Tackling Data Non-IIDness in Multi-server Asynchronous FL

Master's Thesis

Gefei Zhu

Delft University of Technology

TU Delft

Tackling Data Non-IIDness in Multi-server Asynchronous FL

Master's Thesis

by

Gefei Zhu

Student Name	Student Number
Gefei Zhu	5651727

Thesis Advisor:	Jérémie Decouchant
Daily co-supervisor:	Bart Cox
Project Duration:	Nov, 2023 - Aug, 2024
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science
Program:	Embedded Systems

TU Delft

Preface

Federated Learning (FL) has revolutionized machine learning by decentralizing the training process across multiple devices or servers, thereby addressing privacy concerns by ensuring that raw data remains localized. This decentralized framework, however, presents unique challenges when applied to real-world scenarios, particularly in environments characterized by heterogeneous data distributions (non-IID data) and varying computational resources.

This thesis is grounded in the study of an existing multi-server asynchronous FL architecture for geo-distributed clients, with a focus on understanding and mitigating the impacts of heterogeneous environments on system performance. Recognizing the complexity of real-world conditions, a key contribution of this work is the development of a reproducible multi-server FL model with heterogeneous resources. This model is designed to simulate various real-world scenarios, allowing for controlled experimentation and a deeper understanding of how different factors, such as network latency in geo-distributed settings and data distribution, affect FL systems.

Building on this robust experimental framework, we propose three innovative methods to mitigate the adverse effects of non-IID data distributions and asynchronous communication in FL systems: (1) dynamically reallocating clients between servers to enhance data balance and improve global model accuracy while reducing training time; (2) allowing clients to alternatively train two separate models from different servers, ensuring that each server receives tailored updates, which is particularly effective in large-scale client scenarios; and (3) having clients alternate between servers and average the models received before local updates, combining the strengths of both models, which is a method that has been outlined in related works and that we find to be less effective in certain conditions.

The research presented in this thesis demonstrates through extensive experimentation on datasets such as MNIST and CIFAR-10 that our methods significantly improve the efficiency and accuracy of FL systems under non-IID settings. These contributions pave the way for more scalable and efficient FL systems, capable of handling the complexities of real-world data distributions and communication constraints.

I would like to express my heartfelt thanks to my thesis advisor, Dr. Jérémie Decouchant, and my daily co-supervisor, Bart Cox. Their support throughout this project has been invaluable, offering both insightful theoretical guidance and much-needed encouragement. I am deeply grateful for their help, especially during challenging times, when their advice and understanding made a significant difference. Their assistance and mentorship have been greatly appreciated, and I feel fortunate to have had their guidance during this journey.

Gefei Zhu
Delft, August 2024

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
FL	Federated Learning
Non-IID	Non-Independent and Identically Distributed
IID	Independent and Identically Distributed
AWS	Amazon Web Services
FedSGD	Federated Stochastic Gradient Descent
FedAvg	Federated Averaging
Hier-FL	Hierarchical Federated Learning
Spyker	Asynchronous Multi-Server Federated Learning for Geo-Distributed Clients

Symbols

Symbol	Definition
N_s	Total number of servers
N_c	Total number of clients
S	A list to store all servers
C	A list to store all clients
N_{s_ic}	The number of required clients for server i
Avg_{s_i}	Average speed of all clients associated with server i
A_{s_i}	The area (location) of server i in the map
A_{c_i}	The area (location) of client i in the map
$Dist_{s_i,c_j}$	The communication delay between server s_i and client c_j
S_{s_i,c_j}	Speed of client j of server i (training delay)
D_{s_i,c_j}	Communication delay (latency) between server s_i and client c_j
L	The set of labels of dataset

1

Research Paper

Tackling Data Non-IIDness in Multi-server Asynchronous FL

Gefei Zhu
Delft University of Technology

***Abstract*—Federated Learning (FL) is a distributed machine learning approach that enhances data privacy by training models across multiple devices or servers without centralizing raw data. Traditional FL frameworks, which rely on synchronous updates and homogeneous resources, face significant performance challenges in real-world deployments. These challenges include handling heterogeneous data distributions (non-IID) and varying computational resources across clients, leading to performance degradation. To address these issues, asynchronous FL frameworks have been proposed, but they introduce new complexities such as communication latency and workload management across geo-distributed servers.**

This paper focuses on the impact of network latency and non-IID data distributions on asynchronous multi-server FL systems. We propose and evaluate three methods to mitigate the adverse effects of data heterogeneity on model accuracy: (1) transferring clients between servers; (2) sharing clients among servers so that they alternatively train their models; and (3) sharing clients among servers with model averaging. Our contributions include a reproducible experimental framework for multi-server FL, strategies for optimizing client-server interactions, and an analysis of the effectiveness of these strategies in reducing the impact of non-IID data distributions. Experimental results demonstrate that our methods can reduce training time by up to 85.67%, decrease the number of updates required by 85.82%, and improve accuracy by 4.815% in heterogeneous environments, compared to Spyker, a state-of-the-art multi-server FL algorithm.

***Index Terms*—Federated Learning, Non-IID, Heterogeneous Clients, Geo-Distributed Systems**

I. Introduction

Federated Learning (FL) [1] is a distributed machine learning approach where a global model is collaboratively trained across multiple devices or servers holding local data samples. This technique enhances data privacy and security over centralized machine learning by ensuring that the raw data remains local and that only model updates are shared with a central server. FL frameworks initially assumed a single server, a synchronous training process, and homogeneous client resources. This early framework relied on the FedAvg algorithm [2].

However, in practical deployments, clients often have heterogeneous computational resources and possess data that differ in both quantity and class distribution. Heterogeneous environments with varying computational resources and data distributions among clients can negatively impact performance [3]. In recent years, extensive research has been addressing the non-independent and identically distributed (non-IID) data problem in FL along three directions. First, data optimization, [4] focuses on preprocessing and restructuring data to make it more balanced across different devices or sharing portions of data. Second, model update optimization, [5]–[8] improves the aggregation process of model parameters, reducing biases. Finally, model training optimization, [9]–[11] adjusts strategies during the training process to improve the generalization and stability of models.

In addition to non-IID datasets, heterogeneous clients, such as slow clients, can significantly impact the training process by extending the overall training time, causing the ”straggler effect”, and leading to imbalanced contributions to the global model. Slow clients, due to lower processing power or poor network connectivity, take longer to compute their local updates [12]. Since the server often waits for all clients to finish their updates before proceeding, these slow clients delay the entire training process. This effect is exacerbated in heterogeneous environments where there is a wide disparity in client capabilities, leading to inefficient use of resources and prolonged training times. Asynchronous FL training schemes have been developed to mitigate these negative effects [13]–[15]. Asynchronous FL allows clients to send updates independently, reducing dependency on slow clients and improving resource utilization. This approach decreases overall training time by aggregating updates as they arrive. However, asynchronous FL still faces performance-related challenges, such as the impact of long-distance communication on performance and potential queuing delays due to limited computational power on a single server. These factors can affect timely update aggregation and system performance, necessitating robust strategies to manage communication latency and server workload.

To address the challenges introduced by asynchronous FL, several FL algorithms have been developed. Hierarchical Federated Learning (Hier-FL) [16]–[18] introduces intermediate aggregation layers (e.g., edge servers) to reduce the load on the central server and manage long-distance communication more efficiently. Moreover, decentralized federated learning [19], [20] eliminates the central server, with clients communicating and aggregating updates in a peer-to-peer manner. This reduces the central server’s computational load and network congestion, enhancing overall system efficiency.

In this paper, we aim to analyze how network latency, influenced by the geographical distribution of servers and clients, affects the performance of multi-server asynchronous FL. Additionally, we aim to simulate non-IID data distribution

conditions to conduct a comprehensive study of real-world data diversity. Moreover, we aim to ensure the ability to control variables during experiments, enabling us to conduct reproducible experiments. Last, we aim to decrease the impact of heterogeneous resources to ensure a robust and efficient Asynchronous Multi-server FL.

In summary, this paper makes the following contributions:

- We provide a methodology for reproducible experiments for multi-server Federated Learning systems. Our methods support heterogeneous datasets and resources.
- We propose three methods to reduce the impact of non-IID data on training accuracy in those systems. First, we select suitable clients from servers and move them to other servers (Move-Clients approach). Second, we share clients among servers with two different methods (Share-Clients approach).
- We provide an algorithm for selecting the appropriate clients for our Move-Client and Share-Clients methods.
- We evaluate the Move-Client and Share-Clients methods with non-IID scenarios.

This paper is organized as follows. In section II, we provide an overview of Federated Learning (FL) systems, discuss their advantages, limitations, and the challenges associated with their deployment in real-world scenarios. In section III, we introduce a reproducible multi-server FL model that allows for controlled experimentation with non-IID scenarios at the server and client levels. In section IV, we propose three novel methods to address the challenges of Non-IID data distributions and multi-server asynchronous FL. In section V, we conduct a comprehensive performance evaluation of the proposed methods. In section VI, we review related work in the field of Federated Learning, highlighting how our contributions build upon and differ from existing studies. Section VII concludes this paper by summarizing the findings of our study, discussing the implications of our results.

II. Background

In this section, we first introduce the most conventional and widely recognized federated system frameworks. Then, we review three different but typical multi-server FL models. Finally, we present a multi-server asynchronous FL system model, which forms the basis for our study.

A. Federated Learning

McMahan et al. [1] proposed the first single-server synchronous federated learning framework, introducing two key algorithms, FedSGD and FedAvg, and validating their effectiveness through experiments.

The experimental environment used in [1] is a controlled environment, and assumes a perfectly IID data distribution:

- Experiments use K clients, and the client datasets are IID;
- At the beginning of each round, $C \times K$ clients $(0 \leq C \leq 1)$ are selected (experiments have shown that the effectiveness declines when the number of clients exceeds a certain value, so only a subset of clients is chosen). The server sends the global model to the selected clients, who then perform federated learning (including training and federated aggregation).

Selecting $C = 1$ and using SGD for federated learning [1] defines this baseline as Federated SGD (FedSDG) in this paper. The local update for client k is:

$$\mathbf{w}_{t+1}^{k} = \mathbf{w}_t - \eta \nabla F_k(\mathbf{w}_t)$$

where η is the learning rate and ∇F_k is the gradient of the local objective function at $\mathbf{w}_t$. The server collects the local updates and aggregates them to form the new global model:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} \nabla F_k(\mathbf{w}_t)$$

where n_k is the number of data samples at client k, and n is the total number of data samples across all clients. To enhance the computation of each client, the authors propose Federated Averaging (FedAvg). Each client k performs multiple steps of gradient descent using their local data. Let W_t be the global model parameters at round t, The local update after E epochs for client k is:

$$\mathbf{w}_{t+1}^{k} = \mathbf{w}_t - \eta \sum_{i=1}^{E} \nabla F_k(\mathbf{w}_t^i)$$

where $\mathbf{w}_t^i$ represents the model parameters after the ith local epoch. The server collects the local models and averages them to form the new global model:

$$\mathbf{w}_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} \mathbf{w}_{t+1}^{k}$$

where n_k is the number of data samples at client k, and n is the total number of data samples across all clients.

The advantage of this method is that clients can perform multiple parameter updates before sending their parameters, thereby increasing their computational workload. FedSGD is a synchronous algorithm (requiring all clients to wait until all gradients are computed), whereas FedAvg is not fully synchronous (clients can adjust the number of computations performed, though some synchronization is still required).

B. Multi-Server Federated Learning

Multi-server Federated Learning architectures have been introduced to address inherent challenges in single-server setups. The motivation arises from the need to mitigate communication latency between the server and clients, especially in delay-sensitive applications. There exist three primary categories of multi-server approaches, which comprise: Hierarchical Federated Learning (HFL) [16]–[18], [21], Clustered Federated Learning [22]–[24] and Multi-server Overlapping Federated Learning [25], [26].

Hierarchical FL adopts a layered structure that allows multiple edge servers to perform partial model aggregation, enabling hierarchical model aggregation. The advantage lies in leveraging the proximity of intermediate edge servers to reduce communication costs through aggregation on the edge servers.

However, its drawback is the challenge posed by non-IID data. In scenarios involving non-IID data, Hierarchical FL may fail to achieve the desired level of accuracy in certain situations, such as when there is a significant disparity between the edge and the cloud or when multiple edge servers are involved.

Clustered FL is an approach that divides customers into different clusters and trains a separate machine learning model for each cluster. Since each cluster trains its own model independently, there is a reduction in the amount of communication needed between clusters, potentially lowering communication overhead. However, the frequent re-clustering of clients can elevate overall training complexity and duration. Additionally, certain Clustered FL implementations may neglect physical network connectivity constraints, restricting a client's connection to only a subset of servers. This oversight can hamper the efficiency of communication between clients and servers. Moreover, the requirement to handle distinct models for each cluster adds complexity to tasks such as model coordination and maintenance.

Multi-server Overlapping FL leverages a distributed infrastructure comprising multiple servers with intersecting coverage areas, facilitating efficient client localization. The key idea demonstrated by [25] is that clients download multiple models from all the edge servers they can access and train their local models based on the average of these models and send their updated model to multiple edge servers by broadcasting. Consequently, the suggested approach obviates the need for expensive communication with the central cloud server to achieve model synchronization. This results in a substantial reduction in the overall training time when compared to traditional cloud-based federated learning systems. However, [25] introduced a strongly-convex loss function, a highly restrictive choice given that the majority of learning models, such as neural networks, are non-convex. Moreover, the convergence results fail to demonstrate the implications on overlapping areas. Hence, [26] improves the architecture, introduces a new algorithm with two-sided learning rates and provides theoretical convergence analysis of the more general non-convex loss function. It achieves a linear speedup under full/unbiased partial client participation strategies compared to the existing multi-server FL algorithms.

C. Multi-Server Asynchronous FL

Most multi-server Federated Learning (FL) systems are synchronous [16]–[18], [21]–[24], [26]. In scenarios where clients are geographically distributed, communication delays and bandwidth consumption between clients and the central server are significantly high. This issue is particularly pronounced in synchronous FL systems, where these delays can severely impact the convergence speed of the model.

