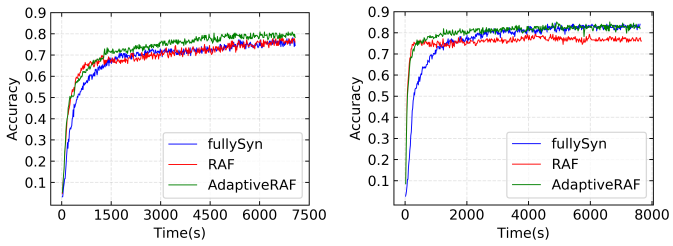
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

number of edge devices is large. Communication at each iteration can be expensive and become the performance bottleneck in bandwidth-limited edge environments. In the *fullySyn*, edge devices speed much time in transmission and waiting rather than doing local computation. Thus the ratio of computation to communication time is low. We can observe in Fig. 9 (column 1) and Table 5 that the convergence speed and final accuracy of *fullySyn* gradually decrease as the number of edge devices increases. In contrast, *RAF* and *HFL* can tackle this problem by letting edge devices perform more local updates and model aggregations before transmitting the model parameters, which increase the compute to communication ratio. The reason why *RAF* outperforms *HFL* is that *RAF* lets faster nodes make full use of resources instead of waiting for the stragglers in the same group. Thus, the final accuracy and convergence speed of *RAF* is better than *fullySyn* and *HFL*. **For the convergence quality comparison, *RAF* achieves much better accuracy compared to *fullySyn*** because *RAF* does more local computation in a similar round time.

Different Model Training Structures. We validate the effectiveness of RAF using CNN model on Fashion-MNIST in the 2-level, 3-level and 4-level model training structures with 20 edge devices. As expected, the average round time shown in Fig. 10a is similar to Fig. 9a. However, in the 2-level structure, the average round time of *RAF* is a little longer than that of *fullySyn*. The reason is that in the *RAF*, the time that model parameters of each child node arrive at the aggregation node is the same as possible. This brings a large communication traffic jam on the aggregation node in the 2-level structure with a relatively large number of child nodes. This also takes a certain amount of time to schedule. Despite this, ***RAF* still has a greater performance gain over *fullySyn* and *HFL* under different model training structures in terms of convergence speed and convergence round**, see in Fig. 10 (column 1) and Fig. 10 (column 2) respectively. *RAF* also converges to a higher final model accuracy than *fullySyn* and *HFL*. Thus, the results illustrated that *RAF* also helps in different model training structures, including 2-level federated learning framework.

5.4 Evaluation of RAF with Adaptive Aggregation Frequency

To study the impact that when the aggregation frequencies among edge devices in the same group differ significantly on training performance, we train EMNIST on CNN model with 20 edge devices in a heterogeneous environment under 8-class non-IID and 15-class non-IID settings. As shown in Fig. 11, when there are large discrepancies between the aggregation frequencies on edge devices, the convergence curve of *RAF* rises rapidly in the initial stage of training but results in a lower final accuracy than that of *fullySyn*. This is because if the aggregation frequencies among edge devices differ greatly, the faster node will deviate from the model too much in extreme non-IID cases, which negatively impacts the convergence speed and final model accuracy. To achieve a better trade-off between training time and accuracy, we let each edge device train the model with a larger aggregation frequency at the beginning of training and gradually decrease it at the different phrase. Fig. 11



(a) 8-class non-IID data. (b) 15-class non-IID data.

Fig. 11: Comparisons of convergence speed with fullySyn, RAF, and AdaptiveRAF for the CNN model on EMNIST.

shows that **the convergence curve of *AdaptiveRAF* has a faster initial increase in the training accuracy and results in higher final accuracy than that of *RAF* and *fullySyn***.

6 Discussions

Optimization of Resource Modeling. Instead of directly modeling the resources of each edge device, we estimate the accurate computation and communication time for each edge device. Due to unstable wireless channels and inference, it is hard to accurately estimate the communication time between edge devices in wireless communication. One of the statistical methods is to measure the communication time between edge devices multiple times and then calculate the average value as the estimated communication time. This method is simple and can alleviate the uncertainty of communication time between edge devices. This method can also be applied to our method because RAF is currently based on a static edge computing environment. In our future work, we will design an algorithm to adapt to a dynamic edge computing environment considering the constrained and dynamically changing resources and training converge.

