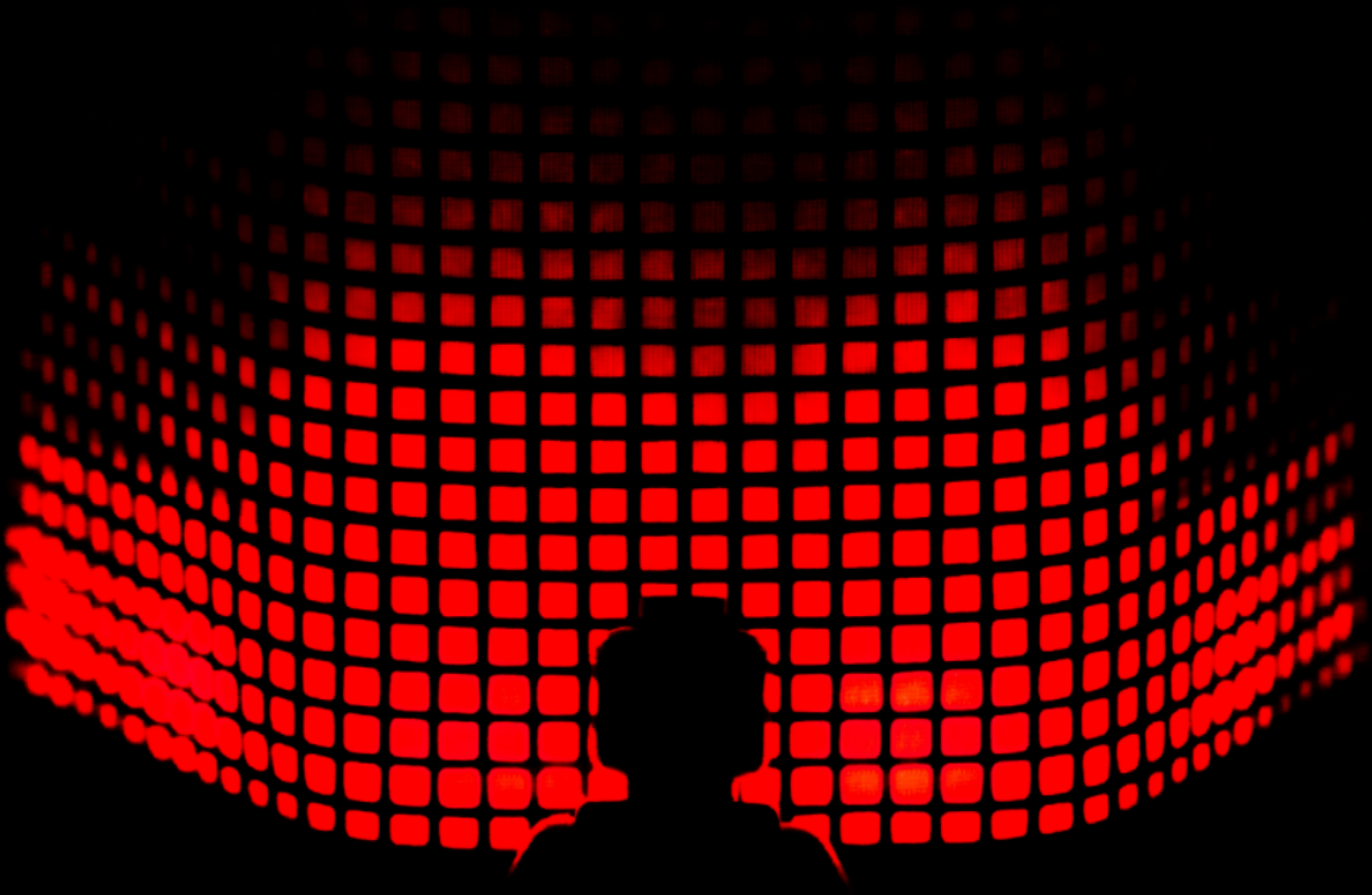
Multi-server Asynchronous Federated Learning

Master's Thesis

Yuncong Zuo

Delft University of Technology



TU Delft

Multi-server Asynchronous Federated Learning

by

Yuncong Zuo

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 23rd, 2023 at 10:00 AM.

Student number:	5443857
Programme:	Embedded Systems
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science
Project duration:	November 11th, 2022 – August 23rd, 2023
Thesis committee:	Dr. Lydia Chen, TU Delft, Supervisor
	Dr. Jérémie Decouchant, TU Delft, Supervisor
	Dr. Qing Wang, TU Delft

Preface

In recent years, federated learning (FL) has gained paramount importance, finding applications across diverse fields such as mobile applications, healthcare, and autonomous vehicles. It harnesses the strengths of machine learning by capitalizing on the expansive distributed data residing on clients' devices while safeguarding the privacy of clients' raw data from external entities.

Privacy, performance, and efficiency are three pivotal domains commanding researchers' attention. In this study, our primary focus revolves around enhancing the efficiency of FL systems, balancing the delicate trade-off between diminished performance and reduced execution time. The evolutionary journey of FL, extending beyond the conventional FedAvg algorithm, encompasses a spectrum of solutions including asynchronous FL and multi-server FL.

The practical deployment of algorithms is intricately intertwined with the substantial impact of network and computation delays on system efficiency. Many applications of FL are set in a landscape characterized by disparate client-server computing power and geographical distribution, underscoring the significance of network communication latencies and varying local training times. While asynchronous strategies liberate the system from waiting for tardy clients, they are relatively less tolerant to extended geographical communication due to their non-geographically replicable nature compared to multi-server alternatives.

In this paper, we introduce MultiAsync, a novel methodology that bridges the benefits of geo-replication in multi-server systems with the conventional asynchronous FL paradigm. By strategically assigning clients to servers based on their geographical locations, MultiAsync orchestrates asynchronous updates aggregation while enabling model exchanges among servers. This server-to-server interaction eliminates synchronization latency and accentuates global asynchronism. Furthermore, for scenarios of minimal internet delays among servers, a synchronous server model exchange approach is offered, providing an alternative that judiciously balances synchronization gains against marginal accuracy loss.

In the evaluation, we compared our algorithm against three canonical benchmarks—FedAvg as a synchronous solution, FedAsync as an asynchronous contender, and HierFAVG as a multi-server exemplar. Through meticulous emulation of network and computation delays based on empirical measurements and real-world network characteristics, MultiAsync emerged as the swiftest converging algorithm. Notably, its performance demonstrated commendable proximity to the baseline methods concerning the cumulative update count, affirming the algorithm's compelling efficiency gains even in light of the negligible trade-off in accuracy.

Thanks to the insightful guidance and attentive supervision from Prof. Jérémie Decouchant and Prof. Lydia Chen, and PhD candidate Bart Cox. Their collective expertise in the field has imbued this article with greater depth and substance. Thanks to my boyfriend Han who supports me throughout the endeavour.

Yuncong Zuo
Delft, August 2023

Contents

1
Research Paper

Multi-Server Asynchronous Federated Learning

Abstract—In federated learning (FL) systems, a server maintains a global model trained by a set of clients based on their local datasets. Conventional synchronous FL systems are very sensitive to system heterogeneity since the server needs to wait for the slowest clients in each round. Asynchronous FL partially addresses this bottleneck by dealing with updates once they are received. But with a single server, the system performance would be influenced if the clients are located far from the server and require very high communication costs. Another issue in single-server settings is that the client scale is limited since the server can be overloaded with heavy communication and computation workload. Moreover, a crash on the central server is fatal to the single-server system. Multi-server FL reduces the average communication cost by decreasing the distance between servers and clients. However, the bottleneck brought by the slowest clients still exists in multi-server systems that preserve synchrony, such as Hierarchical FL. The approach we follow in this paper consists in replicating the server in a way that the global training process remains asynchronous. We propose MultiAsync, a novel asynchronous multi-server FL framework that aims to address the single-server and synchronous-system bottleneck.

Nomenclature

D	The total dataset
D_k	The sub-dataset on $Client_k$
d	The number of data in D
d_k	The number of data in D_k
m	The number of clients
n	The number of servers
W^t	The weight vector of the model
τ_k^t	The staleness of W_k^t
η_k	The learning rate of $Client_k$
x_k	The number of local updates contributed by $Client_k$
P	The period of synchronization among servers
T	The number of global rounds
T_k	The number of local epoch on $Client_k$
$c_{process}$	The estimated duration of $process$

I. Introduction

FL is a decentralized approach to machine learning proposed by McMahan et al. in 2017 [1]. It aims to make good use of the vast amount of data distributed among a large number of users to train a model in a privacy-preserving way. It was initially introduced to improve privacy preservation in machine learning that requires data from edge devices [2]–[6]. Now it is also developed to train a shared model among some large organizations such as finance companies and health care centers [7]–[9]. The two corresponding scenarios are termed cross-device and cross-silo FL [10]. FL can also be categorized as horizontal and vertical FL. In vertical FL, clients don't share the same feature space and have unique features. They collaboratively train the predictor to have a more comprehensive knowledge of all the features. In this paper, we focus on cross-device and horizontal FL.

Different from conventional machine learning where the training is completed on a single machine with a single dataset, FL requires local training happening in different machines in parallel and with different local datasets. The most commonly used framework is the centralized and synchronous FL framework [1]. In each round, a set of clients are selected to receive the current global model from the server, train the model with their local dataset, and send the update back to the server. The server collects all the updates and computes a new model according to the aggregation rule of the FL system and applies the new aggregated model to its global state. The process is repeated in rounds until the global model converges. Figure 1 gives a general view of FL applications.

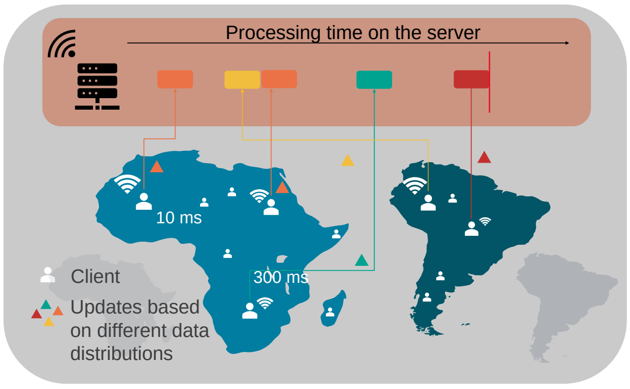

Processing time on the server
10 ms
300 ms
Client
Updates based on different data distributions

Figure 1: An FL round over heterogeneous devices and networks.

FL is a significant improvement over conventional machine learning because it does not require collecting all data onto a single centralized server, which provides stronger privacy guarantees for users who contribute their data. As shown in Figure 1, only the trained models are sent to the server without revealing clients' local data. As a similar counterpart, distributed learning also requires a set of clients to participate in training one model without collecting all data on the server. But it focuses on distributing the training workload to decentralized clients and is developed towards the benefit of parallel training. It has the assumption of independent and identically distributed (IID) datasets. The advantage of FL over distributed learning, as we illustrate in Figure 1, is that different data distributions can exist in clients, e.g. different types of images stored in different people's smartphones, which is a more realistic setting in real-life applications. In other words, FL can train a model with non-IID data distributed to different clients. Dealing with non-IID data is more difficult, especially in the context of distributed and private datasets. Two main goals of developing FL are achieving higher efficiency in the training process and higher accuracy of the resulting model.

Except for statistical heterogeneity in clients, resource heterogeneity is another challenge in FL. Multiple devices conduct local training and contribute to one global model, and they have varying computing power and communication delays. In a synchronous FL framework, the duration of each round is bounded by the slowest client whose update is the last to be received by the server. A larger heterogeneity in devices' computing power leads to a larger amount of idle time on both the server and fast clients in terms of the training process. Asynchronous FL [11]–[13] and aim at addressing this performance bottleneck. Asynchronous FL improves the system to deal with local updates without waiting for all the selected clients and thus greatly reduces the idle time on the server. However, long-distance communication now exposes as a major influence on overall performance. Moreover, the limited computational power on a single server may cause queuing of the updates of clients.

To address the remaining problems, Hierarchical FL [14]–[16], introduces multiple servers and assigns each with a set of clients. It can reduce the average communication costs since clients upload the model to its nearest server. Also, with more computational power on the server side, fewer updates from clients will be pending. However, due to its synchronous execution, the time of each round is still bounded by the slowest clients.

