Accelerating Federated Learning with Cluster Construction and Hierarchical Aggregation

Zhiyuan Wang, *Hongli Xu, *Member, IEEE,* Jianchun Liu, *Student Member, IEEE,* Yang Xu, *Member, IEEE,* He Huang, *Member, IEEE, ACM,* Yangming Zhao

Abstract—Federated learning (FL) has emerged in edge computing to address the limited bandwidth and privacy concerns of traditional cloud-based training. However, the existing FL mechanisms may lead to a long training time and consume massive communication resources. In this paper, we propose an efficient FL mechanism, namely FedCH, to accelerate FL in heterogeneous edge computing. Different from existing works which adopt the pre-defined system architecture and train models in a synchronous or asynchronous manner, FedCH will construct a special cluster topology and perform hierarchical aggregation for training. Specifically, FedCH arranges all clients into multiple clusters based on their heterogeneous training capacities. The clients in one cluster synchronously forward their local updates to the cluster header for aggregation, while all cluster headers take the asynchronous method for global aggregation. Our analysis shows that the convergence bound depends on the number of clusters and the training epochs. We propose efficient algorithms to determine the optimal number of clusters with resource budgets and then construct the cluster topology to address the client heterogeneity. Extensive experiments on both physical platform and simulated environment show that FedCH reduces the completion time by 49.5-79.5% and the network traffic by 57.4-80.8%, compared with the existing FL mechanisms.

Index Terms—*Hierarchical Federated Learning, Mobile Edge Computing, Cluster Construction, Optimization.*

1 Introduction

In the era of big data, billions of Internet of Thing devices and smartphones around the world produce a significant amount of data per second [1]. Therefore, the traditional way of uploading those data to the remote cloud for processing will encounter many issues, including privacy leakage, network congestion, and high transmission delay [2]. Since data are generated at the network edge, mobile edge computing (MEC) is a natural alternative [3], [4], which uses the computing and storage resources of devices to perform data processing close to the data generators. According to Cisco's survey, most IoT-created data will be stored, processed, and analyzed close to or at the network edge [5]. With more data and advanced applications (*e.g.*, autonomous driving, virtual reality, and image classification), machine learning (ML) tasks will be a dominant workload in MEC [6]. To alleviate the network bandwidth burden and avoid the privacy leakage, FL becomes an efficient solution to analyze and process the distributed data on end devices for those ML tasks [7], [8].

Among previous FL frameworks, the dominant one is the parameter server (PS) based framework [9]. This framework comprises two components, the PS and clients, which form a two-layer architecture. Clients mainly use their local data to cooperatively train models in a synchronous manner. After each client performs local training, the local model will be forwarded to the PS for processing, which is called *global aggregation*. Under this framework, the PS maintains the globally shared up-to-date model to solve large-scale ML problems, which can protect privacy [10], and relieve network burden [11], [12], compared with the traditional centralized training approach.

However, the PS-based framework will also encounter some challenges due to the limited and heterogeneous resources (*i.e.*, computing and communication) [13], [14] in MEC. 1) *Bandwidth constraint on parameter server.* The first challenge is to aggregate models from massive devices in the FL framework. The bottleneck lies in the high communication cost on the centralized parameter server, which is always located on a remote cloud platform to provide reliable services [9] with limited ingress bandwidth [7], [9]. Specifically, for a system with hundreds of devices, the total size of models transmitted in each epoch will reach at least 50GB when training model (*e.g.*, VGG16 [15]), leading to possible network congestion. 2) *Limited and heterogeneous communication resource on clients.* Different from the traditional model training, where the server clusters communicate with each other through high-speed links, the clients may frequently communicate with the PS for model transmission over heterogeneous wireless links under the PS-based framework. Therefore, the client with the slowest link will restrict the training speed of most FL mechanisms [7], [11]. 3) *Limited and heterogeneous computing resource on clients.* The clients (*e.g.*, smartphones, and vehicles) are generally have limited and heterogeneous computing capacities [16], [17]. Since model (*e.g.*, deep neural network) training always imposes heavy computing requirements and the amount of data among clients are imbalanced [18], the performance

- *A preliminary version of this paper titled "Resource-Efficient Federated Learning with Hierarchical Aggregation in Edge Computing" was accepted by IEEE INFOCOM 2021.*
- *Z. Wang, H. Xu, J. Liu Y. Xu and Y. Zhao are with University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: cswangzy@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, jsen617@mail.ustc.edu.cn, xuyangcs@ustc.edu.cn, and zhaoyangming.uestc@gmail.com.*
- *H. Huang is with the School of Computer Science and Technology, Soochow University. E-mail: huangh@suda.edu.cn*

gap between clients is further widened, which will severally deteriorate the training efficiency and lead to long training time.

Some recent studies have proposed various solutions to address the above challenges. To alleviate the total communication cost in FL, a natural solution adopts the model compression [19], [20]. Unfortunately, those approaches will sacrifice the accuracy of the trained model and incur high computation overhead. Moreover, several solutions [21]–[23] have built a basic hierarchical FL architecture for alleviating the bandwidth pressure on the PS. However, those works ignore the possible heterogeneous resources among clients, adopt the pre-defined system architecture and train models in a synchronous manner, which can seriously deteriorate the training efficiency. Each training epoch of those studies only progresses as fast as the slowest clients [24], called *straggler effect*. To conquer the negative impact of client heterogeneity, the asynchronous FL [17], [25], [26] and partial update [27], [28] are proposed. The asynchronous FL performs global aggregation as soon as the PS collects one local update from an arbitrary client. Therefore, the global model will be simply and severely destroyed by the local model from a single client, leading to poor convergence performance. For partial update, those approaches allow to aggregate local models from partial devices for global aggregation, and some stragglers' gradients are dropped. Since a certain proportion of weak clients will always be discarded and never participate in global aggregation, it is difficult for the partial update to achieve the comparable accuracy as [6], [21], [23].

To effectively deal with the limited resource as well as the client heterogeneity in MEC, in this paper, we design an efficient FL mechanism, called FedCH. Different from most of the existing work which performs model aggregations in the synchronous or asynchronous manner based on a given architecture (including the hierarchical FL architecture), FedCH will construct an efficient *C*luster topology and then adopt the proposed *H*ierarchical aggregation approach for training. Specifically, FedCH will arrange all clients into $\mathcal{K}$ clusters based on their heterogeneous training capacities (*i.e.*, computing and communication) for distributed machine learning. This procedure enables the clients in the same cluster to be more similar to each other than those in different clusters. To adapt to the constructed cluster topology, FedCH further adopts the *hierarchical aggregation*, *i.e.*, those clients in the same cluster perform the synchronous method for aggregation, called *cluster aggregation*, while all clusters perform global aggregation in an asynchronous method. Based on this, each cluster performs training independently at its own training speed, and those clusters composed of clients with low training capacities will not restrict the training speed of the whole system. Besides, considering the low-cost intra-cluster communication among clients, the time consumption of model aggregation will be significantly reduced and the bandwidth burden of the parameter server will be obviously relieved. Under hierarchical aggregation, the theoretical analyses show that the training performance mainly depends on the cluster topology as well as the resource budgets (Section 4). As a result, it is critical to determine how to construct the cluster topology under the given resource constraints for training, *i.e.*, how many clusters should we create and to which cluster should each client be assigned? The main contributions of this paper are as follows:

1) We design a novel FL mechanism, namely FedCH, to address the resource constraints and client heterogeneity in edge computing. Furthermore, we theoretically prove that FedCH can provide the convergence guarantee for model training.
2) To accelerate federated learning, we propose efficient algorithms to determine the optimal number of clusters with resource constraints and construct the cluster topology for hierarchical aggregation. We also extend our algorithms to deal with dynamic scenarios.
3) We conduct extensive experiments using various models and datasets on both physical platform and large-scale simulated environment. The experimental results show that the proposed algorithms reduce completion time by 49.5-79.5% and the network traffic by 57.4-80.8% while achieving similar accuracy, compared with the well-known FL mechanisms.

The rest of this paper is organized as follows. We review the related work in Section 2. We propose our proposed mechanism in Section 3. In Section 4, we present our algorithms and extend them to dynamic scenarios. Section 5 evaluates the performance of our proposed algorithms. We conclude this paper in Section 6.

2 Related Work

This section briefly reviews the related work on FL acceleration and the hierarchical FL in edge computing.

2.1 Federated Learning Acceleration

To address the resource constraint and accelerate FL in MEC, several mechanisms have been proposed recently [6], [29]–[32], including asynchronous aggregation, adaptive training, partial update (client selection), model pruning or compression and some other advanced technologies. 1) For asynchronous aggregation, Xie *et al.* [17] first propose the FL algorithm with asynchronous aggregation and staleness treatment to improve flexibility and scalability. Based on this, some other approaches have been proposed for asynchronous FL acceleration in the past two years. For example, Wu *et al.* [33] propose a semi-asynchronous FL mechanism and introduce novel designs with respect to model distribution and global aggregation to mitigate the impacts of stragglers, and model staleness. The authors in [34] adopt a novel weighted aggregation strategy to aggregate the models with different versions to take full advantage of the acceleration effect of asynchronous strategy on heterogeneous training. 2) For adaptive training in FL, Wang *et al.* [6] propose a solution that adjusts the frequency of global aggregation to minimize the learning loss under a fixed resource budget dynamically and adaptively. The authors in [35] introduce two approaches for FL acceleration by adaptive sampling and top-k selective masking. The former controls the fraction of selected client models dynamically, while the latter selects parameters with top-k largest values. The authors in [29] target to accelerate the FL by jointly optimizing local

training batch size and communication resource allocation. 3) There is also a strand of literature that focuses on client selection in FL. Chen *et al.* [30] enable the implementation of FL over wireless networks by considering user selection under limited bandwidth resource. Jin *et al.* [31] study how to select suitable participants and set the number of local updates for resource-efficient FL. [36] solves a client selection problem with resource constraints, which allows the server to aggregate as many client updates as possible and to accelerate performance improvement. 4) Model compression is also a natural solution to alleviate communication cost and accelerate FL. Xu *et al.* [37] propose the ternary FL protocol, which compresses both uploading and downloading communications to nearly one-sixteenth of the standard method. The authors in [38] design a compression control scheme to balance the energy consumption of local computing and wireless communication from the long-term learning perspective. 5) There are also some other approaches to conduct resource allocation or hyper-parameter adjustment in FL for improving the training performance. The authors in [32] introduce the game-theoretic incentive mechanisms to provide efficient resource management for FL at the network edge for federated learning acceleration. Ma *et al.* [39] study the relationship between batch size and learning rate, and formulate a scaling rule to guide the setting of learning rate in terms of batch size.

Some approaches (e.g., adaptive training, model compression, and hyper-parameter adjustment) are orthogonal to our work and can be used in conjunction with our techniques for achieving better training performance, while the other approaches (e.g., asynchronous aggregation and client selection) are difficult to obtain good convergence performance and address resource constraints simultaneously. Besides, the aforementioned studies mainly focus on the traditional parameter server based FL architecture. The bottleneck of this architecture lies in the high communication cost on the centralized parameter server, which can also cause network congestion and worse model convergence.

2.2 Hierarchical Federated Learning

To conquer the above challenges, the hierarchical FL has received increasing attention in the past two years. The approach in [22] has built a basic client-edge-cloud federated learning architecture for achieving faster convergence, and reducing the communication cost. However, this work adopts a pre-defined system architecture and ignores the possible heterogeneous communication conditions and computing resources among different clients, which will seriously affect the training efficiency, especially for a scenario with large-scale clients. Chai *et al.* [21] first propose a hierarchical blockchain-enabled FL framework in the internet of vehicles. They aim to model the training process as a trading market and each client chooses to sell their models to servers according to the bidding prices, rather than considering the resource consumption and the client heterogeneity.

Three researches [23], [40], [41] further consider the heterogeneity among clients. Luo *et al.* [40] aim to perform resource allocation of each device for training cost minimization in terms of energy and delay. This approach will restrict the clients' maximum computing capacity (slows down the training speed of fast clients) to reduce energy consumption. Besides, it performs synchronous training among all clients and edge servers, thus, each training epoch only processes as fast as the slowest client. The author in [41] employs gradient sparsification and periodic averaging to minimize the communication latency. This work has not considered the heterogeneous computing resource among clients. Besides, the gradient sparsification technology (with additional computing cost) and synchronous training method further deteriorate its system performance. To release the computing workload of less powerful devices and fully utilize the computing resource of edge servers or even cloud, the authors in [23] propose the solution. They target to determine which controllers (edge servers) should process data from which sensors (clients), and make the data collection and transfer solutions based on the heterogeneous resources of edge servers. Since the clients' private data is transmitted through the network, this approach violates the increasing privacy concern. The work [42], which was presented very recently, joint consider hierarchical FL and cluster construction for non-independent and identically distributed (non-IID) data. They propose to assign clients to edge servers by considering both the statistical properties of local datasets and wireless communication connectivity. TiFL [43] groups clients into several tiers (e.g., fast, and slow tiers) for training. However, it groups clients into a predefined number of tiers while we will determine the number of clusters based on the resource constraints and convergence bound. Besides, it randomly selects partial clients from a tire for synchronous training in each epoch, leading to a waste of resource and poor convergence performance. In the contrast, all clusters in our mechanism perform training simultaneously and asynchronously, by which we can take the full use of resources of clients and achieve good convergence performance. The authors in [44] aim to relieve global imbalance by adaptive data augmentation and downsampling, and address local imbalance by creating the mediators to reschedule the training of clients based on Kullback–Leibler divergence of their data distribution. The advanced technologies (e.g., data augmentation) proposed in [44] can be simply combined with our approach for achieving performance improvement.

