CSMAAFL: Client Scheduling and Model Aggregation in Asynchronous Federated Learning

Xiang Ma*, Qun Wang†, Haijian Sun‡, Rose Qingyang Hu*, and Yi Qian§
*Department of Electrical and Computer Engineering, Utah State University, Logan, UT
†Department of Computer Science, San Francisco State University, San Francisco, CA
‡School of Electrical and Computer Engineering, University of Georgia, Athens, GA
§Department of Electrical and Computer Engineering, University of Nebraska–Lincoln, Omaha, NE

***Abstract*—Asynchronous federated learning aims to solve the straggler problem in an environment with heterogeneity, where certain clients may possess limited computational capacities, potentially leading to model aggregation delay. The core concept behind asynchronous federated learning is to empower the server to aggregate the model as soon as it receives an update from any client without waiting for updates from multiple clients or adhering to a predetermined waiting time, which is typical in synchronous mode. Because of the asynchronous setting, a potential concern is the emergence of a stale model issue, wherein slow clients might employ an outdated local model for their data training. Consequently, when these locally trained models are uploaded to the server, they may impede the convergence of the global training. Therefore, effective model aggregation strategies play a significant role in updating the global model. Besides, client scheduling is critical when heterogeneous clients with diversified computing capacities participate in the federated learning process. This work first investigates the impact of the convergence of asynchronous federated learning mode when adopting the aggregation coefficient in synchronous mode. Effective aggregation solutions that can achieve the same convergence result as in the synchronous mode are proposed, followed by an improved aggregation method with client scheduling. The simulation results in various cases demonstrate that the proposed algorithm converges with a similar level of accuracy as the classical synchronous federated learning algorithm but effectively accelerates the learning process, especially in its early stage.**

***Index Terms*—Asynchronous federated learning, client scheduling, model aggregation**

I. Introduction

The distributed learning nature of Federated Learning (FL) [1] can effectively address the privacy concerns associated with machine learning by sharing only the refined model instead of exposing raw data to other devices. Under the coordination of the central server, clients collaboratively train a global model through an iterative process. The training process is organized into discrete rounds, and each round is based on the previous round's training results. This classical synchronous federated learning (SFL) could suffer straggler issues caused by slow clients [2].

With the rapid advancement of pervasive intelligence, a wide range of heterogeneous devices can now serve as clients for FL. These devices include desktop computers, laptops, smartphones, Raspberry Pis, and more. These devices can conduct local training and learning by collecting data from nearby sensors. However, computing resource-constrained devices, such as Raspberry Pis, may experience delays and become stragglers when processing substantial volumes of data. Numerous strategies have been suggested for integration into synchronous federated learning (SFL) frameworks to tackle these straggler issues.

In [3], a method was proposed where a small subset of clients is sampled in each round, eliminating the need to wait for updates from all clients. However, it does not fully address the straggler issue, as slower clients can still be selected within the subset. A predefined synchronous window was suggested to aggregate as many client updates as possible in each round in [4]. However, slow clients may not get the opportunity to upload their updates during the entire learning process. Furthermore, it does not ensure the convergence of the model. An adaptive local computation scheme in resource-constrained edge computing systems was proposed in [5], where fast clients can execute more local iterations than slow clients. Nevertheless, the global model still needs to wait for all clients to complete their local computations before getting the updates.

