Article

Asynchronous Hierarchical Federated Learning Based on Bandwidth Allocation and Client Scheduling

Jian Yang [1], Yan Zhou [1], Wanli Wen [2,*], Jin Zhou [3,*] and Qingrui Zhang [4]

1 NUPT-JSTZ Carbon Peaking & Carbon Neutrality Application Technology R&D Institute, College of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210042, China; yangj@njupt.edu.cn (J.Y.); 1221045609@njupt.edu.cn (Y.Z.)
2 School of Microelectronics and Communication Engineering, Chongqing University, Chongqing 400044, China
3 School of Physics and Information Engineering, Jiangsu Second Normal University, Nanjing 210013, China
4 School of Philosophy, Religion and History of Science, University of Leeds, Leeds LS2 9JT, UK; yh.marjory@gmail.com
* Correspondence: wanli_wen@cqu.edu.cn (W.W.); zhou@jssnu.edu.cn (J.Z.)

Abstract: Federated learning (FL) offers a promising solution in edge computing to overcome bandwidth limitations and privacy concerns associated with traditional cloud-based training. However, current FL methods often suffer from transmission delay and excessive communication resource usage. In this paper, we introduce an innovative asynchronous hierarchical FL approach based on bandwidth allocation and client scheduling. Specifically, we propose an efficient algorithm that dynamically assigns clients to edge servers based on client mobility during training and accelerates parameter uploading while taking into account the remaining bandwidth of the edge servers. Our experimental results demonstrate the effectiveness of our approach, particularly in scenarios with frequent client mobility. This research strongly supports the application of FL in edge computing and underscores the crucial role of resource allocation in addressing communication resource constraints and reducing the training time of FL.

Keywords: federated learning; resource allocation; client selection

Citation: Yang, J.; Zhou, Y.; Wen, W.; Zhou, J.; Zhang, Q. Asynchronous Hierarchical Federated Learning Based on Bandwidth Allocation and Client Scheduling. *Appl. Sci.* **2023**, *13*, 11134. https://doi.org/10.3390/app132011134

Academic Editor: Christos Bouras

Received: 14 August 2023
Revised: 15 September 2023
Accepted: 4 October 2023
Published: 10 October 2023

1. Introduction

The advent of digital technology has fueled significant advancements in various transformative technologies such as big data and artificial intelligence. Machine learning (ML)-powered mobile applications are revolutionizing various aspects of modern life [1]. Nevertheless, ML training tasks typically demand a significant volume of data from diverse terminals with varying computational capabilities. Uploading data to remote cloud servers for processing, which is the conventional approach, presents challenges like privacy infringement, network congestion, and significant transmission delays. These challenges hinder the complete utilization of the abundant data [2]. In 2016, the Google Research Institute introduced the concept of federated learning (FL) with the aim of mitigating network bandwidth constraints and addressing vulnerabilities in data privacy [3]. FL is a collaborative training and sharing approach that eliminates the need for accessing raw data, aligning with decentralized collection and data minimization principles, where updates are shared with a central server or a coordinator. This decentralized approach adheres to principles of data minimization and eliminates the requirement for centralized data collection. Raw data remain securely stored on individual devices without direct access. FL facilitates local model training on each device while uploading only aggregated updates or model parameters to the central server.

Hierarchical FL (HFL) [4] is an FL framework where edge servers serve as intermediaries between mobile devices and the cloud. These servers facilitate the aggregation of local models received from neighboring devices at the edge. By enabling the cloud server to effectively process data from a larger number of terminal devices, HFL successfully addresses the challenges encountered during data uploading to the cloud. However, the common HFL framework employs synchronous aggregation of global models, wherein the server does not conduct cloud aggregation until it receives the model parameters of all clients. This synchronous approach leads to a phenomenon known as the straggler effect, wherein the delay in cloud aggregation is determined by the slowest client. The straggler effect adversely affects model training in terms of delayed convergence and greater resource consumption. Notably, a delay in uploading parameters by one or more clients result in an overall delay in the training process, prolonging the training time. Moreover, a delayed aggregation of client parameters impedes the convergence of the global model, potentially affecting the overall accuracy performance of the trained model. These negative effects underscore the significance of addressing the straggler effect in FL to ensure efficient and timely model updates.

2. Related Works

In order to address the issue of low communication efficiency with peripheral devices, Zhang et al. proposed an FL framework with cooperative mobile edge networking [5]. The structure involves the use of multiple edge cloud servers collaborating instead of a single cloud server, which allows devices located at the periphery to upload parameters to the cloud server at a lower cost, enabling it to cover a broader range of clients. The authors in [6] designed a compression control scheme to balance the energy consumption of local computing and wireless communication from the long-term learning perspective. Several approaches have been proposed to address the straggler effect in FL, including works in [7–9]. Xie et al. in [7] introduced the FedAsync algorithm, which enables clients to promptly upload their local models to the server upon completing an update. However, these uploaded models may become outdated by the time they reach the server. To mitigate the influence of obsolete models on the global model, some researchers attempted to incorporate the FedAsync algorithm with an obsolescence function as a hyperparameter. Nguyen et al. proposed the FedBuff algorithm [8], which involves the server establishing a buffer of a specific capacity and leveraging trusted execution environments to enhance privacy protection. In this approach, clients asynchronously upload their updated models to the buffer. Once the buffer holds a sufficient number of models, the server conducts a round of aggregation. The FedBuff algorithm has faster convergence compared to the FedAsync algorithm while also preserving privacy. Chen et al. introduced an aggregation method based on update awareness in the asynchronous FL framework [9]. This approach effectively addresses the negative impact of variations in local training speeds on aggregation parameters by appropriately controlling parameter proportions among clients. Furthermore, the aggregation parameters of slower clients are dynamically updated in this method.