FL is applied in many applications, including mobile keyboard prediction [17], spoken language understanding [18], healthcare [19], and so on. These applications require a large amount of user or patient data. In applications such as next-word prediction and spoken language understanding tasks, vast data from users over the world can be utilized well. In this case, where clients are distributed on a large scale physically, the network heterogeneity introduces a bottleneck to the conventional synchronous FL systems. In synchronous cloud-based FL, one centralized cloud server is connected to all the clients in the system. With a large number of clients in the system, the server needs to handle a heavy load of model computing and frequent communications, which makes the system more sensitive to the number of clients and the computing power of the server. The limitation of a single centralized cloud server also reflects on the low tolerance of a single point of failure on the server. Hierarchical FL addresses the bottleneck led by the server's heavy computing and communication load and improves the scalability of the number of clients. However, existing hierarchical FL frameworks client_edge_cloud , [15] have not addressed the bottleneck brought by synchronous algorithms and cannot tolerate the crash on the central server.

In this paper, we introduce MultiAsync, a novel framework for multi-server Asynchronous FL systems. The average staleness led by asynchrony is reduced by distributing the local updates to multiple servers so that high training accuracy can be maintained. Efficiency is also obtained thanks to parallel computing in servers.

The remaining sections are organized as follows:

1) In section II, we provide background and related work, including the most commonly used synchronous frameworks, asynchronous FL algorithms, and hierarchical FL algorithms.
2) In section III, we describe the system model of our method as well as the centralized system and multi-server system.
3) In section IV, we introduce our method MultiAsync in detail, including the local update and global model exchange strategies. MultiSync provides a synchronous version of global model exchange, while MultiAsync moves to a fully asynchronous system.
4) In section V, we provide our performance evaluation. We compared our method with baselines: FedAvg, FedAsync, and HierFAVG. We also explored the impact of the number of clients and servers in the system.
5) In section VI, the conclusion and future work are given.

II. Related Work

In this section, we discuss the related background and previous works. We first introduce the readers to the conventional and most commonly known federated system frameworks and move to asynchronous and hierarchical FL systems.

A. Synchronous Federated Learning

FL was first proposed by McMahan et al. in 2017 [1]. The topology includes one central server and many clients that connect with the server. They proposed the first synchronous single-server FL algorithms FedAvg and FedSgd. In each round, a group of clients is selected to receive the global model from the server and perform local training. After Client k receives a global model W^t, it conducts local training on their local dataset D_k as follows:

$$W_k^{t+1} = W_k^t - \eta_k \frac{\partial F(W^t, D_k)}{\partial W^t}, \tag{1}$$

where F is the loss function of the classification, and η_k is the learning rate.

At the end of the round, the server collects the local updates and aggregates them according to the following equation:

$$W^{t+1} = \sum_{k=1}^{K} \frac{n_k}{N} W_k^t, \tag{2}$$

where $\frac{n_k}{N}$ is its fraction of data points included in this aggregation since the influence of a model greatly relies on the data it is trained on. W^{t+1} is then broadcast to the selected group of clients in the next round and the process is repeated until the global model converges. These fundamental algorithms converge eventually but may suffer from the high delay of the slowest clients.

Chai et al. proposed TiFL, a tier-based FL framework to reduce the influence of resource heterogeneity in clients [20]. In TiFL, the selection of clients is adaptive to their computing power. Initially, the central server runs a profiling algorithm to collect the latency metrics of all the available clients based on which the clients are divided into groups called tiers. Different from the conventional FL systems where clients are randomly selected, TiFL randomly selects the clients from one tier in each round. The stragglers' negative effect on the training efficiency of a single round is significantly reduced. However, the algorithm still needs to wait for the slowest tier to complete its round in order to proceed.

Mobile edge computing is a practical scenario for FL. For smartphone clients, except for computing power, resource heterogeneity often appears as different wireless channel conditions. If some clients lose connection in the middle of a round, it is unpredictable how long the system has to wait. Li et al. proposed a hierarchical pace control FL framework SmartPC which focuses on production FL in real-time settings [16]. They consider the cross-device FL and conduct experiments with Android smartphones. SmartPC aims to balance the model accuracy, training time, and energy consumption of clients' devices. Similar to another client selection solution called FedCS [21], its main idea is to assign deadlines to bound the waiting time. In SmartPC, before every round, the central server evaluates the status of the selected devices and determines a global deadline. Then the local layer running on each device will adjust the system configuration of the device to meet the deadline with the least energy consumption. The global deadline is determined based on a feedback mechanism that ensures a specific percentage of clients are able to meet the deadline. This method ensures a round to terminate on time, but some important training results from certain clients might be discarded during the system configuration adjustment.

Cox et al. proposed Aergia, an FL framework that leverages model offloading to enhance the training efficiency [22]. The federator evaluates and profiles the performance of selected clients and matches the weak with the strong ones. The matching algorithm also considers the feasibility of offloading by computing the pair-wise similarity of the two clients' data inside a trusted execution environment (TEE). Once the schedule is decided by the federator and known by the related clients, the weak ones freeze part of the model and send it to their matched strong clients. The heterogeneity of training phases is also measured and discussed in their work. The most time-consuming phase is chosen to be frozen and offloaded.

B. Asynchronous Federated Learning

Asynchronous systems could provide faster convergence. With the heterogeneity in clients' computing power and communication delay, the interval time between the server sending out the new global model and receiving the model from each client varies. In synchronous FL systems, large heterogeneity would profoundly reduce the efficiency of the training process, since both the server and the fast clients need to wait for the slow client updates to come before entering the aggregation phase. Asynchronous systems allow clients to follow their own speed when participating in the training. The server does not wait for the slow clients but aggregates the arrived client updates immediately. In this section, we introduce some existing solutions in asynchronous FL.

In conventional FL systems, the resource heterogeneity can significantly prolong the time in each round while the data heterogeneity brings an upper bound for the result accuracy. As FL is broadly used by edge device training, global synchronization encounters a great challenge due to the varying computing power and infrequent communication. Asynchronous FL overcomes the impact of resource heterogeneity by aggregating an update immediately when it is received by the server. The asynchronous version of FedAvg can be described as

$$W^{t+1} = W^t - s(\tau)\frac{n_k}{N}(W_k^t - W_k^{t+1}) \qquad (3)$$

where W^t is the global model with timestamp t, W_t^k is the local update of client k with timestamp t, $\frac{n_k}{N}$ is the data proportion of the update, and $s(\tau)$ is a staleness parameter which we describe in the following section.

Chen et al. proposed ASO-Fed, an asynchronous online FL framework that enables wait-free computation and communication [13]. ASO-Fed allows client updates to stream into the federator in different rounds and each local training is based on the newly streamed-in data. A decay coefficient is introduced to adjust the training result according to the last model trained on previous data. And a central feature representation learning is used to reduce the influence of asynchronous aggregation. To address the staleness problem, ASO-Fed scales the learning step size by the client's communication delays. Since clients with long delays lose their impact on the intermediate models, a larger step size is assigned to compensate for the loss.

Xie et al. proposed FedAsync, an asynchronous optimization algorithm for FL [11]. They guarantee a near-linear convergence and address the staleness problem in the asynchronous setting. Each time an update arrives, the weighted averaging is applied for the current global model and the update. The weight of the client decreases when its staleness increases. The global model is updated by the equation

$$W^{t+1} = W^t - \frac{n_k}{N}r_k^t(W_k^t - W_k^{t+1}), \qquad (4)$$

where $r_k^t = max\{1, \log \bar{d_t^k}\}$, and $\bar{d_t^k} = \frac{1}{t}\sum_{\tau=1}^{t} d_k^{\tau}$ is the average computing time of client k in past rounds.

Wang et al. proposed AsyncFedED, an adaptive asynchronous aggregation based on Euclidean distance [12]. If the global model is updated before a client sends back its update, the learning rate of the server should be adjusted based on the staleness of the current update. AsyncFedED evaluates the staleness of an update by its Euclidean distance from the server. A larger staleness leads to a smaller learning rate for its aggregation. The staleness of update w_k^{t+1} is defined as

$$\gamma_k^{t+1} = \frac{\|W_k^t - W_k^{t+1}\|}{\|\triangle_i W_k^{t+1}\|} \tag{5}$$

The global learning rate of the current update is calculated by

$$\eta_{k+1}^t = \frac{\lambda}{\gamma_k^{t+1} + \epsilon,} \tag{6}$$

where λ and ϵ are hyperparameters to be tuned.

In asynchronous FL, the number of updates each client contributes depends on its computing power and on its network conditions. Due to the asynchrony, each update's influence on the global model should be adjusted. From the previous asynchronous FL algorithms, we see that staleness is one of the most important elements to consider when aggregating asynchronous updates. Staleness is the number of interval updates between the current model and the last model that a client has received. The number of updates received and aggregated by a server is defined as the age of the server, which is also sent along as a timestamp when global model delivery occurs.

Staleness is calculated by $\tau = t - t_k$, where t is the current age of the server, and t_k is the timestamp indicating when Client k received the global model for the current local training. In the interval of the model delivery and receiving of Client k, other clients have updated the global model W^{t_k} to the new stage W^t. When the server receives the update W_k from Client k upon the previous model $W^{t-\tau}$, the influence adjustment should be based on the number of intervals updates τ. Different strategies of staleness-weighted aggregation have been proposed. FedAsync [11] introduces a staleness strategy where the influence of a client's update decreases when its staleness grows. Different weighting functions are proposed to calculate the weight $s(\tau)$. In this paper, we only refer to the following polynomial weighting function

$$s(\tau) = (\tau + 1)^{-\alpha}, \tag{7}$$

where α is an adaptive parameter that decides the influence of staleness. A larger α indicates the more the influence of stale updates should be reduced.

FedAsync performs best when the staleness is small, and it can converge faster than FedAvg. In the paper, a max staleness of 4 and 16 is tested. However, the average staleness is decided by the number of clients in the system. When there are N clients in the system, each update would contribute to $N-1$ staleness in total since every update is pushing the model to another stage. The average staleness of each aggregation is $N-1$. In a larger system such as one with 100 clients, the system has to tolerate the average staleness of 99. By leveraging multiple servers in the system, we can reduce the average staleness of local updates and improve training efficiency. When staleness becomes extremely large, the influence of these ancient updates would be greatly reduced according to Equation 7. Additionally, with a single server in the system, high communication latency that is caused by clients far away from the server can dramatically slow down the training process.

C. Multi-server Federated Learning

Another scheme to address resource heterogeneity is to spread the computation and communication burdens instead of bearing them all on a single server. And the distinguished global models maintained at multiple servers also facilitate scenarios where personalized models are required. Existing methods [14], [15], [23]–[25] add some edge servers or assign proxy clients to undertake part of the workload. In addition, clustering techniques become very valuable in multi-server contexts as they help organize data more efficiently and identify inherent patterns, promoting better collaboration among servers. Clustering is frequently used in multi-task and unsupervised FL [26]–[28]. We put it into future work to incorporate clustering algorithms in our framework for potential refinement.

Xie et al. proposed a multi-center FL algorithm, FeSEM [29], to address the challenge of data heterogeneity. The main idea is to cluster the updates and assign them to their closest global model (i.e. center). They showed in experiments that the clusters of local updates characterize clients' data distribution thus models generated from similar data distributions are gathered on the same center and aggregated.

Hierarchical FL represents a novel approach that improves the capabilities of multi-server FL [14]–[16], [25]. In hierarchical FL, there is a leader server and multiple edge servers. Two levels of aggregations happen in each round. The first level happens at the edge server which aggregates a group of clients and sends the update to the leader. The leader executes the second-level aggregation and generates the global mode for the new round.

L. Liu introduced the client-edge-cloud hierarchical framework, HierFAVG, which can be considered as a multi-server version of the FedAvg algorithm. Each edge server employs averaging aggregation for its clients. After every certain number of rounds, the cloud server applies averaging strategy to the edge servers. The introduction of the hierarchical structure reduces the communication burden on servers and improves the capability of the single-server FedAvg algorithm while maintaining the stability of synchronous systems.

Despite the advantages, current multi-server and hierarchical-server algorithms do not directly address the commonly existing drawback of synchronous systems, which is their sensitivity to large system heterogeneity. This gives the motivation to our MultiAsync method: a combination of multi-server and asynchronous FL that deals with all heterogeneity at the same time.

III. System Model

We consider a system with multiple servers and a group of clients. Compared to a hierarchical system, it is a decentralized system without a central cloud server. Figure 2 illustrates the topology of centralized, hierarchical, and multi-server systems. The centralized system contains one server which is connected to all the clients in the network. In hierarchical systems, a centralized server at the highest level is connected to the second-level servers, which are connected to separate clusters of clients in the system. In the multi-server system which is used in our algorithm, each server is the center of a star network, with a group of clients connecting to it. And servers are able to communicate with each other. With m clients in the system, one round of local updates costs at least $O(m)$ communication contributed by model receiving and delivery. In multi-server settings, this communication workload is spread to n servers. The system could bear $(m-1)n_{max}$ more clients if n_{max} is the maximum number of clients a server could tolerate.

