Article

STAFL: Staleness-Tolerant Asynchronous Federated Learning on Non-iid Dataset

Feng Zhu ^{1,2}, Jiangshan Hao ^{1,*} , Zhong Chen ^{1,2}, Yanchao Zhao ^{1,*} , Bing Chen ¹ and Xiaoyang Tan ¹

¹ Collage of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; smc@nuaa.edu.cn (F.Z.); 736531683@nuaa.edu.cn (Z.C.); cb_china@nuaa.edu.cn (B.C.); x.tan@nuaa.edu.cn (X.T.)

² Nanjing Research Institute of Electronics Engineering, Nanjing 210007, China

* Correspondence: jiangshanhao@nuaa.edu.cn (J.H.); yczhao@nuaa.edu.cn (Y.Z.)

Abstract: With the development of the Internet of Things, edge computing applications are paying more and more attention to privacy and real-time. Federated learning, a promising machine learning method that can protect user privacy, has begun to be widely studied. However, traditional synchronous federated learning methods are easily affected by stragglers, and non-independent and identically distributed data sets will also reduce the convergence speed. In this paper, we propose an asynchronous federated learning method, STAFL, where users can upload their updates at any time and the server will immediately aggregate the updates and return the latest global model. Secondly, STAFL will judge the user's data distribution according to the user's update and dynamically change the aggregation parameters according to the user's network weight and staleness to minimize the impact of non-independent and identically distributed data sets on asynchronous updates. The experimental results show that our method performs better on non-independent and identically distributed data sets than existing methods.

Keywords: federated learning; edge computing; weight divergence; non-iid

Citation: Zhu, F.; Hao, J.; Chen, Z.; Zhao, Y.; Chen, B.; Tan, X. STAFL: Staleness-Tolerant Asynchronous Federated Learning on Non-iid Dataset. *Electronics* **2022**, *11*, 314. <https://doi.org/10.3390/electronics11030314>

Academic Editor: Claus Pahl

Received: 22 December 2021

Accepted: 17 January 2022

Published: 20 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Mobile phones, wearable devices, and autonomous vehicles are just a few of the modern distributed networks that are generating a wealth of data each day. Due to the growing computational power of these devices, coupled with concerns over transmitting private information, it is increasingly attractive to store data locally and push network computation to the edge.

The concept of edge computing is not a new one. Indeed, computing simple queries across distributed, low-powered devices is a decades-old area of research that has been explored under the purview of query processing in sensor networks, computing at the edge, and fog computing [1]. Recent works have also considered training machine learning models centrally but serving and storing them locally; for example, this is a common approach in mobile user modeling and personalization [2].

As the storage and computational capabilities of the devices within edge computing grow, it is possible to leverage enhanced local resources on each device. However, privacy concerns over transmitting raw data require user-generated data to remain on local devices. This has led to a growing interest in federated learning, which explores training statistical models directly on remote devices. In federated learning, data will not appear in other places except the data source, and each edge device cooperates to train a shared global model.

The existing research on federated learning is primarily about synchronous communication. Researchers use the method of client selection to reduce the negative impact of stragglers on the global model in federated learning. However, synchronous communication will waste computing resources, since different devices have different computing

capabilities. In addition, the server can not efficiently use the data in all s due to only a small number of s participating in the model training of each global round. Compared with synchronous communication, asynchronous federated learning will increase the burden on the server and consume a large amount of communication resources; however, it can significantly improve training efficiency. Training efficiency is crucial in Internet of Things tasks since they increasingly focus on real-time. Most of the existing asynchronous algorithms are semi-asynchronous strategies. These methods allow s to upload updates independently; however, they still need to synchronize their information. Another challenge of federated learning is the heterogeneity of data. Since federated learning does not allow data to be transmitted in the communication link, data heterogeneity cannot simply resolve by traditional methods such as scheduling data. The convergence speed of the asynchronous federated learning mechanism will be severely affected if each user has the same aggregation parameter. Moreover, as a type of distributed machine learning, more than 90% of the information exchange in federated learning is redundant. Compressing updates sent from the can reduce the consumption of communication resources in asynchronous federated learning and reduce the overall convergence time of federated learning.

In this paper, we design a staleness-tolerate asynchronous federated learning method, STAFL. STAFL can asynchronously receive and aggregate updates from the and reduce the impact of stragglers on the global model by adding penalty parameters. Secondly, STAFL will adjust the aggregation parameters according to the heterogeneity of the user's data and dynamically change the aggregation factor for each epoch by maintaining a local model parameter list. Finally, this paper uses the method of bit quantization. We compress the updates uploaded by the user and the global information sent by the server. Specifically, the contributions of this paper are as follows:

- We have designed a federated learning architecture for asynchronous communication, STAFL. After the user completes the local iteration, it can update the local model information at any time. The server immediately aggregates the update and delivers the latest global model to the user when receiving the update information, thereby reducing the waste of computing resources;
- We use the weight divergence of the local model to group users and maintain a list of users' updated information on the server-side. The longer the list, the more users are considered when aggregating. Different aggregation weights are assigned based on the arrival time of the user update, the amount of data of the user, and the group to which the user belongs to reduce the negative impact of non-independent and identically distributed data sets on asynchronous aggregation;
- We conducted many experiments to prove the effectiveness of the proposed method and used the communication compression method to reduce the communication cost of asynchronous federated learning. The experimental results show that STAFL has a significant advantage in convergence speed compared with other methods.