In contrast to the above researches, our work addresses the resource constraints and client heterogeneity by constructing an efficient cluster topology and then performing the proposed hierarchical aggregation for FL. Each global aggregation consumes resources of the clients in one cluster, and the global model eventually converges after hundreds of global aggregations. Intuitively, the more clusters, the fewer clients will be assigned to each cluster and the less resources will be consumed for each global aggregation. But as the number of clients in each cluster decreases, the number of global aggregations required for model convergence will increase. Therefore, it is challenging but meaningful to construct the cluster topology under a given resource budget for achieving good training performance.

3 Preliminaries

In this section, we first describe the training process of FedCH and then prove its convergence. We finally give the

TABLE 1: Summary of main notations.

Symbol	Definition
N	the number of clients
$\mathcal{K}$	the number of clusters
n_k	the number of clients in each cluster k
H	the number of local updates
T	the total number of training epochs
M	the total number of resource types
R_m	the total budget of resource m
E	the total time budget
T_b	the preset constant for the fixed re-clustering
T_k	the epochs that cluster k performs under time budget
$\widetilde{\mathcal{K}}$	the preset constant for the adaptive re-clustering
x_i^k	whether client i is assigned to cluster k or not
w	the model parameter
$F(w^T)$	the global loss function after T epochs
$F(w^*)$	the optimal value of loss function $F(w)$

problem definition.

3.1 Gradient Descent-based FL

In FL, each client trains its own local model based on a collection of data samples. Let j denote a training sample, including feature x_j and label y_j. Given a model, *e.g.*, logistic regression (LR) [45], or convolutional neural network (CNN) [46], the loss function is denoted as $f_j(w, x_j, y_j)$, written as $f_j(w)$ for simplicity, where w is the model vector. The loss function on dateset D is

$$f(w) = \frac{1}{|D|} \sum_{j \in D} f_j(w) \tag{1}$$

where $|D|$ is the number of training samples in D. Then, the learning problem is to find the optimal model vector w that minimizes the loss function $f(w)$, expressed as

$$w^* = \arg\min f(w) \tag{2}$$

It is almost impossible to solve Eq. (2) directly, especially for deep learning models. Alternatively, each client will perform gradient-descent in each local update (*i.e.*, iteration) to gradually approach the optimal solution. For iteration t, the local update rule is described as follows:

$$w(t) = w(t-1) - \eta \nabla f(w(t)) \tag{3}$$

where $\eta > 0$ is the step size.

3.2 Training Process of FedCH

Assume that there are N clients, which cooperate to train a model in edge computing. For FedCH, we consider that all clients are divided into $\mathcal{K}$ clusters, with n_k clients in each cluster $k \in \{1, 2, ..., \mathcal{K}\}$ (*i.e.*, $\sum_{k=1}^{\mathcal{K}} n_k = N$). Let T be the total number of training epochs, with one global aggregation in each epoch. Let H denote the number of local updates (*i.e.*, iterations) that each client performs between its two consecutive cluster aggregations. Then, we introduce the training process of FedCH in three steps, *i.e.*, local updates, cluster aggregation and global aggregation.

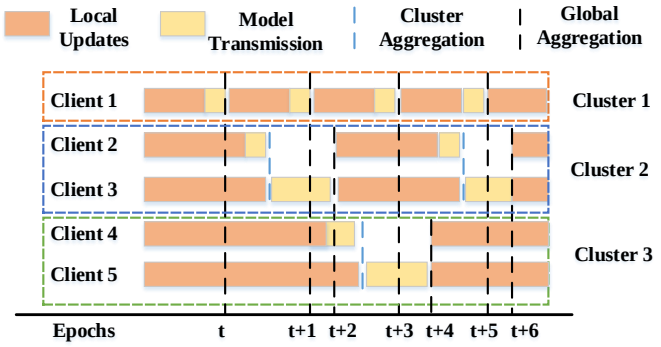


Fig. 1: Illustration of training process of FedCH.

Local Updates. Each client $i \in \{1, 2, ..., N\}$ in cluster k is associated with a local loss function based on the local dataset D_k^i, *i.e.*,

$$F_k^i(w_k^i) = \frac{1}{|D_k^i|} \sum_{j \in D_k^i} f_j(w_k^i) \tag{4}$$

where w_k^i is the local model of client i in cluster k. To minimize the local loss function, client i iteratively updates the local model by Eq. (3).

Cluster Aggregation. After H iterations, each client in cluster k will forward the updated local model to the selected leader node, denoted as LN_k, for aggregation. Once collecting local models from all clients in cluster k, LN_k will perform the cluster aggregation. The new model after cluster aggregation is defined as

$$w(k) = \frac{\sum_{i=1}^{n_k} |D_k^i| w_k^i}{\sum_{i=1}^{n_k} |D_k^i|} \tag{5}$$

This model will be further forwarded to the PS for global aggregation.

Global Aggregation. Since each cluster asynchronously performs the cluster aggregation, the parameter server performs global aggregation as soon as it receives one model from an arbitrary cluster without waiting for all other clusters. The PS updates the global model w^t at epoch $t \in \{1, 2, ..., T\}$ by staleness-aware global update approach (Section 3.3), and distributes the up-to-date global model back to the corresponding cluster. Then, the global loss function $F(w^t)$ after t epochs is

$$F(w^t) = \frac{\sum_{k=1}^{\mathcal{K}} \sum_{i=1}^{n_k} |D_k^i| F_k^i(w^t)}{\sum_{k=1}^{\mathcal{K}} \sum_{i=1}^{n_k} |D_k^i|} \tag{6}$$

The global loss function $F(w^t)$ cannot be directly computed without sharing global model to all clients by the PS.

To illustrate the training timeline of FedCH more clearly, we give an example in Fig. 1, where there are 5 clients, numbered from 1 to 5, respectively. The length of each bar denotes the time consumption of the corresponding operation (*e.g.*, local updates). For FedCH, we first assign 5 clients into 3 clusters based on their heterogeneous training speed, *i.e.*, client #1 belongs to cluster #1, clients #2 and #3 belong to cluster #2, and clients #4 and #5 belong to cluster #3. Based on the constructed cluster topology, each cluster then performs the training process independently,

Algorithm 1 Training process of FedCH

Initialize w^0, and $t = 0$;
Determine the cluster number by Section 4.1 ;
Construct the cluster topology based on Alg. 2;
repeat
  Global Aggregation at the Parameter Server
  Receive update from LN_k, and set $t \leftarrow t + 1$;
  Compute w^t according to Eq. (7);
  Send w^t back to LN_k;
  Cluster Aggregation at LN_k
  Receive local updates from all clients in cluster;
  Compute $w(k)$ by Eq. (5);
  Send $w(k)$ to PS;
  Receive w^t from PS and return it back to clients;
  Local Updates at Client i **in Cluster** k
  Receive w^t from LN_k;
  Perform H local updates;
until Exceed the resource budget;
Return the final model parameter w^t;

and different clusters always experience different global aggregation frequencies. From epoch t to $t+6$, clusters #1, #2, and #3 performs 4, 2 and 1 global aggregations, respectively. For cluster #2, assume client #3 is chosen as the cluster header. During training, client #2 sends the updated local model to clients #3 for cluster aggregation after local updates, and client #3 forwards the aggregated model to the parameter server for global aggregation.

We finally conclude the training process of FedCH as shown in Alg. 1. We should note that FedCH is the generalization of the previous FL solutions. For example, if the cluster number $\mathcal{K}$ is 1, it becomes the synchronous FL [47]. If $\mathcal{K}$ is N, it is exactly the asynchronous FL [17].

3.3 Staleness-aware Global Update

Since clients in each cluster perform the synchronous FL method, their staleness, denoted as τ, is the same. For cluster k, its staleness is defined as the number of experienced epochs since its last global update. PS updates the global model w^t at epoch t with staleness treatment, that is, the weight of each newly received model $w(k)$ from an arbitrary cluster k will be determined by τ,

$$w^t = (1 - \alpha_\tau^t) w^{t-1} + \alpha_\tau^t w(k) \tag{7}$$

where α_τ^t is the weight of $w(k)$ at epoch t with staleness τ. Then, we adopt a function [17] to determine the value of α_τ^t, that is,

$$\alpha_\tau^t = \begin{cases} \alpha, & \tau \le a \\ \alpha \cdot \tau^{-b}, & \tau > a \end{cases} \tag{8}$$

where $a > 0$, $b \ge 0$, and $\alpha \in (0, 1)$ is an initial model weight. This function implies that when $\tau > a$, the weight of model drops rapidly as the staleness becomes higher. In fact, when we divide clients into different numbers of clusters, the model weight of each cluster will drop with the increasing number of clusters. We initialize weight α as $\alpha = \phi(\mathcal{K}) = 1 - \frac{\mathcal{K}-1}{N}$, with $\mathcal{K} \in \{1, .., N\}$, and we also set a lower bound for α based on the analysis in Section 4. After substituting α in Eq. (8), we obtain the expression of α_τ^t as follows

$$\alpha_\tau^t = \begin{cases} 1 - \frac{\mathcal{K}-1}{N}, & \tau \le a \\ (1 - \frac{\mathcal{K}-1}{N})\tau^{-b}, & \tau > a \end{cases} \tag{9}$$

By Eq. (9), we have $\alpha_1^t = 1$ if $\mathcal{K} = 1$. As a result, $w^t = \frac{\sum_{i=1}^N |D_i| w_i}{\sum_{i=1}^N |D_i|}$, which is same as that in the synchronous FL [6].

3.4 Convergence Analysis

To analyze the convergence of FedCH, we first make some widely used assumptions as follows: [17].

Assumption 1. Assume that the loss function f satisfies the following conditions:
1) f is μ-strongly convex, where $\mu \ge 0$, *i.e.*,

$$f(y) - f(x) \ge \nabla f^{\mathbb{T}}(x)(y - x) + \frac{\mu}{2}\|y - x\|^2, \forall x, y$$

2) f is β-smooth, where $\beta > 0$, *i.e.*,

$$f(y) - f(x) \le \nabla f^{\mathbb{T}}(x)(y - x) + \frac{\beta}{2}\|y - x\|^2, \forall x, y$$

3) There exists at least one solution x^* for global optimization that can minimize the loss function, *i.e.*,

$$x^* = \inf_x f(x) \text{ and } \nabla f(x^*) = 0, \exists x^* \in R^d$$

The above assumptions can be satisfied for many models with convex loss functions, *e.g.*, linear regression [48], LR [45] and SVM [49]. The loss functions $f(w, x_j, y_j)$ of those models are listed below.

- Liner regression: $\frac{1}{2}\|y_j - w^{\mathbb{T}} x_j\|^2, y_j \in R$
- LR: $-\log(1 + \exp(-y_j w^{\mathbb{T}} x_j)), y_j \in \{0, 1\}$
- SVM: $\frac{\lambda}{2}\|w\|^2 + \frac{1}{2}\max\{0; 1 - y_j w^{\mathbb{T}} x_j\}, y_j \in \{-1, 1\}$

According to the above assumptions, we prove the convergence of our two-layer FL mechanism in two steps. We analyze the convergence bound after H local updates. Based on that, we will derive the bound after T epochs.

Definition 1. Assume that the global loss function F is β-smooth and μ-strongly convex. $\forall w \in R^d$ and $\forall j \in D_k^i$, where $k \in \{1, ..., \mathcal{K}\}$ and $i \in \{1, ..., n_k\}$, we define an upper bound Q_1 of $\|\nabla f(w; j) - \nabla F(w)\|^2$, *i.e.*,

$$\mathbb{E}\|\nabla f(w; j) - \nabla F(w)\|^2 \le Q_1$$

We also define Q_2 as the upper bound of $\|\nabla f(w; j)\|^2$, *i.e.*,

$$\mathbb{E}\|\nabla f(w; j)\|^2 \le Q_2$$

Theorem 1. When the following conditions are satisfied: 1) $\eta < \frac{1}{\beta}$; and 2) $F(w^0) - F(w^*) > \frac{Q_1 + Q_2}{2\eta\mu^2}$, the convergence bound of the global loss function F after T epochs is,

$$\begin{aligned} &\mathbb{E}[F(w^T) - F(w^*)] \\ &\le [\frac{\mathcal{K}-1}{N} + \alpha(1 - \eta\mu)^H]^T (F(w^0) - F(w^*)) \\ &+ \frac{(Q_1 + Q_2)(1 - [\frac{\mathcal{K}-1}{N} + \alpha(1 - \eta\mu)^H]^T)}{2\eta\mu^2} \end{aligned} \tag{10}$$

where w^0 is the initial model parameter, and w^* denotes the optimal model which minimizes the global loss function F.