IV. MultiAsync

This section introduces our method MultiAsync. The multiple servers are distributed in a flat instead of hierarchical structure, thus no central cloud server is required. Each server is assigned a group of clients based on geographical proximity, which is a key feature for reducing communication overhead. Locally, clients continuously retrieve the latest global model from the server, train it on its local dataset, and send back the update to the server. Globally, the exchange of server models is essential in harnessing the power of distributed data and computing resources across multiple servers. In our main algorithm MultiAsync, global model exchange happens in an asynchronous fashion. We introduce a token-based strategy to allow servers to exchange their models asynchronously. The model exchange can only be initiated by the server that holds the token, which prevents concurrent and redundant broadcasting. Model aggregation happens peer to peer and asynchronously, which leads to different global model states after exchanging. We also introduce a synchronous version, MultiSync, where global synchronization is initiated by time and happens periodically. Servers wait for each others' global model and obtain the same global model state after averaging. It provides better system consistency, reducing potential discrepancies in the model state but its effectiveness may be impacted by network heterogeneity.

In multi-server settings, the strategy of exchanging models among servers significantly influences the performance of the system for two reasons: (a) Data heterogeneity: each server wants to prevent its model from being biased toward its own group of clients. The exchange of global models gets good use of the vast data distributed to different clients. (b) Resource heterogeneity: due to the difference in clients' computing power and network conditions, the number of updates each server has handled can vary a lot. The model exchange allows advanced models cast their influence on others. In other words, in multi-server settings, a global model is not only updated by the clients of the server but it should also be improved by other servers' global models from time to time.

In the following sections, we will explain MultiAsync from a top-down view, and give a synchronous alternative, MultiSync, in the end.

A. Server-Server Aggregation

The server-server aggregation of MultiAsync consists of two main algorithms: a token-based algorithm for asynchronous server model exchange and an aggregation algorithm to merge two server models.

1) *Model Exchange Process:* Algorithm 1 is the pseudocode of the model exchange procedure in MultiAsync. Parameter h_{inter} is the threshold for the largest age difference between different servers. h_{intra} is the threshold for the age difference between a server's current age and the age after its last completed aggregation. Either excess of the two thresholds will initialize the model exchange.

Lines 1-8 describe how the token is distributed and triggers the model exchange process. The token is initialized with an empty hash map and a broadcast ID of 0. It starts in one server and travels to each server in a ring-based topology. Upon the arrival of the token, Server i updates its current age onto the hash map. It then checks the current largest age difference between servers and the age difference between its current age and its age after the last broadcast phase. If the inter-server age difference is over the threshold h_{inter}, or the intra-server age difference is over h_{intra}, it triggers Server i to initialize the server model exchange process by broadcasting its current model W_i^t, age A_i, and broadcast ID bid_t to all the other servers.

Lines 9-22 describe how models are exchanged among servers and how the token is redistributed by the server that initiated the exchange. Upon receiving a broadcast, Server j compares the bid_t with its own broadcast ID bid_j. In line 12, if $bid_t \geq bid_j$, the broadcast is the latest in Server j's knowledge so it should be disposed of and Server j should update its own broadcast ID bid_j as $bid-t$. Server j, in line 15, would then execute a server-server aggregation algorithm $serverAgg$. This algorithm will be explained in the next subsection. When Server j that initializes the algorithm has collected and aggregated all

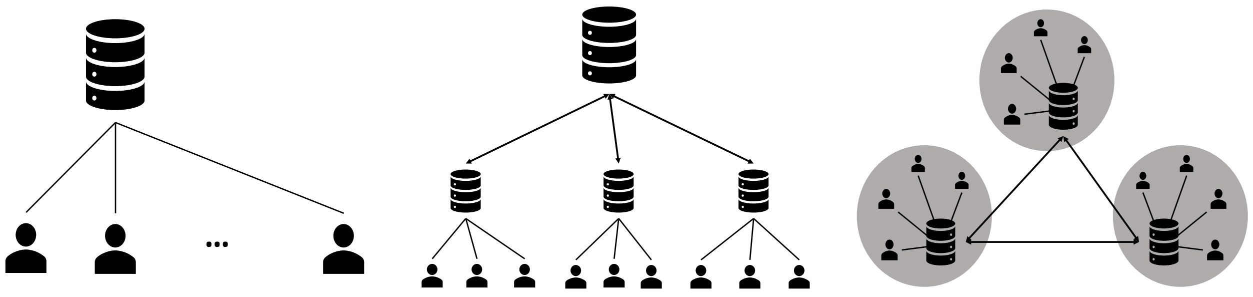

(a) Centralized system (b) Hierarchical system (c) Multi-server system

Figure 2: Layout of 3 different FL system models

the other servers' models, it increases the token broadcast ID bid_t by one, re-initializes the hash map, records its current age, and passes it to the next server. In line 10, $bid_t < bid_j$ happens when the broadcast bid_t takes too long to arrive due to a network delay, and it should be ignored since Server j has been updated by newer broadcasts. Concurrent initialization is avoided since the initializer server keeps the token during the algorithm running.

2) Model Aggregation: The aggregation algorithm 2 between servers consists of two parts: calculating the aggregation weights and updating the server models.

When deciding the weights, the number of data points on servers should be taken into consideration. The age of a server is defined by the number of local updates it has applied. When comparing a server's age with others at the same time, the number of data points these server covers are compared indirectly. Thus, we use the following weighted aggregation strategy to calculate the weights of W_j^t according to their relative age difference. The Sigmoid function is used to measure the difference in age taking the relative age difference as the variable.

$$w_i^t = \frac{1}{e^{-ka+1}}, \quad \text{where} \quad a = \frac{A_j - A_i}{A_i} \tag{8}$$

The age difference $A_j - A_i$ indicates how is W_j^t more mature than W_i^t in terms of the number of updates they are trained on, and the influence of W_j^t should increase when the difference grows. Denominator A_i makes the absolute weight relative to the current age of Server i. As the age increases, the model is in a more stable and mature phase of training and the learning ratio towards other servers should be influenced less by the absolute age difference. The Sigmoid function ensures the weight is between the range of 0 and 1. The derivative becomes 0 and results in a weight of 1 when the relative difference is too large. Parameter k indicates in which range the Sigmoid function is active. A larger k leads to a smaller active range, leaving a larger area for the weight of 0 and 1.

Algorithm 1 Token-based global model broadcast in MultiAsync

```
Require: Age difference threshold between servers $h_{inter}$, Age difference threshold within servers $h_{intra}$, Server i age $A_i$
 1: Server i Procedure Token Loop
 2:     Initialization: $ages_t \leftarrow \{\}, bid_t \leftarrow 0, bid_i \leftarrow 0, cnt_i \leftarrow \{\}, A_{i,pre} \leftarrow A_i$
 3:     Token arrives at Server $i$
 4:     $ages[i] \leftarrow A_i$
 5:     if $max(ages) - min(ages) >= h_{inter}$ or $A_i - A_{i,pre} >= h_{intra}$ then
 6:         Broadcast($W_i^t, bid_t, sender_id$)
 7:     else
 8:         Pass the token to the next server
 9: Server j Procedure Receive Broadcast($W_i^t, bid_t$)
10:     if $bid_t < bid_j$ then                ▷ the broadcast is stale
11:         Return
12:     if $bid_t \geq bid_j$ then             ▷ Latest broadcast arrives
13:         $bid_j \leftarrow bid_t$
14:         Broadcast($W_j^t, bid_t$)
15:     $A_j, W_j^{t+1} = serverAgg(W_i^t, A_i)$
16:     $cnt[bid_t] \leftarrow cnt[bid_t] + 1$
17:     if $cnt[bid_t] == \#servers - 1$ and Server $j$ is initializer then
18:         $bid_t \leftarrow bid_t + 1$     ▷ Prepare the next broadcast id
19:         $ages \leftarrow \{\}$
20:         $ages[j] \leftarrow A_j$
21:         $A_{j,pre} \leftarrow A_j$       ▷ Record age of last broadcast
22:         Pass the token to the next server
```

The aggregation rate η_a scales the influence of other servers in peer-peer aggregation. When the aggregation with Server i happens locally at Server j, a large η_a indicates that Server i could influence Server j's model greatly, and $\eta_a = 1$ means it could replace Server j's model when their ages are equal. η_a needs to be carefully

Algorithm 2 Server-server aggregation in MultiAsync

```
Require: Aggregation rate $\eta_a$, active rate $k$
1: Server i Procedure ServerAgg($W_j^t, A_j^t$)
2:     $a = \frac{A_j - A_i}{A_i}$
3:     $w_j = \frac{1}{e^{ka+1}}$
4:     $W_i^{t+1} \leftarrow W_i^t + \eta_a w_j (W_j^t - W_i^t)$
5:     $A_i \leftarrow (1 - \eta_a w_j) A_i + \eta_a w_j A_j$
```

tuned to ensure the exchange of models is efficient. If η_a is too large, the influence of the server itself is eliminated; if η_a is too small, the server learns too little from its peers thus its model could be biased to its clients' data distribution.

The server age should also change after model exchange and aggregation with other servers. According to the weight w_j that has been applied to the model aggregation, Server i updates its age locally as shown in the last line of Algorithm 2. Server-server aggregation contains more than one update's complexity, thus we apply the weighting strategy to age, indicating how many ages a server grows after considering another server's model.

B. Client-server Aggregation

In an asynchronous context where there is one server and a group of clients, the aggregation process is described in Algorithm 3. Line 1-8 describes the perspective of the client, it waits for the global model, trains it upon its local dataset, sends the trained model back to the server, and repeats the process. The $Decay(\cdot)$ in line 4 represents a decay function that will be explained in detail in section IV-C. Line 9-13 describes the perspective of the server, it receives an update from the client, aggregates it immediately, sends the updated model back to the client, and repeats the process. In MultiAsync and MultiSync, local learning rate decay is applied to limit the clients with more frequent updates. In our experimental study, it shows that the decay of the local learning rate reduces the distance of servers' global models indicated by a smaller standard deviation in the resulting accuracy.

C. Client Local Training

Due to the resource and network heterogeneity in clients, clients send their updates to the server at different paces. Fast clients are active contributors, consistently sending their updates at smaller intervals. In contrast, slow clients contribute infrequently, offering fewer updates. As a result of this disparity in the number of updates, the server faces challenges in handling the data distributions of all clients, primarily because of the flooding updates received from the fast clients. To address the difficulty of maintaining a balanced data distribution, adaptive strategies are needed to manage client updates. We introduce a customized local learning rate to avoid potential biases in the global model caused by clients' data heterogeneity. The main idea is to limit fast clients' influence on the

Algorithm 3 Client-server update in MultiAsync

```
Require: Client Learning rate $\eta$, server learning rate $\eta_i$,
    the number of local updates $x_k$
1: Client Procedure Local Training
2:     Receive global model $W_i^t$ from Server $i$
3:     Initialization: $W_k^t \leftarrow W_i^t$
4:     $\eta_k \leftarrow Decay(\eta_k)$
5:     for epoch e $\in [E]$ do
6:         Update $W_k^t$ with learning rate $\eta_k$
7:     $W_K^{t+1} \leftarrow W_k^t$
8:     Send the $W_k^{t+1}$ to Server $i$
9: Server Procedure Aggregation
10:    Receive model $W_k^{t+1}$ from Client $k$
11:    $w_k^t \leftarrow GetWeight(a_k, a_i)$
12:    $W_i^t \leftarrow W_i^{t-1} + \eta_i w_k^t$
13:    Send $W_i^t$ to Client $k$
```

server while finding a balance in the trade-off of the advantage of their high computing power.

Before local training, client k adjusts its local learning rate according to the following function:

$$\eta_k = \begin{cases} \eta_k, & x < \lambda \\ \max\left(\eta_{min}, \eta_k - \beta(x - \lambda)\right), & x \geq \lambda \end{cases} \tag{9}$$

where η_k is the learning rate without decay, η_{min} is the lower bound of the decayed learning rate, β is the decaying rate, and λ determines when the decay function is activated for each client. λ should be tuned according to the average number of updates each client should contribute. If λ is set to an excessively small value, the learning process may slow down too much, which can hinder the model's ability to explore and learn from the data of fast clients effectively. On the other hand, when λ is too large, the learning rate might not decrease enough to mitigate the impact of fast clients.