The organizational structure of our paper is as follows: in Section 1, we introduce the unique challenges and difficulties of asynchronous federated learning, and propose corresponding solutions. Section 2 describes the related work of this paper. We will introduce the design ideas and a detailed description of the STAFL system in Section 3. In order to verify the effectiveness of the proposed method, we show the performance of STAFL in different scenarios in Section 4. Finally, Section 5 summarizes the main work of this paper.

2. Related Work

In recent years, federated learning has received widespread attention as an efficient cooperative machine learning method. Federated learning can address the concerns of a response time requirement, battery life constraint, bandwidth cost-saving, and data safety and privacy [3–6]. Guo et al. [7] explore the security issues of federated learning and how to design efficient, fast, and verifiable aggregation methods in federated learning. As IoT applications have higher and higher requirements for real-time performance, asynchronous

federated learning has begun to receive attention as a method that can speed up convergence and reduce wasted computing resources. Although asynchronous federated learning can improve real-time performance, some challenges still need to be resolved [8,9]. In [10], Lu et al. designed an asynchronous federated learning scheme for resource sharing on the Internet of Vehicles and solve many challenges in the Internet of Vehicles scenario with very ingenious methods. In [11], the author proposes an age-aware communication strategy that realizes federated learning through wireless networks by jointly considering the parameters on user devices and the staleness of heterogeneous functions. However, these schemes pay more attention to security and privacy in federated learning, sacrifice part of the training efficiency, and ignore the negative impact of non-IID datasets on the shared model. Damaskinos et al. [12] designed an online system that can be used as middleware on Android OS and machine learning applications to solve the staleness problem. Wu et al. proposed an asynchronous aggregated federated learning architecture [13]. The author designed a bypass to store user updates arriving later in the global epoch and used the user selection method to improve training efficiency. In [14], Chai et al. proposed the use of a tier to distinguish users arriving at different times in a global epoch, aggregate each tier, and analyze convergence. However, these methods are semi-asynchronous communication and do not consider completely asynchronous scenarios, therefore staleness will still affect the system.

Statistical heterogeneity is also one of the problems that need to be solved in federated learning and attracts the attention of researchers [15,16]. The baseline FL algorithm FedAvg [17] is known to suffer from instability and convergence issues in heterogeneous settings related to device variability or non-identically distributed data [18]. Since McMahan et al. [17] proposed a benchmark synchronization federated learning data aggregation method, many researchers have explored communication methods, communication architectures, and user selection methods to make machine learning algorithms more efficient. Zhao et al. [19] proposed a method to ensure accuracy when the data are not independent and identically distributed. Sahu et al. [20] improved the algorithm of FedAvg. Chen et al. proposed a federated learning communication framework and joint learning method based on wireless networks [21]. Regarding the combination of non-independent identical distribution and asynchronous communication, Chen et al. [22] proposed a novel asynchronous federated learning algorithm and conducted a theoretical analysis of non-iid scenarios and asynchronous problems.

3. System Design

According to what we proposed in the previous section, this paper improves the existing work as follows. We believe that our method is better than existing methods in federated learning based on asynchronous communication. The symbols used in this paper are shown in Table 1. We assume that, in federated learning, users can upload updates independently and are not restricted by server selection. The central server has sufficient resources to encourage the user to participate in federated learning. In addition, the server has sufficient computing capacity to perform asynchronous aggregation operations without consuming a large amount of time. We consider that there are n users participating in the federated learning. The training data in each user's device are subject to non-independent and identical distribution. That is, each user will have missing labels. The missing data will pull the weight of local updates in the wrong direction, and users need to cooperate in offsetting the negative impact of missing labels. Each local dataset will not be leaked to any other users during the entire training process.

Table 1. Notation and Parameters.