Proof: Since some proof procedure can be found in the previous work [6], [17], we only show the differences. After client i in an arbitrary cluster k performs H local updates by the model $w^{t-\tau}$, the convergence bound is

$$\begin{aligned}
&\mathbb{E}[F(w^{t-\tau,H}) - F(w^*)] \\
&\leq (1-\eta\mu)^H[F(w^{t-\tau,0}) - F(w^*)] + \frac{\eta Q_1}{2}\sum_{h=1}^{H}(1-\eta\mu)^{h-1} \\
&\leq (1-\eta\mu)^H[F(w^{t-\tau,0}) - F(w^*)] + \frac{\eta Q_1}{2}\frac{1-(1-\eta\mu)^H}{1-(1-\eta\mu)} \\
&\leq (1-\eta\mu)^H[F(w^{t-\tau,0}) - F(w^*)] + \frac{\eta Q_1}{2}\frac{H\eta\mu}{1-(1-\eta\mu)} \\
&\leq (1-\eta\mu)^H[F(w^{t-\tau,0}) - F(w^*)] + \frac{H\eta Q_1}{2} \qquad (11)
\end{aligned}$$

where $w^{t-\tau,H}$ is derived from $w^{t-\tau}$ by H iterations. Apparently, $w^{t-\tau,0}$ and $w^{t-\tau}$ are equal. Then, the asynchronous FL mechanism in [17] will perform one global aggregation immediately. But for our FL mechanism, only after n_k clients in cluster k have performed H iterations and the LN_k aggregates their local models based on Eq. (5), the PS will update the global model by Eq. (7). Thus, the convergence bound of FedCH after t epochs is:

$$\begin{aligned}
&\mathbb{E}[F(w^t) - F(w^*)] \\
&\leq (1-\alpha_\tau^t)F(w^{t-1}) + \alpha_\tau^t\mathbb{E}[F(w(k))] - F(w^*) \\
&\leq (1-\alpha_\tau^t)F(w^{t-1}) + \alpha_\tau^t\mathbb{E}[F(\frac{\sum_{i=1}^{n_k}|D_k^i|w_k^i}{\sum_{i=1}^{n_k}|D_k^i|})] - F(w^*) \\
&\leq (1-\alpha_\tau^t)F(w^{t-1}) + \alpha_\tau^t\sum_{i=1}^{n_k}\frac{|D_k^i|}{\sum_{i=1}^{n_k}|D_k^i|}\mathbb{E}[F(w_k^i) - F(w^*)] \\
&\leq (1-\alpha_\tau^t)[F(w^{t-1}) - F(w^*)] + \alpha_\tau^t\mathbb{E}[F(w_k^i) - F(w^*)] \qquad (12)
\end{aligned}$$

From the second term and the fifth term, we note that the expected loss of the aggregated model of each cluster k is smaller than that of any client i in this cluster. This is because the aggregated model learns information from all local dataset of clients in cluster k while the local model of client i is only trained over the local dataset. Since the clients in cluster k adopt $w^{t-\tau}$ to perform H local updates, the weight of the new model w_k is α_τ^t, where $\alpha_\tau^t \leq \alpha_1^t$ according to Eq. (9) and $\mathbb{E}[F(w_k^i) - F(w^*)]$ can be replaced with $\mathbb{E}[F(w^{t-\tau,H}) - F(w^*)]$ in Eq. (11). Based on Eqs. (11) and (12), we can derive the convergence bound after T epochs as shown in Theorem 1. □

For the potentially non-convex loss function, the convergence can also be guaranteed as in [17], and the result is shown in Theorem 2. However, this paper mainly focuses on the convergence bound of convex functions for ease of discussion, and the experimental results also verify the efficiency of FedCH for non-convex functions.

Theorem 2. The loss function f is μ-weakly convex if the function $\overline{f}(x) = f(x) + \frac{\mu}{2}\|x\|^2$ is convex, where $\mu \geq 0$. If $\mu = 0$, f is convex. Otherwise, f is potentially non-convex. Thus, when $\eta < \min\{\frac{1}{\beta}, \frac{2}{\rho-\mu}\}$ where $\rho > \mu$, we have

$$\begin{aligned}
&\mathbb{E}[F(w^T) - F(w^*)] \\
&\leq \beta^T(F(w^0) - F(w^*)) + (1-\beta^T)\mathcal{O}(Q_1 + Q_3) \qquad (13)
\end{aligned}$$

where $\beta = 1 - \alpha + \alpha(1 - \frac{\eta(\rho-\mu)}{2})^H$ and Q_3 is the upper bound of $\mathbb{E}\|\nabla\overline{f}(w;j)\|$ (*i.e.*, $\mathbb{E}\|\nabla\overline{f}(w;j)\|^2 \leq Q_3$).

3.5 Problem Formulation

To train models among distributed clients by FL, it is inevitable to consume resources (*e.g.*, network traffic and CPU cycles). Formally, we consider M different types of resources, and each resource $m \in \{1, 2, ..., M\}$ has a budget R_m. We assume that it consumes c_m of resource m for performing H local updates at each client. Meanwhile, let b_m denote the average consumption of resource m for each global aggregation of an client. Thus, the total resource consumption of T epochs is $T \cdot n_k \cdot p_k^t \cdot (c_m + b_m)$, where p_k^t is a binary variable to indicate whether cluster k is involved in global aggregation at epoch t or not. Since each cluster runs asynchronously, p_k^t is determined in real-time during training. Due to the client heterogeneity, we will assign clients with similar training speed together (Section 4.2) and performs asynchronous training among clusters. Specifically, we denote the time consumption between two consecutive global aggregations of cluster k as e_k. For a given time budget E, total number of training epochs that $\mathcal{K}$ clusters can perform is $\sum_{k=1}^{\mathcal{K}}\frac{E}{e_k}$. To construct an efficient cluster topology for performing hierarchical aggregation, we formulate the problem as follows:

$$\min_{T\in\{1,2,3,...\}} F(w^T)$$

$$s.t. \begin{cases} \sum_{t=1}^{T} n_k p_k^t(c_m + b_m) \leq R_m, & \forall m, k \\ T \leq \sum_{k=1}^{\mathcal{K}}\frac{E}{e_k}, & \forall m \\ \sum_{k=1}^{\mathcal{K}} p_k^t = 1, & \forall t \\ n_k = \lfloor\frac{N}{\mathcal{K}}\rfloor + \beta_k, & \forall k \\ p_k^t \in \{0,1\}, \beta_k \in \{0,1\} & \forall t, k \end{cases} \qquad (14)$$

The first set of inequalities indicates that the resource consumption on each type during T epochs should not exceed its budget. The second set of inequalities indicates that T should not exceed the total number of training epochs that all clusters can perform under a given time budget E. The third set of formulas represents that all clusters asynchronously perform the global update, and the PS updates the global model when it receives the model parameters from an arbitrary cluster. The fourth set of formulas denotes that we adopt the balanced clustering algorithm to balance the cluster size [50], [51]. Our objective is to minimize the global loss function after T epochs. In fact, it is difficult to directly solve the problem in Eq. (14) for the following two reasons: 1) It is impossible to solve Eq. (14) by finding an exact expression among $\mathcal{K}, T$ and $F(w^T)$ [6]. 2) For each cluster k, its per-epoch training time e_k is always associated with the constructed cluster topology. Hence, we will decompose the original problem into two subproblems and solve them step by step.

4 Algorithm Design

To construct an efficient cluster topology so as to accelerate FL, in this section, we formally propose the algorithms

for solving the problem in Eq. (14) in two steps. We first determine how many clusters (*i.e.*, $\mathcal{K}$) should be created under the given resource (time) budget. Then, we target to decide to which cluster should each client be assigned.

4.1 Cluster Number Determination of FedCH

Given a loss function, its minimum value $F(w^*)$ is always a constant. Therefore, we can rewrite the objective function $F(w^T)$ in Eq. (14) as $F(w^T) - F(w^*)$. Then we replace $F(w^T) - F(w^*)$ by the approximate value which is derived by convergence analysis in Eq. (13). As a result, we rewrite the objective function as:

$$\min_{T,\mathcal{K}} \gamma(F(w^0) - F(w^*)) + \frac{(Q_1+Q_2)(1-\gamma)}{2\eta\mu^2} \tag{15}$$

where $\gamma = [\frac{\mathcal{K}-1}{N} + \alpha(1-\eta\mu)^H]^T < 1$, which is related to T and K. To determine the cluster number based on Eq. (15), we make an assumption as follows,

Assumption 2. Assume the executed number of training epochs $\frac{E}{e_k}$ under the time budget E of each cluster k is with the Gaussian distribution, in which the constant expectation is denoted as T_e.

By Theorem 1, we note that Eq. (15) always decreases as T increases. Based on the first constraint of Eq. (14), we derive that $T \leq \lfloor \min_m \frac{R_m}{\lceil \frac{N}{\mathcal{K}} \rceil (c_m+b_m)} \rfloor$ since $\beta_k \in \{0,1\}$. We omit the rounding operation for simplicity. By the second constraint of Eq. (14), we have $T \leq \sum_{k=1}^{\mathcal{K}} \frac{E}{e_k} = \mathcal{K}T_e$. As a result, the optimal value of T can be expressed as,

$$T = \min\{\min_m \frac{R_m\mathcal{K}}{N(c_m+b_m)}, \mathcal{K}T_e\} \tag{16}$$

We first assume that $\min_m \frac{R_m\mathcal{K}}{N(c_m+b_m)} \leq \mathcal{K}T_e$, and substitute T into the objective function in Eq. (15), yielding

$$L(\mathcal{K}) = [\frac{\mathcal{K}-1}{N} + \alpha(1-\eta\mu)^H]^{\frac{R_m\mathcal{K}}{N(c_m+b_m)}}[F(w^0) - F(w^*)] + \frac{(Q_1+Q_2)[1 - (\frac{\mathcal{K}-1}{N} + \alpha(1-\eta\mu)^H)^{\frac{R_m\mathcal{K}}{N(c_m+b_m)}}]}{2\eta\mu^2} \tag{17}$$

After that, the optimal value of $\mathcal{K}$ can be determined as

$$\mathcal{K}^* = \arg \min_{\mathcal{K} \in \{1,2,...,N\}} L(\mathcal{K}) \tag{18}$$

from which we can obtain the value of T, *i.e.*,

$$T^* = \min_m \frac{R_m\mathcal{K}^*}{N(c_m+b_m)} \tag{19}$$

Theorem 3. We set $R_{min} = \min_m R_m$. When $R_{min} \to \infty$, we have $F(w^T) - F(w^*) \leq \frac{Q_1+Q_2}{2\eta\mu^2}$.

Proof: Because $R_{min} \to \infty$, that is, $R_m \to \infty, \forall m$, we have $T = \lfloor \min_m \frac{R_m\mathcal{K}}{N(c_m+b_m)} \rfloor \to \infty$. We also have $\frac{\mathcal{K}-1}{N} + \alpha(1-\eta\mu)^H < 1$ based on Eq. (9). Thus, $[\frac{\mathcal{K}-1}{N} + \alpha(1-\eta\mu)^H]^T \to 0$ and $\gamma \to 0$. Then it follows $\gamma(F(w^0) - F(w^*)) \to 0$ and $(1-\gamma) \to 1$. As a result, $F(w^T) - F(w^*) \leq \frac{Q_1+Q_2}{2\eta\mu^2}$. This result denotes that the global model will eventually converge regardless of the value of $\mathcal{K}$ in the condition without resource constraint. □

However, the resource constraints are unavoidable in MEC. Therefore, we demand to study the impact of $\mathcal{K}$ on Eq. (17) under limited resources, which is important for improving the training performance of FedCH. We set

$$g(\mathcal{K}) = [1 - \frac{N+1-\mathcal{K}}{N}(1-(1-\eta\mu)^H)]^{\frac{R_m\mathcal{K}}{N(c_m+b_m)}} \tag{20}$$

Then, we focus on the monotonicity of this function instead of $L(\mathcal{K})$ in Eq. (17). For simplicity, we define

$$A = \frac{1}{N}(1-(1-\eta\mu)^H), B = \frac{R_m}{N(c_m+b_m)} \tag{21}$$

We rewrite Eq. (20) as $g(\mathcal{K}) = (1 + \mathcal{K}A - (N+1)A)^{\mathcal{K}B}$. Taking the derivative, we get

$$\frac{\partial g(\mathcal{K})}{\partial \mathcal{K}} = (1+\mathcal{K}A-(N+1)A)^{\mathcal{K}B}[B\ln(1+\mathcal{K}A-(N+1)A) + \frac{\mathcal{K}BA}{1+\mathcal{K}A-(N+1)A}] \tag{22}$$

The second derivative result is

$$\frac{\partial^2 g(\mathcal{K})}{\partial^2 \mathcal{K}} = (1+\mathcal{K}A-(N+1)A)^{\mathcal{K}B}[B\ln(1+\mathcal{K}A-(N+1)A) + \frac{\mathcal{K}BA}{1+\mathcal{K}A-(N+1)A}]^2 + \frac{2BA[A(\mathcal{K}-1)+(1-NA)]}{[1+\mathcal{K}A-(N+1)A]^2} \tag{23}$$

According to Eq. (21), we have $A \in (0, \frac{1}{N})$. It follows $A(\mathcal{K}-1)+(1-NA) > 0$, and $\frac{\partial^2 g(\mathcal{K})}{\partial \mathcal{K}} > 0$. So $\frac{\partial g(\mathcal{K})}{\partial \mathcal{K}}$ is monotonically increasing with $\mathcal{K}$. We define

$$H(\mathcal{K}, A) = B\ln(1+\mathcal{K}A-(N+1)A) + \frac{\mathcal{K}BA}{1+\mathcal{K}A-(N+1)A}$$

The partial derivative of function $H(\mathcal{K}, A)$ on A is

$$\frac{\partial H(\mathcal{K}, A)}{\partial A} = \frac{B[A(\mathcal{K}-N-1)^2 + 2\mathcal{K} - N - 1]}{[1+(\mathcal{K}-N-1)A]^2} \tag{24}$$

We note that $H(\mathcal{K}, 0) = 0$ if $A = 0$.