Performance Guarantee for Weak Synchronization. FedNova [32] provides a general theoretical framework to analyze the convergence of FL with heterogeneity in the number of local updates. It illustrates that heterogeneity in local updates slow down the convergence due to objective inconsistency. The paper [32] proposes a normalized averaging method FedNova, eliminating object inconsistency by providing a generalized update rule for updating the shared global model. Our work is to consider the objective inconsistency from the perspective of optimizing the aggregation frequencies to achieve a trade-off between maximizing resource utilization and preserving fast convergence. We adjust the number of local iterations of each edge device by finding out the optimal value of aggregation frequencies of each edge device (local update or model aggregation). We provide solutions for quantitatively determining the aggregation frequencies, which is one of the most challenging and important problem. Furthermore, we will continue to provide a theoretical analysis or a performance guarantee for hierarchical model training using weak synchronization in our future work.

Overhead of AdaptiveRAF. The *AdaptiveRAF* requires running *fullySyn* to make such a comparison. If the training performance of *RAF* is worse than *fullySyn*, the *AdaptiveRAF* starts another model training with adaptively adjusting the

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

aggregation frequency. People may argue that why not just run *fullySyn* instead of consuming twice the resources for training the same model. In fact, the time of running *fullySyn* is controllable. The purpose of running *fullySyn* is to find out the initial time point to adjust the aggregation frequency, that is, choose a specific point in time t when the difference of model accuracy between *RAF* and *fullySyn* begins to be less than a certain threshold. Therefore, we can terminate the *fullySyn* and start another model training if we find out the initial time point. For instance, it can be seen from fig. 11(a) that the accuracy of model training to achieve convergence is about 78%. The *fullySyn* takes about $7007.12s$ to achieve 78.1% training accuracy, while *AdaptiveRAF* takes about $4151.92s$ to achieve the same accuracy. Compared with *AdaptiveRAF*, *fullySyn* takes about $2855s$ more time to reach the same accuracy. Moreover, we find out the initial time point t (about $1000s$) of adjusting the aggregation frequency by running *fullySyn*. Then we can terminate the model training. This helps to save more time and resource to achieve the same accuracy compared to *fullySyn*.

7 Related Work

Communication efficiency is a significant issue of distributed model training. Most of the current works focus on *strong synchronization* in a centralized model training structure. These works demonstrate how to determine the optimal aggregation frequency (τ) for centralized model training structure from a theoretical perspective. For example, [4] has shown that for general strongly convex and smooth functions when T iterations are performed at N devices with a fixed mini-batch size, the linear speedup is attainable only with $O(\sqrt{NT})$ rounds of communication. Each device performs $\tau = O(\sqrt{T/N})$ local updates for each round without hampering the convergence. [33] improves upon the previous works by obtaining a linear speedup for strongly convex and smooth function with $\tau = O(T^{\frac{1}{2}}/N^{\frac{3}{2}})$, when data is identically distributed among devices. [33] also considers the case that data is heterogeneous among different devices. The optimal value of $\tau = O(T^{\frac{1}{4}}/N^{\frac{3}{4}})$ gives $O(1/\sqrt{NT})$ convergence speed for not necessarily strongly-convex function. [34] focuses on smooth and strongly convex functions with a very general noise model and proposes a communication method that requires only $\Omega(N)$ communication rounds to achieve a linear speedup. For non-convex optimization problems, [9] achieves $O(1/\sqrt{NT})$ convergence by averaging only every $\tau = O(T^{\frac{1}{4}}/N^{\frac{3}{4}})$ iterations. [35] shows that by choosing $\tau = O(T^{\frac{2}{3}}/N^{\frac{1}{3}})$, model averaging achieves a linear speedup for non-convex functions under Polyak-Lojasiewicz condition, i.e., only $O(N^{\frac{1}{3}}T^{\frac{1}{3}})$ communication rounds are sufficient to achieve asymptotic performance compared to previous work.

The value of τ among devices is fixed over different communication rounds for centralized model training structure in the above methods. However, recent works [34], [35] have shown that adjusting the value of τ over different communication rounds leads to better performance over fixed periodic averaging methods. Besides, considering the resource-constrained dynamic environments, [36] has proposed a method that dynamically adapts the value of τ every communication round based on a fixed resource budget under the theoretical convergence bound that has considered non-IID data distributions among devices. [37] has also proposed a method that adaptively adjusts the τ in real time based on deep reinforcement learning in given computation and communication energy budget.