To address these limitations, Spyker [27] introduces a novel FL architecture that leverages multiple servers and asynchronous communication. Spyker, as shown in Fig.1, allows clients to interact with their nearest server, which minimizes communication latency and ensures efficient integration of updates. Unlike traditional hierarchical FL frameworks that still rely on synchronous procedures for global model updates, Spyker enables fully asynchronous interactions not only between clients and their nearest server but also among the servers themselves. This approach significantly reduces idle times and accelerates model convergence, especially in geo-distributed settings. Spyker defines new mechanisms to handle model staleness and update aggregation, ensuring that model updates are effectively weighted based on their relevance and the timing of their arrival. Specifically, in Spyker, clients receive the global model W^t from their nearest server, perform local training using the formula

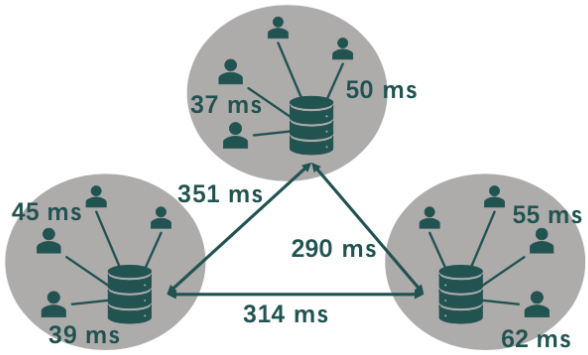


Fig. 1. The architecture of Spyker [27]

$$W_k^{t+1} = W^t - \eta_k \nabla F_k(W^t),$$

where W_k^{t+1} is the updated model at client k after local training, W^t is the global model received from the server, η_k is the learning rate for client k, and $\nabla F_k(W^t)$ represents the gradient of the loss function F_k with respect to the model W^t on the local dataset of client k.

Clients send back updates asynchronously. This allows continuous operation without waiting for other clients. When the server receives an update from a client, it incorporates this update using the aggregation formula

$$W_i^{t+1} = W_i^t + \eta_i \cdot weight \cdot (W_k^{t+1} - W_i^t),$$

where W_i^{t+1} is the updated server model after aggregating the client update, W_i^t is the current server model before aggregation, η_i is the server's learning rate, W_k^{t+1} is the model update received from client k, and $weight$ is a factor that may include considerations for model staleness or client importance.

The server also updates the age of its model as

$$A_i^{t+1} = A_i^t + 1,$$

where A_i^{t+1} is the updated age of the server model after incorporating the client's update, and A_i^t is the current age of the server model.

Additionally, Spyker employs a token-based strategy to manage server-to-server communications, which helps maintain synchronization without introducing bottlenecks. Servers periodically exchange their models asynchronously using a token-based mechanism to trigger model exchanges. This prevents simultaneous model broadcasts and ensures that server models are synchronized effectively. When a server i receives a model W_j from another server j, it updates its model using the formula

$$W_i^{t+1} = W_i^t + \eta_a \cdot w_{ij} \cdot (W_j^t - W_i^t),$$

where W_i^{t+1} is the updated server model after aggregating the model from server j, W_i^t is the current model of server i, η_a is the aggregation rate, W_j^t is the model received from server j, and w_{ij} is a weight computed based on the relative ages of the models from servers i and j.

The server also adjusts the age of its model according to

$$A_i^{t+1} = (1 - \eta_a \cdot w_{ij}) A_i^t + \eta_a \cdot w_{ij} \cdot A_j,$$

where A_i^{t+1} is the updated age of the server model after aggregating the model from server j, A_i^t is the current age of the server model, A_j is the age of the model received from server j, η_a is the aggregation rate, and w_{ij} is the weight based on the ages of the models. This process ensures that updates are weighted based on the staleness and relevance of the models being aggregated, maintaining the effectiveness and efficiency of the FL system.

The primary objective of this paper is to improve over Spyker's accuracy. By addressing the limitations in its latency calculation method and improving the precision of data distribution to better reflect real-world scenarios, we aim to achieve more reliable and accurate model performance in geo-distributed settings. Our research focuses on refining the data resource heterogeneity handling and optimizing the asynchronous communication among the whole system to ensure a robust and efficient FL system.

III. Reproducible Multi-Server Federated Learning with Heterogeneous Resources

In this section, we first examine the impact of network latency and present a more accurate method for calculating communication delays based on the geographical distribution of servers and their clients. Then we provide a realistic data distribution. Finally, we construct a reproducible multi-server FL model to control experimental variables.

A. Impact of network latency and client distribution

1) Network latency mapping: The communication delays between clients and servers, and the affectation of clients to servers can significantly impact the experimental outcomes of multi-server FL.

CloudPing monitors the latency between Amazon Web Service (AWS) sites [28]. In theory, the proximity of servers is anticipated to exert an influence on the communication delay between them. Geographic distance is a pivotal factor in this regard. The latency observed between AWS regions in close proximity is considerably lower when compared with regions that are geographically dispersed. However, numerous other variables that may impact communication delay will not be taken into consideration in these experiments.

Hence, it is necessary to construct a map based on communication delays, which involves establishing a matrix of communication delays to represent distances between locations. According to the region of the provided servers, six locations are chosen as shown in Tab. I. The chosen locations are then mapped to coordinates, ensuring that locations with shorter communication delays are closer on the map.

TABLE I
Communication delays between servers according to CloudPing

From\To	Hong Kong	Tokyo	Sydney	Canada	London	California
Hong Kong	3.04	50.92	130.49	200.7	196.92	153.27
Tokyo	51.82	1.61	109.53	157.2	225.27	110.4
Sydney	130.65	109.18	4.24	198.25	266.31	139.71
Canada	202.62	157.04	199.75	2.42	78.47	79.26
London	205.89	226.41	268.6	78.75	0.88	147.81
California	152.52	111.05	139.08	79.54	148.45	4.43

Nevertheless, there are minor discrepancies in communication delays between each pair of locations in both directions. In order to accurately map these communication delays onto the geographical map, it is imperative that the distances between two locations remain consistent. Therefore, the communication delays matrix must be symmetric. To achieve this symmetry, the disparities in communication delays for the bidirectional link between two locations need to be addressed. An effective approach is to compute the average of the two distinct delays, allowing for the construction of a symmetric matrix that accurately represents the communication delays across the network. Moreover, the distance between the same location should be zero so that the values on the diagonal of the matrix should be zero as shown in Tab. II .

TABLE II
Communication delays matrix

From\To	Hong Kong	Tokyo	Sydney	Canada	London	California
Hong Kong	0	51.37	130.57	201.35	201.41	152.90
Tokyo	51.37	0	109.36	157.12	225.84	110.73
Sydney	130.57	109.36	0	199	267.46	139.4
Canada	201.35	157.12	199	0	78.61	79.4
London	201.41	225.84	267.46	78.61	0	148.13
California	152.90	110.73	139.4	79.4	148.13	0

This mapping is achieved using the Multi-Dimensional Scaling (MDS) method [29], which projects communication delays onto coordinates in a two-dimensional plane. The objective is to visually represent the spatial arrangement of locations on the map, reflecting their connectivity and communication efficiency.

In order to reduce latency, minimize network congestion and optimize resource utilization, it is essential for each client to efficiently identify and interact with the nearest server. In order to achieve this goal, we need to partition the map obtained

by the MDS method based on the locations of servers. This partitioning ensures that the distance from clients in these regions to their respective servers is minimized. The Voronoi diagram [30] divides a plane into multiple regions based on the coordinates provided by the MDS method (the positions of servers), where each region contains points that are closest to the corresponding server.

The Voronoi diagram is shown in Fig. 2. The blue points represent the location of each server and the polygon formed by dashed and solid lines around the blue dot corresponds to the region of this server. After determining the number of servers, ranging from 1 to 6, each server can randomly choose a unique blue point as its location.

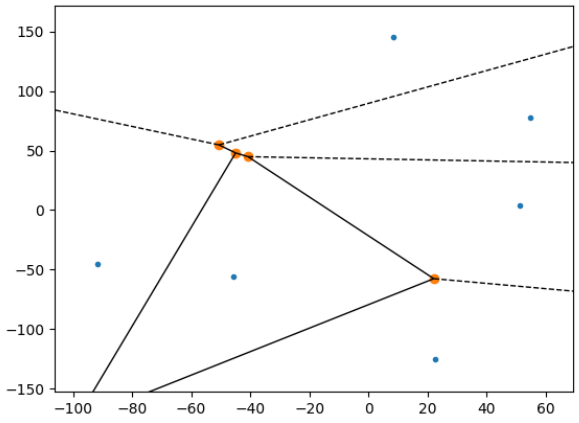


Fig. 2. The Voronoi diagram

2) Client distribution: After the completion of the map construction, the clients can be created. The number of total clients is determined first, and then it needs to be decided whether each server should be allocated an equal number of clients or whether the allocation should follow a normal distribution. Each client is randomly assigned a location within the corresponding region of each server. To ensure the distance from each clients to its associated server is minimized, a KD-tree is established. This method guarantees that each client is efficiently identified and associated with its nearest server. Fig. 3 illustrates the distribution of 8 clients for each of 4 server on the Voronoi diagram.

B. Impact of non-IID server datasets

1) Dirichlet distribution for generating Non-IID scenarios: In constructing non-IID scenarios for federated learning experiments, the Dirichlet distribution offers a highly adaptable means to mimic the diverse data distributions found across different network nodes or clients. This adaptability is crucial for simulating a broad spectrum of real-world situations, where each client may possess varying quantities and types of data.

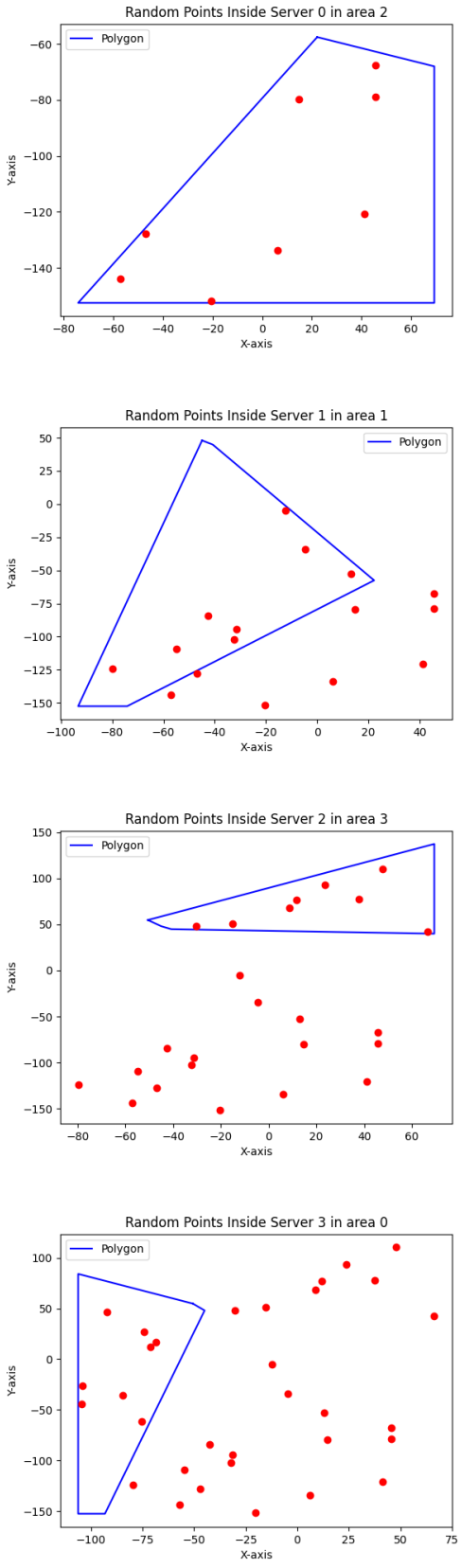


Fig. 3. The distribution of clients

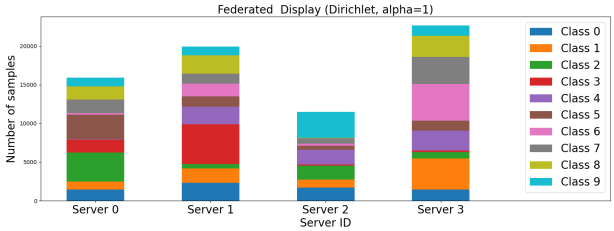


Fig. 4. The dataset distribution at the server level

Utilizing the Dirichlet distribution, the dataset for each client is partitioned in such a way that it reflects the potential real-life variability in data across clients.

In the Dirichlet distribution, the parameter α controls the shape of the distribution, which directly influences the degree of data heterogeneity among the clients. The density function [31] is given by:

$$f(x_1, ..., x_K; \alpha_1, ..., \alpha_K) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

When all the α_k are very large and approximately equal (i.e., $\alpha_k = \alpha$ for all k), and $x_i \in$ [0,1], we can observe that:

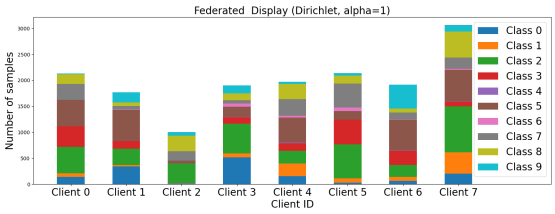
- each $x_i^{\alpha-1}$ in the probability density function approaches a constant because the value of x_i tends towards $\frac{1}{K}$.
- the formula for variance indicates that the dispersion of each X_k decreases.
- therefore, this distribution approaches a uniform distribution over the hypercube $[0, 1]^K$.

As all α_k tend towards infinity, the Dirichlet distribution approaches a uniform distribution. To evaluate the influence of different alpha values on the models' performance, we would systematically vary the alpha parameter of the Dirichlet distribution that governs the data distribution across clients and servers. This examination would reveal how the concentration parameter alpha, which controls the degree of similarity among the distributed data subsets, affects the convergence speed, accuracy, and overall learning efficiency of the models. By comparing the models' performance across a range of alpha settings, we can determine the robustness of the models to varying degrees of data heterogeneity.

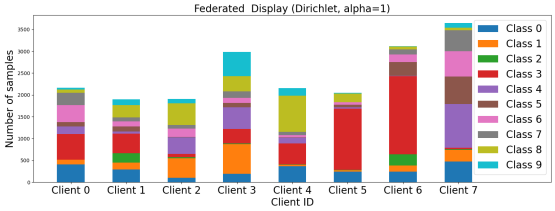
As an example, let us consider a scenario where both the server-level and client-level datasets exhibit non-IIDness. In this setup, there are 4 servers, each managing eight clients, and the dataset used is MNIST. The distribution of datasets at both the server and client levels follows a Dirichlet distribution with an alpha value of 1.0. Fig. 4 shows the dataset distribution at the server level for each server, while Fig. 5(a) represents the dataset distribution at the client level for each server.