Another strategy to tackle the straggler problem is called asynchronous federated learning (AFL) [6], in which the local model uploading at clients and the global aggregation at the server are decoupled. Thus, aggregation of the global model is executed without waiting to receive all the client models. The authors in [7] presented an online AFL algorithm with data being non-Independent and Identically Distributed (non-IID). The server initiates the aggregation process upon receiving an update from any single client. The newly aggregated model is subsequently distributed to the clients considered "ready". Nevertheless, a clear definition for "ready" clients is absent. And it does not provide a methodology for their selection either. Similarly, the researchers in [8] proposed to decouple scheduling and aggregation but did not define the criteria for client selections in the scheduling. A Euclidean distance-based adaptive federated aggregation algorithm was introduced in [9] to solve the stale model problem in AFL. The staleness is measured using the distance between the current and stale global models. This evaluation process requires the server to store all the global models, starting from the initial training phase up to the current iteration, leading to significant consumption of storage resources on the server. AFL over wireless networks is considered in [10], where a global model is broadcast to all the clients in each global iteration. Each client must independently

ICC 2024 - IEEE International Conference on Communications | 978-1-7281-9054-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICC51166.2024.10622903

determine whether to commence training with the latest model. This has resulted in significant wastage of both energy and time resources. Despite the various works mentioned above, few of them have offered a comprehensive architectural outline of AFL. Moreover, no comparative analysis between SFL and AFL has been provided.

This work develops a new AFL framework with client scheduling and model aggregation. Unlike the existing work, our method allows the recently aggregated global model to be sent exclusively back to the client that has just uploaded its local model. This eliminates the need to broadcast to a group of clients, thereby circumventing the necessity of addressing the client selection challenge globally. Instead, we introduce a client scheduling algorithm as part of our algorithm. The proposed scheduling mechanism considers both computational capabilities and fairness. The model aggregation component addresses the staleness problem inherent in AFL. We employ the iteration difference as the metric for staleness. Only one hyperparameter is introduced to keep the aggregation simple. Besides, a detailed AFL architecture and a comparative analysis between AFL and SFL are given. The major contributions of this paper are summarized as follows:

- A new AFL architecture is introduced. One client is selected to upload its local model in each global iteration. Subsequently, the newly aggregated global model is returned only to the client that uploaded its local model.
- A comparative analysis of the completion time between AFL and SFL is presented. While the total learning completion time is not necessarily shorter in AFL, its distinct advantage lies in the timely updating of the global model within a significantly reduced timeframe.
- A general baseline AFL framework is first introduced, which can achieve the same accuracy performance as SFL. Then, the solution of the aggregation coefficients is developed based on this framework setting.
- A new AFL framework incorporating client scheduling and model aggregation is proposed, which considers client computational capabilities, fairness, and model staleness as import metrics. As a result, the proposed AFL framework provides a solution to the inherent stale model issue.

The rest of the paper is organized as follows. Section II introduces the system model of SFL and AFL, where the two models are compared. Section III presents the baseline AFL framework to achieve the same learning performance as SFL. The advanced client scheduling and model aggregation framework is then developed. Simulation results are given in Section IV. The paper is concluded in Section V.

II. System Model

The classical federated averaging (FedAvg) algorithm is a synchronous communication model where the server performs aggregation after receiving models from a predetermined number of clients or after a set amount of time has elapsed. While in an asynchronous setting, the server commences model aggregation immediately upon receipt of an update. This ensures that the server is always updated with the most recent model. The considered FL system comprises a central server and M clients. Each client m has $|D_m|$ amount of dataset.

A. Synchronous Federated Learning

In the SFL framework, the learning process unfolds iteratively between the server and clients. Each iteration consists of four basic steps. In step 1 (S1), the server disseminates the current global model to all clients. In step 2 (S2), clients utilize the global model as the initial point and employ an optimization method such as stochastic gradient descent (SGD) to derive a new local model. Following this, in step 3 (S3), the updated local model is uploaded to the server. Finally (S4), the server awaits either a fixed amount of time or a fixed number of model updates from a predetermined number of clients before performing aggregation. This process can be observed in Fig. 1 (left). Notably, in SFL, a "wait" stage allows all clients to upload their respective local models, thereby preventing the server from aggregating prematurely.