While these algorithms can improve model training efficiency and reduce data traffic, several challenges still require further investigation. For example, optimization algorithms tend to exhibit longer processing times for non-IID (non-Independent and Identically Distributed) data compared to IID data, resulting in increased time costs [10]. Additionally, compression algorithms, capable of reducing the data volume of communication, may also have a significant impact on convergence performance, posing a challenge for achieving a balance between communication efficiency and convergence [11]. Consequently, there is an urgent need for effective algorithms that can overcome issues related to non-uniform computing capacity and discrete distribution in the context of a large number of clients.

In response to the challenges posed, we introduce an innovative asynchronous hierarchical FL approach and the main contributions of this work are summarized as follows.

- To mitigate the straggler effect, we introduce an asynchronous hierarchical FL approach which jointly optimizes bandwidth allocation and client scheduling.
- In order to realize load balance, we dynamically assign clients to edge servers taking into account client mobility during training. Then, we propose a client scheduling

algorithm which selects clients with high-quality data and low latency to achieve fast convergence and high model accuracy.

- We accelerate model parameter uploading by taking into account the remaining bandwidth of the edge servers.
- The simulation results validate the potential of the proposed asynchronous HFL approach in improving the overall performance and scalability in the context of a large number of clients.

3. System Model

As shown in Figure 1, we consider an HFL system, including one cloud server, edge servers, and clients. The cloud server is responsible for selecting participating clients for training and managing the global model parameters of the entire FL system. The edge servers receive updates of local model parameters from the clients, instead of data samples, thus reducing communication overhead. The clients receive the global model from the cloud server and update the global model based on their own data samples. Moreover, these clients are mobile and thus are associated with different edge servers at different time intervals. The sets of clients and edge servers are denoted using $\mathcal{I} = \{i : i = 1, 2, \ldots, I\}$ and $\mathcal{J} = \{j : j = 1, 2, \ldots, J\}$, respectively. The cloud server is denoted using C. Each client i has an independent local dataset $D_i = \{(x_m, y_m)\}_{m=1}^{|D_i|}$, where x_m represents the mth input sample, y_m represents the corresponding label, and $|D_i|$ is the size of the dataset on client i. In the following, we detail the training process and then formulate an optimization problem to improve the training performance.

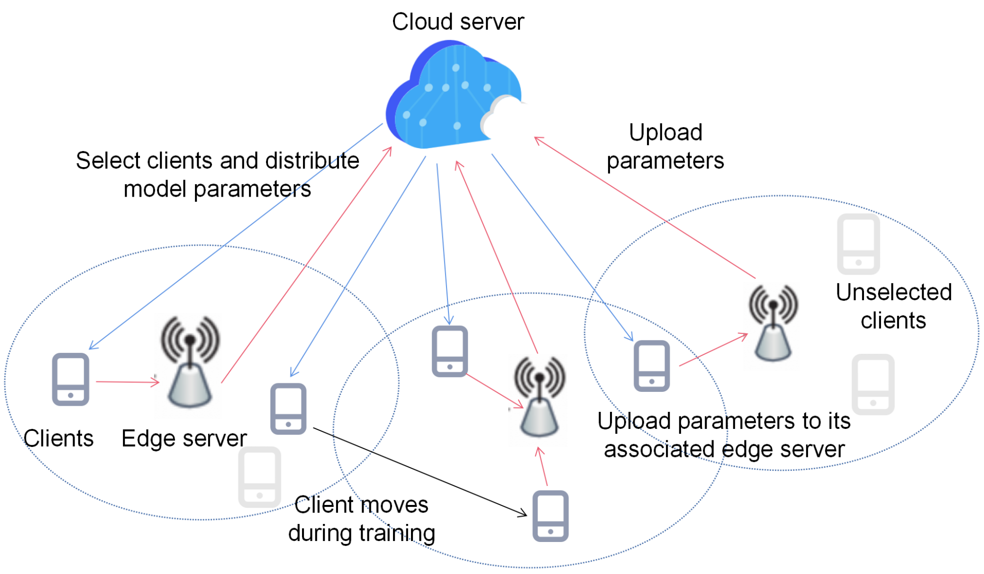
Cloud server
Select clients and distribute model parameters
Upload parameters
Unselected clients
Clients
Edge server
Upload parameters to its associated edge server
Client moves during training

Figure 1. A snapshot of the considered system model.

3.1. Training Process

Figure 1 depicts the training process, starting with the cloud server distributing the latest model parameters to the clients. Once it has received the global model parameters, each client performs local training to update model parameters based on its own dataset and then transmits the updated model parameters to its associated edge server, taking into account client location and available bandwidth. An example is given in Figure 2, where $ES1, ES2, \ldots$ represent edge servers and $c1, c2, \ldots$ represent the clients participating in this round of edge aggregation, $\boldsymbol{\omega}^{(t)}$ represents the model parameter in the cloud server in round t, and $\boldsymbol{\omega}_j^{(t)}$ represents the parameter in the edge server j in round t. Each edge server performs edge aggregation on the received model parameters after a predefined time interval of ΔT. Then, the edge servers upload edge model parameters to the cloud server and the cloud sever performs cloud aggregation during the time interval of $\Delta T_{aggregation}$.

Additionally, the edge server optimizes bandwidth allocation to improve the achievable transmission rate and reduce the uploading time of model parameters. Subsequently, the edge server uploads the aggregated model parameters to the cloud server for cloud aggregation. Notably, edge aggregation and model uploading operations can occur concurrently on the edge servers. Such a process is named a round and is repeated many times until convergence.