Theorem 4. If $R_m < \mathbb{R}, \forall m$, where $\mathbb{R}$ a finite real number, we have $\mathcal{K}^* \in \{1, ..., \lfloor \frac{N+1}{2} \rfloor\}$.

Proof: If $\frac{N+1}{2} < \mathcal{K} \leq N$, we obtain $\frac{\partial H(\mathcal{K},A)}{\partial A} > 0$. For $H(\mathcal{K}, 0) = 0$, we have $H(\mathcal{K}, A) > 0$ if $A > 0$. Thus, we get $\frac{\partial g(\mathcal{K})}{\partial \mathcal{K}} > 0$, making the loss upper bound increase with $\mathcal{K}$.

On the contrary, assume that $1 \leq \mathcal{K} \leq \frac{N+1}{2}$. We consider the following two propositions with different conditions.

1) $\frac{N-1}{N^2} < A < \frac{1}{N}$. We have $\frac{\partial H(\mathcal{K},A)}{\partial A} > 0, \forall \mathcal{K}$, and $\frac{\partial g(\mathcal{K})}{\partial \mathcal{K}} > 0$. Thus, the minimum value of function $g(\mathcal{K})$ is $g(1)$, which means $\mathcal{K}^* = 1$. This case becomes the synchronous FL.

2) $0 < A \leq \frac{N-1}{N^2}$. If $\mathcal{K} = 1$, we have $\frac{\partial H(1,A)}{\partial A} < 0$. Since $H(1,0) = 0$, we derive $H(1,A) < 0$. Meanwhile, if $K = \frac{N+1}{2}$, we have $H(\frac{N+1}{2}, A) > 0$ for $\frac{\partial H(\frac{N+1}{2},A)}{\partial A} > 0$, and $H(\frac{N+1}{2}, 0) = 0$. Due to the continuity of $H(\mathcal{K}, A)$, there exists $\mathcal{K}^* \in (1, \frac{N+1}{2})$, making $H(\mathcal{K}^*, A) = 0$. As a result, $g(\mathcal{K})$ will takes the minima at $\mathcal{K}^*$ where $\frac{\partial g(\mathcal{K}^*)}{\partial \mathcal{K}^*} = 0$. □

By Eq. (16), if $\min_m \frac{R_m\mathcal{K}}{N(c_m+b_m)} > \mathcal{K}T_e$, then we have $T = \mathcal{K}T_e$. 1) When $E \to \infty$, we simply obtain the similar result as in Theorem 3. 2) By setting $B = T_e$ in Eq. (21), we also obtain the same result as in Theorem 4.

Since the solution of $\frac{\partial g(\mathcal{K})}{\partial \mathcal{K}} = 0$ is difficult to be obtained directly and $\mathcal{K}^*$ is a positive integer with less than $\frac{N+1}{2}$, we can explore a proper value of $\mathcal{K}^*$ within a finite range that minimizes $g(\mathcal{K})$. To calculate the value of $g(\mathcal{K})$, we should

estimate some parameters (*i.e.*, c_m, b_m, T_e and μ) during training in practice. Specifically, each client can calculate μ based on Assumption 1, and the average value of μ among all client will be adopted by the parameter server. For T_e, we first measure the average per-epoch training time among N clients, and then compute T_e by the time budgets E. For the resources such as network traffic and CPU cycles, c_m and b_m can be estimated offline. Specifically, given fixed batch size and the number of local updates (H), the computing workload for local updates of each client is the same. Besides, the network traffic consumption for global aggregation of each client is also constant for the reason that the model size is constant. Since Eq. (20) increases with $\mathcal{K}$ when $\mathcal{K} > \frac{N+1}{2}$, we can take the lower bound of α as 0.5 for better convergence performance. After that, we can search $\mathcal{K}^*$ by Eq. (20) with a time complexity of $\mathcal{O}(\log \frac{N+1}{2})$.

4.2 Effective Balanced Clustering Algorithm

After determining the number of clusters (*i.e.*, $\mathcal{K}$), we then perform the clustering algorithm to construct the cluster topology for effective federated learning. In this section, we first review the most widely used K-means algorithm [52], [53], and point out three disadvantages. Then, we formally present the proposed clustering algorithm.

4.2.1 K-means Clustering [52]

Clustering is a process of dividing a given set of objects into multiple disjoint clusters based on some pre-defined clustering criteria such that the objects in the same cluster are more similar to each other than those in different clusters [54], [55]. A lot of algorithms have been proposed in the literature, in which the K-means method [53] remains to be the widely used one for many practical applications. Let $V = \{V_1, V_2, ..., V_q\}$ be a set q objects. Each object V_i is denoted as a vector, which is always composed of multiple attributes. Then, we can apply the K-means algorithm for clustering so as to minimize the objective function depending on the clustering criterion and the cluster centers:

$$\mathbb{E} = \sum_{i=1}^{q} \sum_{k=1}^{\mathcal{K}} x_i^k \cdot d(V_i, \mathbb{C}_k^*) \tag{25}$$

where x_i^k indicates whether the object V_i belongs to the cluster k or not and $\mathbb{C}_k^*$ is the centroid of cluster k. The most widely used clustering criterion is the sum of the squared Euclidean distance, *i.e.*, $d(V_i, \mathbb{C}_k^*)$, between each object V_i and the centroid $\mathbb{C}_k^*$ of its cluster [56]. The K-means algorithm is initialized by randomly selecting $\mathcal{K}$ objects as the centroid of each cluster, and then the target function in Eq. (25) is iteratively optimized by two steps. Firstly, the algorithm assigns each object V_i to the cluster k that minimizes $d(V_i, \mathbb{C}_k^*), \forall k \in \{1, 2, ..., \mathcal{K}\}$. Then, it updates the centroid of each cluster according to the current clustering result (*i.e.*, $\mathbb{C}_k^* = \frac{\sum_{i=1}^{q} x_i^k V_i}{\sum_{i=1}^{q} x_i^k}$). The algorithm terminates until the clustering result remains unchanged or the objective function $\mathbb{E}$ does not changes significantly.

4.2.2 Problem Definition of Cluster Construction

There are three drawbacks of the K-means algorithm that prevents us from constructing the efficient cluster topology for FL directly. 1) The K-means algorithm treats all attributes of an object independently and equally, and calculates the mean square error of these attributes between objects to evaluate their difference for deciding the cluster memberships. However, for two attributes (*e.g.*, computing and communication capacities) of clients, they always have different effects on training efficiency of different models. For example, the computing capacity dominates the training speed of computing-intensive models. 2) Except for the centroids of the $\mathcal{K}$ clusters in the initialization process, the centroid of each cluster may be a virtual object with the given attribute values during the iteration [57]. However, the cluster centers in FedCH must be real clients for performing the cluster aggregation. 3) The number of clients in each cluster will range from 1 to N-$\mathcal{K}$+1 by the K-means algorithm. In some cases, each cluster may only contain one client [58]. Due to the non-IID data among clients, the model from a single client may severely destroy the convergence of the global model. Based on the above considerations, we first give computing and communication models of each client and then provide the definition of the clustering problem for FL with N clients.

To organize clients with similar model training capabilities into a cluster, the PS demands to profile the local training time (*i.e.*, $C_{i,cmp}$) and the communication time (*i.e.*, $C_{i,com}^k$) for model transmission of each client i. Specifically, for each client, the local training time (*i.e.*, $C_{i,cmp}$) at each epoch is determined by its computing capacity (*e.g.*, CPU frequency), the dataset size and the number of local updates, *i.e.*,

$$C_{i,cmp} = H \frac{|D_i| f}{f_i} \tag{26}$$

where f and f_i are the computing workload of each data sample (which is a constant for the trained model) and computing capacity of client i, respectively. For the communication delay, we denote the model size as M, therefore, we have

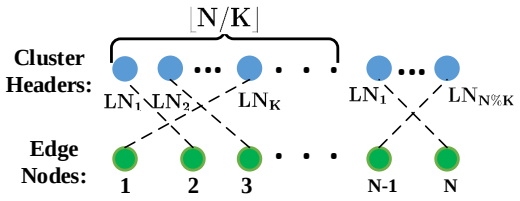
$$C_{i,com}^k = \frac{M}{B_{i,k}} \tag{27}$$

where $B_{i,k}$ is the transmission rate of client i for forwarding the updated model to the corresponding cluster header LN_k. Based on the above definition, we give the problem definition as follows,

$$\min_{x_i^k} \sum_{i=1}^{N} \sum_{k=1}^{\mathcal{K}} x_i^k \cdot d(i, LN_k)$$

$$s.t. \begin{cases} d(i, LN_k) = |C_{i,cmp} + C_{i,com}^k - C_{k,cmp}|, & \forall i, k \\ \sum\limits_{k=1}^{\mathcal{K}} x_i^k = 1, & \forall i \\ \sum\limits_{k=1}^{\mathcal{K}} n_k = N \\ n_k \in \{\lfloor \frac{N}{\mathcal{K}} \rfloor, \lceil \frac{N}{\mathcal{K}} \rceil\}, x_i^k \in \{0, 1\} & \forall i, k \end{cases} \tag{28}$$

Since each client i performs the local updates and forwards the updated model to the cluster header LN_k for aggregation, we define the dissimilarity measure between client i and the centroid of cluster k as $d(i, LN_k)$ as the first set of equalities in Eq. (28), where $C_{k,cmp}$ is the local training time of the client LN_k. The second equation in Eq. (28) denotes that each client must be assigned to one



	LN_1	···	LN_K	· · ·	LN_1	···	$LN_{N\%K}$
1	d(1, LN_1)		d(1, LN_K)		d(1, LN_1)		d(1, $LN_{N\%K}$)
2	d(2, LN_1)		d(2, LN_K)		d(2, LN_1)		d(2, $LN_{N\%K}$)
⋮				⋱			
N	d(2, LN_1)		d(N, LN_K)		d(2, LN_1)		d(N, $LN_{N\%K}$)

Fig. 2: The constructed bipartite graph and the matrix $\mathbb{A}$.

cluster. The third and fourth sets of formulas indicate that we adopt the well-known balanced clustering method to improve the convergence performance of model training and balance the weight of each cluster. The model from a single client will be aggregated with other $n_k - 1$ clients. Even if a local model get stuck at locally optimal value and deviate from the direction of global optimization due to the non-IID data, the intra cluster aggregation can minimize its impact on the global model for the reason that the weight of this model is reduced by n_k times for global aggregation. Besides, Eq. (12) has already shown that the expected loss of the aggregated model of each cluster k is that smaller than that of client i in this cluster. After we minimize the objective function in Eq. (28), the clients with similar model training abilities will be assigned together, and perform the low-cost synchronous training. In the following, we will propose an effective balanced clustering algorithm based on the existing K-means algorithm.

4.2.3 Algorithm for Cluster Construction

In the clustering algorithm, we will match each client with the corresponding cluster header. Since there are only two types of entities (*i.e.*, cluster headers and clients) in this matching problem, a bipartite graph [59], [60] is introduced to deal with this problem. However, a normal bipartite graph, which is composed of K cluster headers and N clients, can not address the fourth constraint in Eq. (28). To this end, we construct a bipartite graph as shown in Fig. 2. Specifically, for each cluster header LN_k with $1 \leq k \leq N\%\mathcal{K}$, we generate $\lfloor N/\mathcal{K} \rfloor + 1$ vertexes. Moreover, we generate $\lfloor N/\mathcal{K} \rfloor$ vertexes for each cluster header LN_k with $N\%\mathcal{K} < k \leq \mathcal{K}$. As a result, there are totally N vertexes on one side of the bipartite graph. On the other side, we also generate N vertexes, one for each client. To construct the optimal cluster structure is equivalent to find a maximum matching of this bipartite graph. For simplicity, we create an N×N matrix $\mathbb{A}$ in Fig. 2, in which each element denotes the distance between the corresponding client and the cluster header. We demand to search N values of different rows and columns in $\mathbb{A}$ to minimize their sum, then, the optimal matching of the bipartite graph is obtained. This problem can be solved by using the Hungarian algorithm [61] in four steps.