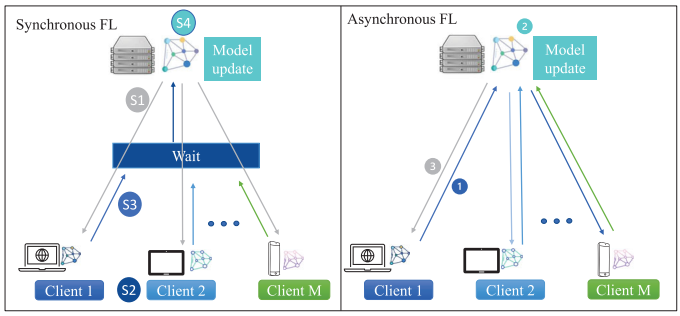


Fig. 1. Synchronous vs Asynchronous FL.

The global model is first initialized as w_0. Clients perform the local learning process as follows:

$$w_t^m = w_t - \eta \nabla F_m(w_t), t = 0, 1, 2, ..., \quad (1)$$

where w_t^m is the local model of client m at round t after local learning, w_t is the global model broadcast by the server at round t, η represents the learning rate, and $\nabla F_m(w_t)$ signifies the gradient of the loss function $F_m(w_t)$. This occurs in step (S2), while the global model w_t is the initial reference point for each client's local learning process.

Once the server receives the models from a predetermined number of clients or when a specified time limit elapses, the server proceeds to perform aggregation using:

$$w_{t+1} = \sum_{m=1}^{M} \alpha_m w_t^m, \quad (2)$$

where α_m is the aggregation coefficient of client m, which is usually defined as $\alpha_m = \frac{|D_m|}{\sum_c |D_c|)}$. Without loss of generality, we let all clients participate in each learning round as in equation (2).

Based on the procedure mentioned above, the model of each client is synchronized with the global model following step (S1) in each round. Subsequently, in step (S2), the local models become different across clients after local learning.

Step (S3) acts as a blocking operation, thus expanding the overall learning time. Lastly, in step (S4), the global model is updated by using equation (2).

B. Asynchronous Federated Learning

Under asynchronous settings, the server is not mandated to wait. Instead, the server initiates the aggregation process once a client model is received. This approach enables faster clients to proceed with local learning without waiting for slower clients. In SFL, a round is uniformly observed by both the server and all clients. However, faster clients perform more rounds in AFL than their slower counterparts. We will utilize the global aggregation number to keep track of the learning process. To differentiate the notation used in SFL, we employ the term "iteration" along with the symbols i and j to denote the global aggregation time in AFL. The aggregation process at the server is then performed as follows:

$$w_{j+1} = \beta_j w_j + (1-\beta_j) w_i^m, \tag{3}$$

where w_{j+1} is the global model in iteration $j+1$, w_j represents the global model from the previous iteration j. The local model w_i^m corresponds to the model obtained from client m after local learning, utilizing the global model from iteration i when client m was reselected during iteration j. $\beta_j \in (0,1)$ denotes the aggregation coefficient. Following the aggregation process, the client m that recently sent its model to the server receives the updated global model from the server. In Fig. 1 (right), it is evident that only one client participates in the learning process during each iteration. Consequently, the subsequent local learning iteration continues based on the received updated global model, that is

$$w_{j+1}^m = w_{j+1} - \eta \nabla F_m(w_{j+1}), \tag{4}$$

where w_{j+1}^m is the updated local model of client m. The proposed AFL follows a different approach where only one client receives the updated model during each global iteration. While client m uploads its model to the server, other clients can either continue their local computations or wait for the channel to idle. This allows for concurrent local computations, achieving efficient resource utilization in the AFL framework.

C. Comparison

There exists extensive idle time in SFL while learning continues uninterrupted in AFL. Based on the details above, we can infer that AFL is expected to learn faster than SFL as more computations can be executed within the same time frame. We analyze both modes in specific scenarios to assess the speed advantage offered by AFL. First, we consider a homogeneous scenario in which all clients have identical computational capabilities. Let τ denote the computation time, which is the same for all clients. Another assumption is that a client can be scheduled to upload the local model again only when all other clients have completed their model uploads. We further assume all the clients have the same channel conditions and power allocations to facilitate the analysis. Thus, uploading time τ^u in step (S3) is identical for all the clients.