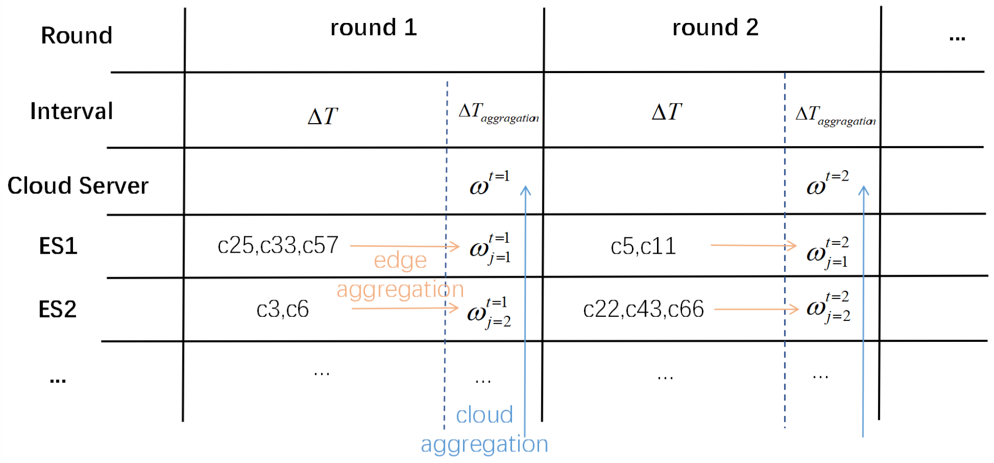

Figure 2. Model uploading and aggregation.

3.1.1. Client Selection and Model Distribution

During the training process, the clients are mobile and their movement is unpredictable, making it challenging for the cloud server to determine the participating clients. Also, the uncertainty in client association may lead to a failure of model parameter uploading, especially when the number of clients is huge but the available bandwidth is limited. To tackle this challenge, it is crucial to determine which clients should participate in the training process and how to associate the clients with different edge servers.

In this paper, we propose an approach that considers the available bandwidth of each edge server and traffic flow to determine the optimal number of clients. Our approach involves distributing the latest model parameters and enabling clients to locally train the model using their own datasets. The goal is to enhance the performance of the models shared among the clients. Given the powerful computation and communication capabilities of the cloud servers, the calculation time required for selected clients and the distribution time can be neglected.

3.1.2. Local Training on Clients

At the beginning of the local training process, the participating clients receive the latest model parameters from the cloud server. Then, each client i utilizes its local dataset D_i to minimize the loss function $f_i(x_m, y_m, \boldsymbol{\omega})$, which facilitates the representation of y_m with the optimal model parameters $\boldsymbol{\omega}$. The loss function of local training on client i is defined as follows [12]:

$$F_i(\boldsymbol{\omega}) = \frac{1}{|D_i|} \sum_{m \in D_i} f_i(x_m, y_m, \boldsymbol{\omega}), \tag{1}$$

which aims to find the optimal model parameters that minimize the loss function, i.e., $\boldsymbol{\omega}_i^* = \arg\min[F_i(\boldsymbol{\omega})]$.

The stochastic gradient descent method is adopted in the local training process. To achieve the desired level of local accuracy $\theta \in (0,1)$, the clients carry out several local iterations such as $L(\theta) = \mu \log(1/\theta)$ [13], where the value of the constant μ relies on the dimensions of the training task. Specifically, the update rule of the l-th local iteration is given as $\boldsymbol{\omega}_i^{(l)} = \boldsymbol{\omega}_i^{(l-1)} - \eta \nabla F_i\left(\boldsymbol{\omega}_i^{(l-1)}\right)$, where η represents the learning rate.

The time consumption of the local training is calculated as follows:

$$\tau_i^{\mathrm{comp}} = L(\theta)P_i\psi, \tag{2}$$

where $L(\theta)$ represents the number of local iterations, P_i represents the time consumed by processing a data sample on client i, and ψ represents the batch size.

3.1.3. Model Uploading

Upon reaching the predetermined accuracy, each participating client promptly sends the model parameters to a certain edge server with available bandwidth. Let β_{ij} denote the ratio of the bandwidth of edge server j allocated to client i. The achievable rate of uploading model parameters of client i can be expressed as follows [14]:

$$r_{ij} = \beta_{ij}B_j \log_2\left(1 + \frac{p^{(T)}|h_{ij}|^2}{N_0}\right), \tag{3}$$

where h_{ij} is the channel gain between edge server j and client i, N_0 is the noise power, and $p^{(T)}$ is the transmit power of the clients. Then, the uploading delay of model parameters can be expressed as $\tau_{ij}^{comm} = \frac{\Gamma}{r_{ij}}$, where Γ is the size of the model parameters to be uploaded.

3.1.4. Edge Model Aggregation

To address the straggler effect in the synchronous FL framework, our approach introduces asynchronous aggregation. In this method, each edge server independently performs edge model aggregation using the collected model parameters and then uploads the aggregated parameters to the cloud server. It is unnecessary to wait for the model parameters from all the participating clients; thus, asynchronous aggregation can significantly reduce waiting time and potential delays caused by slow clients.

However, a potential challenge lies in the possibility of clients uploading outdated parameters. To address the challenge, we introduce an obsolescence function as a hyperparameter to gradually reduce the weight of outdated parameters during the aggregation process. The obsolescence function of each client is determined by the current round of cloud aggregation, denoted as t, and the round of model uploaded from client i, denoted as t'. The function for the uploaded parameters of client i can be expressed as $H_i^{(t)} = \lambda^{t-t'}$, where $\lambda \in (0,1)$ is an attenuation coefficient, which makes the influence of outdated parameters diminish over time.