For hierarchical model training, [38] has proven that local averaging has a benefit to model convergence. Furthermore, by reducing the high communication cost of global aggregation, it has also proposed that increases the value of τ at the aggregation node (except for the root node) can achieve better performance. However, the question of how to determine the optimal aggregation frequency for hierarchical model training with *weak synchronization* lacks solution.

8 Conclusion

In this article, we adopt week synchronization in a tree-based hierarchical model training framework named E-Tree learning and formalize the process of model training with varying aggregation frequencies in E-Tree. We propose an aggregation frequency controlling method termed RAF, which determines the aggregation frequency for all edge devices in E-Tree based on heterogeneous edge resources. To tackle the problem that aggregation frequencies among edge devices differ greatly may result in inferior training performance, we dynamically adjust the aggregation frequencies at different phases during the model training. The performance of RAF is evaluated via extensive experiments on our self-developed edge computing testbed. Evaluation results validate the benefits of RAF in different datasets and models, data distribution scenarios, number of edge devices, and model training structure. Future work can investigate how to optimize the aggregation frequencies in a dynamic edge computing environment.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grant 61972161, in part by Guangdong Basic and Applied Basic Research Foundation under Grant 2020A1515011496 and the Key Research and Development Program of Guangdong under Grant 2019B010154004, and in part by Hong Kong RGC General Research Fund under Grant PolyU 152133/18 and PolyU 15217919.

References

[1] Z. Zhou, X. Chen, E. Li *et al.*, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, pp. 1738–1762, 2019.

[2] H. McMahan, E. Moore, D. Ramage *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[3] T. Lin, S. Stich, M. Jaggi *et al.*, "Don't use large mini-batches, use local SGD," *ArXiv*, vol. abs/1808.07217, 2020.

[4] S. Stich, "Local SGD converges fast and communicates little," *ArXiv*, vol. abs/1805.09767, 2019.

[5] L. Yang, Y. Lu, J. Cao, J. Huang, and M. Zhang, "E-tree learning: A novel decentralized model learning framework for edge AI," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[6] L. Liu, J. Zhang, S. Song *et al.*, "Client-edge-cloud hierarchical federated learning," *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, 2020.

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMC.2022.3149584, IEEE Transactions on Mobile Computing

[7] M. S. H. Abad, E. Ozfatura, D. Gündüz *et al.*, "Hierarchical federated learning across heterogeneous cellular networks," *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8866–8870, 2020.

[8] O. Dekel, R. Gilad-Bachrach, O. Shamir *et al.*, "Optimal distributed online prediction using mini-batches," *J. Mach. Learn. Res.*, vol. 13, pp. 165–202, 2012.

[9] H. Yu, S. Yang, S. Zhu *et al.*, "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *AAAI*, 2019.

[10] M. Li, D. Andersen, J. Park *et al.*, "Scaling distributed machine learning with the parameter server," in *BigDataScience '14*, 2014.

[11] I. Hegedüs, G. Danner, M. Jelasity *et al.*, "Gossip learning as a decentralized alternative to federated learning," in *DAIS*, 2019.

[12] R. Ormándi, I. Hegedüs, M. Jelasity *et al.*, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, 2013.

[13] R. Han, S. Li, X. Wang *et al.*, "Accelerating gossip-based deep learning in heterogeneous edge computing platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, pp. 1591–1602, 2021.

[14] X. Lian, C. Zhang, H. Zhang *et al.*, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *NIPS*, 2017.

[15] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," *CoRR*, vol. abs/1808.07576, 2018.

[16] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *ArXiv*, vol. abs/1802.05799, 2018.

[17] A. Gerbessiotis and L. Valiant, "Direct bulk-synchronous parallel algorithms," *J. Parallel Distributed Comput.*, vol. 22, pp. 251–267, 1994.

[18] S. Ghadimi, G. Lan, H. Zhang *et al.*, "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Mathematical Programming*, vol. 155, pp. 267–305, 2016.

[19] Q. Ho, J. Cipar, H. Cui *et al.*, "More effective distributed ml via a stale synchronous parallel parameter server," *Advances in neural information processing systems*, vol. 2013, pp. 1223–1231, 2013.

[20] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pp. 5451–5452, 2012.

[21] H. Feyzmahdavian, A. Aytekin, M. Johansson *et al.*, "An asynchronous mini-batch algorithm for regularized stochastic optimization," *2015 54th IEEE Conference on Decision and Control (CDC)*, pp. 1384–1389, 2015.