In step (S1), the time required to download the global model is assumed to be τ^d. Hence, in SFL, the total completion time for one round, when employing time-division multiple access (TDMA), can be expressed as $\tau_{ho}^{syn} = \tau^d + \tau + M \cdot \tau^u$. Consequently, the global model receives updates after τ_{ho}^{syn} time has elapsed. On the other hand, in AFL, performing the same operation takes $\tau_{ho}^{asyn} = M \cdot \tau^u + M \cdot \tau^d + \tau$ time. Although AFL requires an additional $(M-1) \cdot \tau^d$ time compared to SFL to obtain the same global model, AFL updates the global model every $\tau^u + \tau^d$ time instead of waiting for $\tau^d + \tau + M \cdot \tau^u$ time as in SFL when communication time is significantly longer than computation time.

In the heterogeneous scenario, clients have varying computation capabilities, and channel conditions differ. The computation time for the fastest client is assumed to be τ, while the slowest client requires $a \cdot \tau$ time. Typically, computation takes less than communication, with communication being the latency bottleneck of the system. Nevertheless, in scenarios where slow clients have parallel computation tasks, the value of a can become significant, resulting in $a \cdot \tau$ surpassing $M \cdot \tau^u$. As a result, the completion time for one round is predominantly determined by the computation duration of the slowest client rather than the communication time. In SFL, the global model must wait for $\tau_{he}^{syn} = \tau^d + a \cdot \tau + M \cdot \tau^u$ time to get updated. This implies that the faster clients must remain idle during this waiting period. In contrast, AFL accomplishes model updates within a timeframe spanning from $M \cdot \tau^d + \tau + M \cdot \tau^u$ to $M \cdot \tau^d + a \cdot \tau + M \cdot \tau^u$ time, under the assumption that all client models are uploaded with faster clients being scheduled first. Notably, in AFL, the global model is updated every $\tau^u + \tau^d$ time. The server performs aggregation more frequently in AFL than in SFL.

The differences between AFL and SFL contribute to AFL's accelerated learning pace. However, they can also result in model staleness among slow clients, which may impede the global model's convergence. Hence, it becomes essential to determine the optimal aggregation coefficient, denoted as β, and incorporate client scheduling throughout the learning process. Client scheduling is pivotal in enhancing convergence throughout the learning process while optimizing the aggregation coefficient to minimize individual client model staleness. These strategies work in tandem to ensure overall convergence and reduce the impact of model staleness in AFL.

III. Proposed Algorithm

In this section, we first utilize the SFL aggregation coefficient in AFL. Subsequently, we introduce an AFL algorithm to attain comparable learning performance to SFL. Lastly, we propose a client scheduling and model aggregation framework in AFL (CSMAAFL).

A. SFL Aggregation Coefficient in AFL

In SFL, client scheduling may not be critical since all clients actively participate in model updates. Furthermore,

the aggregation coefficient is determined by considering the relative number of samples present on each client, i.e.,

$$\sum_{m=1}^{m} \alpha_m = \sum_{m=1}^{M} \frac{|D_m|}{\sum_c |D_c|)} = 1. \tag{5}$$