The model aggregation on edge server j can be expressed as

$$\boldsymbol{\omega}_j^{(t)} = \frac{\sum_{i \in S_j^{(t)}} H_i^{(t)}|D_i|\boldsymbol{\omega}_i^{(t)}}{\sum_{i \in S_j^{(t)}} H_i^{(t)}|D_i|}, \tag{4}$$

where $S_j^{(t)}$ is the set of clients associated with edge server j that participates in edge aggregation in round t.

3.1.5. Cloud Model Aggregation

Considering the powerful communication capabilities of both the edge server and the cloud server, the communication delay during parameter uploading from the former to the latter is negligible. Therefore, upon receiving the uploaded parameters, the cloud server promptly performs a round of cloud aggregation. Similarly, it is crucial to incorporate an obsolescence function to mitigate the impact of outdated model parameters on the global model. The obsolescence function is associated with the model parameters uploaded by each edge server; it is linked to the overall obsolescence of the clients who participated in the last round of aggregation and is defined as follows:

$$H_j^{(t)} = \frac{\sum_{i \in S_j} H_i^{(t)}}{|S_j|}. \tag{5}$$

Then, the update of model parameters on the cloud server is given as

$$\boldsymbol{\omega}_0^{(t)} = \frac{\left|D_S^{(t-1)}\right|\boldsymbol{\omega}^{(t-1)} + H_j^{(t)}\left|D_{S_j}^{(t)}\right|\boldsymbol{\omega}_j^{(t)}}{\left|D_S^{(t-1)}\right| + H_j^{(t)}\left|D_{S_j}^{(t)}\right|}, \tag{6}$$

where $D_{S_j^{(t)}}^{(t)} = \bigcup_{i \in S_j^{(t)}} D_i$ is the total dataset of the clients associated with edge server j in round t, and $D_{S_0}^{(t)} = \bigcup_{t'=1}^{t} D_{S_j^{(t')}}^{(t')}$. After the cloud model aggregation, the cloud server starts the next round of training via choosing a set of clients and distributing the latest global model parameters to them.

3.2. Problem Formulation

Due to the asynchronous nature of the aggregation process on edge and cloud servers, it is hard to find the optimal solution of the whole training process. Instead, we aim to achieve a balance between model performance and latency by optimizing the client scheduling and bandwidth allocation, taking into account the delay (including the delay of transmission and local training) and data quality.

After each round of cloud aggregation, the cloud server selects n training clients. These clients are then allocated to the nearest edge server j in order to optimize their training delay (τ_i) and data quality (Q_i). Accordingly, the optimization problem is formulated as

$$\begin{aligned} &\min_{\beta_{ij}, S_j} \sum_{i=1}^{n} \left(\alpha\tau_i + \frac{(1-\alpha)}{Q_i} \right), \\ \text{s.t.}\ \ & S_j \in \mathcal{I}, \\ & S_j \cap S_{j'} = \varnothing, \forall j, j' \in \mathcal{J}, j \neq j', \\ & \sum_{i \in S_j} \beta_{ij} \leq 1, \end{aligned} \tag{7}$$

where α is the weight coefficient and $\tau_i = \tau_i^{\text{comp}} + \tau_{ij}^{comm}$. Note that one client can associate only with one edge server during the training process.

However, solving this optimization problem is challenging due to the vast combinatorial search space of the association decisions and the interdependence with resource allocation. Instead of finding the optimal solution, we aim to find suboptimal solutions by employing a divide-and-conquer approach to split the critical problem into two subproblems, i.e., reducing client training delay and enhancing client data quality. Our proposed approach includes the following steps:

(a) We develop a load balance algorithm that calculates the number of clients associated with each edge server based on the remaining available bandwidth and the state transition probability of client mobility.
(b) Then, we propose a client scheduling algorithm which selects the clients with low latency and high data quality.
(c) Furthermore, we also develop an algorithm that accelerates model parameter uploading by making full use of the available bandwidth of the edge servers.

3.3. Load Balance

Generally, it is hard to predict the movement of each client. This work assumes that the state transition probability of each client is available. Specifically, different edge servers have different coverage areas. The probability that a client staying in the coverage area of edge server ES_j moves to the coverage area of edge server $ES_{j'}$ is known a priori. Thus, we model the movement of clients as a Markov process and an example of the state transition probability matrix is given in (8) with $J = 3$, where the elements in row j and column j' represent the probability that a client moves from the coverage area of edge server j to the coverage area of edge server j' during the training process.

$$\mathbf{M} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix}. \tag{8}$$

Each client participating in the training process has a specific initial bandwidth allocation ratio β_0. First, we define a vector $\mathbf{n} = [n_1, n_2, n_3, \ldots, n_J]^T$ where the j-th element represents the number of clients that are associated with edge server j in the next round. In addition, we also define a vector $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_J]^T$ where the j-th element denotes the number of clients initially allocated by the cloud server to the j-th edge server. Then, we formulate the following optimization problems, which determine the number of clients that are associated with each edge server, i.e.,

$$\begin{aligned} &\min_{\mathbf{x}} \sum_{i=1}^{J} (y_j - n_j)^2, \\ &\text{s.t. } \sum_{j=1}^{J} x_j - \sum_{j=1}^{J} n_j = 0, \\ &\quad x_j \geq 0, j \in \mathcal{J}, \\ &\quad x_j^{\text{remain}} - x_j \geq 0, j \in \mathcal{J}, \end{aligned} \tag{9}$$

where y_j is the j-th element of $\mathbf{y} = \mathbf{x}^T\mathbf{M}$. We rewrite the first constraint as $c_1(x) = 0$, the following J constraints as $c_j = x_j \geq 0, j = 2, 3, \ldots, J+1$, and the last J constraint as $c_j = x_j^{\text{remain}} - x_j \geq 0, j = J+2, J+3, \ldots, 2J+1$, respectively.