When greater data heterogeneity exists in clients, the system is more sensitive to resource heterogeneity in clients since the local optimums vary more and over-approaching any optimum affects the performance more seriously. With the implementation of the local learning rate decay, all clients have a more equitable opportunity to shape the global model, leading to a more comprehensive and accurate representation of the entire dataset.

D. Multisync

As a synchronous alternative for MultiAsync, MultiSync shares the same client-server aggregation and client-local training procedures. MultiSync is preferred in the situation where server-server communication latencies are compensated with better performance and stability. The server-server aggregation for MultiSync is a synchronization process, which is the main difference from MultiAsync. We introduce a periodic synchronous aggregation mechanism

Algorithm 4 Periodic global synchronization in MultiSync

```
Require: Period $p$
1: Server Procedure Aggregation
2:     Broadcast $W_i^t$ and $A_i$ with a period of $p$
3:     Receive models $\{W^t\}$ from all the servers
4:     for $W_j^t \in \{W^t\}$ do
5:         $w_k^t \leftarrow \frac{A_j}{\sum A}$
6:     $W^{t+1} \leftarrow \sum \frac{A_k}{A} W_j^t$
```

for the global models' aggregation. The time-based synchronization has the advantage of simple initiation since it reduces unnecessary messages for the servers to reach a consensus. Algorithm 4 shows the procedure of global synchronization. Periodically, the servers broadcast their model and age and wait for others' messages to arrive. As long as a server receives all the models from its peer, aggregation happens. A weight function of $w_k^t = \frac{A_j}{\sum A}$ is used to decide a server's influence in the synchronization, which is evaluated by the age of the servers. The server with a larger age will have a greater impact during the synchronization. By the end of the algorithm, each server should reach the same model.

In MultiSync, when the server synchronization algorithm is started, servers stop accepting local updates from clients but wait for all the other servers' models to arrive and synchronize. This introduces a common bottleneck of synchronous systems that they are more sensitive to network heterogeneity since the synchronization duration is strictly limited by the worst communication delay.

V. Performance Evaluation

In this section, we introduce the experimental settings and evaluation matrices. We conduct experiments on the MNIST, CIFAR-10, and WikiText2 datasets. In the experimental study, we explore the impact of the number of clients per server when the number of servers is stable as well as when the total number of clients is stable. We simulated the training delays on clients and the aggregation delays on the servers. We compared our methods with fedAvg [1] and fedAsync [11].

A. General Experiment Settings

The dataset is split into equal sizes and assigned to every client. To address clients' data heterogeneity, we assign l labels to each client. A smaller l indicates a more non-IID data distribution and thus larger data heterogeneity in clients. l is set to 2 in non-IID data experiments. To simulate the time, we consider the following delays:

In addition, we set the communication delays between clients and servers to 10 ms. Since servers might be located far from each other (e.g. Amsterdam and Sydney), we consider the communication delays among servers according to the Amazon Web Service network latency [30] as shown in table II.

Table I: Simulated computation delays

Local Training	200 ms
Global Synchronization in MultiSync	2 ms
Peer-to-peer Aggregation in MultiAsync	2 ms
FedAvg Aggregation	15 ms
HierFAVG Cloud Aggregation	15 ms
Aggregation in FedAsync	2 ms

Table II: Communication delays

Delays(ms)	Hongkong	Paris	Sydney	California
Hongkong	1.41	194.9	132.28	155.13
Paris	197.91	0.9	278.83	142.25
Sydney	132.06	280.11	2.56	138.47
California	154.96	142.79	138.57	2.14

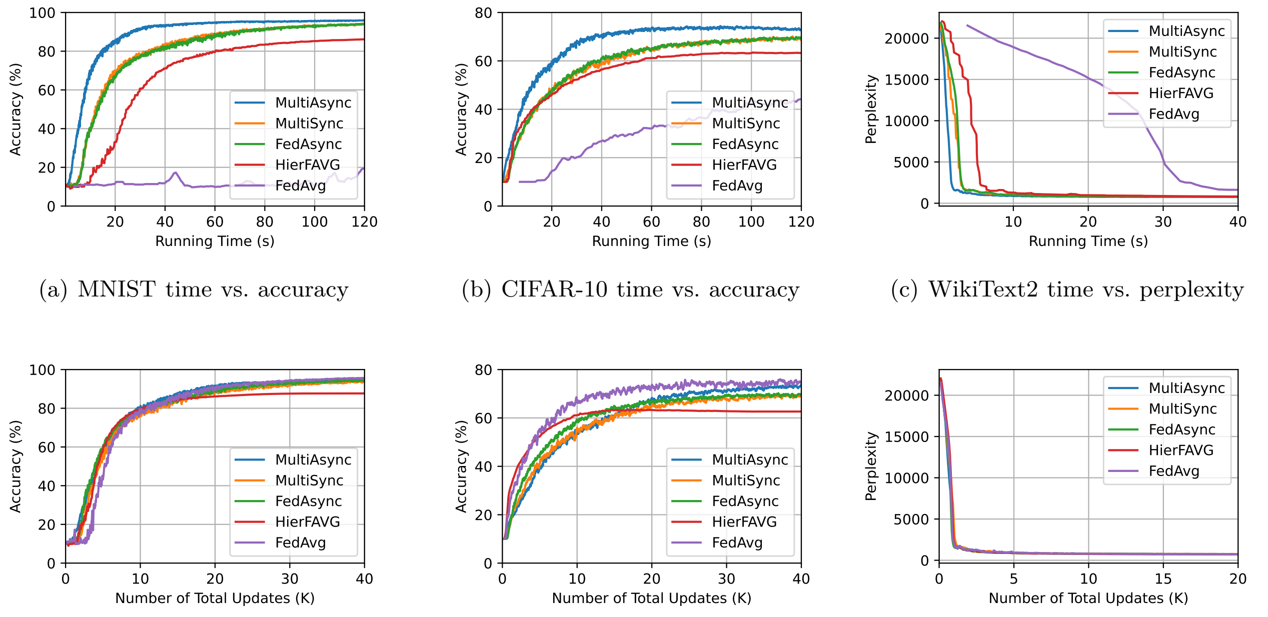
To simulate the resource heterogeneity in clients, we sample each client's training delay from a Gaussian distribution: $X \sim N(\mu, \sigma^2)$, where $\mu = 100$ and $\sigma = 0.4\mu = 40$. To avoid extra randomness in clients, all trials in the experiment share the same clients.

For the neural network models, we use a CNN model with two convolutional layers and two fully connected layers to perform MNIST classification. For CIFAR-10, a CNN model with three convolutional layers and two fully connected layers is used. For the text dataset WikiText2, we use the next character long short-term memory (LSTM) neural network model, designed for character-level text generation tasks. The language model consists of an embedding layer that is used to capture the semantic and syntactic properties of characters, an LSTM layer that captures dependencies between characters and generates coherent text, and a fully connected layer that transforms the LSTM's hidden states into a probability distribution over all possible characters in the vocabulary. The initial local learning rate of clients η_k is 0.05. In Asynchronous settings, we use $\alpha = 0.5$ for the staleness weighting function 7 and a global learning rate of $\eta = 0.6$ for the client-server update 3.

B. Performance Comparison of 5 FL Algorithms

In this experiment, each FL system consists of 100 clients. Figure 3 shows the results of experiments on datasets MNIST, CIFAR-10, and WikiText-2. In terms of time, for each dataset, MultiAsync converges the fastest in terms of time, MultiSync has the same efficiency as FedAsync that together they are only slower than MultiAsync. In terms of the needed number of updates, our algorithms show very similar performance towards FedAsync and FedAvg.

Except for the horizontal comparison of the five algorithms' run-time accuracy, we also conducted a vertical comparison of them in terms of client scalability. In this set of experiments, we changed the client number from the original 100 to 200 and then 300 and evaluate each



(a) MNIST time vs. accuracy (b) CIFAR-10 time vs. accuracy (c) WikiText2 time vs. perplexity

(d) MNIST updates vs. accuracy (e) CIFAR-10 updates vs. accuracy (f) WikiText2 updates vs. perplexity

Figure 3: Accuracy curves of 5 FL algorithms on MNIST, CIFAR-10, and WikiText2 datasets with 100 clients in the system.

Table III: The ratios of required time and update number for 90% accuracy compared with 100-client statistics. Each value subtracted by 1 is the increased ratio relative to the 100-client value.

	200 clients		300 clients	
	Time	Updates	Time	Updates
MultiAsync	**1.21**	**2.43**	**1.43**	**4.34**
MultiSync	1.61	2.68	1.90	4.87
FedAsync	2.42	3.63	5.99	9.01
FedAvg	1.98	4.05	2.44	7.26
HierFAVG	1.64	3.18	1.75	8.19

algorithm by the needed time and number of updates to reach 90% accuracy. The experimental result is shown in Tab III.

From table III we can see that with the increasing number of clients in the system, the convergence time and update amount of the methods increase in different trends. MultiAsync has a trend that is closest to the linear one. For every additional 100 clients, the time ratio approximately increases by a factor of 0.21 while the update's increasing ratio rises slightly from 1.43 to 1.91. These two statistics also show the smallest absolute amount of increase among all. On the other hand, FedAsync displays the most divergent increasing trend in both time and update amounts. The increase of time at 200 clients is 1.42 while in the interval of 200 to 300 clients, the increasing ratio escalates to 3.57. For the number of updates, it experiences a similar trend from 2.63 to 5.38.

From the above analysis, we can expect a more stable trend of increasing time and updates in MultiAsync when the system engages a larger scale of clients. To conclude, MultiAsync not only exhibits the fastest convergence according to previous discussions, but it also demonstrates the best client scalability compared with the three baselines.

C. Impact of Multiple Servers in Asynchronous FL

To further illustrate the benefits of introducing a multi-server system to an asynchronous FL framework.

In asynchronous FL systems, aggregation happens immediately after the update is received by the server. The time spent on each aggregation becomes an issue that can cause a queue of updates that await. Thus, we first conduct experiments to explore this queuing phenomenon on the server.

The experiment is conducted on MultiAsync and FedAsync, where they both possess 200 clients, while MultiAsync has 4 servers and FedAsync with only 1. The local training delays of clients are generated from Gaussian distribution with a mean of 150 ms and a standard deviation of 60 ms. We set the computation delays according to table II.

From figure 4 we observe that, after the training starts, the queue length of FedAsync constantly rises to nearly 80 updates from 300ms to 600ms and stays at a level above 20 during the rest of the experimental period. This illustrates the fact that many updates are not processed on time due to the lack of computational power of the single server, which could be optimized by introducing more

Table IV: The improvement of efficiency of MultiAsync on the MNIST dataset. The top 3 rows are the results when simulating network latency while the bottom 3 rows are those without. In each experiment, the time is measured for either FedAsync or MultiAsync to stably reach an accuracy of 90% or 95% in milliseconds. The boost is the percentage of time decreased in MultiAsync compared with FedAsync.

Network	Method	Time 90%	Time 95%
lat.	FedAsync	59s	125s
	MultiAsync	22s	51s
	boost	61%	58%
No lat.	FedAsync	40s	75s
	MultiAsync	25s	56s
	boost	38%	25%

servers in parallel to process the updates. On the other hand, MultiAsync despite having queues, never exceeds the length of 20 during the whole experimental period, which illustrates the improvement in the efficiency in processing client updates of introducing multiple servers. In real-life applications, MultiAsync can always provide freedom of choice in the number of servers to cope with the actual computational power needed to handle the offload of clients.

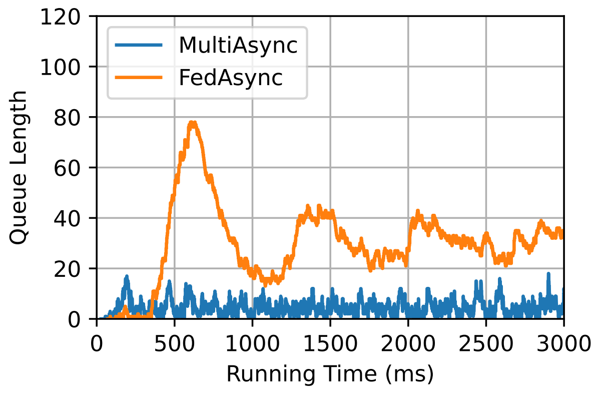


Figure 4: The number of queuing updates at server

To illustrate the efficiency gain in having smaller queues, we further conducted a group of experiments to test the time needed for FedAsync and MultiAsync to stably reach 90% and 95% accuracy on an MNISt dataset. In this group of experiments, only the time difference caused by resource heterogeneity is simulated. The results are shown in the bottom three rows of table IV.