C. Experimental algorithms and models

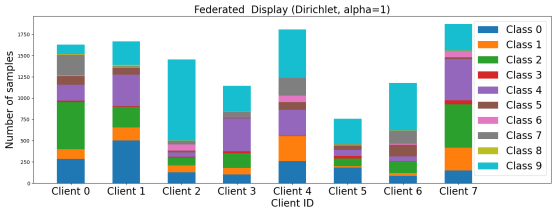
To conduct more experiments to investigate the impact of non-IID server sets, controlling variables is necessary. We implement an experimental algorithm, Alg. 1, to build up the



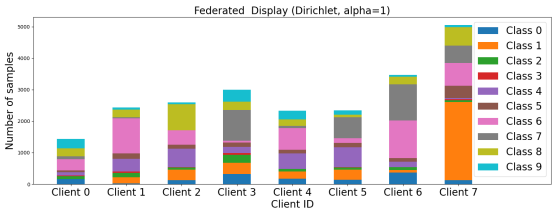
(a) Dataset distribution for server 0's clients



(b) Dataset distribution for server 1's clients



(c) Dataset distribution for server 2's clients



(d) Dataset distribution for server 3's clients

Fig. 5. The dataset distribution at the client level

reproducible multi-server FL model. Tab. III summarizes our notations.

In this model, all controllable parameters are stored in a **Config.json** file which can be read and written. Before starting the experiment, the user should determine the configuration parameters. These parameters set the various behaviors of the federated learning system, including communication policies, optimizer settings, number and behavior of clients and servers, selection and distribution of datasets, etc.

The BUILDINGMODEL procedure (Alg. 1, ll. 1–11) shows the model building process, determining whether to instantiate servers with their clients, and initiate experiments based on the parameters specified in the **Config.json** file. The N_s, Avg_s, N_c, S_{IID}, C_{IID}, $Dataset$, $Flag$, α_s, α_c can all be read in **Config.json** file.

This procedure serves as the entry point for setting up the federated learning environment. It first reads the configuration

TABLE III
NOTATIONS AND PARAMETER VALUES FOR ALG. 1

Symbol	Description
S	A list to store all servers
C	A list to store all clients
N_s	Total number of servers
N_{s_ic}	The number of required clients for server i
Avg_{s_i}	Avg speed of all clients associated to the server i
$Speeds\_S_i$	A list to store all training delay of server i's clients
N_c	Total number of clients.
S_{IID} (True/False)	Indicates whether the dataset distribution among servers's clientsets is IID or Non-IID.
C_{IID} (True/False)	Determines whether the data distribution for each client within a clientset is IID or Non-IID.
Pct_{s_i}	The Percentage of clients owned by server i relative to total number of clients
A_{s_i}	The area (location) of server i in the map
A_{c_i}	The area (location) of client i in the map
$Dist_{s_i,c_j}$	The communication delay between server s_i and client c_j
Ss_i, c_j	Speed of client j of server i (training delay).
D_{s_i,c_j}	Communication delay (latency) between server s_i and client c_j.
L_{c_i}	The set of labels associated with client c_i
L_{s_i}	Represents the collective set of labels for server s_i's clientset
α_s	Represents the parameter of Dirichlet distribution for server level
α_c	Represents the parameter of Dirichlet distribution for client level

file to retrieve necessary inputs. The boolean flag (Alg. 1, ll. 3) retrieved from the configuration file represents whether the experiment is being conducted for the first time. When this flag is set to true, it means that the experiment is being conducted for the first time. In this case, the procedure proceeds to create servers and clients using the CREATESERVER function (Alg. 1, ll. 12–25) and initializes the experiment. After creating servers and determining their properties, the training delay of each server's clients will be calculated using the CALCULATESPEED (Alg. 1, ll. 39–50)) function and be written into the configuration file. Then, the flag is updated to false to indicate that the servers and clients are already configured for subsequent iterations of the experiment. Conversely, if the flag is equal to false, it implies that the experiment has already been set up previously. In this scenario, the procedure skips the server creation step and directly executes the experiment with the existing configurations.

The CREATESERVER function (Alg. 1, ll. 12–25) is used to create servers and obtained each server's clients' parameters. First, it will create a Voronoi diagram and determine the locations that servers can be located, as the blue points shown in the Fig. 2 based on the required number of servers (ll. 14).

If the data distribution method between servers is chosen to be the same (S_{IID} is true), the data is equally distributed to each server; otherwise, it is distributed according to the Dirichlet distribution based on the input α_s (ll. 15–38). The number of clients is determined by the percentage of clients owned by server relative to total number of clients Pct_s(ll. 19).

For each server, an area number A_s in the Voronoi diagram (ranging from 0 to the total number of servers), the number of clients N_{s_ic} each server should have, the dataset S_{i_Data} for each server's client set, and the average update frequency L_{s_i} of each server's client set are assigned. Clients are created using the CREATECLIENT function, and a server instance is spawned, recording the relevant parameters. Then, the information of each server's location, the number of required clients are written into the configuration file. Moreover, the dataset of each server is recorded by writing its data labels and indexes into the file (ll. 20–24).

Algorithm 1 Reproducible Multi-Server FL Model

Input: **Config.json**
Output: S (servers with clients and dataset), C (clients)
1: **procedure** BUILDINGMODEL
2: Read **Config.json** file to get inputs
3: **if** $Flag$ **then** ▷ If true write config, otherwise return
4: CREATESERVER(N_s, L_s, N_c, S_{IID}, C_{IID}, Pct_{s_i}, α_s, α_c)
5: **for** $i = 0$ **to** N_s **do**
6: $Speeds_S_i \leftarrow$CALCULATESPEED(N_c, C, S_i, Avg_s)
7: Write $Speeds\_S_i$ to **Config.json**
8: Write $Flag \leftarrow false$ to **Config.json**
9: Run the experiment
10: **else**
11: Run the experiment
12: **function** CREATESERVER(N_s, L_s, N_c, S_{IID}, C_{IID}, Pct_{s_i}, α_s, α_c)
13: $S \leftarrow \{\}$
14: $Region \leftarrow CreateMap(N_s)$
15: **if** S_{IID} **then**
16: $S_{Data}(i) \leftarrow EqualSplit(Dataset, N_s)$
17: **else**
18: $S_{Data} \leftarrow DirichletSplit(Dataset, N_s, \alpha\_s)$
19: $N_{s_c}[i] \leftarrow N_c * Pct_{s_i}$
20: **for** $i = 1$ **to** N_s **do**
21: $S_i \leftarrow N_{s_c}[i], random\ area\ A_{s_i}, S_{Data}[i], L_s[i]$
22: $S_{i\_C} \leftarrow CreateClient(S_i, N_{s_i,c}, S_{i\_Data}, Region[i], \alpha\_c)$
23: $S[i] \leftarrow S_i$
24: Write $N_{s_i,c}, L_{s_i}$ dataset distribution to **Config.json**
25: **return** S
26: **function** CREATECLIENT($S_i, N_{s_ic}, S_{i\_Data}, Region[i], \alpha\_c$) ▷ For S_i
27: $C \leftarrow \{\}$
28: **if** C_{IID} **then**
29: $C_{Data} \leftarrow EqualSplit(S_{i\_Data}, N_c)$
30: **else**
31: $C_{Data} \leftarrow DirichletSplit(S_{i\_Data}, N_c, \alpha\_c)$
32: **for** $j = 1$ **to** N_c **do**
33: $location = Random\_location(Region[i])$
34: $A_{c_j} \leftarrow KDTreeCheck(location)$

35: $C_{j\_Data} \leftarrow C_{Data}(j)$
36: $C[j] \leftarrow Client(C_{j\_Data}, A_{c_j})$
37: Write $A_{c_j}, C_{j\_Dataset}$ to **Config.json**
38: **return** C
39: **function** CALCULATESPEED[FOR S_i](N_c, C, S_i, Avg_s)
40: $Speed\_All \leftarrow \{\}$
41: $A \leftarrow Matrix\_size(N_c,\ \#\ of\ dataset\ classes)$
42: $B \leftarrow Matrix\_size(\#\ of\ dataset\ classe)$
43: **for** $j = 1$ **to** N_c **do**
44: $Dists_i, c_j \leftarrow CalculateDist(S_i, C[j])$
45: $LCc_j \leftarrow C_j$ dataset labels
46: **for** $label, count$ **in** $Count(LCc_j)$ **do**
47: $A[label][j] \leftarrow count * (1 + 2 * Dists_i c_j)$
48: $B \leftarrow Avg_{s_i} * LS_i$
49: $Ss_i, c_j \leftarrow lstsq(A, B)$
50: **return** Ss_i, c_j

The CREATECLIENT function (Alg. 1, ll. 26–38) is used to create clients for each server. A specified number of client instances are generated for each server based on the provided dataset and map location information. Based on the chosen data distribution method among clients, the dataset is either equally partitioned or distributed to each client according to a Dirichlet distribution(ll. 28–31). For each client, a position is randomly selected, and the location will be checked by the building KD_tree to ensure the client is in the server's region. Then, a client instance is generated and the locations and the dataset distribution information will be recorded into the file (ll. 32–37).

The CALCULATESPEED function (Alg. 1, ll. 39–50) is used to calculate the training delay of each server's clients. The training delay is defined below:

$$\sum_{j=1}^{N} \left((Ss_i c_j + 2 \times Dist_{s_i,c_j} + 2) \times LC_{s_i,c_j} \right) = (Avg_{si} \times LS_i) \tag{1}$$

The rationale behind this equation lies in ensuring a balanced allocation of computational resources in a federated learning setting. $Ss_i c_j$ means the training delay (speed) of client j associate with server i. The total consumption time (speed) for one client is the sum of communication delay between server and clients and the its training delay. The communication delay from the server to the client is marginally greater than that from the client to the server, consistently measured at 2 ms. The total updates of each clients per unit of time is determined by its labels after dataset distribution. The sum of all client updates $((Ss_i c_j + 2 \times Dist_{s_i c_j} + 2) \times LC_{s_i c_j})$ is equal to the total updates server should receive on each label (server's average speed multiply total labels). From this equation, we can get the training delay of each clients.

In order to solve this linear equation, we utilize the Least Squares Method to solve the matrix equation $AX = B$ for the unknown vector X. First, iterating over each client, this method constructs the matrix A by calculating the weights and constant terms obtained from the number of labels in each client's training data and the communication latency which is obtained by transforming $Dist_{s_i c_j}$ into communication delays. Next, it computes the target vector B by tallying the number of labels in the entire server-level training data and utilizing the server-level speed. Then, it invokes the numpy.linalg.lstsq function to solve the matrix equation $AX = B$, obtaining the solution vector X, which represents the training time for each client.

IV. IMPROVED CLIENT-SERVER PAIRING

Due to the negative impact of heterogeneous resources on the performance of Multi-Server Asynchronous FL, we propose three solutions in this section to mitigate these effects and improve accuracy.

A. Overall ideas

The overall idea is to achieve a better data distribution balance. To achieve this, we developed two methods: Move-clients, where selected clients are relocated to communicate with different servers, and Share-clients, where clients alternate communication between their original server and another server.

In addition, to further enhance the effectiveness of our approach, we propose a client selection method specifically designed to identify and select clients that can most effectively balance the data distribution across servers. This method strategically chooses clients whose data characteristics, when shared or moved, will contribute the most to reducing the impact of non-IIDness on the overall model accuracy.

B. Clients-selection

Before starting to select clients, we first need to evaluate the non-IIDness of each server. We will select servers with more balanced data distribution and choose appropriate clients from these servers to perform operations. The servers with more severe non-IIDness, referred to as 'target servers', suffer from greater data distribution bias, leading to models that may overfit to local data and perform poorly on the global model. By introducing clients from other servers, these target servers can access a more diverse data distribution, reducing data bias and balancing model training.

Algorithm 2 illustrates a process for selecting clients. Table IV detais our notations. The whole selection process consists of the following main steps:

Initialization (ll. 1-17): In this stage, the system explores all possible client and server configurations to find an initial configuration that performs the best in the current performance evaluation. Specifically, the system starts by initializing variables such as the step counter and the current best performance value. Then, it iterates through all servers (excluding the target server S_target) and examines the client sets of each server to find suitable clients. For each client, the system generates a tmp_config and evaluates its tmp_perf. If a valid initial configuration is found ($tmp\_perf \neq -\infty$), the system sets it as the cur_config and stores its performance value as the cur_perf. If no valid initial configuration is found, the system raises an error.

Iterative Optimization (ll. 18-32): After identifying the initial configuration, the system enters the iterative optimization stage. During this stage, the system continuously attempts to generate new configurations based on the previously identified best configuration by randomly selects a source server S_source and then randomly selects a client C from that server. It then generates a new_config and evaluates its performance (new_perf). If the new_perf is better than the current best performance ($new\_perf \geq cur\_perf$), the system updates the current configuration and the current best performance value. This process continues until the maximum number of iterations max_step is reached.

Performance Evaluation (ll. 33-38): In each configuration optimization, the system evaluates the overall performance of the client move or share operation based on the non-IIDness of the data and the global system speed. The global speed is calculated through the function CALCULATE_GLOBAL_SPEED (ll. 49-67). The CALCULATE_GLOBAL_SPEED function uses the move flag to determine whether the client is being moved to a different server or sharing its data with another server. If $move = True$, the client is temporarily moved to the target server, and the speed is calculated based on this new configuration before the client is returned to its original server. If $move = False$, the client shares its data with the target server while remaining with its original server, and the speed is calculated with the client contributing to both servers. If the global speed is greater than 0, the system returns a combined performance value, which is a weighted sum of the non-IIDness and the global speed (ll.38). The $non\_iidness + coefficient * speed$ plays a crucial role in balancing the impact of non-IIDness and global speed in the performance evaluation. If the global speed is less than or equal to 0, the system returns an invalid performance value ($-\infty$) to ensure that this configuration is ignored in subsequent optimizations

TABLE IV
NOTATIONS AND PARAMETER VALUES FOR ALG.2

Symbol	Description
S_target	Target server
S_source	A list to store all the servers who has more balanced data distribution than S_target
$move$	A flag. If true, the algorithm is for Moving-clients strategy. If false, for Share-clients strategy
$cur(tmp)\_config$	Current (temporary) configuration, which includes the target server ID and the selected client ID along with its associated server's id
$cur(tmp)\_perf$	Current (temporary) performance represents the calculated evaluation value for the corresponding configuration.