Notation/Term	Description
$\mathcal{D}$	Dataset
T_g	Global epoch
α	staleness hyper-parameter
a	staleness penalty parameter
$S(t)$	staleness penalty function
$E(m)$	data distribution payoff
η	learning rate
$\mathcal{M} = [1, 2, \dots, m, \dots]$	set of users
$\mathcal{L}$	model list
n/β	length of $\mathcal{L}$
n_c	total data of group c
$\mathbf{w}_m$	model weight of user m

3.1. Staleness Tolerant Model

Compared with synchronous communication, asynchronous federated learning requires additional costs to reduce the staleness caused by straggles. Since there is no clear concept of the global epoch, the server needs to perform global aggregation every time it receives an update uploaded by the user. We assume that α is a hyper-parameter that controls the weight of each new user update. Therefore, for each user's update on the server-side, there are the following aggregation methods:

$$\mathbf{w}_{new} = \begin{cases} \mathbf{w}_m, & \text{if } l = 0; \\ \frac{1}{al+b} \mathbf{w}_m + (1 - \frac{1}{al+b}) \mathbf{w}_{old}, & \text{if } 0 < l \leq n; \\ \alpha \mathbf{w}_m + (1 - \alpha) \mathbf{w}_{old}, & \text{otherwise.} \end{cases} \quad (1)$$

where $\mathbf{w}_{old}$ is the global weight information of the last aggregation; l is the total number of times the server aggregates user updates. Whenever the server aggregates a user update, the value of l is increased by one. The time at l denotes as t_l , and $\mathbf{w}_m$ is the update information of user i at time t_l . As shown in Figure 1, if the user i 's update is the first update that arrives at the server (w_1^1), the server will send global weight to both users at the same time (t_1) after the next user's update arrives, because aggregation for only one user is a useless operation. After each update arrives at the server, the server will aggregate this information except at time t_0 .

The term $\frac{1}{al+b}$ in Equation (1) is the gradually decreasing aggregation parameter for newly arrived users' update. When $l < n$, the previous round of global aggregation $\mathbf{w}_{old}$ does not represent the global data distribution. We call $\mathbf{w}_{old}$ at this time the *prototype* global weight. As *prototype* aggregates more and more user updates, the difference between it and centralized updates will become smaller, and it will be significantly better than the update of a certain user. Therefore, we designed a linearly decreasing weight for aggregation. As the number of user updates aggregated by *prototype* increases, the aggregate weight of new arrival user updates will be smaller. In this paper, we set $a = \frac{2}{n-2}$ and $b = \frac{2n-8}{n-2}$. That is, the weight will slowly drop from the original $1/2$ to $1/4$. For *prototype*, we do not set any penalty terms because the server needs to aggregate more updates at this time.

Figure 1. The flow of STAFL between the cloud server and user device.

When the server aggregates enough updates ($l > n$), we think that the global model can represent all users' data and will impose corresponding penalties for stragglers at this time (reflected in α in Equation (1)). Before introducing α , we first introduce the concept of the global epoch of our method. There is a local update counter at the user-side, and the counter's value will increase by one every time the user uploads an update. When the server receives the updates of the next round from more than 10% of users (i.e., when w_1^2 in Figure 1 arrives at the server), the server considers that it has entered the next global epoch. At this time, if the server receives the previous round of user updates, it will impose corresponding staleness penalties. As shown in Figure 1, when the first update information of client 4 arrives at the server, the server has already entered the second global epoch, and the server will impose staleness penalties on client 4's update. The staleness penalties term $S(t)$ will make the aggregation term α smaller. However, α will not only be affected by $S(t)$. The difference in user distribution will also affect the size of α . This will be explained in detail in the following subsection. The server considers that the user's update information has been lost when it still does not receive a user's update after two global epochs and will send the latest global parameters to this user. Second, the server will not aggregate updates two rounds away from the global epoch because this information will seriously slow down the convergence speed of the global model.

3.2. Weight Divergence Discussion

Since data cannot be transmitted in the federated learning scenario, it is necessary to find a method to improve convergence efficiency without data scheduling. The server can adjust the appropriate aggregation parameters if it learns the user's data distribution so that the weights of the global model quickly move closer to the weights of the centralized update. However, due to privacy considerations, users are unlikely to tell the server the number of each label in their local dataset. This would mean that the server cannot accurately obtain the user's data distribution. Therefore, we need to use the information updated by users to obtain data distribution divergence between users.

We first define the concept of the weight difference in this paper. That is the sum of the weight divergence of all layers of the same neural network. Recall the calculation equation of the user's local iteration:

$$\mathbf{w}_m^{k+1} = \mathbf{w}_m^k - \eta \nabla F(\mathbf{w}_m^k, \mathcal{D}_m), \quad (2)$$

where η is the local learning rate, F is the loss function, and $\mathcal{D}_m$ is the dataset owned by user m . It can be seen from Equation (2) that, when all users have the same neural network model, the data owned by the users are the main factor that affects the neural network weight of user m . Intuitively speaking, if the weight divergence in the models owned by two users is smaller, the data distribution is more similar. According to this insight, the

server can assign different aggregation parameters to the user's update based on the weight divergence of the update to the user.