- Each value in $\mathbb{A}$ subtracts the minimum value of its row.
- Each value in $\mathbb{A}$ subtracts the minimum value of its column.
- We count the minimum number of lines (horizontal or vertical) that can cover all zeros in the $\mathbb{A}$. If the number is equal to N, the algorithm terminates and outputs the assignment.
- We find the smallest uncovered value from $\mathbb{A}$ after the previous step, subtract it from all uncovered values and add it to all values that are covered twice. We return to the previous step until the algorithm stops.

We have stated the matching method for a given centroid of each cluster, however, the cluster headers dynamically change during the whole clustering process. We formally introduce the effective balanced clustering algorithm in three phases, as shown in Alg. 2.

1) *Initialization (Line 1-2):* We select $\mathcal{K}$ arbitrary clients from N nodes as the centroid of each cluster and create the matrix $\mathbb{A}$.
2) *Cluster Construction (Line 4-10):* We compute $d(i, LN_k)$ for each client i and LN_k based on the first set of equation in Eq. (28), and put these values in $\mathbb{A}$. We construct the clusters by the Hungarian algorithm over the matrix $\mathbb{A}$.
3) *Cluster Topology Adjustment (Line 11-16):* After assigning each client i to cluster k (*i.e.*, $x_i^k = 1$), we demand to update the centroid of each cluster. For cluster k with n_k clients, we calculate $\sum_j d(j, i)$ for each node i, where j denotes the other clients in the same cluster as node i. To minimize the objective function in Eq. (28), the client i that can minimize the value of $\sum_j d(j, i)$ in each cluster k will be selected as the new LN_k.

We iteratively perform two phases of *Cluster Construction* and *Cluster Topology Adjustment* until the cluster structure does not change or the objective function in Eq. (28) does not decrease any more. The time complexity of Alg. 2 mainly depends on the process of *Cluster Construction*, which can be solved in $O(n^3)$ time by using the Hungarian algorithm [61]. Since the algorithm is executed by the PS with powerful

Algorithm 2 Balanced Clustering for FedCH

1: Initialize the centroid LN_k of each cluster k;
2: Initialize the $N \times N$ matrix $\mathbb{A}$;
3: **repeat**
4: **for** Each client $i \in \{1, 2, ..., N\}$ **do**
5: **for** Each cluster center $LN_k \in \{LN_1, ..., LN_\mathcal{K}\}$ **do**
6: Compute $d(i, LN_k)$ based on Eq. (28);
7: **end for**
8: **end for**
9: Update the matrix $\mathbb{A}$;
10: Construct clusters using $\mathbb{A}$ by Hungarian algorithm;
11: **for** Each cluster $k \in \{1, 2, ..., \mathcal{K}\}$ **do**
12: **for** Each client $i \in \{1, 2, ..., N\}$ **do**
13: Compute $\sum_j d(j, i)$;
14: **end for**
15: Update the centroid LN_k of cluster k;
16: **end for**
17: **until** the cluster structure does not changes or the objective function in Eq.(28) does not decrease;
18: **Return** the final cluster topology;

computing resource, its running time is negligible compared with the model training time.

4.3 Extension to Dynamic Scenarios

In this subsection, we extend FedCH to dynamic scenario, in which the network conditions may vary with time. We propose two approaches, *i.e.*, fixed re-clustering, and adaptive re-clustering.

4.3.1 Fixed Re-clustering

Fixed re-clustering means clustering all clients after every T_b epochs, where T_b is a pre-set constant. Obviously, the PS maintains a counter t_b to record the number of experienced epochs since the last re-clustering. If $t_b \geq T_b$, re-clustering will be triggered. To determine the value of T_b, we should consider the specific value of $\mathcal{K}$. A small value of $\mathcal{K}$ indicates that there are more clients in one cluster. Thus, the FL mechanism is more susceptible to the stragglers effect, and we adopt a small T_b. But for $\mathcal{K} = 1$ and $\mathcal{K} = N$, we set $T_b \to \infty$, because both FL mechanisms can not implement re-clustering. However, the way to obtain the optimal value of T_b demands future study.

4.3.2 Adaptive Re-clustering

Fixed re-clustering is only triggered after every T_b epochs. Thus, it can not fully adapt to the sudden deterioration of network status. To be more flexible, we propose another approach, called adaptive re-clustering. PS adopt s_k to record the condition of each cluster k, where $s_k = 0$ indicates that no straggler appears in cluster k, otherwise $s_k = 1$. Specifically, PS will set s_k as 1 when it discovers the occurrence of straggler in cluster k, *i.e.*, the time between two consecutive global aggregations of cluster k is greatly increased. When a certain number, *e.g.*, $\widetilde{\mathcal{K}} \in \{1, .., \mathcal{K}\}$, of clusters report the presence of stragglers, $i.e.$,

$$\sum_{k=1}^{\mathcal{K}} s_k = \widetilde{\mathcal{K}} \tag{29}$$

re-clustering will be triggered. After adaptive re-clustering, we reset $t_b = 0$ and $s_k = 0, \forall k$. Even though some clients may die during training, the fixed re-clustering and adaptive re-clustering can be triggered normally to discard the dead nodes and construct new clusters for future training. Therefore, the combination of two approaches can deal with the slow or sudden degradation of training conditions, *e.g.*, straggler effect and nodes failure, and maintain efficient training until the ML task is completed.

We then briefly introduce the training process of FedCH when adopting the above two approaches, namely Dyn-FedCH. For each client, it uploads the real-time computing delay and communication delay together with the updated model to the cluster header and then the PS after local updates. The PS will update the corresponding information of each client for constructing new clusters. The computing delay can be directly obtained by each client, which is set as the time consumption of the latest completed local updates. For the communication delay, each client demand to measure the transmission rate for forwarding the updated model to all other clients, which can be implemented by using some transmission rate measurement tools (e.g., iperf3 [62]) or bandwidth prediction kits [63], [64] to obtain the dynamic network condition. By profiling the real-time computation delay of each client and the communication delay among them, the PS can perform effective re-clustering. In Dyn-FedCH, whenever the PS collects updates from a cluster, it checks whether the conditions for re-clustering are satisfied or not. Re-clustering only occurs when the conditions for fixed re-clustering (*i.e.*, $t_b \geq T_b$) or adaptive re-clustering (*i.e.*, $\sum_{k=1}^{\mathcal{K}} s_k \geq \widetilde{\mathcal{K}}$) are met. Then PS will re-compute the value of $\mathcal{K}^*$, construct new clusters based on the proposed algorithms, and distribute the re-clustering results. Each client will forward the local model after local updates to the new cluster headers based on the updated cluster topology in its next training epoch.

TABLE 2: The different computing capacities of 20 clients in the experiments.

Mode	Denver 2	ARM A57	GPU	Device id
0	2.0GHz×2	2.0GHz×4	1.3GHz	1, 2, 3, 4
1	0.0GHz	1.2GHz×4	0.85GHz	5, 6, 7, 8
2	1.4GHz×2	1.4GHz×4	1.12GHz	9, 10, 11, 12
3	0.0GHz	2.0GHz×4	1.12GHz	13, 14, 15, 16
4	1.4GHz×2	1.4GHz×4	0.0GHz	17, 18, 19, 20

5 Performance Evaluation

In this section, we conduct abound experiments on both the physical platform and the simulated environment to evaluate the effectiveness of our proposed algorithms. We firstly discuss the experimental settings in detail, and then present the experimental results on the prototype system and simulated environment, respectively. Finally, we provide a brief summary of those results.

5.1 Experiment Settings

5.1.1 Experiment Environment

The deployment of our testbed spans two parts: one parameter server and twenty clients. As shown in Fig. 3, we use an AMAX deep learning workstation as the parameter server in the experiments. This machine is carrying an 8-core Intel(R) Xeon(R) CPU (E5-2620v4) and 4 NVIDIA GeForce RTX 2080Ti GPUs with 11GB RAM, and the OS is Ubuntu 16.04.1 LTS. We also adopt 20 NVIDIA Jetson TX2 as devices, labeled from 1 to 20. Each TX2 is equipped with one GPU and one CPU cluster, which consists of a 2-core Denver2 and a 4-core ARM CortexA57 with 8GB RAM, and its OS is Ubuntu 18.04.4 LTS. In the implementation of experiments, we place the PS and clients at different locations at least 1,000 meters apart and let them communicate via WAN. This represents the real-world edge computing environment where the PS is located at a remote cloud and communicates with the clients at the network edge through WAN. The clients are arranged together and communicate through a WLAN. Meanwhile, to reflect the heterogeneity of clients, we by default adopt 5 different modes of computing capacities for 20 TX2 clients, and each computing mode is used by four devices, as shown in Table 2.

5.1.2 Models and Datasets

We carry out the experiments over four different models (*i.e.*, SVM [49], LR [45] and two CNN models [46]) and

Fig. 3: The physical platform for the experiments.

two real datasets (*i.e.*, MNIST [65] and CIFAR-10 [66]). Two models SVM and LR with convex loss functions (Section 3.2) are trained over MNIST, which is composed of 60,000 handwritten digits for training and 10,000 for testing. They divide the digits into odd and even categories. Two CNN models with different structures are trained over MNIST and CIFAR-10, respectively. Though the loss functions of CNN are non-convex and do not satisfy Assumptions 1, we also conduct experiments by using CNN to prove the general applicability of our proposed mechanism. CNN has two 5×5 convolution layer, two max-pooling layers and three fully-connected layers for CIFAR-10 (two fully-connected layers for MNIST). CIFAR-10 includes 50,000 color images for training and 10,000 for testing, and has ten different types of objects. We perform stochastic gradient descent to process the mini-batch samples for training [17].

5.1.3 Benchmarks

We compare our proposed algorithms with two well-known FL algorithms for performance evaluation. 1) The first benchmark is FedAsync [17], which is a typical asynchronous FL algorithm with staleness treatment. The global update of FedAsync is performed as soon as one model from an arbitrary client is received by the PS, and this algorithm can guarantee a near-linear convergence to a global optimum. 2) We also choose an improved communication-efficient synchronous FL algorithm FedAvg [67] as the baseline, which can significantly reduce the communication round during federated learning compared with normal synchronized stochastic gradient descent. It randomly selects a fixed number of clients (*i.e.*, a subset) in each epoch and aggregates the local models from these clients for global update. We set the subset size as z. For a fair comparison, we take multiple values of z in different experiments.

5.1.4 Performance Metrics

We mainly adopt four widely used metrics for performance evaluation. 1) *Loss function* measures the difference between the predicted values and the actual values. 2) *Classification accuracy* is the proportion of correctly classified samples to all samples for testing of the image classification tasks. 3) *Network traffic*, which is defined as the total size of data transmitted through the network. We adopt this metric to quantify the communication cost of different algorithms. 4) *Completion time* denotes the time spent until training terminates, which is used to evaluate the model training speed. Loss function and classification accuracy are used to validate whether a federated learning algorithm can effectively guarantee the convergence of the model or not. Meanwhile, the network traffic and completion time show if the algorithms are resource-efficient.

5.1.5 Data Distribution

Data, as the foundation of model training, will directly influence the training efficiency. Therefore, to evaluate the impact of data distribution on the performance of different algorithms, we will consider two main factors while distributing the entire dataset (with its size D) into separate clients. 1) We first investigate the effect of non-independent and identical distributed (non-IID) data on model training efficiency by distributing the dataset in different non-IID levels. In level 1, the data is IID in each client. In level 2, each node only has the data samples with half labels. In level 3, the data samples in each node are always with the same label. 2) We also consider the data imbalance. In case 1, data samples are assigned to each client uniformly (*i.e.*, $\frac{D}{100}$ for each node). In case 2 and case 3, the dataset is distributed to each client according to the Gaussian distribution with the same expectation (*e.g.*, $\frac{D}{100}$) but different standard deviation σ (*e.g.*, 100 for case 2 and 300 for case 3), which denotes the data size of each client mainly ranging from $\frac{D}{100} - 3\sigma$ to $\frac{D}{100} + 3\sigma$ with a probability of 99.73%.

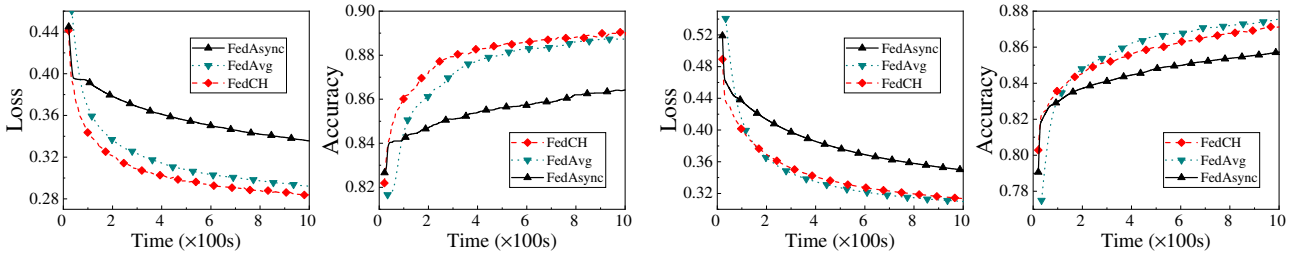
5.1.6 Experiment Parameters

In all experiments, we by default set the learning rate η as 0.01, and the number of local updates in each epoch as $H = 10$ [6]. We also by default distribute the data to all clients uniformly and randomly. For our proposed algorithm, we compare its training performance under the time and network budget constraints. We consider one resource type in each experiment for the convenience of presentation of experiment results. We adopt $a = 5$ and $b = 1$ to deal with staleness. We adopt the average results of 5 independent experiments to avoid accidents.