Algorithm 2 Client selection

Input: S_target $move$ max_step
Output: cur_config

1: **procedure** INITIALIZE(S_target)
2: $step \leftarrow 0$
3: $move \leftarrow True$
4: $cur\_config \leftarrow None$
5: $cur\_perf \leftarrow -\infty$
6: **for** $all(S\ in\ servers \wedge \neq S\_target)$ **do**
7: **for** $all(C\ in\ S's\ clientset)$ **do**
8: $tmp\_config = Config(S, S\_target, C)$
9: $tmp\_perf = Evaluate\_perf(tmp\_config, move)$
10: **if** $tmp\_perf \neq -\infty$ **then**
11: $cur\_config \leftarrow tmp\_config$
12: $cur\_perf \leftarrow tmp\_perf$
13: **break**
14: **if** $cur\_perf \neq -\infty$ **then**
15: **break**
16: **if** $cur\_config = None$ **then**
17: raise ValueError("No valid initial configuration found")
return cur_config, cur_perf
18: **procedure** OPTIMIZEITERATIVELY(S_target,$max\_step, move$)
19: $step \leftarrow 0$
20: cur_config, cur_perf = INITIALIZE(S_target)
21: **for** $step \leq max\_step$ **do**
22: randomly choose one S as S_source
23: randomly choose one C from S_source as C
24: $new\_config = Config(S, S\_target, C)$
25: new_perf = EVALUATE_PERF($new\_config, move$)
26: **if** $new\_perf == -\infty$ **then**
27: $step = step + 1$
28: **continue**
29: **if** $new\_perf \geq cur\_perf$ **then**
30: $cur\_config = new\_config$
31: $cur\_perf = new\_perf$
32: $step = step + 1$
return cur_config, cur_perf
33: **function** EVALUATE_PERF($config, move$)
34: $non\_iidness \leftarrow Evaluate\_iidness(config, move)$
35: $speed \leftarrow Calculate\_global\_speed(config, move)$
36: **if** $speed \leq 0$ **then return** $-\infty$
37: **else**
38: **return** $non\_iidness + coefficient * speed$
39: **function** $Evaluate\_iidness(config, move)$
40: $global\_distirbution \leftarrow bincount(total\_labels\ of\ full\ dataset)$
41: **if** moving **then**: ▷ true: Move-clients strategy; false: Share-clients strategy
42: $target\_subsets$ add $C\_dataset$
43: $source\_subsets$ remove $C\_dataset$
44: $non\_iidness$ = $(calculate\_kl\_divergence(target\_subsets) + calculate\_kl\_divergence(source\_subsets))/2$

```
    else
        target_subsets add C_dataset
        non_iidness =
      (calculate_kl_divergence(target_subsets) +
      calculate_kl_divergence(source_subsets))/2
    return 1 − non_iidness
function CALCULATE_GLOBAL_SPEED(config, move)
    config.S_target.dataset add C.dataset
    global_delay ← 0
    global_speed ← 0
    total_updates ← 0
    C.new_delay =
          config.S_target.CALCULATESPEED(
          True)[C.id]
    if C.new_delay ≤ 0 then
        return −∞
    else
        S_target_total_delay = sum(
            config.S_target.CALCULATESPEED(TRUE))
        if move then:
            config.S_source remove config.C

        S_source_total_delay = sum(

  config.S_source.CALCULATESPEED(FALSE))
        global_delay = S_source_total_delay
                       + S_target_total_delay
        total_updates =
                 count(S_source.clients.labels)
               +count(S_target.clients.labels)
        if move then:
            move config.C back to config.S_source

    global_speed = total_updates/global_delay
    return global_speed
```

C. Move-clients Strategy

Based on the number of servers and clients in the model, one or more target servers can be selected, and one or more clients can be moved accordingly. In this method, a client is first selected from a server (usually a server with more evenly distributed data). The selected client is then "moved" to the target server, where it exclusively exchanges data and model updates with this server. This means that the client's local update results will not be sent back to the original server, but only to the target server. If multiple clients are selected to be moved, this process is repeated until the top n clients (where n is the chosen number) are selected for each target server.

After the clients is moved, the system's Client-Server interactions, including the client's local training and the server's aggregation of models received from clients, as well as the Server-Server aggregation and Token-Based Model Exchange, all follow the processes described in Spyker [27] and Section II-C.

D. Share-clients Strategy

We also propose two alternative approaches. In the previous method, the selected client communicated exclusively with one other server. Now, in the improved method, we allow the client to communicate with both the original server and another target server. Those approaches effectively "shares" the client between the original server and the target server, enabling both servers to benefit from the client's data and updates. The client-server interactions are different from the algorithms proposed in Spyker [27], which will be introduced below.

1) Share-clients with Model Combination: Alg. 3 takes the average of the models received from both servers, and trains it locally on the client. After training, the updated model is alternately sent back to either the original or target server based on a flag.

Algorithm 3 Client-Server Interactions with Alternating Servers with Model Combination

Input: Client learning rate η_k, server learning rates η_i, η_j for servers i and j respectively.
Output: Updated models W_i^{t+1} and W_j^{t+1} for servers i and j.

procedure ALTERNATE_LOCAL_TRAINING(W_i^t, A_i, W_j^t, A_j)
  client k receives model W_i^t with age A_i from server i
  client k receives model W_j^t with age A_j from server j
  $W_k^t \leftarrow$ Average(W_i^t, W_j^t) ▷ Combine models from servers i and j
  $\eta_k' \leftarrow$ Decay(η_k)
  for epoch $\in T_k$ **do**
    update W_k^t with learning rate η_k'
  $W_k^{t+1} \leftarrow W_k^t$
  if $Flag\_trained = True$ **then**
    send (W_k^{t+1}, A_i) to server i
    $Flag\_trained = False$
  else
    send (W_k^{t+1}, A_j) to server j
    $Flag\_trained = True$

2) Share-clients with Dual Models: Alg. 4 takes a different approach by maintaining separate models for each server on the client. The client alternates between training these models, ensuring that each server receives updates tailored specifically to its data distribution.

Algorithm 4 Dual Models Training on Client with Alternating Flag

Input: Initialized models model_0 and model_1 from Server i and Server j, initialized ages age_0 and age_1, learning rate η_k, flag train_on_server_i
Output: Updated model sent to the corresponding server

client receives model and age (model_0 $\leftarrow W_i^t$ age_0 $\leftarrow$ age_i) from $server\_i$
client receives model and age (model_1 $\leftarrow W_j^t$ age_0 $\leftarrow$ age_i) from $server\_j$

if train_on_server_i is True **then** ▷ Train on Server i's model
▷ Local Training on model_0
$\eta'_k \leftarrow \text{Decay}(\eta_k)$
for each epoch $\in T_k$ **do**
model_0 $\leftarrow$ LocalTraining(model_0, η'_k)
SendModel(Server i, model_i, age_0)
train_on_server0 $\leftarrow$ False
else ▷ Train on Server 1's model ▷ Local Training on model_1
$\eta'_k \leftarrow \text{Decay}(\eta_k)$
for each epoch $\in T_k$ **do**
model_1 $\leftarrow$ LocalTraining(model_1, η'_k)
SendModel(Server 1, model_1, age_1)
train_on_server0 $\leftarrow$ True

V. Performance Evaluation

In this section, we first describe our experimental settings. Then, we analyze the impact of non-IID resources across servers as well as clients and study the impact of different non-IID datasets on the accuracy of Multi-Server Asynchronous FL. Moreover, we compare our three proposed solutions with the algorithms from Spyker. The comparison focuses on accuracy.

A. Settings

Baseline and Experimental Models. We compare our three solutions with Spyker [27] which is described in Section II-C. All of our experiments are conducted on the reproducible FL model described in Section III. This model allows experiments to be carried out with controlled variables by specifying the experimental parameters in a configuration file. This approach not only ensures consistency and reproducibility across experiments but also provides flexibility in adjusting parameters to explore different scenarios systematically.

Datasets. We conduct experiments on the MNIST and CIFAR-10 image collections. The MNIST dataset is a collection of 70,000 grayscale images of handwritten digits (0-9). The CIFAR-10 dataset is a collection of 60,000 color images, categorized into 10 classes.

Time Consumption and Network Delays. Our experiments account for both time consumption and network delays. The average server's local training time is approximately 200 ms, which was determined by benchmarking all algorithms using the Python TIME package. Additionally, the communication delays between servers are based on their geographic distribution on the Voronoi diagram, as detailed in Section III-A1. The training delays for the clients are calculated according to the data distribution, using Equation 1 as described in Section III-C. Furthermore, the network delay between a server and a client within its region is calculated by combining the client's training delay, the round-trip communication delay, and an additional 2 ms to account for timing discrepancies between the round-trip delays. This is expressed by the following equation:

$$Network\_delay = C\_training\_delay + 2 * comm\_delay + 2 \quad (2)$$

This comprehensive approach ensures that our model accurately reflects real-world conditions by incorporating both computational and communication factors.

Heterogeneous Data Distribution. In federated learning frameworks, the dataset distribution follows a hierarchical approach. Initially, it is partitioned among the server's client sets, adhering to either IID or non-IID principles. This primary allocation is then refined by distributing the data within each client set to its respective clients, maintaining the designated IID or non-IID scheme. This structured methodology allows for nuanced control over data distribution across both server and client levels, facilitating a comprehensive exploration of various non-IID scenarios. Tab. V encapsulates the possible scenarios.

The non-IID distribution is predicated on the Dirichlet distribution as described in Section III-B1, offering a versatile method to mirror the varied data distributions across different clients, thus simulating real-world data diversity. Conversely, the IID distribution is achieved through an equitable split of the dataset into shards of equal size. This method ensures that each client receives an identical amount of data, irrespective of the total data volume or the specified number of clients, thus fostering uniformity across all clients.

TABLE V
Summary of Data Distribution Scenarios

Scenario	Server Level Distribution	Client Level Distribution
1	IID	IID
2	IID	Non-IID
3	Non-IID	IID
4	Non-IID	Non-IID

B. Comparison among IID and Non-IID scenarios

We compare the 4 scenarios shown in Tab. V for Spyker, a novel multi-server and fully asynchronous FL framework that relies on a flat of architecture of geo-distributed servers, on Fmnist dataset. We employ a distributed computing setup involving 4 servers and 32 clients, with the clients evenly divided among the servers, meaning each server is allocated 8 clients. To simulate non-IID data scenarios, at the server-level, client-level, and both server-client-level, we use the Dirichlet distribution with an alpha value of 0.1.

The results are shown in the Fig. 6. In the figure, the 'IID' scenario shows a rapid increase in accuracy with the initial few updates, suggesting that models converge faster with IID data. The accuracy of both Server_noniid and Client_noniid remains consistently lower than that of the IID scenario. This demonstrates the negative impact of non-IID data on model accuracy, especially when such skewed data distribution is present both at the client and server levels. However, comparing non-IID data at the server and client levels, Server_noniid performs worse than Client_noniid and almost as bad as

all_noniidscenario, suggesting that server-level non-IID data have a greater impact on model accuracy compared to client-level in this instance.

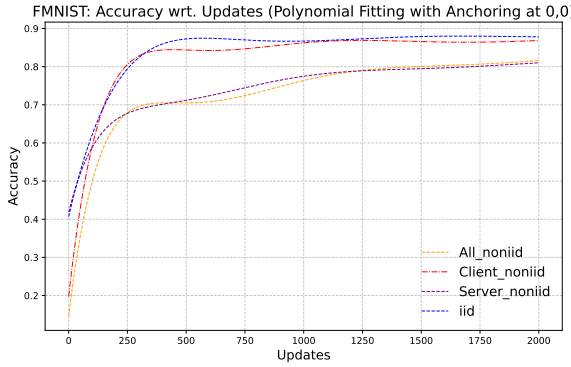


Fig. 6. Data distribution results (MNIST: 32 clients, 4 servers. α = 1)

C. Comparing the impact of different α values

We set up an experiment to compare the results obtained for different α values, which control the degree of server-level non-iidness in Alg. 1, from 0.1 to 1 in the case of server-level non-IID for Spyker algorithm. 4 server and 32 clients are used and each serve has 8 clients. The result is shown in Fig.7:

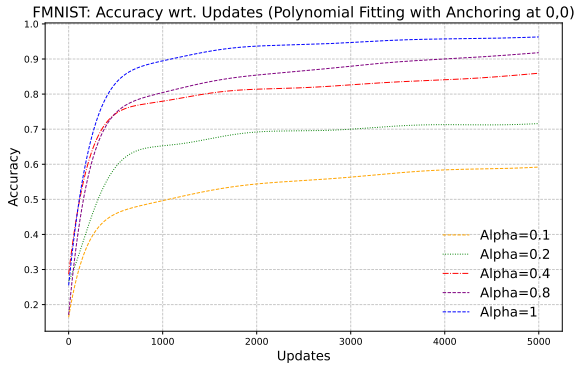


Fig. 7. Impact of different alpha on accuracy (MNIST: 100 clients, 4 servers)

The Fig. 7 demonstrate that lower α values, which represent more skewed non-IID data distributions, result in lower overall accuracy and slower convergence, demonstrating the challenges posed by non-IID data in FL.

D. Comparing the Move-Clients and Share-Clients strategies

As illustrated in Section V-B, it is clear that the server-side non-IID data distribution has significantly reduced the accuracy of the MultiAsync FL model. The impact is particularly pronounced, with server-level non-IIDness leading to a noticeable decline in performance compared to other scenarios. Additionally, as discussed in Section V-C, the system exhibits lower accuracy when the Dirichlet distribution parameter α is 0.1. Given these observations, we have chosen to focus on the server non-IIDness scenario with α set to 0.1 in order to better evaluate the effects of the Move-Clients and Share-Clients strategies on accuracy. This focus allows us to more clearly isolate and understand how these strategies influence the system's performance under conditions of heterogeneous data distribution.

We conduct our experiments using 16 clients and 2 servers with the MNIST and CIFAR10 dataset. The clients are evenly distributed between the two servers, with each server managing 8 clients. A single client is selected for both the Move-Clients and Share-Clients strategies. To ensure consistency and accurate comparisons, we maintain strict control over variables. Notably, the locations of the servers, the distribution of clients, and the training delays of all clients, except for the chosen client, remain identical across all experiments. This approach ensures that any observed differences in performance are solely due to the applied strategies. The results shown in Fig. 8 and Fig. 9

Fig. 8 shows that the Move-Clients strategy (red line) has the fastest accuracy improvement and reaches the highest final accuracy around 4.815%. Tab. VI supports this, showing that this strategy reduces time and updates by 82.39% and 82.64% at 90% accuracy, and by 85.67% and 85.82% at 95% accuracy. Moreover, Fig. 9 indicates the the experiments run on CIFAR10 have the same results as the experiments on MNIST. However, CIFAR-10(Fig. 9) results are ultimately less accurate due to the complexity of the data and the complexity of the model.

The Share-Clients strategy with dual models (represented by the purple line) also shows a noticeable increase in accuracy, though at a slower rate and with a slightly lower final accuracy compared to the Move-Clients strategy. Table VI indicates that this strategy reduces time and updates by 26.79% and 30.76% at 90% accuracy, and by 34.09% and 37.73% at 95% accuracy.