Intuitively, we need to impose penalties on the users whose local data distribution is more different from the global to ensure that these users' data will not negatively affect the global model. However, this is not the case. The situation that weight uploaded by the user is quite different from the global model often indicates that the global model owned by the server has not fully aggregated this user's "knowledge". Each user's data are helpful for machine learning tasks and can represent user preferences or other characteristics. Therefore, we give specific compensation to users with large weight divergence so that the server can adapt to the local models of these users more quickly. In this paper, we use $E(m)$ to denote the data distribution payoff.

Our initial method is based on the following insight: the server may not have the information carried by the arriving user update if this user update is very different from the global model. At this time, the server will allocate a more considerable aggregation parameter α to the user, which is called the data distribution reward in this paper, even if the user's update is stale. However, under the asynchronous and parallel federated learning framework, one concern regarding such a method is that the excessive weight difference may be caused not only by uneven data distribution but by potential attackers sending the model trained by wrong or low-quality data. In addition, even if the data owned by each user are of a certain quality, the server cannot determine the divergence between the user's local weight and the global weight is caused by whether the difference in data distribution is due to staleness. The former is easy to distinguish, since the weight of malicious users will not gradually decrease with the progress of aggregation, and the difference from the global model will always be very high. However, solving the latter server requires additional information.

Recall the global aggregation formulation (1) of federated learning. After performing the first round of aggregation, the server has obtained the first round of updated information for most users. We also use experiments to verify the feasibility of the weight difference to measure the difference in data distribution. Figure 2 shows the effect of different distributions of data on the MNIST data set on the user's model parameters. We can see that the weight divergence of users with the same data distribution will be slight. The data tags owned by $user_2$ and $user_{24}$ are both 1 and 5, and the number of each data tag is the same. It can be seen from the figure that the weight divergence between the two users is minimal. The data held by $user_{38}$ are a fuzzy picture generated by GAN, therefore it is very different from all users.

3.3. Aggregation Parameter Settings

The server will group users according to the user's updates through the updates aggregated in the first few global epochs. Users with similar weights will be classified together, and the server will consider these users to have the same data distribution. Therefore, our aggregation parameter α consists of two parts: data distribution reward and staleness penalty.

The amount of data will affect the time of local iteration. Some users have a small amount of data, and the user's update may reach the server earlier in each global epoch. Similarly to the FedAvg algorithm, the global aggregation parameters will be affected by the amount of data the user has, that is, $\frac{|D_m|}{D}$, where D is the total number of the training data. Besides, when a straggler update arrives at the server, the server will add an exponential penalty term $e^{-a(t_m - T_g^0)}$ to the update information, where a is a parameter that controls the staleness penalty. The larger a is, the greater the penalty for stale updates. t_m is the time when the user arrives at the server, and T_g^0 is the start time of the latest global epoch. The staleness penalty is a necessary setting for asynchronous aggregation. Next, we will mainly introduce the influence of data distribution on aggregation parameters.

Figure 2. Weight difference between users.

When the user's update of m arrives at the server, the server will determine whether the user's update is a delayed update according to the previously described method. When the number of local iterations k_m of the user m is the same as the global iteration counter T_g , the staleness penalty parameter a is always 0. This is because most users' updates can arrive in a short time. Imposing staleness penalties for these users is unnecessary and will have a negative impact on the convergence speed.

As shown in Figure 1, the server maintains a model list (denoted as $\mathcal{L}$) with a length of $\frac{n}{\beta}$. β is a parameter similar to the "client selection" in synchronous communication, which controls the range of the aggregation model in a global epoch. That is, the latest $\frac{n}{\beta}$ users' updated information will be considered during each global aggregation. The server dynamically maintains a topology of user model weight distribution based on this information. The server groups different users according to the weight divergence when the user uploads the update for the first time, and the distribution of users in each group is roughly the same (users 2, 29, 24, and 25 in Figure 2).

As illustrated in Figure 2, when a user update arrives at the server, the server will calculate the weight divergence d_m between the update $\mathbf{w}_m$ and the previous global model $\mathbf{w}_{old}$ according to the stored local model list. Further, it will find the mean value of the distance between $\mathbf{w}_{old}$ and all weights in the model list $\mathcal{L}$, that is, $Mean(\mathcal{L})$ (the black dotted line in the figure). Obviously, when the user's arrival update is in the outer circle, the global model is more susceptible to the user's influence. Accordingly, the server will take corresponding reward and punishment measures based on the difference between the arrival update weight and the global model. Specifically, if $\|\mathbf{w}_m - \mathbf{w}_{old}\| \leq Mean(\mathcal{L})$, the payoff of the data distribution in the aggregate weight $E(m) = 1$. When $\|\mathbf{w}_m - \mathbf{w}_{old}\| \geq$