We see that MultiAsync gained a 38% boost in speed to reach 90% accuracy, and a 25% boost to reach 95% accuracy.

To further demonstrate the fact that MultiAsync also mitigates the impact of high communication delays, we conducted a similar group of experiments that, in addition, introduces network latency according to table II. The results are shown in the top three rows of table IV.

Compared with the bottom three rows where no network latencies are considered, the top three rows show a larger difference between MultiAsync and FedAsync, where MultiAsync gained a 61% boost to reach 90% accuracy, and a 58% boost to reach 95%. In the system, the communication complexity mainly consists of a large amount of update sending and global model delivering, which makes it of vital importance to reduce the latencies of these client-server communications. In MultiAsync, the distance between clients and their assigned server is ensured to be small (within 5 ms), showing the benefit of geo-replication. This enlarged gap compared with non-communication results indicates that MultiAsync not only addresses the limitation of server computing power, it also overcomes the inefficiency brought by long-distance communication.

D. Impact of Learning Rate Decay

To explore the impact of having a client learning rate decay strategy on MultiAsync, we conducted two experiments.

In the first experiment, we used MultiAsync and FedAsync each with 112 clients to perform the classification task on the MNIST dataset. For these two algorithms that represent the single-server and multi-server case, we gathered statistics on the number of updates of every client and plotted a kernel density estimate (KDE) plot to show its distribution. A KDE plot could be considered as a continuous version of a histogram to reveal the distribution of a statistic. The most desirable result of a KDE plot for a FL should be a single concentrated peak, representing that all clients have a similar amount of contribution. The KDE plots of MultiAsync and FedAsync are shown in figure 5.

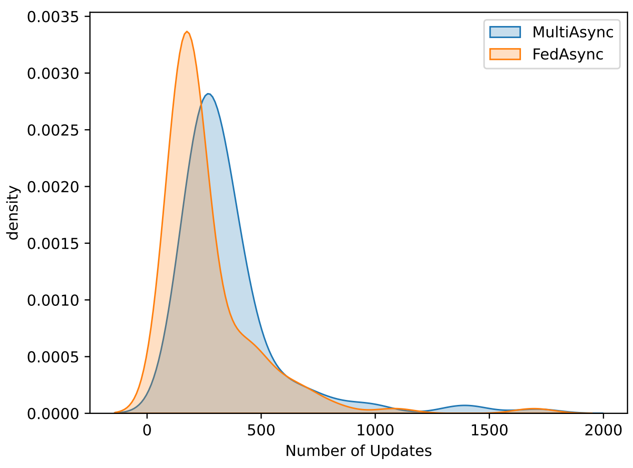


Figure 5: The distribution of client updates in FedAsync and MultiAsync

From the plot, we observe that FedAsync has a more centralized distribution of updates with a steep peak at around 200 updates compared with a gentler peak of MultiAsync at around 300 updates. This is because the major source of delay in FedAsync is communication, and the distribution of communication delay is relatively concentrated. On the other hand, by introducing multiple servers to handle updates from clients in different areas, the communication delay of MultiAsync is significantly shrunk. This makes the difference in clients' training time the major source of heterogeneity in MultiAsync. Moreover, this difference has a less concentrated distribution than communication delay, thus we acquired worse results from MultiAsync than FedAsync. We also observe a small peak at around 1400 updates for MultiAsync and 1700 for FedAsync. This is caused by a coincidence of having clients that train fast and near a server at the same time, which could cause a biased global model due to its high influence.

Based on the analysis above, we consider it necessary of having a client learning rate decay mechanism that customizes the learning rate of clients according to the number of updates they contribute. This would lead to less impact per update for the more active clients to balance the overall contribution of clients. Moreover, as we configured in the decay function, the decay ratio should increase as the number of updates rises to cope with the "lucky" clients that are fast in all aspects.

In the second experiment, we tested the convergence speed of MultiAsync with and without the learning rate decay mechanism.

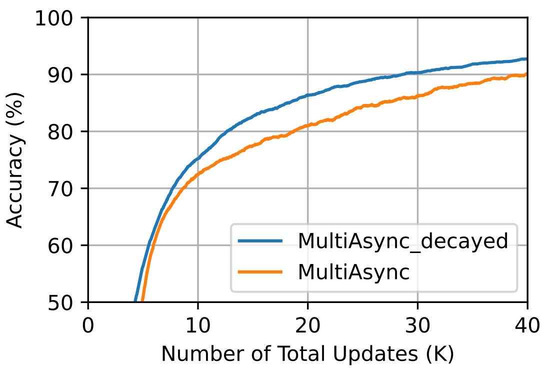


Figure 6: The impact of learning rate decay

From figure 6 we observe that by introducing the learning rate decay mechanism, the overall convergence speed of MultiAsync increased. This proves that this mechanism balances the impact of resource heterogeneity among clients and leads to faster convergence of the system.

VI. Conclusion

In this paper, we proposed a novel asynchronous multi-server algorithm, MultiAsync. As we discussed in the introduction section, multi-server settings are more practical in real-life scenarios where clients are distributed over the world. And asynchronous improvement addresses the bottleneck brought by the slowest client in either communication or computation. We introduce multi-server settings while keeping local updates in an asynchronous fashion. The experimental results show that our method performs better than the three previous significant algorithms FedAsync, HierFAVG, and FedAvg in terms of convergence speed. The design of our token-based broadcast algorithm ensures that we don't sacrifice much accuracy for the trade-off of faster convergence.

We put into future work to investigate the optimal client-server ratio in different configurations to enhance the effectiveness of MultiAsync. We also aim to explore the potential of integrating clustering algorithms in MultiSync to enable servers to group clients based on similarities in their data distributions. In addition, improving the system's robustness towards Byzantine behaviors is also a crucial aspect of our future work.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282.

[2] T. Yang, G. Andrew, H. Eichner, et al., "Applied federated learning: Improving google keyboard query suggestions," CoRR, vol. abs/1812.02903, 2018.

[3] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," CoRR, vol. abs/1906.04329, 2019.

[4] A. Hard, K. Rao, R. Mathews, et al., "Federated learning for mobile keyboard prediction," CoRR, vol. abs/1811.03604, 2018.

[5] M. Chen, A. T. Suresh, R. Mathews, et al., "Federated learning of n-gram language models," CoRR, vol. abs/1910.03432, 2019.

[6] B. Yuan, S. Ge, and W. Xing, "A federated learning framework for healthcare iot devices," CoRR, vol. abs/2005.05083, 2020.

[7] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, Federated learning for healthcare informatics, 2019. doi: 10.48550/ARXIV.1911.06270. [Online]. Available: https://arxiv.org/abs/1911.06270.

[8] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," International Journal of Medical Informatics, vol. 112, pp. 59–67, 2018, issn: 1386-5056. doi: https://doi.org/10.1016/j.ijmedinf.2018.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138650561830008X.

[9] N. Rieke, J. Hancox, W. Li, et al., "The future of digital health with federated learning," CoRR, vol. abs/2003.08119, 2020.

[10] P. Kairouz, H. B. McMahan, B. Avent, et al., Advances and Open Problems in Federated Learning. 2021.

[11] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," CoRR, vol. abs/1903.03934, 2019.

[12] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation, 2022. doi: 10.48550/ARXIV.2205.13797. [Online]. Available: https://arxiv.org/abs/2205.13797.

[13] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 15–24. doi: 10.1109/BigData50022.2020.9378161.

[14] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1–6. doi: 10.1109/ICC40277.2020.9148862.

[15] M. S. H. Abad, E. Ozfatura, D. GUndUz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2020, pp. 8866–8870. doi: 10.1109/ICASSP40776.2020.9054634.

[16] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in 2019 IEEE Real-Time Systems Symposium (RTSS), 2019, pp. 406–418. doi: 10.1109/RTSS46320.2019.00043.

[17] A. Hard, K. Rao, R. Mathews, et al., "Federated learning for mobile keyboard prediction," CoRR, vol. abs/1811.03604, 2018. arXiv: 1811.03604. [Online]. Available: http://arxiv.org/abs/1811.03604.

[18] Z. Huang, F. Liu, and Y. Zou, "Federated learning for spoken language understanding," in Proceedings of the 28th International Conference on Computational Linguistics, Barcelona, Spain (Online): International Committee on Computational Linguistics, Dec. 2020, pp. 3467–3478. doi: 10.18653/v1/2020.coling-main.310. [Online]. Available: https://aclanthology.org/2020.coling-main.310.

[19] X. Li, Y. Gu, N. C. Dvornek, L. H. Staib, P. Ventola, and J. S. Duncan, "Multi-site fmri analysis using privacy-preserving federated learning and domain adaptation: ABIDE results," CoRR, vol. abs/2001.05647, 2020. arXiv: 2001.05647. [Online]. Available: https://arxiv.org/abs/2001.05647.

[20] Z. Chai, A. Ali, S. Zawad, et al., "Tifl: A tier-based federated learning system," ser. HPDC '20, Stockholm, Sweden: Association for Computing Machinery, 2020, pp. 125–136, isbn: 9781450370523. doi: 10.1145/3369583.3392686. [Online]. Available: https://doi.org/10.1145/3369583.3392686.

[21] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1–7. doi: 10.1109/ICC.2019.8761315.

[22] B. Cox, L. Y. Chen, and J. Decouchant, Aergia: Leveraging heterogeneity in federated learning systems, 2022. doi: 10.48550/ARXIV.2210.06154. [Online]. Available: https://arxiv.org/abs/2210.06154.

[23] S. M. Azimi-Abarghouyi and V. Fodor, Multi-server over-the-air federated learning, 2022. doi: 10.48550/ARXIV.2211.16162. [Online]. Available: https://arxiv.org/abs/2211.16162.

[24] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, CoRR, vol. abs/1812.06127, 2018.

[25] X. Wang and Y. Wang, Asynchronous hierarchical federated learning, 2022. doi: 10.48550/ARXIV.2206.00054. [Online]. Available: https://arxiv.org/abs/2206.00054.

[26] A. Adolfsson, M. Ackerman, and N. C. Brownstein, "To cluster, or not to cluster: An analysis of clusterability methods," Pattern Recognition, vol. 88, pp. 13–26, Apr. 2019. doi: 10.1016/j.patcog.2018.10.026. [Online]. Available: https://doi.org/10.1016%2Fj.patcog.2018.10.026.

[27] V. Smith, C. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," CoRR, vol. abs/1705.10467, 2017. arXiv: 1705.10467. [Online]. Available: http://arxiv.org/abs/1705.10467.

[28] D. K. Dennis, T. Li, and V. Smith, "Heterogeneity for the win: One-shot federated clustering," CoRR, vol. abs/2103.00697, 2021. arXiv: 2103.00697. [Online]. Available: https://arxiv.org/abs/2103.00697.

[29] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, and J. Jiang, "Multi-center federated learning," CoRR, vol. abs/2005.01026, 2020.

[30] Latency between different regions of amazon web services, https://cloudping.co/grid, Accessed: 2023-07-12.

2 Extended Related Work

This chapter serves as an essential complement to the first chapter, offering an in-depth exploration of the related work. The basics of FL are given, and more background on multi-server FL and asynchronous FL is provided.

2.1. Federated Learning

FL is a decentralized approach to machine learning proposed by Google in 2017 [1]. It aims to make good use of the vast data distributed to a large number of users and overcome the challenge of privacy preservation at the same time. It was initially introduced to improve machine learning on mobile and edge devices [2]–[6]. Now it is also developed to train a shared model among some large organizations such as finance companies and health care centers [7]–[9]. The two scenarios are termed cross-device and cross-silo FL [10]. In this paper, we only focus on cross-device FL.

Different from conventional machine learning where the training is completed on a single machine with the full dataset, FL requires local training happening in different machines simultaneously and with different datasets. The most commonly used topology is the centralized and synchronous FL framework. As shown in Figure 2.1, in each round, a set of clients are selected to receive the current global model from the server, train the model with their local dataset, and send the update back to the server. The server collects all the updates and computes a new model according to the *aggregation rule* (AGR) of the FL system and applies the new aggregated model to its global state. The process is repeated in rounds until the global model converges.