In contrast, the Share-Clients strategy with the model combination, called Avg Model(blue line), shows the slowest improvement and the lowest final accuracy. Table VI shows that this strategy increases time by 10.43% and updates by 6.88% at 90% accuracy, indicating it is less effective in handling non-IID data distributions.

TABLE VI
MNIST: COMPARISON OF METHODS WITH IMPROVEMENT

Method	90% Accuracy		95% Accuracy	
	Time	Updates	Time	Updates
Baseline	17264.77s **(0%)**	3165 **(0%)**	33669.64s **(0%)**	6178 **(0%)**
Avg Model	19064.03s **(-10.43%)**	3556 **(-6.88%)**	26826.92s **(20.01%)**	6603 **(-6.88%)**
Move	3038.99s **(82.39%)**	549 **(82.64%)**	4826.16s **(85.67%)**	876 **(85.82%)**
Share	12638.63s **(26.79%)**	2204 **(30.76%)**	22188.43s **(34.09%)**	3847 **(37.73%)**

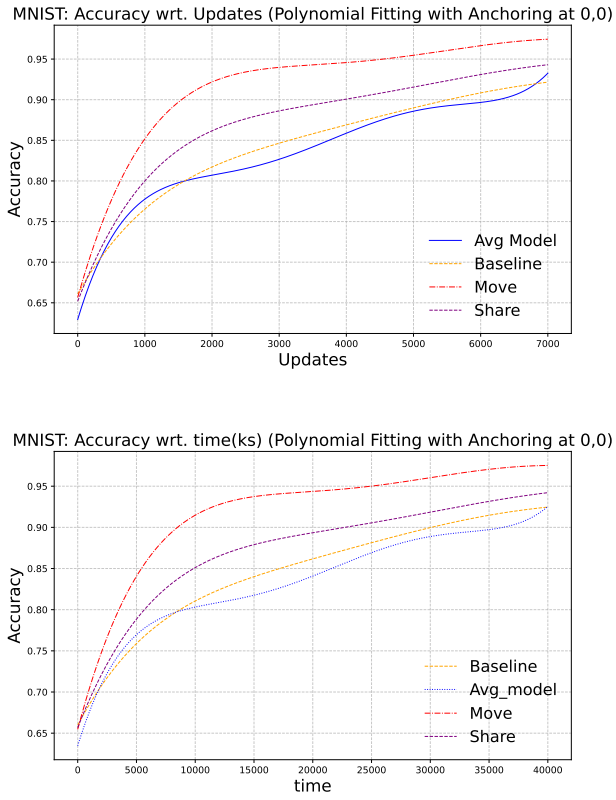


Fig. 8. Comparison of strategies(MNIST: 16 clients, 2 servers, α = 0.1)

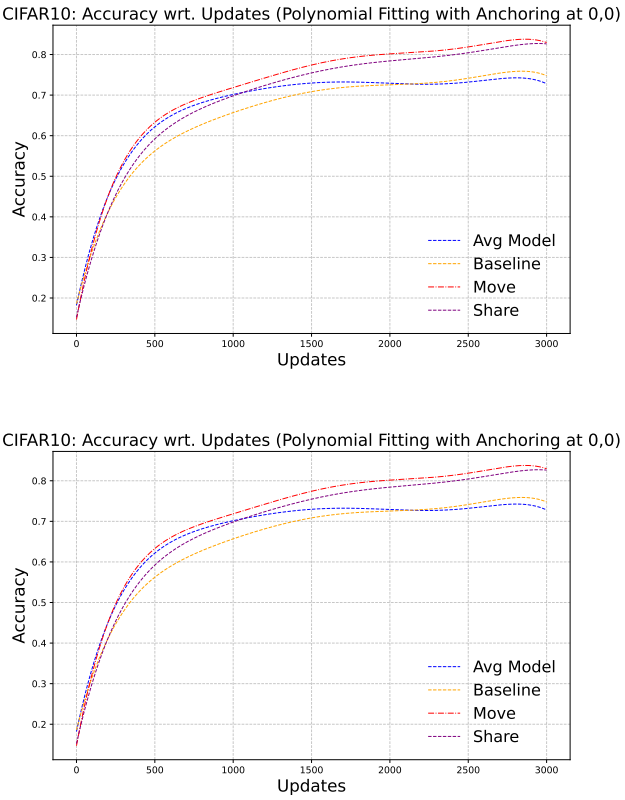


Fig. 9. Comparison of strategies(CIFAR10: 16 clients, 2 servers, α = 0.1)

E. Impact of Number of Clients on the Move-Clients and Share-Clients Strategies

We evaluate the impact of the number of clients from two perspectives: the number of clients per server and the number of chosen clients. Since the results of the Share-Clients strategy with model combination show that it does not effectively mitigate the impact of non-IID data distribution, we will exclude this method from our analysis.

1) Modifying the number of clients per server: Our experiments involve 32 clients and 2 servers using the MNIST dataset. The clients are evenly distributed between the two servers, with each server managing 16 clients.

For both the Move-Clients and Share-Clients strategies, a single client is selected. Fig. 10 shows that the Share-Clients strategy (represented by the purple line) achieves the highest accuracy, surpassing both the Move-Clients strategy (represented by the red line) and the baseline (represented by the yellow ine). Compare with the Fig. 8, which indicates that as the number of clients increases, the Share-Clients strategy becomes more effective, leading to better accuracy. Additionally, the accuracy growth is more pronounced and stable in the Share-Clients strategy, suggesting that sharing client updates between servers can significantly reduce the negative impact of non-IID data distribution when more clients are involved. Through alternate communication, each server's model receives regular updates from different data distributions, reducing overfitting on specific data and improving the generalisation of the model.

2) Modifying the number of chosen clients: Our experiments involve 4 servers and 32 clients, with each server managing 8 clients, using the MNIST dataset. We will compare two scenarios: in the first, a single client from a server with a more severely skewed non-IID data distribution is selected and moved to another server; in the second scenario, two clients from two different servers are selected and moved to two other servers that also have more severely skewed non-IID data distributions.

Fig. 11 and Fig. 12 indicate that both the Move 1 client and Move 2 clients strategies show a improvement in accuracy over the baseline and Share-Clients strategies. In the Move 1 client scenario, the accuracy curve for the Move-Clients strategy (orange line) is slightly higher than the Share-Clients strategy (green line) and the baseline (blue line). This indicates that moving a single client to a different server when the total server number is increasing, cannot significantly improve model performance under non-IID conditions. The Move 2 clients strategy further amplifies this effect, with the accuracy curve for the Move-Clients strategy showing even greater accuracy improvements compared to Move 1 client. This suggests that increasing the number of moved clients enhances

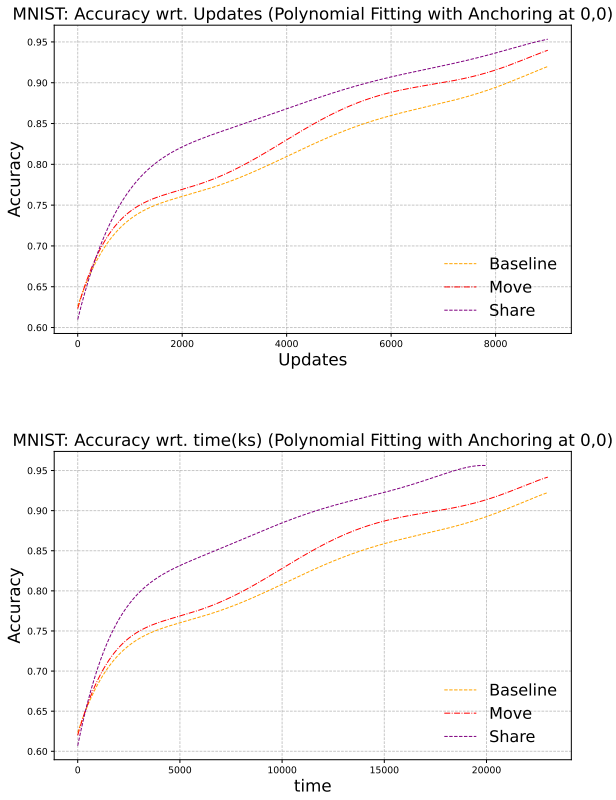


Fig. 10. Comparison of # clients (MNIST: 32 clients, 2 servers, α = 0.1)

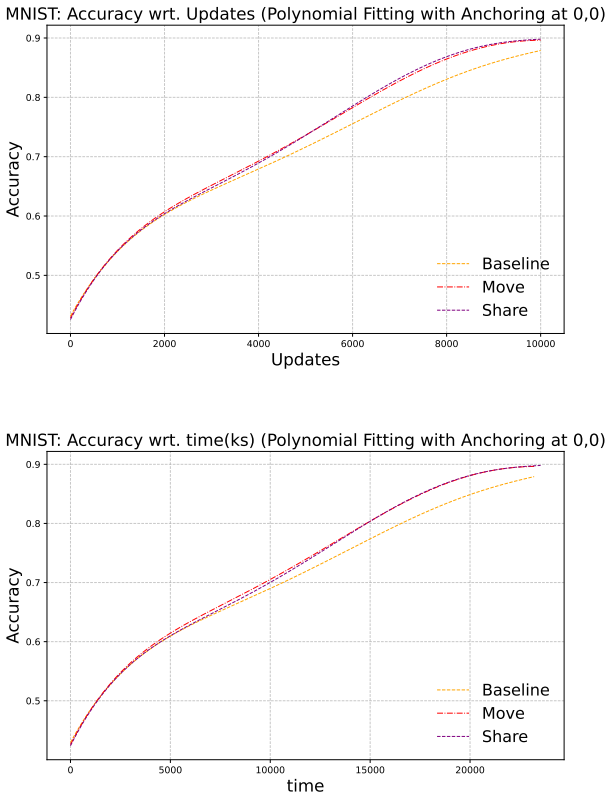


Fig. 11. Move 1 client (MNIST: 32 clients, 4 servers, α = 0.1)

the system's ability to mitigate the effects of non-IID data distributions. In the Move 1 client scenario, the accuracy curves for the Move-Clients strategy and the Share-Clients strategy are quite close, in contrast, in the Move 2 clients scenario, the Move-Clients strategy shows a more pronounced improvement over the Share-Clients strategy. This highlights the scalability of the Move-Clients approach and suggests that its effectiveness increases with the number of clients moved, making it a more powerful tool for addressing non-IID data distributions in federated learning when applied to a larger set of clients.

VI. Related Work

A. Dirichlet distribution's utilization in FL

The Dirichlet distribution was described in [32]. It is a multivariate generalization of the Beta distribution and an important multivariate continuous distribution in probability and statistics [33].

The probability density function (PDF) of the Dirichlet distribution is given by:

$$f(\mathbf{x};\boldsymbol{\alpha}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})}\prod_{i=1}^{K} x_i^{\alpha_i - 1}$$

where $\mathbf{x} = (x_1, x_2, ..., x_K)$ is the vector of K elements, and $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, ..., \alpha_K)$ represents the parameters of the distribution. The multivariate Beta function $\mathrm{B}(\boldsymbol{\alpha})$ is defined as:

$$\mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{K}\Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}$$

Here, $\Gamma(\cdot)$ denotes the gamma function.

The primary applications of the Dirichlet distribution revolve around modeling heterogeneity among clients to enhance the efficiency of both model aggregation and task allocation in various contexts. Unlike methods that impose equal sample sizes among clients or enforce uniform Earth Mover's Distance (EMD) values across clients, the Dirichlet distribution method stands out for its ability to accommodate varying sample sizes and non-uniform EMD values. This characteristic contributes to the realism of the Dirichlet distribution method, aligning well with the complexities often observed in real-world datasets [34].

There are many FL methods utilize the Dirichlet distributions for generating the non-iid dataset [35]–[37]. In order to simulate the label distribution skew which is one of non-IID data setting, [35] introduces distribution-based label imbalance settings that each party is allocated a proportion of the samples of each label according to Dirichlet distribution.

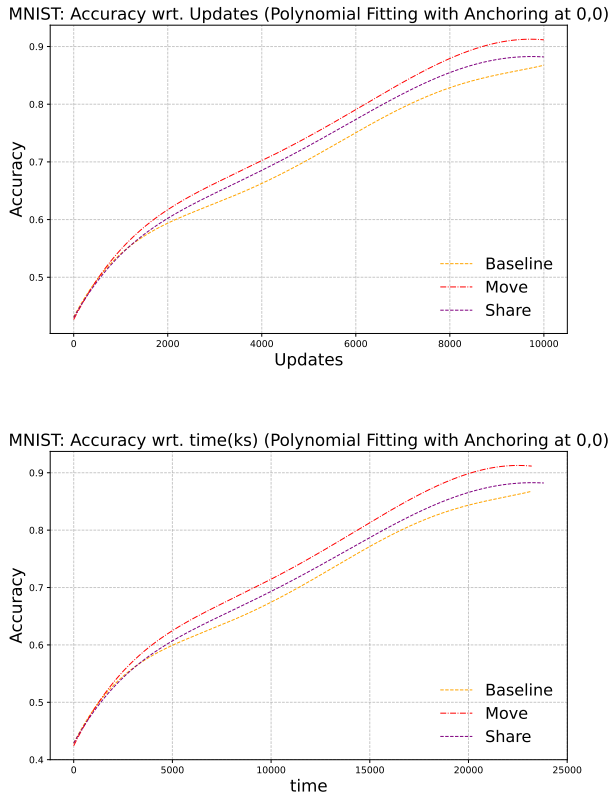


Fig. 12. Move 2 clients (MNIST: 32 clients, 4 servers, α = 0.1)

B. FL with control variates

SCAFFOLD [6] is a method designed to address the convergence issues in federated learning caused by Non-IID data distributions. Traditional federated learning methods, such as FedAvg, often struggle with Non-IID data, leading to local model updates that deviate from the optimal direction of the global model, ultimately slowing down the convergence and potentially harming the overall performance. SCAFFOLD introduces control variates to correct the 'client-drift' introduced during local training, thereby enhancing the convergence of the global model. Specifically, SCAFFOLD maintains both local and global control variates for each client, which are used to adjust the local updates and correct the drift caused by Non-IID data. During each training round, clients adjust their model updates based on these control variates, ensuring that the updates are more aligned with the optimal direction for the global model. When aggregating the models, the server does not simply average the updates but also adjusts the global control variate based on the corrected updates, ensuring faster convergence and improved final accuracy. This approach allows SCAFFOLD to effectively mitigate the challenges posed by Non-IID data in federated learning systems, resulting in faster convergence and higher accuracy, making it particularly suitable for scenarios with heterogeneous data distributions.