$Mean(\mathcal{L})$, the server will determine which group the user belongs to. If the model-list stored on the server has aggregated the data of the group with more than $\frac{1}{\beta}$, then the server will add a penalty term to the user's update. Otherwise, a reward will be given. In conclusion, we have:

$$E(m) = \begin{cases} 1, & \text{if } d_m < d_M; \\ \beta \frac{1-n_c}{n_c}, & \text{if } d_m \geq d_M, \end{cases} \quad (3)$$

where $d_M = Mean(\mathcal{L})$, and n_c is the total amount of data of user group c assigned to user m by the server. We call the user m that satisfies $d_m < d_M$ as the “central” user, otherwise they are the “edge” user. As shown in Figure 3, when the arriving user is an “edge” user, the server will determine whether enough updates of the data distribution have been aggregated. If the server has aggregated $\frac{n_c}{\beta}$ data of the group, then the server will impose a penalty for the update (the red cross in the figure). Otherwise, the corresponding reward will be given (orange cross in the picture). If we use α^* to denote the aggregate parameters of all users in the model list $\mathcal{L}$, and $\alpha^*[m]$ is the aggregate parameter of the user m 's update information in model list $\mathcal{L}$, then, when $l > n$, Equation (1) can be rewritten as:

$$\mathbf{w}_{new} = \sum_{m=0}^{\text{len}(\mathcal{L})} \frac{\alpha^*[m]}{\text{sum}(\alpha^*)} \mathcal{L}[m], \quad \text{if } l > n. \quad (4)$$

where function $\text{sum}()$ is the sum of all items in the aggregate parameter list α^* . The detailed process of the STAFL method is described in Algorithm 1.

Figure 3. The impact of data distribution on user weight.

Algorithm 1 Staleness-tolerate asynchronous federated learning.

Input: user update information $\mathbf{w}_m$, the amount of data n_m owned by the user m , staleness function $S(l)$, data distribution payoff $E(m)$, User m update counter k_m , global epoch counter T_g , aggregation counter l , model-list $\mathcal{L}$.

Output: finalized global model

At server side:

```

while User local update  $\mathbf{w}_m^{k_m}$  arrives at the server do
  if user  $m$  is straggler then
    compute staleness penalty  $S(l) = \exp(a(t_m - T_g^0))$ 
  end if
  if  $l > n$  then
    compute data distribution payoff  $E(m)$  according to Equation (3)
    compute  $\alpha = \frac{|\mathcal{D}_m|}{|\mathcal{D}|} S(l) E(m)$ 
  end if
  perform aggregation using Equation (1)
  update model-list  $\mathcal{L}$ 
  if meet the accuracy requirements then
    break;
  end if
end while

```

At client side:

```

for each user  $m$  do
  update local model according Equation (2)
end for

```

3.4. Model List Update and Weight Divergence Computation

As mentioned above, for the server to judge the aggregated data distribution of the global model, we need to maintain a model list on the server-side. Although storing user update information will consume additional storage costs, our experimental results show that even holding a very short model list can significantly improve the negative impact of the non-IID dataset on the global model. The longer the model list, the more accurately the server can estimate the data distribution required for the global model. When a user update arrives at the server, the server immediately aggregates the update and stores the user's weight in the model list. We will delete the oldest user update stored in the model list if the model list is full. In this way, we maintained a fixed-length sequence of user-local model weights that represents the latest global model.

We use the Euclidean distance between the weights of the two models to represent the difference in data distribution for users. When a user update arrives at the server, the server immediately computes the Euclidean distance d_m between that update and the global model. In addition, it should be noted that, since the computational complexity of calculating the Euclidean distance of the neural network weight will increase exponentially with the number of users participating in the training, we will only group users based on their first-round update information. The local model information uploaded by the user for the first time will only be affected by its local training dataset and will not be affected by the weight of other user models. Therefore, the weight divergence between users with different data distributions in the first epoch will be the largest, thus facilitating server grouping.

Figure 4. Weight quantification steps.

3.5. Reduce Communication Overhead

Since many users participate in asynchronous communication, it is necessary to use the communication compression mechanism to reduce the communication cost. In this paper, we use the method of communication quantization to reduce the weight accuracy of the network from float (32 bits) to 4 bits. The overall scheme of communication compression is shown in Figure 4. First, each user trains their model locally. After the training is completed, the user quantizes the model weight and sends the quantized content to the server. After receiving the quantized model weight, the server performs a de-quantization operation to restore the original update of the user. Subsequently, the server uses the weight obtained by de-quantization to perform model aggregation to obtain a new global model. Then, it quantizes the model and sends it to all users participating in the training. The user then starts the next local epoch based on the new global model obtained by de-quantization.

4. Experiment Evaluations

To make sure that our experiments are closer to the Internet of Things, we integrated the training of federal learning into the actual environment. We assume that 50 users participate in the training process of federated learning. We also assume that all users are evenly distributed. In other words, the distances between all users, the server, and the transmission delay consumed are equal. The network condition is basically stable, and there is no massive packet loss. However, our method can tolerate a certain amount of data loss and is robust to non-large amounts of data loss. In the comparative experiment, the waiting time for each global epoch is the longest time consumed by the user in that global epoch minus the shortest time. In order to simulate the heterogeneity of devices in federated learning, we set different computing capacities for different devices (that is, we reduce the computing capacities of a device by reducing the frequency) and set a different amount of data for all users.