Brendan McMahan et al. proposed the first AGR named Federated Averaging (FedAvg) [1]. It is the distributed version of traditional stochastic gradient descent (SGD) and it is provably communication-efficient and convergent. Each round a fraction of clients are selected to update their training results to the server. The server aggregates the updates according to the following equation:

$$W^t = \frac{1}{K}\sum_{i=1}^{K}\frac{n_i}{N}W_i^{t-1}$$

W^t is the resulting global model on the server at time t, W_i^{t-1} is Client i's updated model based on the last global model W^t, K is the number of selected clients, n_i is the number of datapoints upon which Client i trains the model, and N is the total number of datapoints. The key idea is to average the model updates from different clients. The global model benefits from

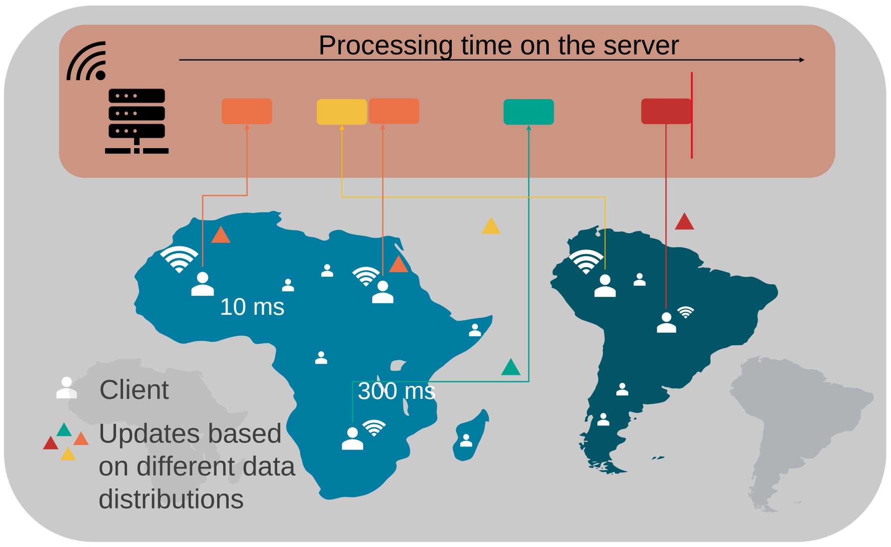

Processing time on the server
10 ms
300 ms
Client
Updates based on different data distributions

Figure 2.1: The framework of FL

the diverse dataset of clients while mitigating the privacy concerns associated with sharing raw data.

The emphasis on privacy preservation is the driving force behind the considerable attention that FL has gained from numerous researchers. To further enhance privacy in FL, researchers have introduced advanced techniques such as differential privacy [11], [12] and secure aggregation [13], [14]. Differential privacy defends the system against inference attacks that aim to learn sensitive information in data points, while secure aggregation protects the system against attacks that attempt to compromise the integrity of model updates. In this paper, we do not focus on privacy preservation in FL, instead, we focus on the performance and efficiency of the FL systems.

2.2. Data Heterogeneity and Resource Heterogeneity

FL presents a common yet challenging scenario where the dataset distributed across individual devices exhibits notable variations in features, quantity, and label distribution [15]. This diversity in data distribution is visualized in Figure 2.1, wherein clients are depicted with different colors, highlighting their distinct datasets.

The distribution of local data within a dataset is profoundly shaped by the methodologies employed during data collection. Data that originate from varying timeframes, geographical locations, and diverse devices introduce a significant degree of heterogeneity to the dataset. For instance, consider the scenario of gathering data to investigate the correlation between individuals' sleeping habits and their occupations. Should a substantial proportion of the data originate from a specific age bracket (such as 20 to 30 years old), any conclusions drawn from this dataset could potentially distort the relationship between individuals' bedtime and their age. Consequently, this engenders a form of data heterogeneity that can be classified as non-identical, as the data have been sourced from disparate probability distributions owing to the uneven collection process.

Aside from data heterogeneity, another substantial challenge in FL is resource heterogeneity. FL has gained extensive adoption across diverse fields, with instances such as Apple's employment of FL to enhance Siri's functionality and Google's utilization of FL in predictive text suggestions and emoji recommendations within the Gboard keyboard application. The participating client devices exhibit a wide spectrum, spanning various smartphone models to full-fledged computers. Moreover, resource disparities come to the forefront during local model training. Some clients may execute parallel processes, while others might have idle processing units. These variations underscore the existence of resource heterogeneity within the FL ecosystem. It is evident that clients equipped with limited resources often experience prolonged training durations, inadvertently impeding training efficiency.

Also, during the local training, some clients have other processes running in parallel while other clients have some idle kernels. The above possibilities address the existence of resource heterogeneity. Clients with fewer resources often prolong the training time thus binding the training efficiency. Various discussions and solutions have been proposed to address resource heterogeneity. We will introduce synchronous and asynchronous solutions in later sections.

2.2.1. Synchronous FL

Most conventional FL systems are synchronous. Clients and servers are working under the global clock. An exemplar instance of this is observable in the FedAvg framework[1], wherein the progression of local training on clients is governed by global round synchronization. After every selected client uploads its model to the server, the FedAvg aggregation algorithm can be performed. Fast clients who finish the training and transfer earlier should wait for the slow ones so that they could synchronously move to the next round. On the other hand, the synchronous nature of these FL systems introduces notable advantages. One key advantage lies in the establishment of synchronized model updates. By ensuring that all clients contribute their updates at the same point, the global model can be updated coherently, fostering convergence towards a consensus model. Additionally, this synchronization simplifies the implementation of aggregation strategies, enabling efficient and accurate integration of client contributions. Synchronous systems also simplify equitably distributing the contributions of each client, which could avoid potential bias that could arise from an uneven allocation of updates originating from any individual client. In this section, we introduce some synchronous federated solutions that aim to improve the efficiency of training.

Chai et al. proposed a tier-based FL frame TiFL to reduce the influence of resource heterogeneity in clients [16]. In TiFL, the selection of clients is adaptive to their dynamically adapted to their individual computational capabilities. The initiation of the process involves the central server executing a profiling algorithm, which systematically accumulates latency metrics across all available clients. Based on these metrics, the clients are subsequently stratified into distinct groups known as ”tiers.” Different from the conventional FL systems where clients are randomly selected, TiFL randomly selects the clients from one tier in each round. The stragglers’ negative effect on the training efficiency of a single round is significantly reduced.

Mobile edge computing presents a pragmatic use case for FL, particularly concerning smartphone clients where resource heterogeneity extends beyond computing power to encompass disparities in wireless channel conditions. Within this context, the occurrence of interrupted connections during rounds introduces unpredictability to the waiting period. In response, Nishio et. al. have introduced a hierarchical pace control FL framework SmartPC which focuses on production FL in real-time settings [17]. They consider the cross-device FL and conduct experiments with Android smartphones. SmartPC aims to balance the model accuracy, training time, and energy consumption of clients’ devices. Analogous to the federated client selection (FedCS) approach [18], SmartPC introduces the concept of imposing deadlines to regulate waiting times. In the SmartPC framework, this process is initiated by the central server, which evaluates the status of selected devices prior to each round. A global deadline is determined according to the evaluation result. Subsequently, the local layer operating on each device undertakes system configuration adjustments to adhere to the imposed deadline while minimizing energy expenditure. Crucially, the global deadline is determined based on a feedback mechanism that ensures a predefined proportion of clients are able to meet the deadline.

The aforementioned study primarily directs its attention toward addressing resource heterogeneity. However, while emphasizing resource disparities, the work may inadvertently overlook the pivotal aspect that the resultant model's quality remains intricately linked to the balance and equality of clients' contributions. Cox et al. proposed an FL framework Aergia that leverages model offloading to enhance the training efficiency while managing the intrinsic challenges of data heterogeneity across diverse client devices [19]. The federator evaluates and profiles the performance of selected clients and matches the weak with the strong ones. The matching algorithm also considers the feasibility of offloading by computing the pair-wise similarity of the two clients' data inside a trusted execution environment (TEE). Once the schedule is decided by the federator and known by the related clients, the weak ones freeze part of the model and send it to their matched strong clients. The heterogeneity of training phases is also measured and discussed in their work. The most time-consuming phase is chosen to be frozen and offloaded.

Aergia is an FL framework that leverages the heterogeneity in its clients [19]. Aergia aims at reducing the delay brought by clients with less computing power. In Aergia, the central server is assigned the task of estimating clients' performance, distinguishing the slow and fast clients, and deciding on a schedule where the weak clients offload part of their training to the strong ones.

Figure 2.2 illustrates the system architecture of Aergia. The same as the conventional FL system, clients are distributed and able to communicate with the central server ”federator”. The latest global model is constantly updated and broadcast to all the clients. And a set of selected clients train the global model upon their local private data for a customized number of epochs, send their update back to the federator, and wait for the next round's global model. With the aggregation running at the federator, the updated global model gathers the training result of enormous data without leaking any of the raw private data to the central federator.

Below we list the differences between Aergia and the conventional FL framework.2.2.

1. Clients are equipped with clocks to evaluate their local training time and communication delay. In each round, the performance is reported to the federator after the first local epoch.
2. During the round, the federator is tracing the performance of the selected clients while running a scheduling algorithm. It evaluates the current computing power of each client and identifies the strong and weak ones. It indicates a schedule where slow clients should offload part of the training job to the fast ones. The schedule also takes the data similarity into consideration which is elaborated in the following.
3. The federator is equipped with a TEE to evaluate the data similarity between two matched clients. It ensures code integrity and data security. The resulting similarity matrix is then used by the federation to evaluate if the match is appropriate. A high data similarity indicates that offloading training tasks between these two datasets are more reasonable.
4. Clients are able to directly communicate with each other. Once a slow client is matched with a fast one, she will be informed of the IP address of the fast client so that the offloaded model can be sent via the link.

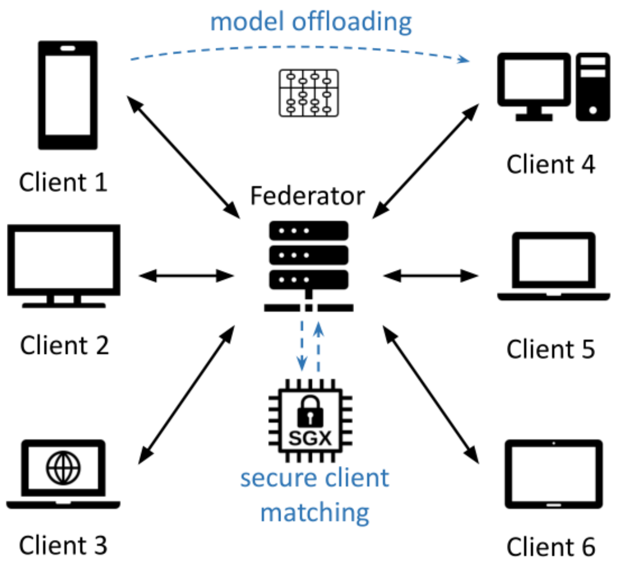


Figure 2.2: The system model of Aergia

2.2.2. Asynchronous FL

Within synchronous FL systems, the presence of resource heterogeneity introduces the potential for considerable round-duration extensions. Particularly in the context of edge device training, where FL finds widespread application, the notion of global synchronization confronts formidable challenges arising from divergent computing capacities and intermittent communication. Addressing this, asynchronous FL emerges as a remedy, deftly surmounting the impact of resource disparities through immediate update aggregation upon server reception. This asynchronous variant of the conventional FedAvg algorithm can be encapsulated as follows:

$$w^{t+1} = w^t - \alpha \frac{n_k}{N}(w_k^t - w_k^{t+1}) \tag{2.1}$$

where w^t is the global model with timestamp t, w_t^k is the local update of Client k with timestamp t, $\frac{n_k}{N}$ is the data proportion of the update, and α is a staleness parameter. In this section, we introduce previous work in asynchronous FL.

Chen et al. proposed ASO-Fed, an asynchronous online FL framework that enables wait-free computation and communication [20]. ASO-Fed introduces a paradigm where clients' updates seamlessly flow into the federator across different rounds, facilitating an uninterrupted progression. This distinct approach orchestrates local training rooted in freshly streamed-in data, thereby enhancing model adaptability. ASO-Fed introduces a decay coefficient to adjust the training result according to the last model trained on previous data. Augmenting this, a pivotal central feature representation learning mechanism is employed to attenuate the influence of asynchronous aggregation, further refining the training process. To address the staleness problem, ASO-Fed scales the learning step size by the client's communication delays. Since clients with long delays lose their impact on the intermediate models, a larger step size is assigned to compensate for the loss. The global model is updated as

$$w^{t+1} = w^t - \frac{n_k}{N} r_k^t (w_k^t - w_k^{t+1}), \tag{2.2}$$

where $r_k^t = max\{1, \log \bar{d_t^k}\}$, and $\bar{d_t^k} = \frac{1}{t}\sum_{\tau=1}^{t} d_k^\tau$ is the average computing time of Client k in past rounds.