The constraint problem (9) can be solved with the projected hypergradient descent method [15], for which the main steps are given as follows:

Step 1: We first initialize the penalty factor θ_1, error control factor $\varepsilon > 0$, constant $\theta \in (0, 1)$, and $k = 1$.

Step 2: For the initial point $\boldsymbol{X}_{k-1}$, we solve the following unconstrained problem:

$$\begin{aligned} M(x, \lambda, \sigma) = f(x) + \frac{1}{2\sigma} \sum_{i=2}^{2J+1} \left\{ [\max(0, \lambda_i - \sigma c_i(x))]^2 - \lambda_i^2 \right\} \\ -\lambda_1 c_1(x) + \frac{\sigma}{2} c_1(x), \end{aligned} \tag{10}$$

then we obtain the optimal solution $\boldsymbol{X}_k$.

Step 3: If $\varphi_k = \left\{ c_1^2(x) + \sum_{i=2}^{2J+1} \left[\min\left(c_i(x_k), \frac{(\lambda_k)_i}{\sigma} \right) \right] \right\}^{\frac{1}{2}} < \varepsilon$, then $\boldsymbol{X}_k$ is the optimal solution; stop. Otherwise, go to step 3.

Step 4: If $\varphi_k / \varphi_{k-1} \leq \theta$, go to step 4. Otherwise, let $\sigma_{k+1} = c\sigma_k$, and go to step 4.

Step 5: Update the multiplier vector according to

$$(\lambda_{k+1})_1 = (\lambda_k)_1 - \sigma c_1(x_k), \tag{11}$$

and

$$(\lambda_{k+1})_i = \max[0, (\lambda_k)_i - \sigma c_i(x_k)], i = 2, 3, \ldots, 2J+1. \tag{12}$$

Let $k = k+1$, then go to step 2.

Here, the first constraint is an equality constraint and the other constraints are inequality constraints. The output $\boldsymbol{X}$ may not an integer and the rounding method can be adopted. Then, the obtained integer solution is the number of clients that are associated with each edge server. The proposed client selection method reduces the insufficient bandwidth allocation, especially when the number clients is large but the available bandwidth is scarce.

3.4. Client Scheduling

During the phase of client scheduling, the computing capability and the quality of the local data are considered simultaneously. In this work, data quality is measured using the entropy weight method. Specifically, assume that each client i has K_i attributes. We randomly select M samples from the local dataset $\mathcal{D}_i$ of each client i and calculate

$$Q_i = \frac{1}{M}\sum_{k=1}^{K_i}\sum_{m=1}^{M} w_k^{(i)} y_{km}^{(i)} \tag{13}$$

where $w_k^{(i)} = \frac{1-e_k^{(i)}}{\sum_{k=1}^{K_i} 1-e_k^{(i)}}$ is the entropy weight of the k-th data attribute of client i, y_{km}^{i} is the value after data attribute standardization, and the information entropy of each data attribute [16] can be expressed as follows:

$$e_k^{(i)} = -\frac{1}{\ln m}\sum_{n=1}^{m} \frac{y_{kl}^{(i)}}{\sum_{n=1}^{m} y_{kl}^{(i)}} \ln\left(\frac{y_{kl}^{(i)}}{\sum_{n=1}^{m} y_{kl}^{(i)}}\right). \tag{14}$$

The client scheduling metric is based on its computing capability P_i and the indicator of local data quality Q_i. It is assumed that the P_i and Q_i of each client are stable during the training process and the clients with a higher value of $\phi_i = \gamma P_i + (1-\gamma)Q_i$ have priority to be selected to participate in the FL training. The steps of selecting x_j clients from the range of edge server j are as follows:

Step 1: The value of $|N_{selected}|$ updates. If $|N_{selected}| > k$, select x_j^{best} clients from $N_{selected}$ within the coverage of edge server j and with the highest value of ϕ.

Step 2: If $x_j^{best} < x_j$, $x_j - x_j^{best}$ clients are randomly selected from $I - N_{selected}$ that are within the range of the edge server j. The value of ϕ of these clients is calculated, and the clients are then added to $N_{selected}$.

Step 3: Finally, all clients in $N_{selected}$ participate in the next round of training, and the cloud server distributes the latest model parameters to these clients.

The time complexity of this algorithm for calculating the entropy weight of each client is related to the product of the number of data samples chosen and also the number of clients selected, which is $\mathcal{O}(M|N_{selected}|)$.

3.5. Uploading Acceleration

To accelerate the uploading process of model parameters, the available bandwidth is allocated based on the distance between the clients and the edge server. When n clients are uploading simultaneously, the allocation ratio for client i is adjusted to ensure that the transmission time for all clients is nearly equal, i.e., the allocation ratio for client i is given by

$$\min\{\frac{s_{ij}^2 \beta_{remain}^{(j)}}{\sum_{i=1}^{n} s_{ij}^2}, \beta_{\max}\}. \tag{15}$$

If there are no other clients uploading at the same time, then the $\min\{\beta_{remain}^{(j)}, \beta_{\max}\}$ of the bandwidth is allocated to client i, where $\beta_{\max}$ is the constant, which limits the maximum ratio of bandwidth allocated by an edge server to a client.

4. Simulation Results

In this section, we will show the simulation results of the asynchronous HFL approach based on bandwidth allocation and client scheduling. To illustrate the significant advantages of this approach, some baseline algorithms are introduced.

4.1. Simulation Settings

We consider an FL system consisting of a cloud server in the center and five edge servers randomly distributed within the coverage of the cloud server. A total of 250 clients are randomly distributed within the coverage area of the edge servers, but the number of clients under each edge server may vary.