4.1. Data Settings

Datasets: we use MNIST, FEMNIST, and CIFAR-10 datasets in this experiment to prove the effectiveness of our proposed method. The MNIST dataset comprises 60,000 training samples and 10,000 testing samples. The image is a fixed size (28×28 pixels) with a value of 0 to 1. Each image is flattened and converted into a one-dimensional numpy array of 784 (28×28) features. The CIFAR-10 data set contains 60,000 32×32 color images which comprise 50,000 training images and 10,000 test images. These samples are divided into five training batches and one test batch. Each batch has 10,000 samples. The test batch contains 1000 randomly selected images from each category. Training batches contain the remaining images; however, some may contain more images from one category than another. The five training sets contain exactly 5000 images from each class. The Federated Extended

MNIST dataset (FEMNIST) is the extended MNIST dataset based on the writer of the digit and character.

Non-independent and identically distributed setting: in order to simulate a non-independent and identically distributed experimental environment, each user in the experiment randomly selects a part from different categories of the training data set, that is, each user does not completely own all the categories of the training data set, and the number of data in each user is different. In particular, we will replace the data set of one of the users with fuzzy data (noise is added to the image or an image data set without complete GAN training).

4.2. Experimental Results of Model List and Weight Divergence

We first focus on the performance of our method under different data distributions. In order to simulate different degrees of non-independent and identically distributed data sets, we divide the experimental scenarios into two types. In the first type of non-independent and identically distributed data set, most users have five or more data labels, and a small number of users only have data with one or two labels. Under this scenario, the data are not extremely non-independent and identically distributed. User updates that arrive continuously within a specific time period can cover all data labels when performing federated learning under this data distribution. In the second non-independent and identically distributed case, each user has at most two data labels. In this case, if the global aggregation operation is performed, the global model will be severely affected by the “non-independent and identically distributed”, and the accuracy jitter will be fairly obvious.

As shown in Figures 5 and 6, we compare STAFI with the traditional asynchronous method (i.e., heuristic method, the aggregation parameter α is large at the beginning of training, and, as the global aggregation progresses, the updated aggregation parameter α assigned to new users update will be smaller). In addition, in order to verify the influence of the weight difference grouping on the convergence speed, we use STAFI-1 to represent the accuracy change in the model when we only perform the staleness penalty without the data distribution payoff. We use STAFI-2 to describe the situation where we impose the staleness penalty on stragglers and add the data distribution payoff to “edge” users. When the user’s training data has one or two labels, as shown in Figure 6, the test accuracy of STAFI-2 improved the fastest. Compared with the heuristic method, only aggregating the model list $\mathcal{L}$ can reduce the impact of the previous stale user update on the global model. The staleness penalty can also reduce the degree of accuracy jitter. As shown in Figure 5, when most users’ data have more than five labels, the weight divergence between users is not particularly obvious. Therefore it is difficult for the server to divide users into multiple groups. In this case, the effect of adding data distribution payoff to each user is not very obvious. However, it still has a certain outcome.

Figure 5. STAFI performance under scenario 1. (a) Test accuracy of scenario 1. (b) Test loss of scenario 1.

Figure 6. STAF performance under scenario 2. (a) Test accuracy of scenario 2. (b) Test loss of scenario 2.

4.3. Model List Length Discussion

As mentioned in Section 3, the parameter β controls the range of user updates considered during server aggregation. Compared with the traditional asynchronous aggregation method, aggregating the continuously updated model list $\mathcal{L}$ can reduce the stale updates that the server aggregated a long time ago. The global model will only be affected by the latest few user updates. Correspondingly, the length of the model list $\mathcal{L}$ will affect the scope of server aggregation. Similar to the client selection of synchronous federated learning, the larger the β , the more user updates in a global aggregation, and the greater the possibility of covering all data distributions.

Correspondingly, the length of the model list $\mathcal{L}$ will affect the scope of server aggregation. Similar to the client selection in synchronous federated learning, the larger the β , the more user updates in a global aggregation, and the greater the possibility of covering all data distributions. However, in asynchronous federated learning, the length of $\mathcal{L}$ is not as long as possible. First of all, storing a user's model requires storage space, especially a complex neural network, which requires keeping a large amount of model weight information. Secondly, the longer the model list, the more obvious the impact of stale updates during aggregation. Further, a one-time aggregation of some users' updates is enough for the global model to learn the corresponding knowledge for the server. We explore the impact of different β on the global model in the FEMNIST data set. We use the final aggregated model in each global epoch as the evaluation standard. As can be seen from Figure 7, because the aggregation parameters can compensate for the weight divergence, when the length of the model list is within a certain range, the difference in the accuracy of the overall model is not very obvious.