Xie et al. proposed FedAsync, an asynchronous optimization algorithm for FL [21]. They guarantee a near-linear convergence and address the staleness problem in the asynchronous setting. Each time an update arrives, the weighted averaging is applied for the current global model and the update. The weight of the client decreases when its staleness increases. The global model is updated by the equation

$$w^{t+1} = w^t - \frac{n_k}{N} r_k^t (w_k^t - w_k^{t+1}), \tag{2.3}$$

where $r_k^t = max\{1, \log \bar{d_t^k}\}$, and $\bar{d_t^k} = \frac{1}{t}\sum_{\tau=1}^{t} d_k^\tau$ is the average computing time of client k in past rounds.

Wang et al. proposed AsyncFedED, an adaptive asynchronous FL aggregation based on Euclidean distance [22]. If the global model is updated before a client sends back its update, the learning rate of the server should be adjusted based on the staleness of the current update. AsyncFedED evaluates the staleness of an update by its Euclidean distance from the server. A larger staleness leads to a smaller learning rate for its aggregation. The staleness of update w_k^{t+1} is defined as

$$\gamma_k^{t+1} = \frac{\|w_k^t - w_k^{t+1}\|}{\|\triangle_i w_k^{t+1}\|} \tag{2.4}$$

The global learning rate of the current update is calculated by

$$\eta_{k+1}^t = \frac{\lambda}{\gamma_k^{t+1} + \epsilon,} \tag{2.5}$$

where λ and ϵ are hyperparameters to be tuned.

2.2.3. Multi-server FL

Another scheme to address resource heterogeneity is to spread the computation and communication burdens instead of bearing them all on a single server. In addition, distinguished global models maintained at multiple servers facilitate scenarios where personalized models are required. Existing methods [23]–[26] add some edge servers or assign proxy clients to undertake part of the workload. And clustering can be applied to characterize data distributions. Most existing multi-server federated frameworks are hierarchical, where servers of multiple levels exist. Figure 2.3 illustrates the scenario and settings in classical hierarchical FL. In hierarchical FL [25], [26], there is a leader server and multiple edge servers. Two levels of aggregations happen in each round. The first level happens at the edge server which aggregates a group of clients and sends the update to the leader. The leader executes the second-level aggregation and generates the global mode for the new round.

One of the most classical hierarchical FL frameworks, HFL, was proposed by Abad et al. [25]. In the HFL system, there are two levels of aggregators and a group of mobile users. The clients are clustered according to their location and assigned to the nearest station, which can be considered as an edge server. Inter-cluster aggregation happens synchronously at each iteration. And the global synchronization occurs at the central station in every predefined period of time. The servers' computation burden is greatly reduced by collaboratively aggregating and communication overheads between clients and their directly-connected servers shrink effectively through geographic clustering. However, the synchronous nature of both levels brings a bottleneck to the efficiency of the system.

HierFedAsync is an asynchronous hierarchical FL framework proposed by Wang et al. [24]. A central cloud server and a group of ”aggregators” spread the workload of communication and aggregation. A clustering algorithm is run at the central server, and in each cluster, one client is selected as an "aggregator" to be responsible for aggregating the updates within the cluster and transferring models to and from the central server. In this asynchronous system, the delivery of the global model and aggregation on both "aggregators" and the central server happens periodically. The staleness is evaluated to adjust the learning rate of each client and "aggregator".

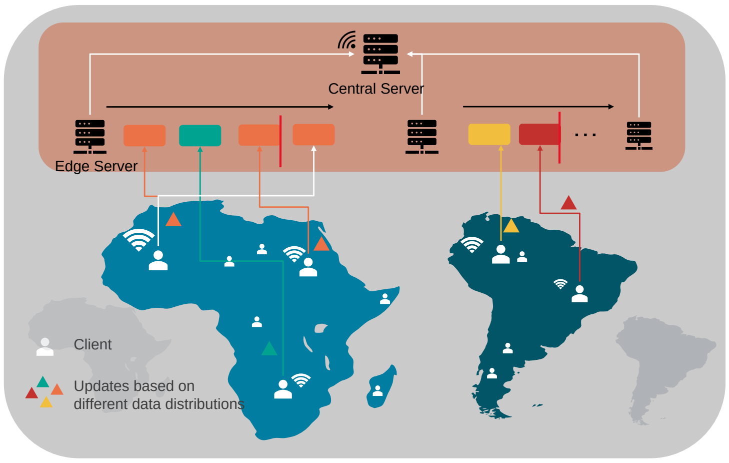


Figure 2.3: The framework of hierarchical FL

Apart from the hierarchical structure, the approach of single-level multi-server FL also flourishes. To address the challenge of data heterogeneity, Xie et al. proposed a multi-center FL algorithm, FeSEM [27]. The main idea is to cluster the updates and assign them to their closest global model (i.e. center). They showed in experiments that the clusters of local updates characterize clients' data distribution thus models generated from similar data distributions are gathered on the same center and aggregated. The integration of the clustering algorithm addresses the difficulty in dealing with non-independent and identically distributed (IID) data distributed among clients. But the lack of consideration for the system's communication efficiency makes it less practical for real-world application.

To conclude, multi-server FL overcomes the heavy burden brought by frequent aggregations on one server and gains the potential of integrating clustering algorithms for more efficient aggregation. However, existing multi-server FL systems either bear the disadvantage of synchronization or lack the ability to address long-distance communication. Moreover, hierarchical Fl introduces additional communication overheads involved between levels, which prolongs the whole training process.

3 Additional Experiments

In this chapter, we provide details of used datasets, models, and additional experiments to demonstrate a more comprehensive evaluation of MultiAsync.

3.1. Datasets

Three datasets are used for experimental analysis: MNIST, CIFAR-10, and WikiText-2. MNIST and CIFAR-10 are both image datasets, while WikiText-2 is a text dataset. MNIST stands for ”Modified National Institute of Standards and Technology”. It contains a collection of grayscale handwritten digit images. Every image is a 28x28-pixel square, representing a number from 0 to 9. There are 60,000 training images and 10,000 test images in MNIST. MNIST is often used as an introductory dataset in the research of image classification algorithms since it is relatively simple. On the other hand, CIFAR-10 provides a more challenging scenario. It stands for ”Canadian Institute for Advanced Research - 10”, which consists of 60,000 color images of 32x32 pixels that are labelled with 10 different classes. Both MNIST and CIFAR-10 are renowned for their extensive usage in pioneering research within the field of FL. They play an indispensable role in evaluating the potential of novel models, architectures, and techniques. These datasets serve as essential tools to assess the prowess of innovative approaches, providing a foundation upon which the advancements in FL are built and tested.

WikiText-2 is a commonly-used text dataset in FL. It is derived from the content of Wikipedia articles and is often used for language modeling and text prediction tasks. It contains a wide array of words and phrases, covering a rich vocabulary.

3.2. The Impact of Network and Computing Power on Server

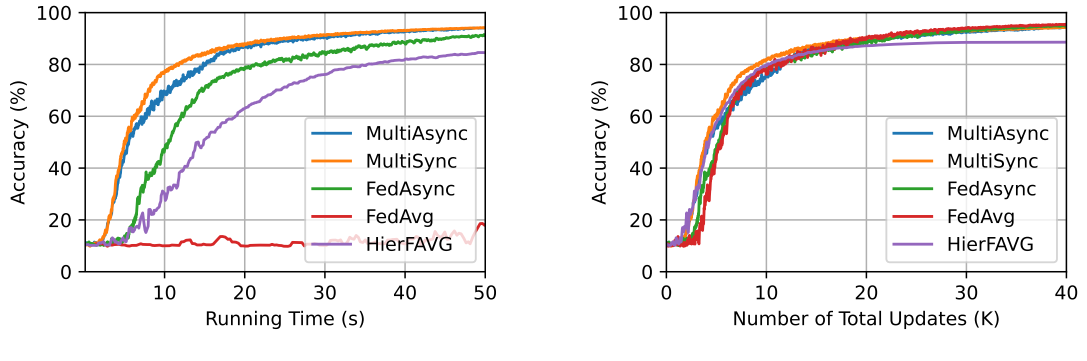
Before taking network conditions into consideration, we conducted a set of experiments without network delays but only computation delays which are shown in Table 3.1. The values are the average running time of each process measured using Python *time* package.

Table 3.1: Simulated computation delays

Local Training	200 ms
Global Synchronization in MultiSync	2 ms
Peer-to-peer Aggregation in MultiAsync	2 ms
FedAvg Aggregation	15 ms
HierFAVG Cloud Aggregation	15 ms
Aggregation in FedAsync	2 ms

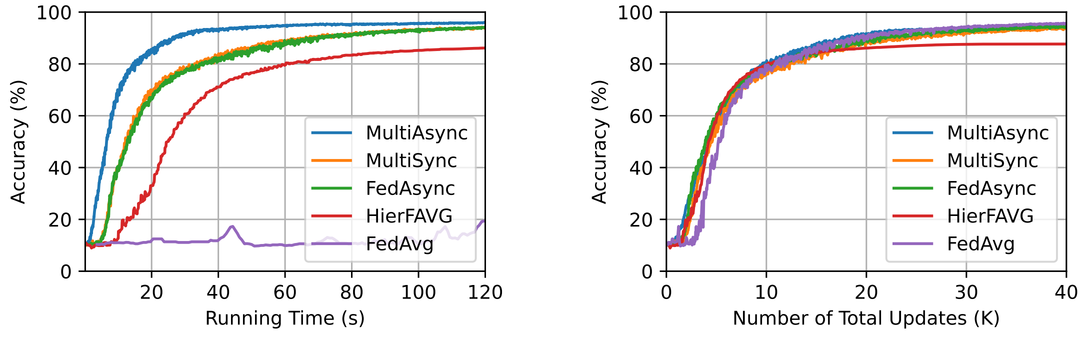
In Figure 3.1, we show the experiment results without considering network delays. In Fig 3.1a, MultiAsync displays the fastest convergence among all the algorithms. More specifically, MultiAsync and MultiSync both show a quicker rise in the beginning phase of training. This boost in convergence time is mainly due to the advantage of MultiAsync and MultiSync dispatching needed computing power to multiple servers instead of bearing them on one. As we analyzed, there is more amount of time in FedAsync that a queue of updates is waiting to be aggregated.

In addition, from 3.1a, we see MultiSync replaces MultiAsync to be the best-performing method. This provides a practical scenario where MultiSync becomes an alternative for MultiAsync - when the communication latencies between servers are extremely small. Without the network bottleneck, MultiSync exhibits the advantage of synchronization among servers, which provides a more stable rise compared to MultiAsync.



(a) Time **(b)** Updates

Figure 3.1: Accuracy curves of 5 algorithms on dataset MNIST without network delays.



(a) Time **(b)** Updates

Figure 3.2: Accuracy curves of 5 algorithms on dataset MNIST with network delays.

For further exploration of the impact of network latencies, we conduct experiments that put servers into four different areas (Paris, Hongkong, Sydney, and California) in order to simulate real-life network delays. Clients are distributed equally in each area. For single-server algorithms FedAvg and FedAsync, the server is set in Paris. Communication delays among areas are shown in Table 3.2 and the resulting accuracy curves are plotted and shown in Fig 3.2.

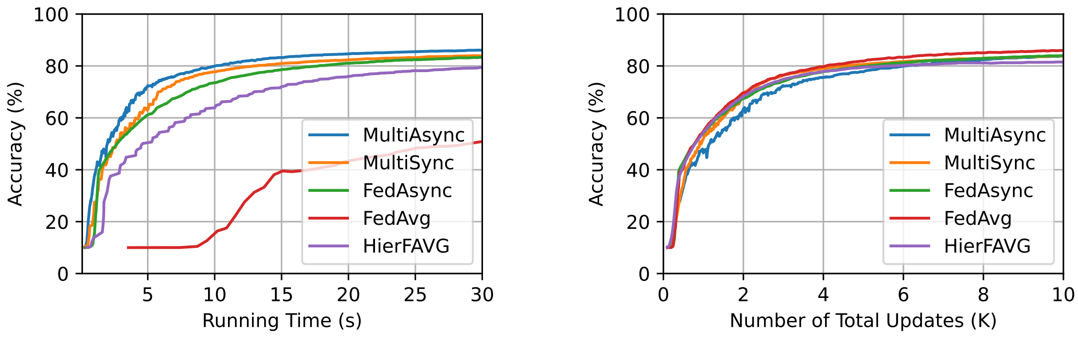
Table 3.2: Communication delays

Delays(ms)	Hongkong	Paris	Sydney	California
Hongkong	1.41	194.9	132.28	155.13
Paris	197.91	0.9	278.83	142.25
Sydney	132.06	280.11	2.56	138.47
California	154.96	142.79	138.57	2.14

Compared with Fig 3.1 where no network latencies are considered, Fig 3.2 shows a larger difference between MultiAsync and other algorithms. In the system, the communication complexity mainly consists of a large amount of update sending and global model delivering, which makes it of vital importance to reduce the latencies of these client-server communications. In MultiAsync, the distance between clients and their assigned server is ensured to be small (within 5 ms), showing the benefit of geo-replication. This enlarged gap compared with non-communication results indicates that MultiAsync not only addresses the limitation of server computing power, it also overcomes the inefficiency brought by long-distance communication.