5.2 Experimental Results on the TestBed

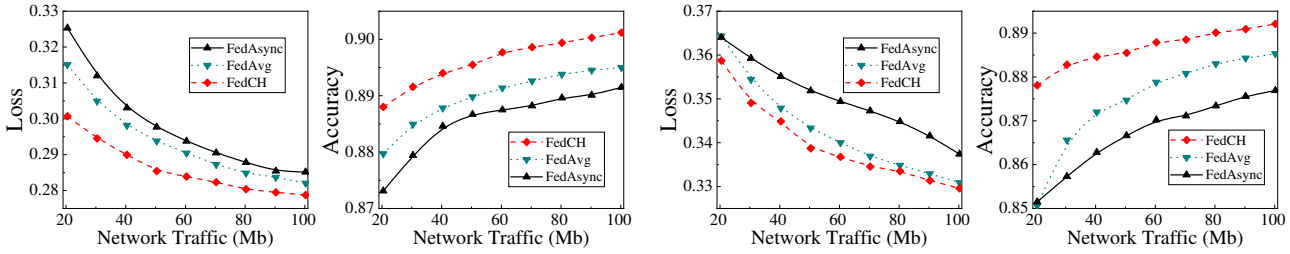
5.2.1 Effect of Time Budgets

Our first set of experiments compares the different performances of FedCH and baselines with time budgets. The results are shown in Figs. 4-6. Firstly, we observe how the loss function and classification accuracy change with the training time for SVM and LR over MNIST with a total time budget of 1,000s in Figs. 4(a) and 4(b). We note that FedCH converges faster than the other two algorithms at the beginning of training and always achieves a lower loss and a higher accuracy compared with FedAsync. Although we set different computing capacities of clients to simulate the system heterogeneity, the model training speed of the fastest clients is only about twice as fast as that of the slowest one. As a result, FedCH only achieves slightly better performance for SVM over MNIST and obtains a similar performance for LR over MNIST compared with FedAvg ($z = 20$). As the heterogeneity among clients increases, FedCH will be more effective. Secondly, we measure the completion time of FedAsync and FedAvg when they achieve the final accuracy of FedCH with a smaller time budget (*i.e.*, 100s). The left plot of Fig. 6 shows that FedCH substantially outperforms



(a) SVM over MNIST. (b) LR over MNIST.

Fig. 4: Loss and Accuracy vs. Time for SVM and LR over MNIST.



(a) SVM over MNIST. (b) LR over MNIST.

Fig. 5: Loss and Accuracy vs. Network Traffic for SVM and LR over MNIST.

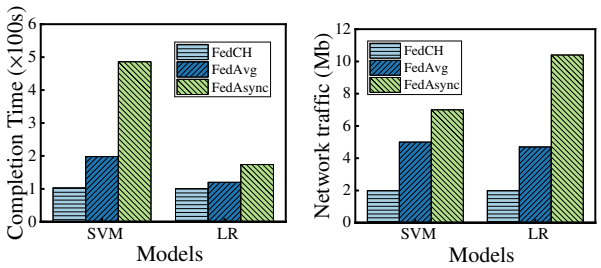


Fig. 6: The Comparison of Resource Consumption. *Left plot:* Time; *Right plot:* Network Traffic.

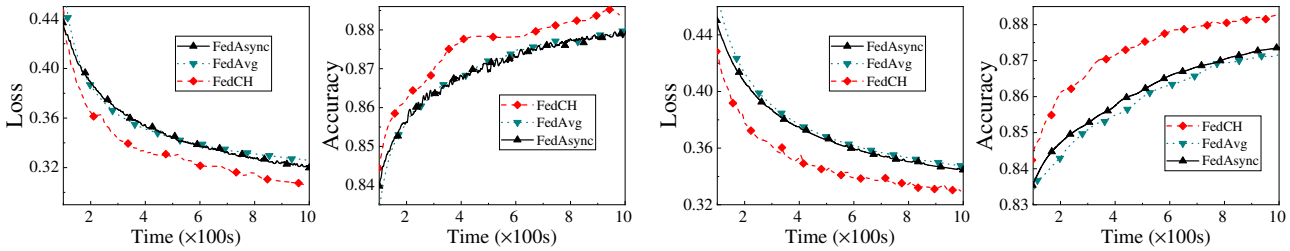
two benchmarks for both SVM and LR over MNIST. For example, the completion time of FedAvg and FedAsync is 198s and 486s, respectively. Thus, compared with baselines, FedCH achieves a completion time reduction of 49.5-79.5%. That is because FedCH clusters the clients with similar training capacities together and adopts the proposed hierarchical aggregation for training, which can effectively handle the straggle effect and help to achieve a faster convergence with time budgets.

5.2.2 Effect of Network Traffic

We also conduct experiments to evaluate the communication cost of different algorithms by analyzing the influence of the network traffic on the training efficiency. Figs. 5(a) and 5(b) show that FedCH always achieves the best performance under different traffic budgets for SVM and LR over MNIST, compared with both FedAvg and FedAsync. For example, the classification accuracy of FedCH with a budget of 100Mb is 90.1% for SVM over MNIST, which is higher than that of FedAvg (89.5%) and FedAsync (89.15%). Meanwhile, the right plot of Fig. 6 shows the network traffic by two baselines when they achieve the accuracy of FedCH with a budget of 20Mb. We observe that FedAvg and FedAsync consume 47Mb and 104Mb for LR over MNIST respectively. In other words, FedCH reduces the consumption of network traffic by about 57.4-80.8% compared with the two benchmarks. The reason for the performance improvement of FedCH under the network traffic budgets is that FedCH performs global aggregation asynchronously among clusters and guarantees the model convergence. On the contrary, FedAvg consumes too much traffic in each epoch and FedAsync requires a large number of training epochs to achieve a similar accuracy.

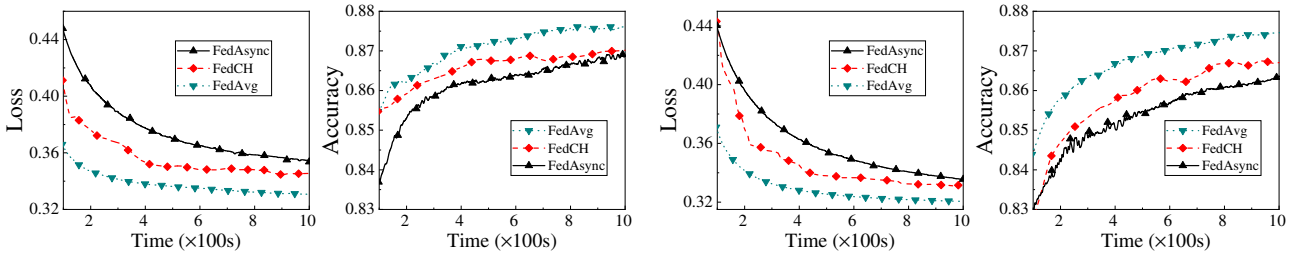
5.2.3 Effect of Different non-IID levels

We also evaluate how the data distribution with different non-IID levels influences the training performance. We conduct the experiments for SVM and LR over MNIST with different non-IID levels on the prototype system. The results are shown in Fig. 7. Since we have presented the performance for SVM and LR with IID data (*i.e.*, non-IID level 1) in the first set of experiments, we omit it here. We make three observations from the results. Firstly, Figs. 7(a)-7(b) show that FedCH performs better than FedAvg and FedAsync with non-IID level 2 where each client only have the data samples with half labels. For example, FedCH achieves an accuracy of 88.2% for LR over MNIST, which is higher than that of two baselines. Secondly, when we only assign the data samples with the same label to each client (*i.e.*,non-IID level 3), the synchronous algorithm FedAvg achieves a better performance than FedCH. Finally, we also note that as the non-IID level increases, the highest accuracy that each algorithm can achieve also decreases. Since FedCH can perform more training epochs than FedAvg under the same time budgets, and aggregate multiple local models from clients in the same cluster in each epoch to alleviate the negative effect of data distribution, FedCH reasonably converges faster than baselines with low non-IID levels. Besides, FedAvg performs model aggregation by collecting local models from all clients, the global model will not be severely affected by the data distribution. For FedAsync, the global model will be easily deteriorated by the adverse model from an arbitrary client as the non-IID level increases, leading to poor convergence performance.



(a) SVM over MNIST with non-IID level 2.

(b) LR over MNIST with non-IID level 2.



(c) SVM over MNIST with non-IID level 3.

(d) LR over MNIST with non-IID level 3.

Fig. 7: Loss and Accuracy vs. Time for SVM and LR over MNIST with Different non-IID Levels.

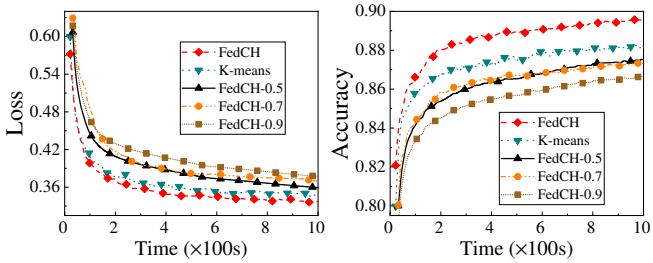


Fig. 8: Loss and Accuracy vs. Time with Different Clustering Algorithms for SVM.

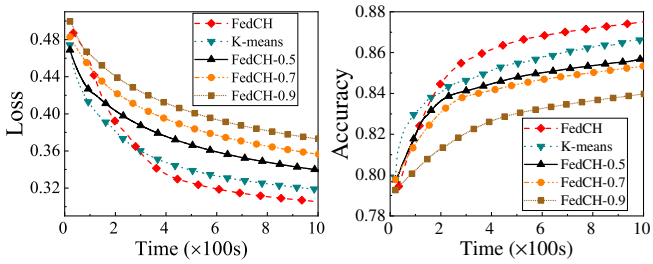


Fig. 9: Loss and Accuracy vs. Time with Different Clustering Algorithms for LR.

5.2.4 *Performance of Clustering Algorithm*

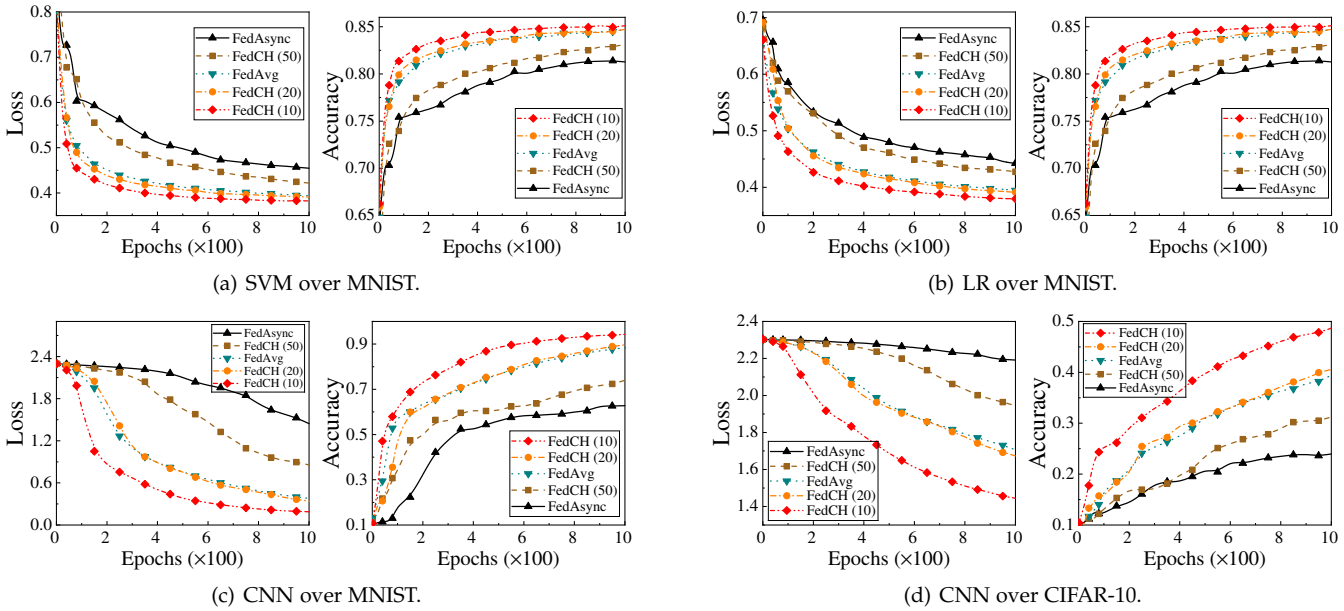
Figs. 8-9 compare the loss and accuracy of the proposed clustering algorithm with different clustering algorithms under non-IID level 2. For FedCH, the clients are assigned to each cluster uniformly. We also introduce three baselines to evaluate the impact of distribution of client number among clusters on the performance of FedCH, namely FedCH-χ (*e.g.*, 0.5, 0.7, and 0.9). Specifically, FedCH-χ (*e.g.*, 0.5, 0.7, and 0.9) denotes that the number of clients in each cluster can fluctuate between $(1-\chi)\frac{N}{K}$ and $(1+\chi)\frac{N}{K}$, where $N = 20$ in the experiments. Besides, we also take the K-means clustering algorithm as one of the baselines. The experiments for SVM and LR over MNIST are conducted on the physical platform. Two observations are as follows. Firstly, we see that our proposed clustering algorithm outperforms the other four clustering algorithms during training. For example, when training MNIST by SVM, FedCH achieves an accuracy of 89.5% when adopting the proposed clustering algorithm after 1,000s. But the accuracy is 88.2%, 87.5%, 87.3% and 86.6% for K-means, FedCH-0.5, FedCH-0.7 and FedCH-0.9, respectively. Secondly, as χ increases, the greater the difference in the number of clients in each cluster, and the worse the convergence performance of the trained model. This result verifies the efficiency of the balanced clustering algorithm. This is because if the number of clients in some clusters is too small, the aggregated cluster models of those clusters are more possible to get stuck at locally optimal values and deteriorate the global model, leading to a poor convergence performance.