Figure 7. Different β on FEMNIST dataset. (a) Test accuracy. (b) Test loss.

4.4. Comparison of STAFI with Other Methods

In this subsection, we compare the performance of STAFI with existing methods. These methods are:

- **FedAvg** [17]: one of the most traditional synchronous aggregation methods of federated learning randomly selects a part of users to participate in training in each global epoch and simply discards the stragglers;
- **FedProx** [23]: FedProx is an improvement over FedAvg. Compared to FedAvg, FedProx still aggregates updates from some stragglers. In addition, similar to STAFI, it also uses Euclidean distance to improve the model's performance on non-IID datasets;
- **ASO** [24]: ASO is a novel asynchronous federated learning algorithm that adaptively trades off convergence speed and accuracy based on staleness.

We evaluate the performance of different methods on the FEMNIST and CIFAR-10 datasets. We assume that the data are extremely non-IID and that 50% of users are stragglers. Table 2 shows the evaluation results. We highlight the optimal performance in different situations in bold. As can be seen from the table, our proposed method does not perform well at the beginning of training. The reason for this is that STAFI needs to calculate the weight divergence between users at the beginning of training, as opposed to other schemes that only perform weighted averages or focus only on staleness. As mentioned above, when a large number of users participates in training, the server needs a lot of time to calculate the Euclidean distance between them, even if the server can perform many calculations in parallel. As training progresses, STAFI can dynamically set aggregation parameters based on the model list and previous grouping results, resulting in better performance than other methods. In contrast, the synchronous aggregation strategy of FedAvg and FedProx cannot maintain the original convergence efficiency when there are a lot of stragglers. The convergence effect of the ASO is also unsatisfactory when the data distribution is extremely heterogeneous.

Table 2. Comparison with other methods.

	Dataset	Accuracy at 90 s	Accuracy at 180 s	Accuracy at 300 s
FedAvg	FEMNIST	0.39	0.51	0.70
	CIFAR-10	0.27	0.49	0.64
FedProx	FEMNIST	0.37	0.58	0.75
	CIFAR-10	0.39	0.47	0.69
ASO	FEMNIST	0.45	0.49	0.71
	CIFAR-10	0.32	0.54	0.67
STAFI	FEMNIST	0.41	0.62	0.81
	CIFAR	0.24	0.56	0.72

4.5. Communication Cost Comparison

In order to reduce the communication overhead of asynchronous federated learning, we adopt the communication compression method described in Section 3. We set the number of users to 2, 3, 4, and 5 to verify the time spent before and after communication compression and the total time required to reach 90% accuracy. We distributed the data to the corresponding number of Raspberry Pis for experimentation. It can be seen from Figure 8a that, after compression, the communication time has been reduced by 10%. Compressing the transmission data does not increase the convergence time. As shown in Figure 8b, the time required for the global model to reach 90% accuracy is reduced. Figure 8c shows the accuracy comparison chart before and after communication compression on the MNIST data set. Since there are fewer users participating in the training, the influence of non-independent and identical distribution of data on convergence is not very obvious.

Figure 8. Comparison before and after communication compression. (a) comparison of communication time. (b) comparison of training time. (c) comparison of test accuracy.

5. Conclusions

In this paper, we design a federated learning system architecture for asynchronous communication, called STAFL. In STAFL, users can upload local updates at any time. The server can use the information stored in a model list to determine whether enough information about a certain data distribution has been aggregated in the global model so that it can better impose corresponding penalties or rewards on arriving updates. We also use communication compression to reduce the communication cost caused by asynchronous aggregation. Compared with other methods, our method focuses more on the effect of data heterogeneity on the global model. STAFL can control the negative impact of non-IID datasets on the convergence rate with the least cost. The experimental results show that our method has a significant improvement in convergence efficiency compared with other methods. In future work, we will focus on model performance on other complex datasets and focus on the points where STAFL can be improved, such as obtaining prior knowledge of data distribution more efficiently and conducting theoretical research.

Author Contributions: Funding acquisition, F.Z., Y.Z. and Z.C.; Methodology, F.Z., J.H., Y.Z. and Z.C.; Validation, X.T.; Visualization, B.C.; Writing—original draft, J.H.; Writing—review and editing, Y.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102000 and in part by the National Natural Science Foundation of China under Grant(No. 62172215) and in part by the Natural Science Foundation of Jiangsu Province(No. BK20200067), in part by the A3 Foresight Program of NSFC (Grant No. 62061146002).