Here, α_m represents the relative significance of the client m in terms of its model's contribution. However, when using α_m as the aggregation coefficient in AFL, the influence of initially selected clients diminishes as the iterations progress. Given a specific client scheduling sequence $\phi(1), \phi(2), \ldots, \phi(M)$, where $\phi(i)$ indicates the index of the client that is scheduled to upload its local model in iteration i, equation (3) can be expressed as follows:

$$\begin{aligned} w_{j+1} &= (1-\alpha_{\phi(j)})w_j + \alpha_{\phi(j)} w_i^{\phi(j)} \\ &= (1-\alpha_{\phi(j)})((1-\alpha_{\phi(j-1)})w_{j-1} + \alpha_{\phi(j-1)} w_k^{\phi(j-1)}) \\ &\quad + \alpha_{\phi(j)} w_i^{\phi(j)}, \end{aligned} \tag{6}$$

where $\phi(j-1)$ represents the client scheduled in iteration $j-1$, and k denotes the iteration when client $\phi(j-1)$ is scheduled to upload its model last time. Consequently, the aggregation coefficient for client $\phi(j-1)$ can be calculated as $\alpha_{\phi(j-1)}(1-\alpha_{\phi(j)})$. For the first client in the scheduling sequence, the aggregation coefficient is $\alpha_{\phi(1)}(1-\alpha_{\phi(2)})(1-\alpha_{\phi(3)})\cdots(1-\alpha_{\phi(j)})$. Since α falls within the $(0,1)$ range, the aggregation coefficient diminishes over time.

B. Baseline

To achieve comparable learning performance to SFL, AFL needs to adopt the same client scheduling strategy and aggregation coefficient. In AFL, a client is scheduled to upload its model again only when all other clients have finished uploading theirs. Additionally, faster clients are prioritized in the scheduling, allowing them to upload, while slower clients still perform computations. As for the aggregation weight β, it varies in each global iteration. To ensure that clients contribute the same as in SFL, the aggregation weight β should also be calculated according to the contribution of each client m. This relationship is formulated in the following equation:

$$\sum_{m=1}^{M} \alpha_m w^m = w_{M+1} = \beta_M w_M + (1-\beta_M) w^{\phi(M)}. \tag{7}$$

In Equation (7), the left-hand side (LHS) corresponds to the global model after aggregation in SFL, whereas the right-hand side (RHS) represents the global model after completing one iteration through all clients in AFL. By analyzing Equation (7), we can infer that β_j is associated with both the iteration j and the scheduled client $\phi(j)$.

$$\begin{aligned} w_M &= \beta_{M-1} w_{M-1} + (1-\beta_{M-1}) w^{\phi(M-1)} \\ &= \beta_{M-1}(\beta_{M-2} w_{M-2} + (1-\beta_{M-2}) w^{\phi(M-2)}) \\ &\quad + (1-\beta_{M-1}) w^{\phi(M-1)} \\ &= \beta_1(\ldots) + (1-\beta_1) w^{\phi(1)}. \end{aligned} \tag{8}$$

From Equation (8), when the client scheduling $\phi(1), \phi(2), \ldots, \phi(M)$ is predetermined, the only unknown parameters are $\beta_1, \beta_2, \ldots, \beta_M$. On the other hand, $\alpha_1, \alpha_2, \ldots, \alpha_M$ are known, and this knowledge allows us to formulate a set of M non-linear equations. Non-linear equations often possess multiple solutions. By examining Equation (7), we can deduce that client $\phi(M)$ is chosen during iteration M. The following equation

$$\alpha_{\phi(M)} = 1 - \beta_M, \tag{9}$$

is formulated. Given that $\alpha_{\phi(M)}$ is a known value, we can solve for β_M. By considering Equation (7) and Equation (8), it becomes apparent that

$$\alpha_{\phi(M-1)} = \beta_M(1-\beta_{M-1}). \tag{10}$$

Consequently, we can solve for β_{M-1}. Using this method, we can sequentially compute β_{M-2}, β_{M-3}, and so forth, until we decide β_1.

As outlined above, we have established a baseline for AFL to achieve comparable learning performance to SFL. This baseline entails the following requirements: a) a client is scheduled again for upload only when all other clients have been scheduled once; b) client scheduling is predetermined before the learning process; and c) the global model is distributed to all clients every M iterations. However, requirement a) results in the under-utilization of faster clients' computational capabilities, which hampers AFL's full potential. Furthermore, requirement b) imposes a relatively strong assumption, necessitating the server's knowledge of each client's computational capabilities beforehand. Lastly, requirement c) entails that clients halt local learning or discard their local learning models in favor of the global model.