C. FL with overlapping area clients update models via regional model averaging

The paper "On the Convergence of Multi-Server Federated Learning With Overlapping Area" [26] explores the dynamics of federated learning within a multi-server environment, particularly focusing on clients situated in overlapping regions that are influenced by multiple servers. The primary contribution of this work is its use of a model averaging technique, where clients in overlapping areas update their local models based on an averaged model derived from the accessible regional models. This method is designed to improve the convergence of the global model by mitigating biases caused by data heterogeneity across different regions.

Importantly, the approach taken in this paper is synchronous, meaning that model updates are synchronized across clients at regular intervals, with all clients typically waiting to complete their updates before aggregation occurs. This synchronization helps manage the integration of updates from overlapping areas, ensuring consistency in the global model.

VII. Conclusion

In this paper, we addressed the challenges of implementing Federated Learning (FL) in environments characterized by heterogeneous data distributions and asynchronous communication across multiple servers. Traditional FL systems, which rely on synchronous updates and homogeneously distributed data, often struggle in real-world scenarios where data is non-IID and resources vary widely.

To better study the impact of non-IID data distributions on FL systems, we provided a reproducible Multi-Server FL model with heterogeneous resources. This framework allows for controlled and consistent testing of various FL strategies under different conditions, offering a robust model for analyzing the effects of data heterogeneity on system performance.

Within this framework, we introduced three novel strategies to enhance the robustness and efficiency of multi-server asynchronous FL: moving clients between servers, sharing clients among servers with dual models, and sharing clients with model combination. To further support the effectiveness of these strategies, we developed a client selection method that aids in optimizing client contributions by strategically selecting and reassigning clients in scenarios with non-IID data distributions. This method enhances the overall approach by ensuring that clients most suitable for balancing data distribution and improving model accuracy are prioritized.

Our experiments demonstrated that while the Share-clients with model combination method did not perform as well in improving model accuracy, the Move-clients method showed significant effectiveness, especially when the number of servers remains constant and the number of clients is relatively small. This method was able to reduce training time by up to 85.67% and decrease the number of updates required by 85.82%. However, as the number of clients increases, the Share-clients with dual models method proved to be more advantageous, offering a balanced improvement in model accuracy (by 4.815%) while maintaining efficiency in large-scale client environments.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016. [Online]. Available: https://arxiv.org/abs/1602.05629

[2] ——, "Communication-efficient learning of deep networks from decentralized data," 2023.

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

[4] T. Zhou and E. Konukoglu, "Fedfa: Federated feature augmentation," 2023. [Online]. Available: https://arxiv.org/abs/2301.12995

[5] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2020. [Online]. Available: https://arxiv.org/abs/1812.06127

[6] S. P. Karimireddy, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," 2021. [Online]. Available: https://arxiv.org/abs/1910.06378

[7] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," 2020. [Online]. Available: https://arxiv.org/abs/2007.07481

[8] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," 2021. [Online]. Available: https://arxiv.org/abs/2003.00295

[9] A. Fallah, A. Mokhtari, and A. Ozdaglar, "Personalized federated learning: A meta-learning approach," 2020. [Online]. Available: https://arxiv.org/abs/2002.07948

[10] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," 2018. [Online]. Available: https://arxiv.org/abs/1705.10467

[11] Q. Li, B. He, and D. Song, "Model-contrastive federated learning," 2021. [Online]. Available: https://arxiv.org/abs/2103.16257

[12] C. Xu, Y. Qu, Y. Xiang, and L. Gao, "Asynchronous federated learning on heterogeneous devices: A survey," *Computer Science Review*, vol. 50, p. 100595, 11 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157401372300062X#sec3

[13] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala, "Asynchronous online federated learning for edge devices with non-iid data," 2020. [Online]. Available: https://arxiv.org/abs/1911.02134

[14] Z. Wang, Z. Zhang, Y. Tian, Q. Yang, H. Shan, W. Wang, and T. Q. S. Quek, "Asynchronous federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6961–6978, 2022.

[15] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "Fedsa: A semi-asynchronous federated learning mechanism in heterogeneous edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3654–3672, 2021.

[16] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[17] 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.

[18] S. M. Azimi-Abarghouyi and V. Fodor, "Scalable hierarchical over-the-air federated learning," 2023.

[19] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on bayesian deep learning (NeurIPS)*, vol. 2, 2018.

[20] Y. Shi, L. Shen, K. Wei, Y. Sun, B. Yuan, X. Wang, and D. Tao, "Improving the model consistency of decentralized federated learning," 2023. [Online]. Available: https://arxiv.org/abs/2302.04083

[21] M. S. H. Abad, E. Ozfatura, D. Gündüz, and Ö. Erçetin, "Hierarchical federated learning across heterogeneous cellular networks," *CoRR*, vol. abs/1909.02362, 2019. [Online]. Available: http://arxiv.org/abs/1909.02362

[22] G. Long, M. Xie, T. Shen, T. Zhou, X. Wang, and J. Jiang, "Multi-center federated learning: clients clustering for better personalization," *World Wide Web*, vol. 26, no. 1, p. 481–500, Jun. 2022. [Online]. Available: http://dx.doi.org/10.1007/s11280-022-01046-x

[23] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," 2021.

[24] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 59–71, 2021.

[25] D.-J. Han, M. Choi, J. Park, and J. Moon, "Fedmes: Speeding up federated learning with multiple edge servers," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3870–3885, 2021.

[26] Z. Qu, X. Li, J. Xu, B. Tang, Z. Lu, and Y. Liu, "On the convergence of multi-server federated learning with overlapping area," 2022.

[27] J. D. L. Y. C. Yuncong Zuo, Bart Cox, "Asynchronous Multi-Server federated learning for Geo-Distributed clients," 6 2018. [Online]. Available: https://arxiv.org/html/2406.01439v1

[28] Matt, "Home — CloudPing - AWS Latency Monitoring." [Online]. Available: https://www.cloudping.co/grid

[29] I. Borg and P. J. F. Groenen, "Modern Multidimensional Scaling: theory and applications," *Journal of educational measurement*, vol. 40, no. 3, pp. 277–280, 9 2003. [Online]. Available: https://doi.org/10.1111/j.1745-3984.2003.tb01108.x

[30] J. Chen, C. Li, Z. Li, and C. M. Gold, "A Voronoi-based 9-intersection model for spatial relations," *International journal of geographical information science (Print)*, vol. 15, no. 3, pp. 201–220, 4 2001. [Online]. Available: https://doi.org/10.1080/13658810151072831

[31] W. contributors, "Dirichlet distribution," 1 2024. [Online]. Available: https://en.wikipedia.org/wiki/Dirichlet_distribution

[32] P. G. Dirichlet, "Sur la distribution des fréquences," *Bulletin de la Société Mathématique de France*, vol. 17, pp. 204–207, 1868.

[33] J. Lin, "On the dirichlet distribution," *Department of Mathematics and Statistics, Queens University*, pp. 10–11, 2016.

[34] I. Cornelis, "Federated learning from non-iid data: Improving accuracy through data-augmentation and communication efficiency," 2022.

[35] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 965–978.

[36] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE journal on selected areas in communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[37] H.-Y. Chen and W.-L. Chao, "Fedbe: Making bayesian model ensemble applicable to federated learning," 2021.

2

Extended Related Work

In Chapter 1, we introduced related work in the Background and Related Work sections, covering topics such as the basic theory of FL, Multi-Server FL Models, and the Multi-Server Asynchronous FL System Model. In this chapter, we will delve deeper into related work, specifically focusing on methods for constructing heterogeneous data, algorithms designed to address the challenges posed by FL with heterogeneous data, and client selection in FL.

2.1. Common Methods to Construct Non-IIDness

2.1.1. Label skew

- **Shard-Based Partitioning:** Mc Mahan et al. [4] presented the FedAvg algorithm, and investigated a learning technique that allows users to collectively reap the benefits of shared models trained from their collective datasets, without the need to centrally store it. They perform extensive experiments on this algorithm, demonstrating that it is robust to unbalanced and non-IID data distributions. They create non-IID data by sorting data by label and partitioning it into shards. Each client receives a few shards, resulting in clients having data from only a subset of classes. However, all the shards are equal-sized.
- **Label Preference Assignment:** This method assigns each client a preference for certain labels based on a predefined probability distribution or preference matrix. Canh T. Dinh et al. proposes an algorithm for personalized FL (pFedMe) [7] using Moreau envelopes as clients' regularized loss functions, which help decouple personalized model optimization from the global model learning in a bi-level problem stylized for personalized FL. They simulate non-iidness by assigning a preference for certain classes to each client. This is done by modifying the distribution of labels each client sees, ensuring that some clients have a higher probability of receiving samples from specific classes.
- **Class-Based Partitioning:** Yue Zhao et al. [10] focuses on the statistical challenge of FL when local data is non-IID. The authors created a non-iid setting by sorting the data by class and then partitioning it to simulate two extreme scenarios: in the 1-class non-iid case, each client received data exclusively from a single class, resulting in highly skewed distributions; in the 2-class non-iid case, the sorted data was divided into 20 partitions, and each client was randomly assigned two partitions, leading to each client having data from exactly two classes, which introduced a moderate level of non-iidness.
- **Random Sampling with Constraints:** Hongyi Wang et al. proposes Federated matched averaging (FedMA) algorithm [8] designed for FL of modern neural network architectures e.g. convolutional neural networks (CNNs) and LSTMs. To simulate non-iid data distributions, they used random sampling with constraints to ensure that each client receives data from a limited subset of the classes for the Shakespeare dataset. They preprocess the Shakespeare dataset by filtering out the clients with less than 10k datapoints and sampling a random subset of clients. This constraint on the random sampling process creates a realistic scenario where different clients have

access to different types of data, reflecting the heterogeneity in real-world applications.

The method we used to construct non-iid data distribution is a form of label skew based on the Dirichlet distribution. This approach assigns different label distributions to each client, resulting in imbalanced data with significantly varying label proportions across clients. The details can be found in the Section III.B of the research paper.

2.1.2. Quantity skew

Quantity Skew happens when different clients have very different amounts of data. The paper [6] written by Qu et al. explores how various types of data heterogeneity can negatively affect FL methods.

In this approach, Qu et al. create Quantity Skew by generating four sets of data partitions where the number of samples varies across simulated institutions. Each partition represents a different institution, with all institutions sharing the same feature and label distributions. However, the key difference lies in the quantity of data each institution possesses. The variability in sample sizes across these institutions introduces the Quantity Skew, which allows the authors to study its impact on FL methods.

To quantify the degree of Quantity Skew, they use the sample standard deviation (STD) of the sample sizes across the institutions. A higher standard deviation indicates a greater difference in the amount of data between institutions, thereby reflecting a higher degree of Quantity Skew. This setup simulates scenarios where some institutions have significantly more data than others, enabling the authors to analyze how FL methods perform under these imbalanced conditions.

2.1.3. Imaging acquisition skew

Imaging Acquisition Skew arises when the images in a dataset are collected under different acquisition settings, resulting in variations in the image characteristics (e.g., color, contrast, noise). These variations can be due to different cameras, environments, times of day, or even different protocols for image capture.

Liangqiong Qu et al. employs three different splits to explore the impact of Imaging Acquisition Skew on FL in the medical domain [6]. They generated a real data partition on the ADNI dataset that provides a longitudinal multi-institutional observation study on Alzheimer's disease patients. In the first method, they introduced resolution skew by progressively decreasing the image resolution across four institutions, with Institution 1 having the lowest resolution and Institution 4 the highest. In the second method, involved signal-to-noise ratio (SNR) skew, where different types of noise and blurring (such as Gaussian noise, motion blur, and combinations thereof) were applied to the images at each institution, leading to varying image quality across institutions. Finally, In the third method, the ADNI dataset was resplit based on the scanner vendors, dividing the data into four institutions according to the imaging devices from manufacturers like GE Healthcare, Philips Healthcare, and Siemens Healthcare, while ensuring minimal quantity and label distribution skew.

These variations mimic the real-world scenario where medical images, such as X-rays, MRIs, or PET scans, may differ significantly depending on the equipment or protocols used at different healthcare facilities.

2.2. Algorithms for Mitigating Challenges in FL with Heterogeneous Data

In this section, we review the key algorithms designed to mitigate the impact of heterogeneous data across different FL setups. We explore and categorize these algorithms based on the following FL setups: Single Server Synchronous, Single Server Asynchronous, and Multi-Server Synchronous algorithms. For each category, we will introduce one key algorithm.

2.2.1. Single-Server Synchronous FL

FedAvg is one of the foundational algorithms in FL, explained in the background section of research paper in Chapter 1. It is a heuristic method that updates the global model by averaging local stochastic gradient descent (SGD) updates from the participating clients. While effective in practice, FedAvg is not well-suited for heterogeneous environments due to its reliance on local updates, the limited number

of devices participating in each round, and the frequent heterogeneity in data distribution. FedAvg largely overlooks system heterogeneity. In real-world applications, if some local models fail to complete training within the allotted time, those models are discarded, which can reduce the accuracy of the global model. Inspired by FedAvg, the broader FedProx [5] framework was proposed by Li Tian et al. to address the challenges of heterogeneous federated environments while maintaining similar privacy and computational cost advantages.

FedProx builds upon FedAvg by introducing a regularization term that penalizes the deviation of the model parameters from the global model parameters. Specifically, FedProx aims to minimize the following objective function:

$$L(w) = \sum_{k=1}^{K} \frac{n_k}{n} \left(F_k(w) + \frac{\mu}{2} \|w - w^t\|^2 \right) \tag{2.1}$$

where w represents the model parameters, K is the number of clients, n_k is the number of data points in the kth client, n is the total number of data points across all clients, $F_k(w)$ is the loss function on the kth client, μ is the regularization parameter controlling the strength of consistency between the local updates and the global model, and w^t represents the current global model parameters. $\sum_{k=1}^{K} \frac{n_k}{n} F_k(w)$ represents the weighted average loss across all clients, ensuring that the model optimization is effective not just on a specific client but across all participating clients. $\frac{\mu}{2} \|w - w^t\|^2$ ensures that during each training round, the local updates do not deviate too much from the initial global model parameters, which is particularly important in scenarios with communication constraints or heterogeneous data distribution among clients. Through this approach, the FedProx algorithm aims to improve the efficiency and effectiveness of standard FL methods (such as FedAvg) when dealing with non-iid data or clients with varying computational capabilities and data volumes.