A training task of handwritten digit recognition based on the MNIST dataset is considered. The LeNet is a classic convolutional neural network for handwritten digit recognition and was proposed by LeCun et al. in 1998 [17]. For simplicity, we still use the MNIST dataset and the LeNet as the basis of our simulations. This network comprises an input layer for 32×32 grayscale images, followed by two convolutional layers (C1 and C3) using 5×5 kernels and Sigmoid activation. Max-pooling layers (S2 and S4) reduce dimensions, while fully connected layers (C5 and F6) introduce intermediate neurons. The output layer, with Softmax activation, has ten neurons for digit classification.

In our study, we randomly select 1000 labeled data samples from the MNIST dataset of 60,000 data samples and form a test dataset on the cloud server. In addition, 5000 labeled data samples are randomly chosen and evenly allocated among the five edge servers to evaluate the model accuracy. The remaining 54,000 data samples are equally distributed among the 250 clients in an independent and identically distributed manner.

We initialize the learning rate as 0.01. After completing each training round, we reduce the learning rate by 0.995 times its previous value. This decay strategy plays a crucial role in gradually adjusting the learning rate over time, leading to improved model convergence. Each client performs forty iterations of local updates. Consequently, every client performs forty iterations of gradient descent using its respective local dataset to update the model parameters. This iterative approach enables the model to learn from the data and continually enhance its performance across multiple training rounds.

The channel is modeled as Rayleigh fading. The small-scale fading coefficient follows complex Gaussian distribution with zero mean and unit variance and the large-scale fading mainly depends on the path-loss model given by $PL[dB] = 128.1 + 37.6log_{10}(d)$, where d is the distance. The transmit powers of each client, each edge server, and the cloud server are set to 20 dBm, 30 dBm, and 40 dBm, respectively. The total available bandwidth is 1 MHz and each sub-channel is of 15KHz.

When simulating, we defined the following classes in our program:

- Cloud Server Class: It implements functions including load balance among edge servers, client scheduling, model distribution, and cloud aggregation.
- Edge Server Class: It implements functions including model collection from clients and edge aggregation.
- Client Class: It implements functions including local updates, model uploading, and client mobility.

Then, the following steps are executed:

- The cloud server performs cloud aggregation and determines the client scheduling and bandwidth allocation scheme in the next round of model training.
- The selected clients perform local updates and then upload the updated model parameters to their associated edge servers, respectively. The uploading delay is recorded. In addition, we use a binomial distribution to simulate model uploading failure and success events.
- Each edge server executes edge aggregation of model parameters from clients and then uploads the aggregated model to the cloud server.
- Such a process is repeated until convergence.

4.2. Introduction to Baseline Algorithms and Proposed Approach

(a) Baseline 1 (hierarchical FL in [4]): In this baseline, after each round of cloud aggregation and before local updates, five edge servers randomly select 10 clients in their coverage to participate in the training process. The client selection takes into account the possibility of client relocation during the training. Once the clients complete their local updates, the edge servers receive the uploaded model parameters if the clients remain in their coverage. The edge servers proceed to collect the parameters from all the clients. If the waiting time exceeds a threshold, the edge server performs a round of edge aggregation and uploads the aggregated model parameters to the cloud server. Upon receiving the parameters from the edge servers, the cloud server performs a round of cloud aggregation.

(b) Baseline 2 (greedy optimization): In this baseline, 10 clients are randomly selected from the clients in the coverage of each edge server for training. The selection process takes into account the potential client relocations during the training process. Once the local training is completed, each selected client is connected with its closest edge server and uploads the model parameters. After a specific time period, each edge server performs edge aggregation on the uploaded model parameters. The parameters that are not uploaded on time are buffered and used in the next edge aggregation based on the uploading delay, which is adjusted according to the outdated function. Finally, the cloud server receives the model parameters from all the edge servers simultaneously, along with the information about the number of participating clients in the cloud aggregation.

(c) Proposed asynchronous hierarchical FL: In our approach, after each round of cloud aggregation and before local updates, we first determine the number of clients that are associated with each edge server. Then, the client scheduling and bandwidth allocation are optimized simultaneously based on training delay and data quality. In addition, client mobility is also considered and is modeled as a Markov process. After finishing model uploading, the edge server releases the idle bandwidth resource. If a client fails to upload its model parameters to its edge server within a given time interval, the edge server is forced to release the corresponding bandwidth. Each edge server performs edge aggregation on the uploaded model parameters in a predetermined interval and uploads them to the cloud server. The cloud server receives all the model parameters uploaded by the edge servers, along with information about the number of clients participating in the cloud aggregation.

4.3. Results

Figure 3 shows that the proposed approach achieves the best performance, compared to baselines 1 and 2, in terms of model convergence. Specifically, to achieve a certain accuracy (such as 0.9), the proposed asynchronous HFL approach based on bandwidth allocation and client scheduling takes only 33 time units, whereas baseline 1 takes 56 time units and baseline 2 takes 113 time units. Furthermore, the proposed approach can achieve an accuracy of 0.95, which serves as the upper bound of baselines 1 and 2. It is found that the proposed approach can reduce the training time by 70.8% compared to baseline 1, and by 41.1% compared to baseline 2. Moreover, the proposed approach exhibits more stable convergence performance compared to baseline 1. This is because the proposed approach takes into account both the training delay and data quality of the clients.

Figure 4 presents a similar conclusion to that drawn from Figure 3. Specifically, the proposed approach has a smaller test loss and converges faster, compared to baselines 1 and 2. This is because the proposed approach tends to select the clients that behave well, i.e., the clients with lower delay and higher data quality.