C. CSMAAFL: Client Scheduling and Model Aggregation in AFL

To fully leverage the advantages offered by AFL, we propose a client scheduling approach incorporating a model aggregation scheme. This scheduling method considers both clients' computational capabilities and the principle of fairness. When a client completes its local computation, it requests a time slot for uploading its updated local model. Upon server authorization of the request, the client transmits its local model and the estimated computational capacity to the server. Subsequently, the server undertakes global model aggregation and returns the aggregated model exclusively to the client who recently uploaded its local model. Furthermore, the server allocates more local iterations to clients with more computational capabilities, while clients with limited computational resources are assigned fewer local iterations. The server learns the computational capabilities of clients after the clients report. And then set the slowest client as the iteration basis. Subsequently, upon receiving the most recent aggregated global model, the client proceeds with learning for the next iteration. A slotted ALOHA protocol as in [10] is employed. Priority is given to the client with the older model when two clients complete their local computations simultaneously and apply for an uploading time

slot. If clients m and n complete their local computations at the current time and intend to upload their updated local models during time slot k, and if the previous upload slots for clients m and n are labeled as m' and n' respectively, then client m will receive priority if $(k - m') > (k - n')$.

The client scheduling approach described above is practical when there is relatively slight variation in the computation capabilities among clients. However, two extreme scenarios need to be considered. The first scenario arises when there are a few extremely fast clients, potentially operating at significantly accelerated speeds (e.g., 10 times faster). The second scenario occurs when there are some excessively slow clients. To ensure that all clients have a fair opportunity to contribute to the global model, we employ a policy similar to the one outlined in [5]. This policy allows clients with greater computation capabilities to perform more local iterations, dedicating more time to the learning process. Conversely, clients with lower computation capabilities perform fewer local iterations, enabling them to spend less time on the task. By adopting this approach, we can maintain a balanced contribution from all clients, regardless of their varying computation speeds.

The client scheduling problem has been addressed, ensuring that each client has an equitable opportunity to upload their local updates. It also needs to be addressed how to balance the current global model and the uploaded local models in each iteration. In equation (3), it can be observed that the contribution of client m to the global model diminishes over time. Furthermore, the difference between the current iteration and the iteration when client m last uploaded its model, denoted as $j-i$, also influences the process. A smaller value of $j - i$ indicates a lower level of staleness. To account for this, we introduce the moving average μ_{ji} to capture the average value of $j - i$ over time. Let

$$(1 - \beta_j)w_i^m = \min(1, \frac{\mu_{ji}}{\gamma j(j - i)})w_i^m, \tag{11}$$

for equation (3), where γ is a positive constant value. The term $\frac{1}{j}$ reflects the gradual decrease in the contribution of individual client models over time. The effect of staleness is represented by $\frac{\mu_{ji}}{j-i}$. When the learning starting point i of a client m is recent (i.e. when $j-i$ is small), the value of $\frac{\mu_{ji}}{j-i}$ is large, indicating a significant contribution from the individual client. As previously mentioned, extremely fast or slow clients are instructed to perform more or fewer local computations during their learning process, ensuring that every client has a comparable opportunity to access the channel for uploading the updated local models. This approach results in only slight changes in $j - i$, leading to the value of $\frac{\mu_{ji}}{j-i}$ close to 1. This helps maintain the stability in the system while accounting for staleness's effects. The complete algorithm is summarized in Algorithm 1.