The core of FedProx is the introduction of a regularization term that penalizes the extent to which the model parameters deviate from the initial global model parameters, thereby controlling the local updates. The specific steps are as follows:

1. The central server initializes the global model parameters w^0
2. A subset of clients is selected, and the current global model parameters w^t are sent to these clients.
3. Each client trains the model on its local dataset. Unlike FedAvg, a regularization term $\frac{\mu}{2} \|w - w^t\|^2$ is added during training, with the aim of constraining the direction of the local model updates so that they do not deviate too far from the global model.
4. After training is completed, the clients send the updated local model parameters or the differences from the global model parameters back to the central server.
5. The server aggregates these updates, potentially weighted by the contribution of each client (which may be based on data volume or other metrics), and updates the global model accordingly.
6. Steps 2 to 5 are repeated until convergence is achieved or sufficient training rounds are completed.

Through this method, FedProx not only addresses the issue of data distribution but also reduces the fluctuation of the model during training via the regularization term, making the model more robust, particularly in scenarios where client environments are highly heterogeneous. This gives FedProx an advantage over traditional FL methods in practical applications.

2.2.2. Single-Server Asynchronous FL

To improve the flexibility and scalability of FL, a new asynchronous federated optimization algorithm, FedAsync [9], has been proposed by Cong Xie et al. FedAsync also helps to alleviate heterogeneity and accelerate model convergence.

Fig. 2.1 provides an overview of the FedAsync system, illustrating how asynchronous FL is managed between devices and servers.

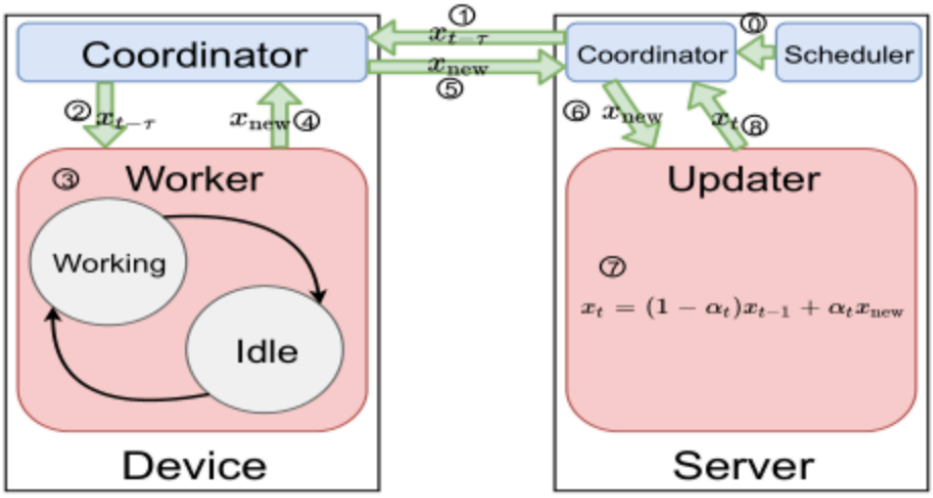


Figure 2.1: Overview of FedAsync's training process [9]

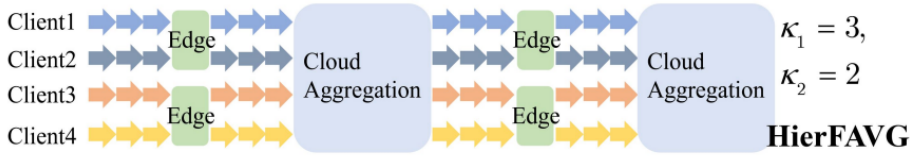


Figure 2.2: Overview of HierFAVG's training process [3]

- **Step 0:** The scheduler triggers training through the coordinator.
- **Steps 1 and 2:** The server sends the model $x_{t-\tau}$ to the worker via the coordinator.
- **Step 3:** The worker computes local updates. The worker can switch between the "Working" and "Idle" states.
- **Steps 4, 5, and 6:** The worker sends the locally updated model back to the server via the coordinator. In Step 5, the coordinator queues the received local models; in Step 6, the updated local models are sequentially submitted to the Updater.
- **Steps 7 and 8:** The server updates the global model and transmits the updated model to the coordinator.

In the FedAVG algorithm, the server waits until the number of responding clients reaches $C \times n$ before performing the weighted averaging to obtain the global model parameters. However, in FedAsync, the server does not wait. As soon as a worker sends its parameters (model parameters and staleness), the server performs weighted averaging immediately. The global model update is influenced by the staleness, with a mixing parameter α used to control the weight. This asynchronous approach allows FedAsync to adapt more quickly to the diverse and often imbalanced contributions from clients, which is a common scenario in non-iid settings.

2.2.3. Multi-Server Synchronous FL

Most FL models always assumed a central server, either in the cloud or at the edge. Cloud servers have access to more data but incur excessive communication overhead and longer latency, while edge servers can communicate more efficiently with clients. To combine the advantages of both, Paper [3] propose a client-edge-cloud hierarchical FL system, which supports the HierFAVG algorithm that allows multiple edge servers to perform partial model aggregation. HierFAVG has three main steps shown in Fig. 2.2: client local training, edge server aggregation and global aggregation. HierFAVG operates by having each edge server aggregate the models from its clients after every $k1$ local updates. Subsequently, after every $k2$ edge model aggregations, the cloud server aggregates the models from all edge

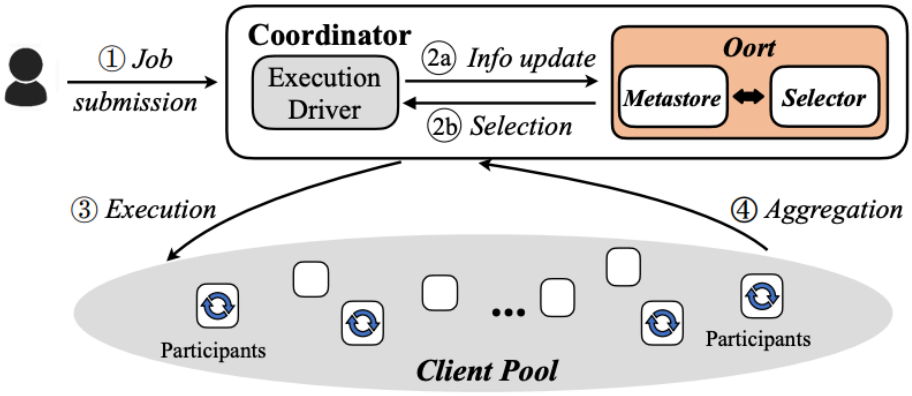


Figure 2.3: The structure of Oort [1]

servers. This means that communication with the cloud occurs after every $k1k2$ local updates.

HierFAVG reduces the impact of non-iidness by performing localized model aggregation at the edge server level before global aggregation. This approach allows for smoothing out extreme variations in model updates from clients with heterogeneous data, ensuring that more similar data distributions are aggregated first. By reducing the divergence in model updates at the edge level, the global model becomes more stable and consistent, effectively mitigating the challenges posed by non-iid data.

2.3. Client selection in FL

The primary reason for client selection in FL is to enhance the time-to-accuracy performance and improve the final model accuracy. In traditional FL approaches, clients are often randomly selected, which can lead to inefficient training because of the large heterogeneity in data characteristics and system performance across clients. Random selection might include clients with poor connectivity, slow processing power, or less relevant data, which can elongate training rounds and degrade model performance. Therefore, selecting clients that are both statistically useful (i.e., they can provide significant improvements in model accuracy) and system-efficient (i.e., they can process and transmit data quickly) is essential for achieving better and faster convergence in FL.

Fan Lai et al. introduces Oort [1], an approach designed to enhance FL and testing performance through guided participant selection, which is shown in Fig. 2.3. To optimize time-to-accuracy during model training, Oort prioritizes clients who possess data with the highest potential to improve model accuracy and who can execute training efficiently.

Oort selects clients by assessing both their statistical utility and system efficiency to optimize FL performance. It evaluates the potential impact of each client's data on the overall model, prioritizing those whose data is most valuable for reducing model error. Simultaneously, it considers the system capabilities of each client, favoring those who can process and transmit data quickly. To maintain a balance, Oort dynamically manages the trade-off between selecting high-utility clients and those with better system performance. Additionally, it employs an exploration-exploitation strategy, probabilistically selecting clients to ensure robustness and adaptability over time.

In Oort, clients are designed to upload their complete model updates to the central server during each round. However, this approach fails to account for the fact that not all updates contribute equally to model training. Due to the diversity in client data sizes and the varying importance of their samples, uploading less significant updates can substantially reduce system efficiency.

Chenning Li et al. studies the under-exploited system efficiency in Oort, and proposes an algorithm called PyramidFL [2]. The overview of PyramidFL is shown in Fig. 2.4. It begins by calculating a utility score for each client, taking into account the relevance of their data and their system capabilities. Clients are then ranked based on these utility scores, which determines their selection for training and the configuration of their local training processes, such as the number of iterations and the amount of data to be used. This ranking-based approach ensures that the most beneficial clients are prioritized,

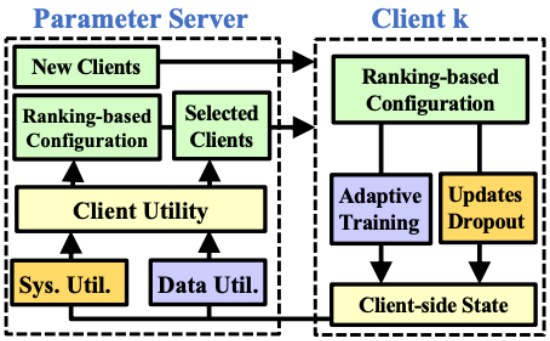


Figure 2.4: Overview of PyramidFL [2]

while others can adjust their training to optimize their contributions. Additionally, PyramidFL allows clients to adapt their local training based on their ranking, utilizing idle time more effectively and reducing communication overhead by dropping less significant updates. An exploration-exploitation mechanism is also integrated, balancing the selection of consistently high-utility clients with the need to explore and involve new clients. This process enables PyramidFL to achieve faster convergence and higher model accuracy by fully exploiting the heterogeneity in data and system capabilities among clients.

3

Additional Experiments

In this chapter, to complement our experiments described in the research paper, we conduct additional experiments to further evaluate and validate our proposed solutions. These additional experiments aim to provide a more comprehensive assessment of our methods and their effectiveness in addressing the challenges posed by our research paper.

3.1. Computation time to select clients

In solution section of research paper, we proposes a Clients-selection method. As what describe in that section, we first evaluate the data distribution on each server to identify those with significant non-IIDness, known as target servers. We then select clients from servers with more balanced data distributions and introduce them to the target servers. In addition, we also need to consider the global speed of the system to comprehensively evaluate the performance. This approach increases data diversity on the target servers, reduces data bias, and balances model training, ultimately improving the performance of the global model. In this section, we further investigated the computation time involved in client selection.

First, we considered a scenario with 16 clients distributed across 2 servers, with each server hosting 8 clients. The dataset used for this experiment is MNIST. At the server level, the data distribution is non-IID, generated using a Dirichlet distribution with alpha = 0.1. However, within each server, the data is evenly distributed among its clients to create IID datasets. Upon evaluating the non-IIDness of both servers, we determined that Server 1 exhibited a higher degree of non-IIDness. To address this, we iterated through each client on Server 0, simulating scenarios where a client could either be moved to or shared with Server 1. The decision to select a specific client was based on a thorough evaluation of the impact on Server 1's non-IIDness after the client's relocation or sharing, as well as the effect on the system's global computation speed.

We utilized Python's time module to measure the duration of each individual round as well as the overall process. In addition to this, We separately recorded the time taken to evaluate the non-IIDness of the servers and to compute the global speed. The results are presented in the Tab. 3.1 and Fig. 3.1.

Tab. 3.1 shows that the time consumption of calculating global speed is 10 times evaluation server's non-IIDness. By analysing the two calculations, we can obtain the reasons.

The global speed represents the total update speed of the system with all communication delays and training delays, which is following the equation below:

$$Global\_speed = total\_updates / total\_delay \tag{3.1}$$

where the total updates is the amount of data processed by all clients during training, and the total delay is the sum of all delays. To further assess the impact of the movement, the system simulates the process of transferring the client from the original server to the target server, during which the client is added to the target server and its training delay in the new environment is calculated. When deciding

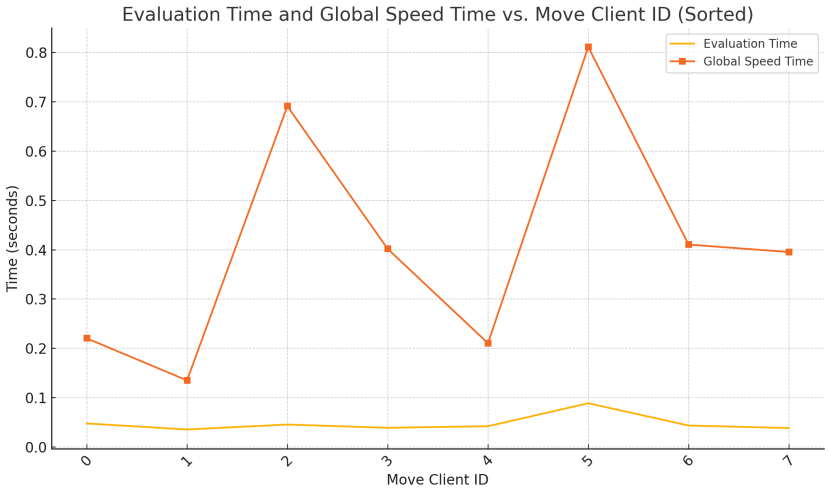


Figure 3.1: The computation time of each clients

whether to move or share the client, the system calculates the impact on global speed according to different strategies: if the client is moved, the system recalculates the total delay and total updates for both servers; if the client is only shared, the system calculates the relevant data post-sharing. After the simulation, the system removes the newly added client and restores the original state of the servers to ensure that the actual operation of the system is not affected. Finally, the system returns a new global speed value based on these calculations, helping to evaluate the effectiveness of the client movement or sharing strategy, and ensuring that the global model training is completed in the shortest possible time. In this process, it is necessary to repeatedly calculate the training delay for each client on both servers before and after the client movement. This calculation applies the least squares method, a computationally intensive process that requires a significant amount of time. Additionally, since the client's training delay is calculated based on the relationship between the client's dataset and the server's dataset as the Equation(1) shown in the Section III.B of the research paper, this approach involves applying the least squares method to a relatively large matrix, further contributing to the computational intensity. Due to the varying locations of each client, the computation required for calculating the training delay with the corresponding server differs, leading to significant variations in computation time as shown in Fig. 3.1.