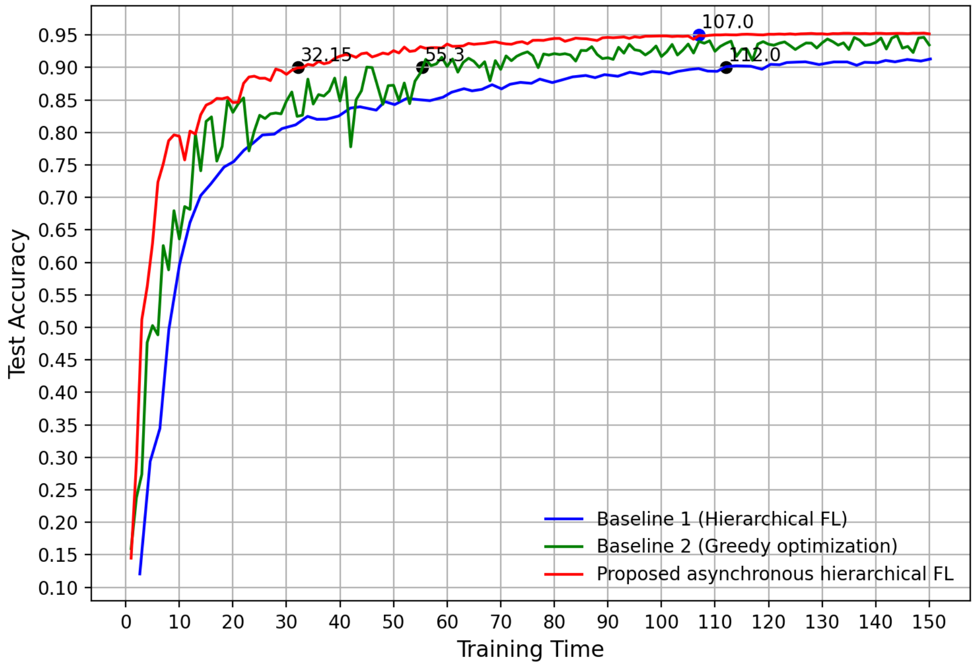

Figure 3. Test accuracy over training time.

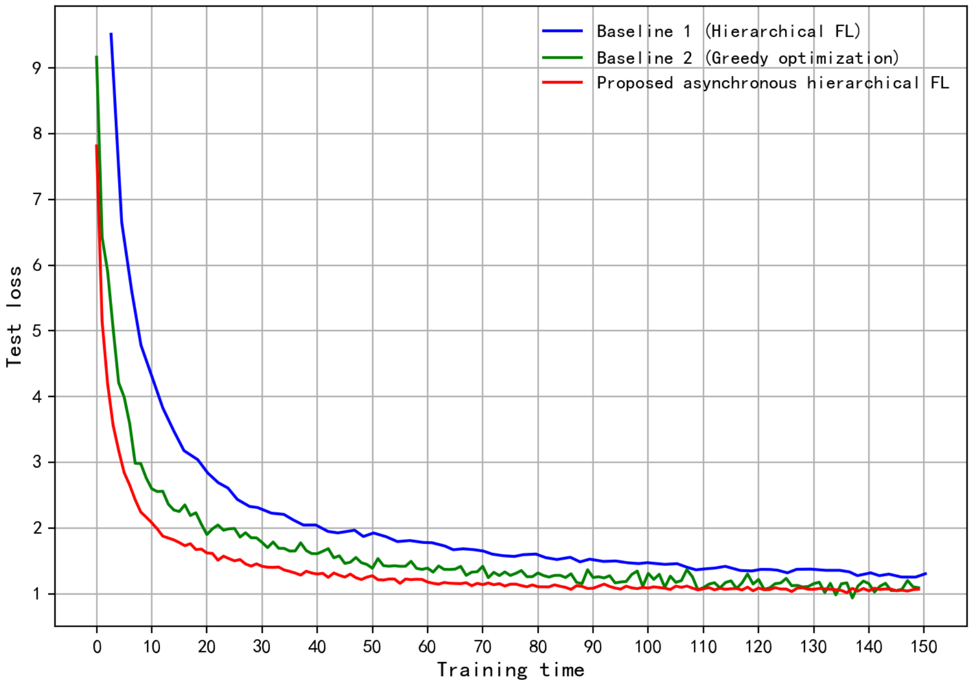

Figure 4. Test loss over training time.

During the training process, the clients' movements may prevent the clients from uploading model parameters to the corresponding edge server in time. This results in some clients not participating in the edge model aggregation, leading to unstable and poor accuracy performance. The greedy optimization algorithm for uploading model parameters can significantly reduce the uploading delay in an HFL framework. As shown in Figure 5, the horizontal axis represents the normalized time required for the clients to upload their models in existing hierarchical FL methods, and the vertical axis represents the time needed in greedy optimization. However, the greedy algorithm may result in initially allocating clients to a group in one particular area post training. This would cause some edge servers to receive additional clients, potentially leading to uploading congestion. If the bandwidth is not well allocated, the clients will be blocked in the uploading stage, resulting in slow model convergence. However, our proposed algorithm achieves load balancing based on bandwidth allocation and client scheduling, which prevents the occurrence of the above situation.