Algorithm 1 Asynchronous Federated Learning with Client Scheduling and Model Aggregation

1: **Initialization: Server** initializes w_0 and broadcasts to all **Clients**.
2: **while** not converge **do**
3: **Client:**
Receives the most recent aggregated global model.
Performs local computation as Eq. (4).
Applies for uploading time slot.
Upload the calculated local model and estimated local computational capability when the request is approved.
4: **Server:**
Approves the first client m requested the time slot.
Receives the local model and computational capability from client m.
Performs aggregation by Eq. (7) and Eq. (11).
Sends the aggregated global model and the number of local computation iterations to client m.
5: **end while**

IV. Simulation Results

In this section, we begin by outlining the simulation settings. Subsequently, we present the simulation results for MNIST and Fashion-MNIST datasets in both IID and non-IID cases. Finally, we analyze and discuss the impact of the constant γ on the results.

A typical FL setting is considered for simulation here. The setup involves 100 clients connected to the server. In the case of SFL, all clients participate in the learning process during each round. However, for AFL, a client only waits for its next upload when all other clients have completed their current uploads. To simulate the heterogeneity in clients' computation capabilities, client selection is randomized at each time, corresponding to the round time in SFL. The communication is assumed to be uniform for each client. Consequently, this random selection affects the values of $j - i$ and μ_{ji}. Two public image datasets, i.e., MNIST and Fashion-MNIST, are used for simulation. MNIST consists of handwritten digit images, while Fashion-MNIST comprises images of Zalando's articles. Both datasets feature 10 classes, with 60,000 training and 10,000 testing images. Under the IID case, the images are randomly allocated equally among the clients. However, in the non-IID case, each client is assigned two classes, resulting in approximately 600 training images per client. We employ Convolutional Neural Networks (CNN) for the machine learning tasks with two convolutional layers, two max-pooling layers, and two fully connected layers. Given the complexity of the Fashion-MNIST images, the hidden layer sizes in the CNN for Fashion-MNIST are larger. The activation function for the last layer is the log softmax function, while ReLU is used in other layers. The learning rate η is 0.01, and the local batch size is 5. The constant γ in equation (11) can be considered a hyperparameter. A larger γ value leads to smaller contributions from individual client models. To investigate the effect of γ, we set its value as 0.1, 0.2, 0.4, and 0.6, respectively.

Four simulation cases are considered, incorporating two datasets, MNIST and Fashion-MNIST, and two data distributions: IID and non-IID. In each case, we conduct simulations using both SFL and AFL approaches. The classical FedAvg

algorithm is employed for SFL simulations, while our proposed CSMAAFL scheme is utilized for AFL simulations.

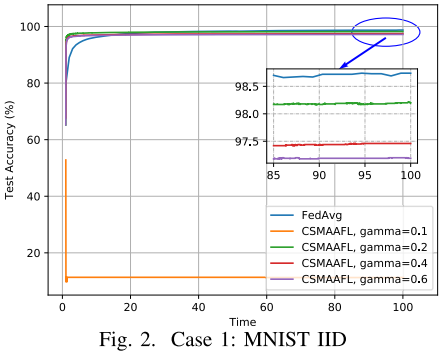


Fig. 2. Case 1: MNIST IID

In Fig. 2, the MNIST dataset with an IID data distribution is used. All schemes, except for CSMAAFL with $\gamma = 0.1$, demonstrate comparable performance. This indicates that our proposed CSMAAFL approach converges and reaches a similar outcome as FedAvg when the value of γ is tuned correctly.

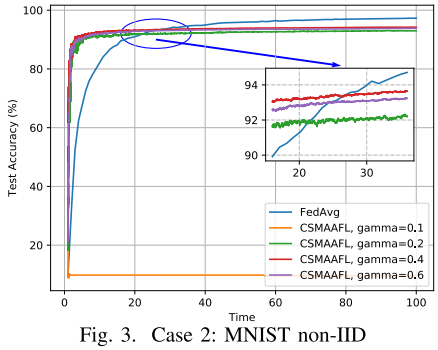
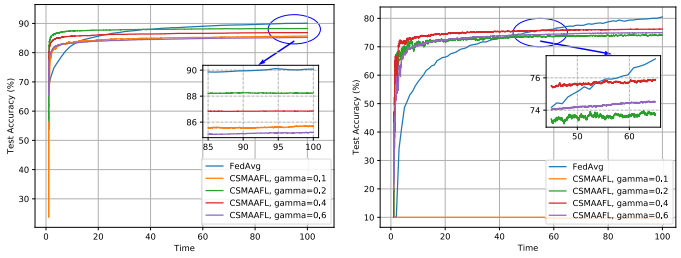