[22] D. Yang, W. Rang, D. Cheng *et al.*, "Mitigating stragglers in the decentralized training on heterogeneous clusters," *Proceedings of the 21st International Middleware Conference*, 2020.

[23] J.-W. Lee, J. Oh, Y. Shin *et al.*, "Accurate and fast federated learning via iid and communication-aware grouping," *ArXiv*, vol. abs/2012.04857, 2020.

[24] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd," in *Proceedings of Machine Learning and Systems*, vol. 1, 2019, pp. 212–229.

[25] D. Bernstein, "Containers and cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, pp. 81–84, 2014.

[26] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, p. 2, 2014.

[27] M. Abadi, A. Agarwal, P. Barham *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *ArXiv*, vol. abs/1603.04467, 2016.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] H. Xiao, K. Rasul, R. Vollgraf *et al.*, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *ArXiv*, vol. abs/1708.07747, 2017.

[30] G. Cohen, S. Afshar, J. Tapson *et al.*, "Emnist: Extending mnist to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921–2926, 2017.

[31] Y. Zhao, M. Li, L. Lai *et al.*, "Federated learning with non-iid data," *ArXiv*, vol. abs/1806.00582, 2018.

[32] J. Wang, Q. Liu, H. Liang *et al.*, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *34th Conference on Neural Information Processing Systems, NeurIPS 2020*, vol. 33, 2020, pp. 7611–7623.

[33] A. Khaled, K. Mishchenko, P. Richtárik *et al.*, "Tighter theory for local SGD on identical and heterogeneous data," in *AISTATS*, 2020.

[34] A. Spiridonoff, A. Olshevsky, I. Paschalidis *et al.*, "Local SGD with a communication overhead depending only on the number of workers," *ArXiv*, vol. abs/2006.02582, 2020.

[35] F. Haddadpour, M. M. Kamani, M. Mahdavi *et al.*, "Local SGD with periodic averaging: Tighter analysis and adaptive synchronization," in *NeurIPS*, 2019.

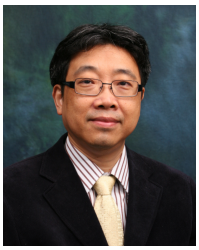
[36] S. Wang, T. Tuor, T. Salonidis *et al.*, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, pp. 1205–1221, 2019.

[37] L. Wang and S. Lei, "Adaptive federated learning and digital twin for industrial internet of things," *ArXiv*, vol. abs/2010.13058, 2020.

[38] J. Wang, S. Wang, R. Chen, and M. Ji, "Local averaging helps: Hierarchical federated learning and convergence analysis," *ArXiv*, vol. abs/2010.12998, 2020.

Lei Yang received the BSc degree from Wuhan University, in 2007, the MSc degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the PhD degree from the Department of Computing, The Hong Kong Polytechnic University, in 2014, where he worked as a postdoctoral fellow for more than a year. Since 2016, he has been an associate professor at the School of Software Engineering, South China University of Technology, China. He has published over 50 papers in major international journals like IEEE TMC, IEEE TC, IEEE TPDS, IEEE TSC, IEEE TCC, ACM TKDD and top conferences like IEEE INFOCOM and IEEE PERCOM. His research interest includes distributed systems and networks, edge and cloud computing, and distributed machine learning.

Yingqi Gan obtained the B.Eng. degree in Software Engineering from Guangdong University of Foreign Studies, China, in June 2019. She is a 2nd year postgraduate student at School of Software Engineering, South China University of Technology, China. Her research interests are distributed machine learning and edge computing.



Jiannong Cao received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990 respectively. He is currently the Otto Poon Charitable Foundation Professor in Data Science and a Chair Professor of the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. His research interests include distributed systems and blockchain, wireless sensing and networking, big data and machine learning, and mobile cloud and edge computing. He published 5 co-authored and 9 co-edited books, and over 500 papers in major international journals and conference proceedings. He is a member of Academia Europaea, a fellow of IEEE, and an ACM distinguished member.

Zhengyu Wang received the BSc degree from Xiamen University, China, in 1987, and the MSc and the PhD degrees from Harbin Institute of Technology, China, in 1990 and 1993, all in computer science. He is currently a professor and the dean of the School of Software Engineering, South China University of Technology, China. His research interests include cloud and edge computing, social computing and blockchain.

1536-1233 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: SOUTH CHINA UNIVERSITY OF TECHNOLOGY. Downloaded on September 07,2022 at 14:37:39 UTC from IEEE Xplore. Restrictions apply.