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

Abbreviations

The following abbreviations are used in this manuscript:

FL	Federated Learning
iid	independent and identically distributed
non-iid	non-independent and identically distributed
IoT	Internet of Things

References

1. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
2. Kuflik, T.; Kay, J.; Kummerfeld, B. Challenges and solutions of ubiquitous user modeling. In *Ubiquitous Display Environments*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 7–30.
3. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [\[CrossRef\]](#)
4. Wang, X.; Han, Y.; Wang, C.; Zhao, Q.; Chen, X.; Chen, M. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Netw.* **2019**, *33*, 156–165. [\[CrossRef\]](#)
5. Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; Zhang, Y. Federated learning for data privacy preservation in vehicular cyber-physical systems. *IEEE Netw.* **2020**, *34*, 50–56. [\[CrossRef\]](#)

6. Yu, Z.; Hu, J.; Min, G.; Lu, H.; Zhao, Z.; Wang, H.; Georgalas, N. Federated learning based proactive content caching in edge computing. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.
7. Guo, X.; Liu, Z.; Li, J.; Gao, J.; Hou, B.; Dong, C.; Baker, T. VeriFL: Communication-Efficient and Fast Verifiable Aggregation for Federated Learning. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 1736–1751. 10.1109/TIFS.2020.3043139. [\[CrossRef\]](#)
8. Vonderwell, S. An examination of asynchronous communication experiences and perspectives of students in an online course: A case study. *Internet High. Educ.* **2003**, *6*, 77–90. [\[CrossRef\]](#)
9. Chen, Y.; Ning, Y.; Slawski, M.; Rangwala, H. Asynchronous online federated learning for edge devices with non-iid data. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; pp. 15–24.
10. Lu, Y.; Huang, X.; Dai, Y.; Maharjan, S.; Zhang, Y. Differentially Private Asynchronous Federated Learning for Mobile Edge Computing in Urban Informatics. *IEEE Trans. Ind. Inform.* **2020**, *16*, 2134–2143. [\[CrossRef\]](#)
11. Liu, X.; Qin, X.; Chen, H.; Liu, Y.; Liu, B.; Zhang, P. Age-aware Communication Strategy in Federated Learning with Energy Harvesting Devices. In Proceedings of the 2021 IEEE/CIC International Conference on Communications in China (ICCC), Xiamen, China, 28–30 July 2021; pp. 358–363.
12. Damaskinos, G.; Guerraoui, R.; Kermarrec, A.M.; Nitu, V.; Patra, R.; Taïani, F. Fleet: Online federated learning via staleness awareness and performance prediction. In Proceedings of the 21st International Middleware Conference, Delft, The Netherlands, 7–11 December 2020; pp. 163–177.
13. Wu, W.; He, L.; Lin, W.; Mao, R.; Maple, C.; Jarvis, S. SAFA: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Trans. Comput.* **2020**, *70*, 655–668. [\[CrossRef\]](#)
14. Chai, Z.; Chen, Y.; Zhao, L.; Cheng, Y.; Rangwala, H. Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv* **2020**, arXiv:2010.05958.
15. Karimireddy, S.P.; Kale, S.; Mohri, M.; Reddi, S.; Stich, S.; Suresh, A.T. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*; PMLR: McKees Rocks, PA, USA, 2020; pp. 5132–5143.
16. Yu, F.; Rawat, A.S.; Menon, A.; Kumar, S. Federated learning with only positive labels. In *International Conference on Machine Learning*; PMLR: McKees Rocks, PA, USA, 2020; pp. 10946–10956.
17. McMahan, B.; Moore, E.; Ramage, D.; Hampson, S.; y Arcas, B.A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*; PMLR: McKees Rocks, PA, USA, 2017; pp. 1273–1282.
18. Khaled, A.; Mishchenko, K.; Richtárik, P. Tighter theory for local SGD on identical and heterogeneous data. In *International Conference on Artificial Intelligence and Statistics*; PMLR: McKees Rocks, PA, USA, 2020; pp. 4519–4529.
19. Zhao, Y.; Li, M.; Lai, L.; Suda, N.; Civin, D.; Chandra, V. Federated learning with non-iid data. *arXiv* **2018**, arXiv:1806.00582.
20. Sahu, A.K.; Li, T.; Sanjabi, M.; Zaheer, M.; Talwalkar, A.; Smith, V. On the convergence of federated optimization in heterogeneous networks. *arXiv* **2018**, arXiv:1812.06127.
21. Chen, M.; Yang, Z.; Saad, W.; Yin, C.; Poor, H.V.; Cui, S. A joint learning and communications framework for federated learning over wireless networks. *IEEE Trans. Wirel. Commun.* **2020**, *20*, 269–283. [\[CrossRef\]](#)
22. Chen, M.; Mao, B.; Ma, T. FedSA: A staleness-aware asynchronous Federated Learning algorithm with non-IID data. *Future Gener. Comput. Syst.* **2021**, *120*, 1–12. [\[CrossRef\]](#)
23. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [\[CrossRef\]](#)
24. Xie, C.; Koyejo, S.; Gupta, I. Asynchronous federated optimization. *arXiv* **2019**, arXiv:1903.03934.