5.3 Simulation Results

To evaluate the performance of our proposed algorithms in a larger-scale scenario, we also conduct our evaluations in a simulated environment with 100 clients. The simulations are performed on the AMAX deep learning workstation, which is equipped with an 8-core Intel(R) Xeon(R) CPU (E5-2620v4) and 4 NVIDIA GeForce RTX 2080Ti GPUs with 11GB RAM. We adopt the same parameter settings (*e.g.*, $\eta = 0.01$ and $H = 10$) as the experiment in the prototype system. Based on the tests on TX2 devices with different computing modes (0-4) in Table 2, we randomly assign each virtual client with a different capacity factor (0-4) accordingly that denotes the time consumption for performing local updates. Specifically, the computing capacity of a client is gradually decreasing in the order of modes 0, 2, 3, 1 and 4. For mode 0, each client only consumes 4-6s for local updates (*i.e.*, training 6,000 data samples) for SVM over MNIST. But for mode 4, the time consumption is increased to 15-18s. The transmission rates between clients are randomly set to 1-10Mbps by default to simulate the heterogeneous network condition. For a fair comparison, all algorithms are implemented under the same settings.

5.3.1 *Convergence Performance*

We first conduct experiments in a simulated environment to observe the convergence performance of different al-



(a) SVM over MNIST. (b) LR over MNIST.

(c) CNN over MNIST. (d) CNN over CIFAR-10.

Fig. 10: Loss and Accuracy vs. No. of Epochs for SVM, LR and CNN over MNIST and CNN over CIFAR-10.

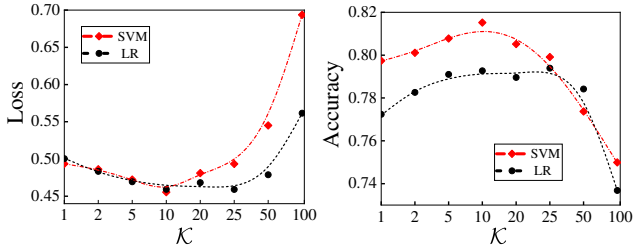


Fig. 11: Loss and Accuracy vs. $\mathcal{K}$ for SVM and LR.

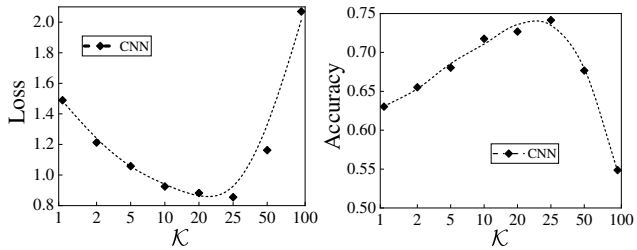


Fig. 12: Loss and Accuracy vs. $\mathcal{K}$ for CNN.

gorithms with different numbers of epochs, ranging from 0 to 1,000. Since Assumption 1 is not satisfied for the loss function of CNN (*e.g.*, non-convex), we conduct the experiments on several values of $\mathcal{K}$ (*e.g.*, 10, 20, and 50) for FedCH without considering the time budget. FedCH(10) denotes the algorithm with $\mathcal{K} = 10$. The results for SVM, LR and CNN over MNIST and CNN over CIFAR-10 are shown in Figs. 10. We make two observations from those results. Firstly, we note that FedCH(10) achieves the best convergence performance compared with other settings, and the increase of $\mathcal{K}$ leads to a performance degradation of FedCH. For example, after training 1,000 epochs over CIFAR-10 using CNN, the accuracy of FedCH(10) and FedAvg is 56% and 39%, and the performance of FedAvg is similar to that of FedCH(20). Besides, FedCH requires about 300 epochs to achieve the same accuracy of FedAsync after 1,000 epochs for CNN over MNIST, which reduces the number of training epochs by 70%. Secondly, the superiority of FedCH is better reflected on the complex model (*i.e.*, CNN) and dataset (*i.e.*, CIFAR-10). For example, FedCH(10) only achieves an accuracy improvement of 4.2% for SVM over MNIST, compared with FedAsync. But for CNN over MNIST and CIFAR-10, the accuracy of FedCH(10) is 40.8% and 103% higher than that of FedAsync. Those results demonstrate the efficiency of FedCH when training models with non-convex functions. By cluster aggregation, each aggregated model w_k of the cluster k can be trained over a virtual big dataset (*i.e.*, the union of local datasets of all clients in cluster k). As $\mathcal{K}$ increases, the number of clients as well as the size of this virtual big dataset in each cluster decreases. Therefore, FedCH(10) converges faster than FedCH(20), FedCH(50), FedAvg and FedAsync with respect to the training epochs.

5.3.2 *Effect of Cluster Number*

We further study the effect of cluster number $\mathcal{K}$ on the training efficiency of FedCH in the simulated environment. The cluster number varies from 1 to 100 for SVM, LR and CNN over MNIST. Apparently, if $\mathcal{K}$ is 100, FedCH becomes FedAsync for the reason that the PS will perform the global update when one model from an arbitrary client is collected. Meanwhile, FedCH is same as FedAvg($z = 100$) if $\mathcal{K} = 1$. The results are shown in Figs. 11-12. Two observations are as follows. Firstly, we note that there are always different optimal values of $\mathcal{K}$ for different models and datasets under a certain time budget (*e.g.*, 600s for SVM/LR and 1,800s for CNN). For instance, FedCH achieves an accuracy of 74.2% with CNN over MNIST when it takes the optimal value of $\mathcal{K}$ (*i.e.*, 25), which is about 15% and 26% higher than FedAvg and FedAsync, respectively. But for SVM and LR, the optimal values of $\mathcal{K}$ are about 10 and 25, respectively. Different models always require different computing and communication resources during training, leading to a different training speed. To handle this, FedCH will construct different cluster topologies for improving the training efficiency. Secondly, from the results, we also observe that loss first decreases then increases when $\mathcal{K} \in [1, 50)$, and gradually increases when $\mathcal{K} \in [50, 100]$, which can verify

the proof in Theorem 3, *i.e.*, there always exists an optimal $\mathcal{K}$ between 0 and $\lfloor \frac{N+1}{2} \rfloor$ under a given time budget.

5.3.3 Effect of Data Imbalance

To analyze the impact of data imbalance, we test the completion time of different algorithms while achieving the same training accuracy in cases 1-3 (Section 5.1.5). We set the target accuracy as the accuracy that all algorithms can achieve, *i.e.*, 80% for SVM and 50% for CNN over MNIST. The completion time of different algorithms (*e.g.*, FedCH, FedAvg, and FedAsync) using SVM or CNN to train MNIST is shown in Fig. 13. Firstly, we observe that as the degree of data imbalance increases, the completion time of FedAvg and FedAsync also rapidly increases. However, the completion time for FedCH remains stable in all cases, especially for CNN over MNIST. We speculate that this may be attributed to the positive effect of clustering, which avoids the adverse effects of single clients (*i.e.*, FedAsync) and prevents the situation for waiting for all clients in each epoch (*i.e.*, FedAvg). In comparison, FedCH demands less training time than both FedAvg and FedAsync. For example, by the left plot of Fig. 13, the completion time of FedCH is about 1,036s in case 1 where data samples are assigned to each client uniformly, which is 56.8% and 67.2% less than that of FedAvg and FedAsync, respectively.

5.3.4 Effect of Dynamic Scenarios

To emulate a dynamic scenario in edge computing, and observe the performance of the Dyn-FedCH algorithm, we conduct the following configurations for communication and computation. We simulate two different network conditions, *i.e.*, the high and low dynamic scenarios. In the high dynamic scenario, we let the transmission rates between clients fluctuate between 1Mbps and 10Mbps, and randomly change the speed of those links every epoch. According to the experiments in the physical platform, we set the computation delay of each client for local updates to vary between 5s and 10s for CNN over MNIST. In the low dynamic scenario, the transmission rates between clients vary between 4Mbps and 6Mbps, and we only change the link speed every 10 epochs. Meanwhile, the computation delay fluctuates between 5s and 7s. For our proposed mechanism, we set the latency of profiling the network bandwidth and the computing capacity of each client for cluster construction as a relatively large proportion (*e.g.*, 10%) of the completion time of the current epoch. For Dyn-FedCH, we also adopt two settings, *i.e.*, Dyn-FedCH-(1,1) and Dyn-FedCH-(10,5). Specifically, Dyn-FedCH-(10,5) means that we adopt $T_b = 10$ and $\widetilde{\mathcal{K}} = 5$ for performing low-frequency re-clustering. Meanwhile, Dyn-FedCH-(1,1) will update the constructed clusters in each training epoch based on the real-time computing power and network status. The results for CNN over MNIST are shown in Figs. 14-15. Three observations are as follows.

Firstly, the right plots in Figs. 14-15 shows that our proposed approaches, including FedCH, Dyn-FedCH-(10,5) and Dyn-FedCH-(1,1), converge faster than baselines. FedAvg ($z = 100$) achieves a slighter better convergence performance with respect to training epochs in two dynamic scenarios, compared with our proposed approaches. However, since its per-epoch completion time depends on

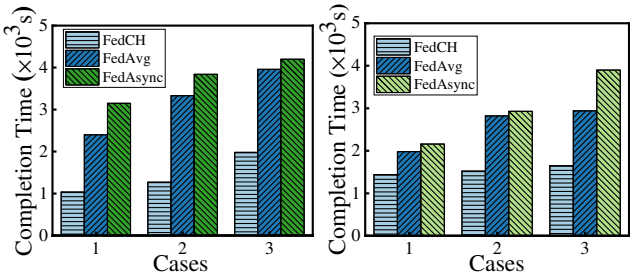


Fig. 13: Completion Time vs. Data Distribution Cases. *Left plot:* SVM; *Right plot:* CNN.

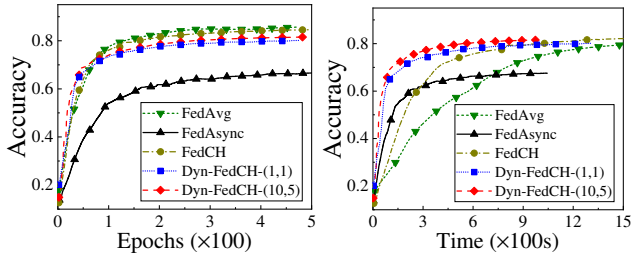


Fig. 14: Accuracy vs. No. of Epochs and Time for CNN over MNIST in Low Dynamic Scenarios.

the maximum training time among all clients, its training speed is much slower than our approaches. For example, when achieving an accuracy of 60% (an accuracy that all algorithms can reach) under high dynamic scenario, Dyn-FedCH-(10,5) reduce the time consumption by about 76.5% and 79.4% compared with FedAvg and FedAsync. Secondly, both Dyn-FedCH-(10,5) and Dyn-FedCH-(1,1) converge faster than FedCH for the reason that the static cluster structure cannot adapt to dynamic scenarios, leading to performance degradation. For example, the time consumption for achieving an accuracy of 60% in the high dynamic scenario is 48s, 130s and 273s for Dyn-FedCH-(1,1), Dyn-FedCH-(10,5) and FedCH, respectively. Thirdly, we also note that the frequency of re-clustering will affect the training performance to a certain extent. Specifically, Dyn-FedCH-(10,5) converges faster than Dyn-FedCH-(1,1) under low dynamic scenario while Dyn-FedCH-(1,1) outperforms Dyn-FedCH-(10,5) in the high dynamic scenario. This is because re-clustering can effectively reduce the per-epoch training time and adapt to the dynamic scenario, but it also incurs additional cost by profiling the network status. However, determining the optimal frequency of re-clustering of Dyn-FedCH for further accelerating FL remains to be studied.