Fig. 3. Case 2: MNIST non-IID

In Fig. 3, the simulation is performed on the MNIST dataset with a non-IID data distribution. After 25 relative time slots, the FedAvg algorithm is starting to approach the performance of CSMAAFL, which indicates the proposed algorithms' faster convergence advantage.



(a) Case 3: Fashion-MNIST IID (b) Case 4: Fashion-MNIST non-IID

Fig. 4. Fashion-MNIST

The results for the Fashion-MNIST dataset with IID distribution are depicted in Fig. 4(a). Among the different γ values, CSMAAFL with $\gamma = 0.2$ exhibits the best performance, closely resembling the performance of the FedAvg algorithm.

In Fig. 4(b), it can be observed that CSMAAFL with $\gamma = 0.6$ achieves a performance that closely matches that of FedAvg. However, it takes FedAvg 55 relative time slots to reach the same performance level as our proposed CSMAAFL scheme. This result demonstrates that our CSMAAFL scheme accelerates the learning performance during the initial stages while maintaining the overall learning performance.

The constant value for γ in different scenarios leads to varying effects. In the case of MNIST IID, MNIST non-IID, and Fashion-MNIST non-IID, a value of $\gamma = 0.1$ results in random guessing. This occurs because the contribution of the individual client model is overly emphasized. On the other hand, for MNIST IID and Fashion-MNIST IID, the best performance is achieved with $\gamma = 0.2$, while for MNIST non-IID and Fashion-MNIST non-IID, the optimal results are obtained with $\gamma = 0.4$. By tuning γ, better learning performance can be achieved.

V. Conclusions

In this study, we introduced a client scheduling and model aggregation scheme for asynchronous federated learning. Our approach considered both the computation capability and fairness of the clients in the scheduling process while also addressing the issues of individual client contribution and model staleness in model aggregation. The results demonstrated that the proposed scheme can accelerate the federated learning process during the initial stages while still achieving comparable performance to the synchronous algorithm.

VI. Acknowledgement

This work was partially supported by the National Science Foundation under grants CNS-2008145, CNS-2007995, CNS-2319486, CNS-2319487.

References

[1] J. Konečný, H.B. McMahan, F.X. Yu, P. Richtárik, A.T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," 2019. [Online]. Available: arXiv:1908.07873.

[3] H.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*, pp. 1273-1282, PMLR, 2017.

[4] T. Nishio, and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," 2018. [Online]. Available: arXiv:1804.08333.

[5] S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, and K. Chan"Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Select. Areas Commun.*, vol. 37, no.7, pp. 1205-1221, Mar. 2019.

[6] M.R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, "Asynchronous federated learning for geospatial applications," in *ECML PKDD 2018 Workshops*, pp. 21-28.

[7] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *IEEE Big Data*, pp. 15-24, 2020.

[8] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," 2019. [Online]. Available: arXiv:1903.03934.

[9] Q. Wang, Q. Yang, S. He, Z. Shui, and J. Chen, "AsyncFedED: Asynchronous federated learning with euclidean distance based adaptive weight aggregation," 2022. [Online]. Available: arXiv:2205.13797.

[10] Z. Wang, Z. Zhang, Y. Tian, Q. Yang, H. Shan, W. Wang, and T.Q. Quek, "Asynchronous federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 6961-6978, Mar. 2022.