In contract, we evaluates the data distribution of the target server and the source server by calculating the KL divergence (Kullback-Leibler Divergence). While calculating KL divergence requires statistical analysis of data distributions, we were able to significantly reduce the computation time by focusing the calculations on the dataset's targets rather than on the raw image data itself. Moreover, evaluating data distribution does not require double-counting.

Table 3.1: The computation time of each rounds

step	move_client_id	round_time	evaluation_time	calculate_global_speed
1	0	0.268330	0.047814	0.220404
2	1	0.170863	0.035609	0.135147
3	2	0.736860	0.045640	0.691047
4	3	0.440941	0.039048	0.401790
5	4	0.253282	0.042213	0.210578
6	5	0.900968	0.088800	0.811131
7	6	0.454293	0.043656	0.410503
8	7	0.434003	0.038512	0.395385
Total	NaN	3.66	NaN	NaN
Average	NaN	NaN	0.047661483	0.409498155

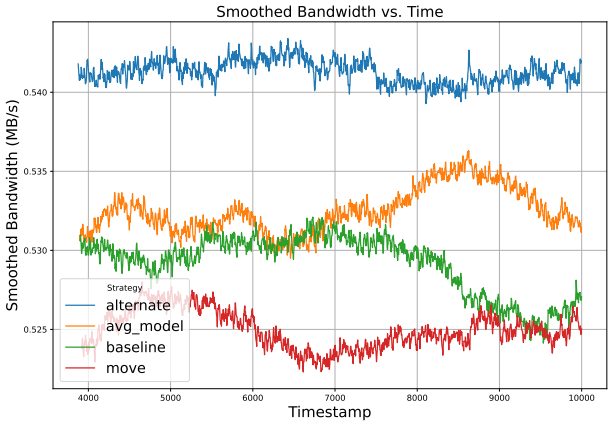


Figure 3.2: Bandwidth consumption of each strategies

3.2. Bandwidth consumption for our three strategies

In this section, we analyze the bandwidth consumption of each strategies. The bandwidth is calculated following this equation:

$$Bandwidth = Total\_Transfer\_Data/Time \tag{3.2}$$

In our Multi-async FL framework as described in the research paper, data transactions occur in three interconnected phases. First, the server distributes the global model to each client, providing a consistent starting point for local training. After training on their local datasets, clients send their updated models back to the server, which reflects the learning achieved during this phase. Additionally, the exchange of server models and aggregation is token-triggered. Servers exchange and aggregate their models by comparing the received token and their own token. Hence, we need to calculate the model size in each phase and count the time consumption during each transaction. We calculate the model size by first iterating through all the parameters of the model to calculate their total size. Then, it proceeds to iterate through all the buffers of the model and calculates their total size. Finally, it adds up the sizes of the parameters and buffers, converts the total to MB, and outputs this value.

In our experiments, we used the MNIST dataset with CNN model, which includes two convolutional layers, two pooling layers, two fully connected layers, two Dropout layers. We set up 2 servers with 8 clients for each and the data distribution is non-IID. We run 3000 rounds for Move-clients, Share-client with dual models(called alternate), Share-client with model combination strategies (called avg_model) and baseline and count the bandwidth consumption in each rounds with timestamps. To reduce the instability in the timer's timing, we selected the time range of 2000-10000 seconds for plotting.

Fig. 3.2 shows the change in bandwidth over time. It indicates that, Share-client with dual models strategy consumes more bandwidth compare to the other three methods and the Move-clients strategy consumes less bandwidth. Fig. 3.3 shows the cumulative bandwidth consumption over time. It shows the similar results as what Fig. 3.2 shown.

The Share-client with dual models strategy consumes more bandwidth primarily due to the way this strategy operates. In this strategy, the client needs to maintain and update two models simultaneously. As a result, during each communication round, the client must send updates for both models to the server, rather than just one. This significantly increases the amount of data being transmitted, thereby consuming more bandwidth compared to strategies that only use a single model.

For the Move-clients strategy, the client is transferred to another server where the server's data is more aligned with the overall data distribution. This alignment allows the model updates to converge faster, potentially leading to fewer communication rounds. Consequently, this reduces bandwidth consumption.

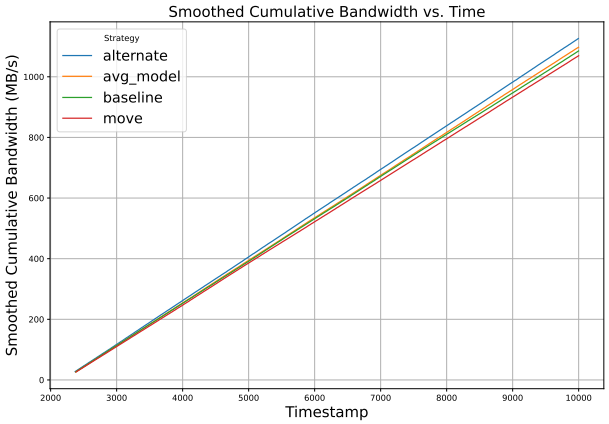


Figure 3.3: Cumulative bandwidth consumption of each strategies

3.3. Combining the Move-Clients and Share-Clients strategies

The Move-clients and Share-clients strategies were initially developed to tackle the challenges posed by non-IID data distribution in FL environments. The Move-Clients strategy involves relocating a selected client to a different server, with the goal of mitigating the negative impacts of non-IID data. Conversely, the Share-Clients strategy allows a client to alternate communication between two servers, enabling both servers to benefit from the client's data and model updates, thereby increasing data diversity across the network.

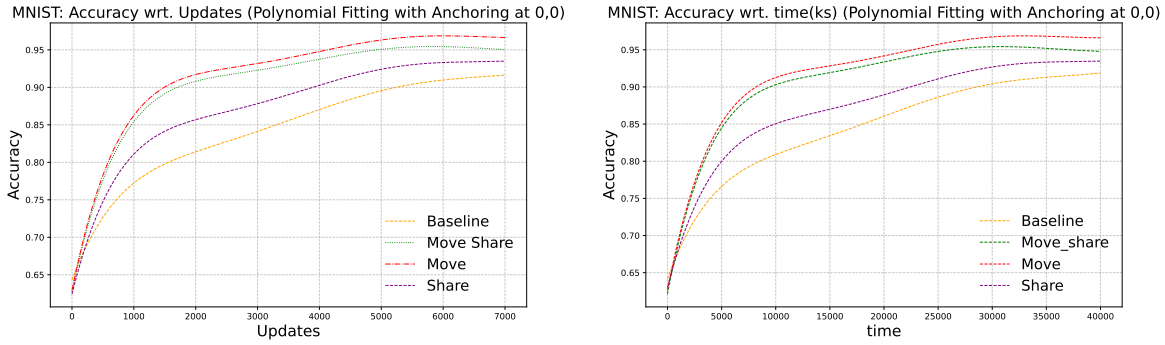
While each strategy offers distinct advantages, they also come with inherent limitations. The Move-clients strategy, despite its effectiveness in boosting data diversity on the target server, completely disconnects the client from its original server, potentially diminishing data diversity there. The Share-Clients strategy, although beneficial in environments with a higher number of clients, may struggle to achieve optimal performance when client numbers are low.

To overcome these challenges, we introduced the Move-Share strategy. This hybrid approach starts by moving the client to a new server to quickly enhance data diversity and model performance. As the training progresses, the strategy shifts to a sharing mode, where the client alternates communication between the original and new servers. Ideally, this dual approach combines the strengths of both strategies: it improves the target server's performance by increasing data diversity while also ensuring that the original server continues to benefit from the client's data. However, we will design experiments to verify its effectiveness.

In our experiments, we used the MNIST dataset and conducted two sets of experiments. In the first set, we used 16 clients, evenly distributed across two servers, with each server managing 8 clients. At the server level, the data distribution is non-IID, generated using a Dirichlet distribution with alpha = 0.1. The total round of each experiment We selected one client from one of the servers and applied three strategies: Move-Clients, Share-Clients, and Move-Share, to compare their effectiveness. In the second set, we increased the number of clients to 32, again evenly distributed across two servers, with each server managing 16 clients. We selected one client from one of the servers and applied the same three strategies. In the Move-Clients strategy, the selected client was moved to another server and communicated exclusively with the new server. In the Share-Clients strategy, the selected client alternated communication between the two servers. In the Move-Share strategy, the client was first moved to another server and then alternated communication between the two servers at the 2500th round. These experimental settings allowed us to evaluate the effectiveness of different strategies in handling non-IID data distribution.

The experimental results indicate that the Move-Share strategy exhibits a nuanced relationship with the Move-Clients and Share-Clients strategies across different client configurations. In the experiment with 16 clients, shown in Fig. 3.4, the Move-Share strategy's accuracy was slightly lower than that

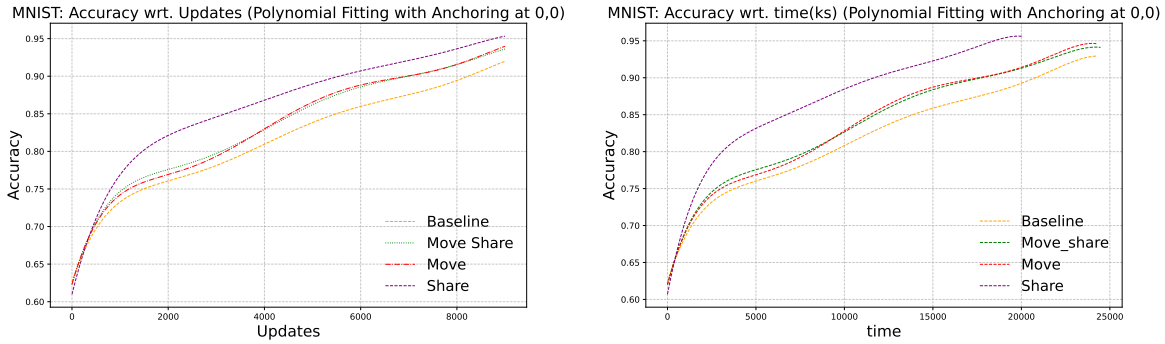
of the Move-Clients strategy, despite its attempt to combine the benefits of both moving and alternating communication. However, in the experiment with 32 clients, shown in Fig. 3.5, the Move-Share strategy performed similarly to the Move-Clients strategy, but it did not show a significant advantage. This suggests that while the Move-Share strategy aims to enhance model generalization by alternating communication, it does not provide a clear additional benefit over the Move-Clients strategy when dealing with non-IID data. Therefore, in both experimental setups, although the Move-Share strategy can achieve similar results to the Move-Clients strategy, it does not demonstrate a marked improvement or advantage.



((a)) Accuracy wrt. updates

((b)) Accuracy wrt. time

Figure 3.4: Comparison of accuracy(MNIST:16 client, 2 server, $alpha$ = 0.1)



((a)) Accuracy wrt. updates

((b)) Accuracy wrt. time

Figure 3.5: Comparison of accuracy(MNIST:32 client, 2 server, $alpha$ = 0.1)

4

Conclusion

In this thesis, we have explored the significant challenges posed by data non-IIDness in multi-server asynchronous federated learning (FL) environments. Traditional FL approaches, which rely on synchronous updates and assume homogeneous data distributions, often fall short when applied to real-world scenarios characterized by data heterogeneity. To better understand the impact of non-IID data distributions and geo-distribution on FL systems, we developed a reproducible multi-server FL model with heterogeneous resources. This framework enables controlled and consistent testing of various FL strategies under different conditions, providing a robust tool for analyzing the effects of data heterogeneity on system performance. Furthermore, we introduced and evaluated three innovative strategies to mitigate the adverse effects of non-IID data distributions: moving clients between servers, sharing clients among servers with dual models, and sharing clients with model combination.

The results demonstrate that the Moving clients strategy is particularly effective in reducing training time and improving model accuracy with less bandwidth consumption when the number of clients is relatively small. However, as the scale increases, the Sharing clients with dual models strategy proves more advantageous, offering balanced improvements in both efficiency and accuracy. On the other hand, while the model combination strategy shows potential, it does not perform as well as the other two methods under certain conditions.

These findings suggest that the proposed strategies can significantly enhance the robustness and efficiency of asynchronous FL systems, especially in environments characterized by high levels of data heterogeneity. By providing a reproducible experimental framework and systematically evaluating different approaches, this thesis contributes to the ongoing development of scalable and effective FL systems that can better handle the complexities of real-world data distributions and communication constraints.

Future research could further refine these strategies, particularly in exploring hybrid methods or adaptive approaches that can dynamically switch between strategies based on the characteristics of the data and the network environment. Additionally, applying these methods to more diverse datasets and FL models could provide deeper insights into their universality and potential limitations.

References

[1] Fan Lai et al. *Oort: Efficient Federated Learning via Guided Participant Selection*. 2021. arXiv: 2010.06081 [cs.LG]. url: https://arxiv.org/abs/2010.06081.

[2] Chenning Li et al. “PyramidFL: A fine-grained client selection framework for efficient federated learning”. In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 2022, pp. 158–171.

[3] Lumin Liu et al. *Client-Edge-Cloud Hierarchical Federated Learning*. 2019. arXiv: 1905.06641 [cs.NI]. url: https://arxiv.org/abs/1905.06641.

[4] H. Brendan McMahan et al. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. 2023. arXiv: 1602.05629 [cs.LG]. url: https://arxiv.org/abs/1602.05629.

[5] Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. *Proximal and Federated Random Reshuffling*. 2021. arXiv: 2102.06704 [cs.LG]. url: https://arxiv.org/abs/2102.06704.

[6] Liangqiong Qu, Niranjan Balachandar, and Daniel L Rubin. *An Experimental Study of Data Heterogeneity in Federated Learning Methods for Medical Imaging*. 2021. arXiv: 2107.08371 [cs.LG]. url: https://arxiv.org/abs/2107.08371.

[7] Canh T. Dinh, Nguyen Tran, and Josh Nguyen. *Personalized Federated Learning with Moreau Envelopes*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 21394–21405. url: https://proceedings.neurips.cc/paper_files/paper/2020/file/f4f1f13c8289ac1b1ee0ff176b56fc60-Paper.pdf.

[8] Hongyi Wang et al. *Federated Learning with Matched Averaging*. 2020. arXiv: 2002.06440 [cs.LG]. url: https://arxiv.org/abs/2002.06440.

[9] Cong Xie, Sanmi Koyejo, and Indranil Gupta. *Asynchronous Federated Optimization*. 2020. arXiv: 1903.03934 [cs.DC]. url: https://arxiv.org/abs/1903.03934.

[10] Yue Zhao et al. *Federated Learning with Non-IID Data*. 2018. doi: 10.48550/ARXIV.1806.00582. url: https://arxiv.org/abs/1806.00582.