5.3.5 Sensitivity of a and b

We finally analyze the influence of parameters a and b on convergence performance, which are adopted to handle the staleness during global aggregation. We adopt three values of a (*i.e.*, 5, 10 and 20) and b (*i.e.*, 0.5, 1 and 2) to train CNN over MNIST for 1,000 epochs in the simulated environment. The results are shown in Fig. 16. Two observations are as follows. Firstly, we note that the loss (accuracy) first decreases (increases) and then remains stable as b increases. In fact, for a model with high staleness τ (*i.e.*, $\tau > a$), its weight decreases with the increase of b during global aggregation. Therefore, a large value of b can effectively prevent the global model from being destroyed by the model with high staleness from the clusters. Secondly, we also observe that

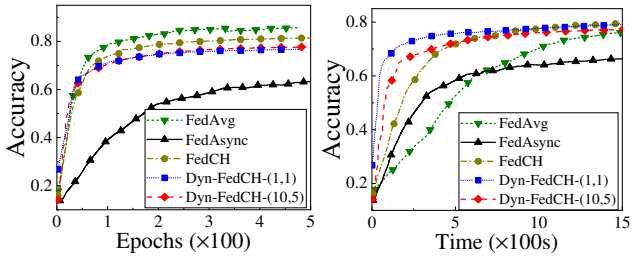


Fig. 15: Accuracy vs. No. of Epochs and Time for CNN over MNIST in High Dynamic Scenarios.

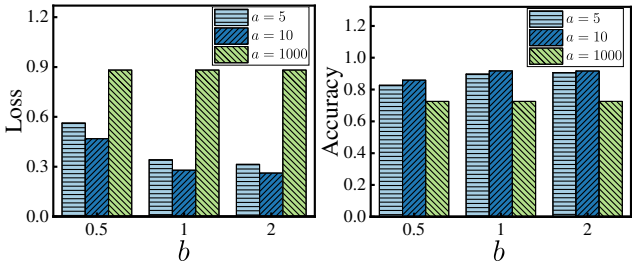


Fig. 16: Impact of Parameters a and b for CNN over MNIST. *Left plot:* Loss; *Right plot:* Accuracy.

different values of a have a slight impact on the training efficiency of FedCH except when $a = 1,000$. For example, when $a = 10$ and $b = 1$, FedCH achieves the accuracy of 91.8%, whereas if $a = 1,000$, the accuracy would be 26% lower. That is because $a = 1,000$ indicates that we have not taken measures to deal with staleness, and all the local models share the same weight when executing global aggregation, leading to a poor convergence performance.

5.4 Summary of Experiment Results

To summarize, our proposed algorithms can substantially outperform benchmarks in the following aspects. Firstly, FEDCH always outperforms two baselines under resource budgets. For example, FedCH reduces the resource consumption (*i.e.*, time and network traffic) by about 49.5-80.8% while achieving a similar performance in Figs. 5-6. Secondly, we observe that FedCH effectively deals with non-IID data. FedCH performs better than or similar to the synchronous method (*i.e.*, FedAvg) in different non-IID levels in Fig. 7. Furthermore, FedCH converges faster than benchmarks for different models and datasets during training. From Fig. 10, FedCH only requires 300 epochs to achieve the same accuracy as FedAsync after 1,000 epochs. We also realize that Dyn-FedCH handles the network dynamics well. Fig. 14-15 show that Dyn-FedCH reduces the time for achieving the target accuracy of about 76.5-79.4%, compared with baselines. Finally, Fig. 16 shows that FedCH with staleness treatment improves the classification accuracy by 26% compared with the algorithm without dealing with staleness.

6 Conclusion

In this paper, we have designed an effective federated learning mechanism, FedCH, to accelerate the model training with the cluster construction and hierarchical aggregation in heterogeneous edge computing. We have proposed efficient algorithms to determine the optimal number of clusters with resource constraints and construct cluster topology for model training. We have further extended our algorithm to deal with the network dynamics in practice. The experimental results have indicated that the proposed mechanism can obtain excellent performance compared with baselines. We believe that our proposed mechanism will provide a valuable solution for federated learning.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] B. M. Gaff, H. E. Sussman, and J. Geetter, "Privacy and big data," *Artificial Intelligence*, vol. 47, no. 6, pp. 7–9, 2014.

[3] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE Network*, vol. 32, no. 1, pp. 96–101, 2018.

[4] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, pp. 1–1, 2020.

[5] C. V. Networking, "Cisco global cloud index: Forecast and methodology, 2015-2020. white paper," *Cisco Public, San Jose*, 2016.

[6] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[7] D. Verma, S. J. Julier, and G. Cirincione, "Federated ai for building ai solutions across multiple agencies." *arXiv: Computers and Society*, 2018.

[8] A. Hard, C. Kiddon, D. Ramage, F. Beaufays, H. Eichner, K. Rao, R. Mathews, and S. Augenstein, "Federated learning for mobile keyboard prediction." *arXiv: Computation and Language*, 2018.

[9] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, B. Mcmahan *et al.*, "Towards federated learning at scale: System design," *arXiv: Learning*, 2019.

[10] M. Ammaduddin, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, "Federated collaborative filtering for privacy-preserving personalized recommendation system." *arXiv: Information Retrieval*, 2019.

[11] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *arXiv: Artificial Intelligence*, 2019.

[12] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi, "Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data," *arXiv: Machine Learning*, 2018.

[13] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," pp. 19–27, 2014.

[14] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1997–2006.

[15] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[16] J. Konecný, H. B. Mcmahan, D. Ramage, and P. Richtarik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv: Learning*, 2016.

[17] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization." *arXiv: Distributed, Parallel, and Cluster Computing*, 2019.

[18] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey." *arXiv: Networking and Internet Architecture*, 2019.

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning," pp. 265–283, 2016.

[20] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv: Computer Vision and Pattern Recognition*, 2017.

[21] H. Chai, S. Leng, Y. Chen, and K. Zhang, "A hierarchical blockchain-enabled federated learning algorithm for knowledge sharing in internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[22] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning." *arXiv: Networking and Internet Architecture*, 2019.

[23] Y. Tu, Y. Ruan, S. Wang, S. Wagle, C. G. Brinton, and C. Joe-Wang, "Network-aware optimization of distributed learning for fog computing," *arXiv preprint arXiv:2004.08488*, 2020.

[24] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey." *arXiv: Networking and Internet Architecture*, 2019.

[25] M. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, "Asynchronous federated learning for geospatial applications," pp. 21–28, 2018.

[26] Y. Chen, Y. Ning, and H. Rangwala, "Asynchronous online federated learning for edge devices." *arXiv: Distributed, Parallel, and Cluster Computing*, 2019.

[27] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[28] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[29] J. Ren, G. Yu, and G. Ding, "Accelerating dnn training in wireless federated edge learning systems," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 219–232, 2020.

[30] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 269–283, 2020.

[31] Y. Jin, L. Jiao, Z. Qian, S. Zhang, S. Lu, and X. Wang, "Resource-efficient and convergence-preserving online participant selection in federated learning," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 606–616.

[32] L. U. Khan, S. R. Pandey, N. H. Tran, W. Saad, Z. Han, M. N. Nguyen, and C. S. Hong, "Federated learning for edge networks: Resource optimization and incentive mechanism," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 88–93, 2020.

[33] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "Safa: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, 2020.

[34] J. Cao, Z. Zhu, and X. Zhou, "Sap-sgd: Accelerating distributed parallel training with high communication efficiency on heterogeneous clusters," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2021, pp. 94–102.

[35] S. Ji, W. Jiang, A. Walid, and X. Li, "Dynamic sampling and selective masking for communication-efficient federated learning," *arXiv preprint arXiv:2003.09603*, 2020.

[36] Y. J. Cho, J. Wang, and G. Joshi, "Client selection in federated learning: Convergence analysis and power-of-choice selection strategies," *arXiv preprint arXiv:2010.01243*, 2020.

[37] J. Xu, W. Du, Y. Jin, W. He, and R. Cheng, "Ternary compression for communication-efficient federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[38] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[39] Z. Ma, Y. Xu, H. Xu, Z. Meng, L. Huang, and Y. Xue, "Adaptive batch size for federated learning in resource-constrained edge computing," *IEEE Transactions on Mobile Computing*, 2021.

[40] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "Hfel: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6535–6548, 2020.

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

[42] N. Mhaisen, A. Awad, A. Mohamed, A. Erbad, and M. Guizani, "Optimal user-edge assignment in hierarchical federated learning based on statistical properties and network topology constraints," *IEEE Transactions on Network Science and Engineering*, 2021.

[43] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng, "Tifl: A tier-based federated learning system," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, 2020, pp. 125–136.

[44] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 59–71, 2020.

[45] S. L. Gortmaker, D. W. Hosmer, and S. Lemeshow, "Applied logistic regression." *Contemporary Sociology*, vol. 23, no. 1, p. 159, 1994.

[46] S. Shalev-Shwart and S. Ben-David, "Understanding machine learning: From theory to algorithms." *Cambridge, U.K.: Cambridge Unix. Press*, 2015.

[47] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," *arXiv: Machine Learning*, 2019.

[48] S. Wassertheil and J. Cohen, "Statistical power analysis for the behavioral sciences," *Biometrics*, vol. 26, no. 3, p. 588, 1970.

[49] T. Joachims, "Making large-scale svm learning practical," *Technical reports*, 1998.

[50] X. Chang, F. Nie, Z. Ma, and Y. Yang, "Balanced k-means and min-cut clustering," *arXiv: Learning*, 2014.

[51] A. Banerjee and J. Ghosh, "Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres," *IEEE Transactions on Neural Networks*, vol. 15, no. 3, pp. 702–719, 2004.

[52] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the royal statistical society. series c (applied statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[53] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[54] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 29, no. 3, pp. 433–439, 1999.

[55] K. Wagstaff, C. Cardie, S. Rogers, S. Schroedl *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[56] T. S. Madhulatha, "An overview on clustering methods," *arXiv preprint arXiv:1205.1117*, 2012.

[57] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering." in *ICML*, vol. 98. Citeseer, 1998, pp. 91–99.

[58] H. Liu, J. Han, F. Nie, and X. Li, "Balanced clustering with least square regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[59] H. Zha, X. He, C. Ding, H. Simon, and M. Gu, "Bipartite graph partitioning and data clustering," in *Proceedings of the tenth international conference on Information and knowledge management*, 2001, pp. 25–32.

[60] F. Serratosa, "Fast computation of bipartite graph matching," *Pattern Recognition Letters*, vol. 45, pp. 244–250, 2014.

[61] D. Bruff, "The assignment problem and the hungarian method," *Notes for Math*, vol. 20, no. 29-47, p. 5, 2005.

[62] ESnet and L. B. N. Laboratory, ""iperf3." http://software.es.net/iperf/."

[63] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei, "Linkforecast: cellular link bandwidth prediction in lte networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1582–1594, 2017.

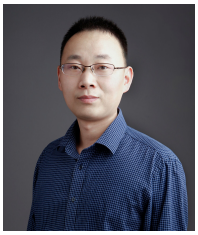
[64] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Predictable 802.11 packet delivery from wireless channel measurements," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 159–170, 2010.

[65] 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.

[66] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[67] H. B. Mcmahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," pp. 1273–1282, 2017.

Zhiyuan Wang received the B.S. degree from the Jilin University in 2019. He is currently studying for a master's degree in the School of Computer Science, University of Science and Technology of China (USTC). His main research interests are edge computing, federated learning and distributed machine learning.



Hongli Xu (Member, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China, China, in 2002, and the Ph. D degree in computer software and theory from the University of Science and Technology of China, China, in 2007. He is a professor with the School of Computer Science and Technology, University of Science and Technology of China, China. He was awarded the Outstanding Youth Science Foundation of NSFC, in 2018. He has won the best paper award or the best paper candidate in several famous conferences. He has published more than 100 papers in famous journals and conferences, including the IEEE/ACM Transactions on Networking, IEEE Transactions on Mobile Computing, IEEE Transactions on Parallel and Distributed Systems, Infocom and ICNP, etc. He has also held more than 30 patents. His main research interest is software defined networks, edge computing and Internet of Thing.

Jianchun Liu received B.S. degree in 2017 from the North China Electric Power University. He is currently a Ph.D. candidate in the School of Data Science, University of Science and Technology of China (USTC). His main research interests are software defined networks, network function virtualization, edge computing and federated learning.

Yang Xu (Member, IEEE) is currently an associate researcher in the School of Computer Science and Technology at University of Science and Technology of China. He got his Ph.D. degree in computer science and technology from University of Science and Technology of China in 2019. He got his B.S. degree in Wuhan University of Technology in 2014. His research interests include Ubiquitous Computing, Deep Learning and Mobile Edge Computing.

He Huang is currently a professor in the School of Computer Science and Technology at Soochow University, P. R. China. He received his Ph.D. degree in School of Computer Science and Technology from University of Science and Technology of China (USTC), in 2011. His current research interests include traffic measurement, computer networks, and algorithmic game theory. He is a Member of both IEEE and ACM.

Yangming Zhao received the BS degree in communication engineering and the PhD degree in communication and information system from the University of Electronic Science and Technology of China, in 2008 and 2015, respectively. He is a research scientist with SUNY Buffalo. His research interests include network optimization, data center networks, edge computing and transportation systems.