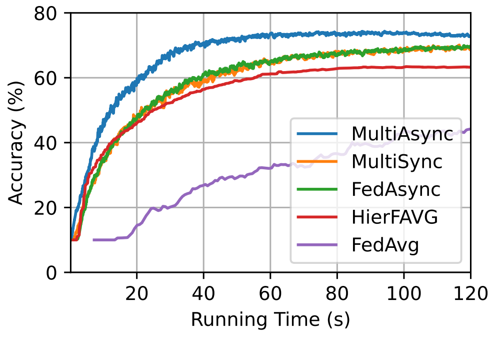
3.3. Experiments on IID Data

With IID settings, each client is relatively similar in terms of its statistical properties. It simplifies the training process since the local optimums of clients are close to each other so updates from different clients are more likely to drive the global model in a similar direction. We conducted experiments on IID data using datasets MNIST and CIFAR-10. The results of non-IID data are displayed for the convenience of comparison.

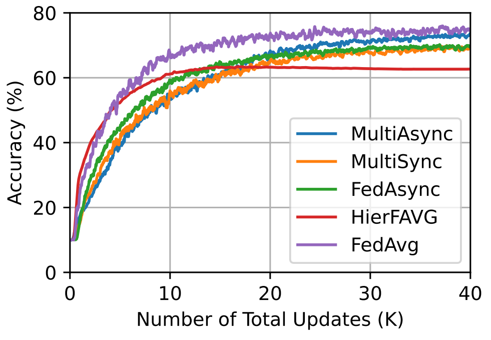


(a) Time **(b)** Updates

Figure 3.3: Accuracy curves of the five algorithms on IID CIFAR-10 Dataset. a) With the x-axis to be the running time. b) With the x-axis to be the number of updates.



(a) Time



(b) Updates

Figure 3.4: Accuracy curves of the five algorithms on Non-IID CIFAR-10 Dataset. a) With the x-axis to be the running time. b) With the x-axis to be the number of updates.

Comparing figure 3.3a and figure 3.4a, we find that the difference between MultiAsync and the baselines is larger with non-IID data, especially the difference of the gap between MultiAsync and FedAsync. Though the difference is smaller with IID data, MultiAsync still exhibits the best performance.

3.3.1. Impact of the number of servers

In this experiment, we explore the impact of the number of clients each server has in MultiSync. Results are evaluated through metrics including average accuracy, standard deviation, and the time needed for the MultiSync to reach 55% accuracy. The average accuracy is the average resulting accuracy of all servers, indicating the overall performance of each setting, while the standard deviation explores the disparity among servers. A large disparity in servers' resulting model displays the instability of the training system. The time needed to reach an average accuracy of 55% is a metric to test the algorithm's convergence speed since we assume the increase in servers may cause the system to slow down. The choice of the threshold at 55% is due to the consideration of the lowest average accuracy in this experiment.

The reason we conduct this experiment with MultiSync instead of MultiAsync is for the uniformity of experiments. In MultiAsync, with the varying number of servers, the age threshold for activating the model broadcast needs to be fine-tuned since it is the ratio related to the number of clients per server. However, in MultiSync, we could simply set the synchronization period to ensure uniformity.

There are 64 clients in total and every server maintains the same number of clients from 2 to 32, which leads to the number of servers decreasing from 32 to 2. We use the dataset MNIST and assign every client $l = 2$ labels of data to simulate the non-IID distribution. We control the number of total updates to be 30000.

Table 3.3: The impact of the number of clients per server

Clients per Server	2	4	8	16	32
Average Accuracy (%)	55.95	81.35	88.60	93.28	93.20
Standard Deviation (%)	11.00	7.32	2.30	0.15	0.03
Time to Reach 55% (ms)	35620	11880	10286	26506	77950

The results in table 3.3 show that when the total number of clients in the system is the same, the configuration with more clients on each server achieves better average accuracy in general, until from 16 to 32 clients per server we see that the average accuracy is saturated at around 93.20%. The standard deviation shows a more consistent trend of decline. Since every server has a committed group of clients, the server model's optimum largely depends on the local optimums of the clients' data. The case of fewer clients on a server leads to a more biased model since the clients' data are less comprehensive and thus makes it harder for the server model to move closer to the global optimum based on its own clients. And the difference among server models also increases because it is less possible for them to achieve similar data distributions with a fewer number of clients.

The time consumed to reach 55%, however, is not monotonically increasing or decreasing. This is because if the number of servers is too high, as mentioned before, it is harder for the global model to reach the global optima due to the more biased models among servers. On the other hand, if the number of servers is too low, meaning that the system architecture tends to FedAsync, computational power will be insufficient to handle clients' updates and cause the queuing phenomenon. Thus, it is very important to choose the appropriate number of servers in the system to achieve a balance in performance and converging time.

4 Conclusion

In this paper, we proposed a novel multi-server asynchronous federated learning approach. Our primary innovation involves integrating wait-free asynchronous mechanisms within a multi-server framework.

As we discussed in previous chapters, the prominence of the multi-server setup lies in its capacity for geo-replication, leading to substantial reductions in long-distance communication and the equitable distribution of computational loads across multiple servers. Furthermore, multi-server frameworks exhibit great scalability, accommodating varying client scales by allowing the dynamic adjustment of server numbers. Additionally, the presence of multiple servers in the system effectively mitigates the risk of single-point failures. On the other hand, asynchronous solutions also improve the efficiency of FL systems by circumventing the bottlenecks posed by synchronous communication. Leveraging the immediate aggregation strategy, asynchronous systems cleverly avoid long idle time on servers that is caused by network heterogeneity in clients.

Our proposed method, named MultiAsync, harnesses the pivotal advantages of both multi-server architectures and asynchronous systems. It involves two levels of asynchrony: client-server update and server-server model exchange. Client-server communication is markedly enhanced as clients are assigned to servers according to their locations. Simultaneously, asynchronous server-server aggregation provides the lowest time cost for servers reaching global consensus.

We undertake an extensive evaluation of our approach through both vertical and horizontal comparisons with three prominent baselines: FedAsync, HierFAVG, and FedAvg. This evaluation is carried out across three classical datasets: MNIST, CIFAR-10, and WikiText-2. The results garnered from our experimental analyses distinctly illustrate the impressive performance of our method in comparison to these baselines. In terms of horizontal comparison, our method outperforms all algorithms in convergence speed, demonstrating the swiftest rate of convergence among them. Furthermore, in the vertical comparison, our method also showcases top client scalability.

In conclusion, MultiAsync distinctly demonstrates its capability to elevate performance across diverse dimensions when evaluated against established benchmarks. Looking ahead, enhancing the system's robustness emerges as a practical avenue for further development. Additionally, we have earmarked the integration of clustering algorithms as a focal point for future endeavors, aiming to augment training effectiveness even further.

Bibliography

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282.

[2] T. Yang, G. Andrew, H. Eichner, *et al.*, "Applied federated learning: Improving google keyboard query suggestions," *CoRR*, vol. abs/1812.02903, 2018.

[3] S. Ramaswamy, R. Mathews, K. Rao, and F. Beaufays, "Federated learning for emoji prediction in a mobile keyboard," *CoRR*, vol. abs/1906.04329, 2019.

[4] A. Hard, K. Rao, R. Mathews, *et al.*, "Federated learning for mobile keyboard prediction," *CoRR*, vol. abs/1811.03604, 2018.

[5] M. Chen, A. T. Suresh, R. Mathews, *et al.*, "Federated learning of n-gram language models," *CoRR*, vol. abs/1910.03432, 2019.

[6] B. Yuan, S. Ge, and W. Xing, "A federated learning framework for healthcare iot devices," *CoRR*, vol. abs/2005.05083, 2020.

[7] J. Xu, B. S. Glicksberg, C. Su, P. Walker, J. Bian, and F. Wang, *Federated learning for healthcare informatics*, 2019. DOI: 10.48550/ARXIV.1911.06270. [Online]. Available: https://arxiv.org/abs/1911.06270.

[8] T. S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I. C. Paschalidis, and W. Shi, "Federated learning of predictive models from federated electronic health records," *International Journal of Medical Informatics*, vol. 112, pp. 59–67, 2018, ISSN: 1386-5056. DOI: https://doi.org/10.1016/j.ijmedinf.2018.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138650561830008X.

[9] N. Rieke, J. Hancox, W. Li, *et al.*, "The future of digital health with federated learning," *CoRR*, vol. abs/2003.08119, 2020.

[10] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, *Advances and Open Problems in Federated Learning*. 2021.

[11] M. Naseri, J. Hayes, and E. De Cristofaro, "Local and central differential privacy for robustness and privacy in federated learning," *arXiv preprint arXiv:2009.03561*, 2020.

[12] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[13] K. Bonawitz, V. Ivanov, B. Kreuter, *et al.*, *Practical secure aggregation for federated learning on user-held data*, 2016. arXiv: 1611.04482 [cs.CR].

[14] D. Byrd and A. Polychroniadou, "Differentially private secure multi-party computation for federated learning in financial applications," in *Proceedings of the First ACM International Conference on AI in Finance*, 2020, pp. 1–9.

[15] H. Zhu, J. Xu, S. Liu, and Y. Jin, "Federated learning on non-iid data: A survey," *CoRR*, vol. abs/2106.06843, 2021.

[16] Z. Chai, A. Ali, S. Zawad, *et al.*, "Tifl: A tier-based federated learning system," ser. HPDC '20, Stockholm, Sweden: Association for Computing Machinery, 2020, pp. 125–136, ISBN: 9781450370523. DOI: 10.1145/3369583.3392686. [Online]. Available: https://doi.org/10.1145/3369583.3392686.

[17] L. Li, H. Xiong, Z. Guo, J. Wang, and C.-Z. Xu, "Smartpc: Hierarchical pace control in real-time federated learning system," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, 2019, pp. 406–418. DOI: 10.1109/RTSS46320.2019.00043.

[18] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7. DOI: 10.1109/ICC.2019.8761315.

[19] B. Cox, L. Y. Chen, and J. Decouchant, *Aergia: Leveraging heterogeneity in federated learning systems*, 2022. DOI: 10.48550/ARXIV.2210.06154. [Online]. Available: https://arxiv.org/abs/2210.06154.

[20] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 15–24. DOI: 10.1109/BigData50022.2020.9378161.

[21] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *CoRR*, vol. abs/1903.03934, 2019.

[22] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, *Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation*, 2022. DOI: 10.48550/ARXIV.2205.13797. [Online]. Available: https://arxiv.org/abs/2205.13797.

[23] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, "On the convergence of federated optimization in heterogeneous networks," *CoRR*, vol. abs/1812.06127, 2018.

[24] X. Wang and Y. Wang, *Asynchronous hierarchical federated learning*, The network topology is a star graph, both between edge servers and the cloud server, and each edge server and the its client.<br/>1. The server runs a clustering algorithm (BASED ON?) to assign clients to their edge servers.<br/>2. The server sends out the the global model periodically with timestamps TO EDGE SERVERS.<br/>3. The edge server is elected from each client cluster, called aggregator.<br/>4. The aggregator is the parent node of each cluser, forwarding the information of center server to its children.<br/>, 2022. DOI: 10.48550/ARXIV.2206.00054. [Online]. Available: https://arxiv.org/abs/2206.00054.

[25] M. S. H. Abad, E. Ozfatura, D. GUndUz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8866–8870. DOI: 10.1109/ICASSP40776.2020.9054634.

[26] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148862.

[27] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, and J. Jiang, "Multi-center federated learning," *CoRR*, vol. abs/2005.01026, 2020.