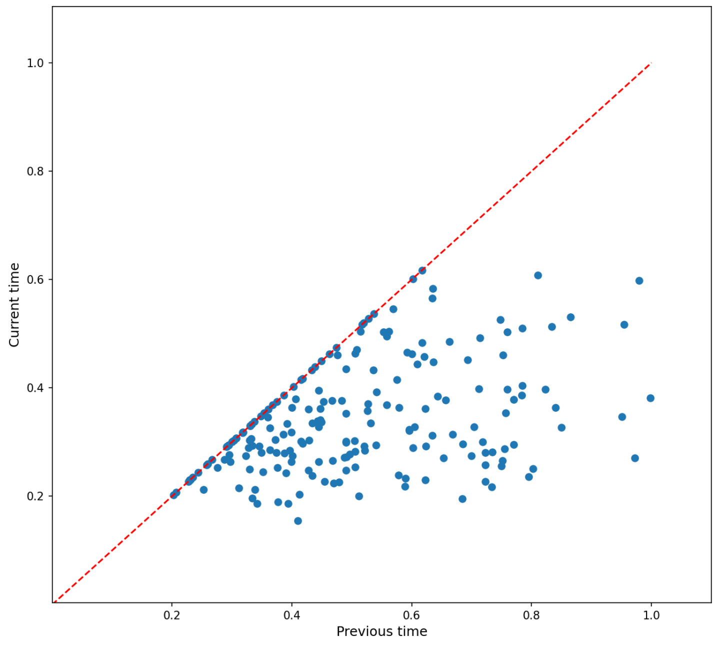

Figure 5. Previous time–current time via greedy optimization algorithm.

5. Conclusions

This work introduced an asynchronous HFL framework with bandwidth allocation and client scheduling that aims to enhance resource utilization while guaranteeing model training performance. We proposed an efficient algorithm to perform bandwidth allocation based on client mobility patterns and also load balance on edge servers. Edge servers further accelerated model uploading using the remaining bandwidth. Ultimately, compared to the existing hierarchical FL methods, our approach can reduce training time by 78.2% and achieve higher model accuracy, which represent promising potential for efficient and robust FL in dynamic environments with client mobility. Our future research endeavors will focus on safeguarding privacy in the considered asynchronous FL by exploring the integration of privacy protection techniques such as differential privacy and homomorphic encryption.

Author Contributions: Conceptualization, J.Y.; methodology, J.Y. and Y.Z.; validation, Y.Z.; analysis, J.Y. and Y.Z.; investigation, J.Y.; resources, J.Y. and J.Z.; writing—original draft preparation, Y.Z. and W.W.; writing—review and editing, W.W., J.Z. and Q.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was sponsored by the National Natural Science Foundation of China under Grant 62201101, the Project funded by China Postdoctoral Science Foundation under Grant 2022M720020 and Grant 2022M722669, the Natural Science Foundation of Chongqing, China, under Grant cstc2021jcyj-msxmX0458, the Chongqing Technology Innovation and Application Development Special Key Project under Grant CSTB2022TIAD-KPX0059, and the open research fund of the National Mobile Communications Research Laboratory, Southeast University, under Grant 2022D06.

Data Availability Statement: Data available within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
2. Li, P.; Li, J.; Huang, Z.; Li, T.; Gao, C.Z.; Yiu, S.M.; Chen, K. Multi-key privacy-preserving deep learning in cloud computing. *Future Gener. Comput. Syst.* **2017**, *74*, 76–85. [CrossRef]
3. McMahan, H.B.; Moore, E.; Ramage, D.; y Arcas, B.A. Federated learning of deep networks using model averaging. *arXiv* **2016**, arXiv:1602.05629.
4. Liu, L.; Zhang, J.; Song, S.; Letaief, K.B. Client-edge-cloud hierarchical federated learning. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 July 2020; pp. 1–6.
5. Zhang, Z.; Gao, Z.; Guo, Y.; Gong, Y. Scalable and Low-Latency Federated Learning with Cooperative Mobile Edge Networking. *IEEE Trans. Mob. Comput.* **2022**, 1–11. [CrossRef]
6. Chen, R.; Li, L.; Xue, K.; Zhang, C.; Pan, M.; Fang, Y. Energy efficient federated learning over heterogeneous mobile devices via joint design of weight quantization and wireless transmission. *IEEE Trans. Mob. Comput.* **2022**, 1–13. [CrossRef]
7. Xie, C.; Koyejo, S.; Gupta, I. Asynchronous federated optimization. *arXiv* **2019**, arXiv:1903.03934.
8. Nguyen, J.; Malik, K.; Zhan, H.; Yousefpour, A.; Rabbat, M.; Malek, M.; Huba, D. Federated learning with buffered asynchronous aggregation. In Proceedings of the International Conference on Artificial Intelligence and Statistics, Virtual, 27–30 March 2022; pp. 3581–3607.
9. Chen, R.; Xie, Z.; Zhu, X.; Qu, Z. An Asynchronous Federated Learning Aggregation Update Algorithm. *Small Micro Comput. Syst.* **2022**, *42*, 2473–2478.
10. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In Proceedings of the Artificial Intelligence and Statistics, Lauderdale, FL, USA, 20–22 April 2017; pp. 1273–1282.
11. Caldas, S.; Konečny, J.; McMahan, H.B.; Talwalkar, A. Expanding the reach of federated learning by reducing client resource requirements. *arXiv* **2018**, arXiv:1812.07210.
12. Bottou, L.; Curtis, F.E.; Nocedal, J. Optimization methods for large-scale machine learning. *SIAM Rev.* **2018**, *60*, 223–311. [CrossRef]
13. Konečnỳ, J.; Qu, Z.; Richtárik, P. Semi-stochastic coordinate descent. *Optim. Methods Softw.* **2017**, *32*, 993–1005. [CrossRef]
14. Vu, T.T.; Ngo, D.T.; Tran, N.H.; Ngo, H.Q.; Dao, M.N.; Middleton, R.H. Cell-free massive MIMO for wireless federated learning. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 6377–6392. [CrossRef]
15. Rockafellar, R.T. The multiplier method of Hestenes and Powell applied to convex programming. *J. Optim. Theory Appl.* **1973**, *12*, 555–562. [CrossRef]
16. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
17. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]