To appear in *Proceedings of 39th IEEE International Parallel and Distributed Processing Symposium* (*IPDPS 2025*), June 2025.

SEAFL: Enhancing Efficiency in Semi-Asynchronous Federated Learning through Adaptive Aggregation and Selective Training

Md Sirajul Islam[1], Sanjeev Panta[1], Fei Xu[2], Xu Yuan[3], Li Chen[1], and Nian-Feng Tzeng[1]
[1]School of Computing and Informatics, University of Louisiana at Lafayette, USA
[2]School of Computer Science and Technology, East China Normal University, China
[3]Department of Computer and Information Sciences, University of Delaware, USA

***Abstract*—Federated Learning (FL) is a promising distributed machine learning framework that allows collaborative learning of a global model across decentralized devices without uploading their local data. However, in real-world FL scenarios, the conventional synchronous FL mechanism suffers from inefficient training caused by slow-speed devices, commonly known as stragglers, especially in heterogeneous communication environments. Though asynchronous FL effectively tackles the efficiency challenge, it induces substantial system overheads and model degradation. Striking for a balance, semi-asynchronous FL has gained increasing attention, while still suffering from the open challenge of stale models, where newly arrived updates are calculated based on outdated weights that easily hurt the convergence of the global model. In this paper, we present *SEAFL*, a novel FL framework designed to mitigate both the straggler and the stale model challenges in semi-asynchronous FL. *SEAFL* dynamically assigns weights to uploaded models during aggregation based on their staleness and importance to the current global model. We theoretically analyze the convergence rate of *SEAFL* and further enhance the training efficiency with an extended variant that allows partial training on slower devices, enabling them to contribute to global aggregation while reducing excessive waiting times. We evaluate the effectiveness of *SEAFL* through extensive experiments on three benchmark datasets. The experimental results demonstrate that *SEAFL* outperforms its closest counterpart by up to $\sim$22% in terms of the wall-clock training time required to achieve target accuracy.**

***Index Terms*—Federated Learning, System Heterogeneity, Asynchronous Federated Learning, Partial Training**

I. Introduction

In recent years, the proliferation of edge devices has resulted in a significant surge in distributed data generation that can be leveraged for machine learning and smart applications. However, with the introduction of stringent laws and regulations such as the GDPR [1] in 2018, traditional methods based on data aggregation into a centralized data center raise serious privacy concerns and become increasingly unfeasible. As a promising alternative, Federated Learning (FL) [2] has emerged to enable collaborative model training without the need of transferring raw data. FL leverages distributed user data while preserving privacy by exchanging the gradients or model updates of participating devices. Due to its superior privacy implications, FL has been applied in diverse areas such as natural language processing [3], computer vision [4], healthcare [5], and human activity recognition [6].

Traditional FL [2] training typically relies on a parameter server to orchestrate the training process across devices using a synchronous mechanism. This synchronous training approach involves multiple rounds, each comprising the following steps. Initially, the server chooses a subset of devices and broadcasts the global model to them. Then, local training is performed on each selected device using its own data. Subsequently, each device sends the model updates back to the server. Finally, the server aggregates the received updates to produce a new global model once all chosen devices finish the aforementioned steps. Despite its efficiency and ease of implementation, the synchronous mechanism is susceptible to stragglers (slow devices), which can significantly prolong the training process [7], particularly when dealing with heterogeneous devices [8], [9]. This could severely impact training efficiency as powerful devices may remain inactive while the server waits for stragglers [10], posing critical challenges that greatly hinder the scalability of synchronous FL methods in large-scale cross-device scenarios.

To tackle these limitations, recent studies have introduced asynchronous FL (AFL) [5], [10]–[12], allowing the server to aggregate uploaded models without waiting for stragglers, which may instead contribute to future aggregation rounds. In fully AFL such as FedAsync [11], the server initiates the global model aggregation immediately upon receiving a single model update. Although this approach alleviates the straggler issue, it introduces stale model updates, leading to slower convergence and accuracy degradation [13]. Additionally, it incurs significant computational overhead due to excessive server aggregation.

As a compromise, semi-asynchronous FL methods [5], [10], [14], [15] buffer a specified number of local updates for aggregation in each round, as illustrated in Fig. 1. Once receiving a sufficient number (*i.e.*, 3 for the example in

The research is supported in part by the NSF under grants OIA-2327452 and OIA-2019511, in part by the Louisiana BoR under LEQSF(2024-27)-RD-B-03, and in part by the NSFC under 62372184 and by the Sci. and Tech. Commission of Shanghai Municipality under 22DZ2229004.
Corresponding author: Li Chen. Email: li.chen@louisiana.edu

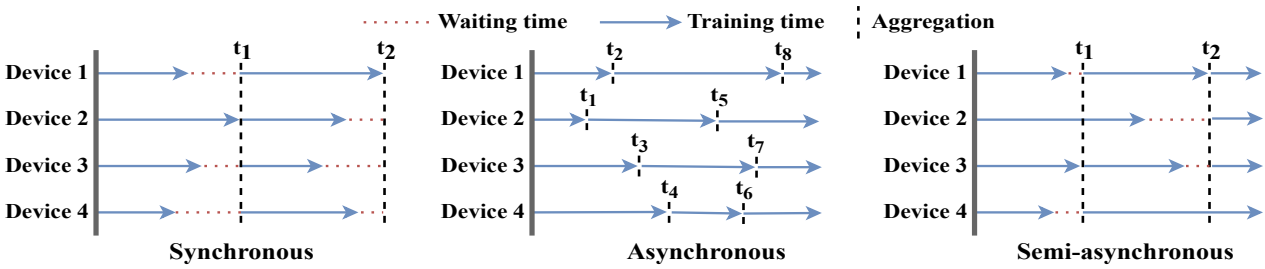


Fig. 1: The working process of synchronous, asynchronous, and semi-asynchronous FL algorithms.

Fig. 1) of updates, the server updates the global model without waiting for slower devices (*i.e.*, Device 2 in the first round). Those devices failing to participate in the aggregation can continue their training to completion and potentially contribute to future aggregation. Some approaches [10], [15] discard local updates from slower devices based on a staleness threshold, resulting in wasted training efforts. Excluding them would also impede the convergence of the global model and delay the training process. To let slower devices contribute to the global aggregation while accounting for the staleness of their updates, existing semi-asynchronous FL methods leverage static polynomial formulas [5], [16] or simple attention mechanisms [17]. However, they are limited in their abilities to determine and dynamically adjust the significance of received updates during aggregation, resulting in suboptimal training efficiency and model accuracy.

To fill this gap, we propose a novel staleness-aware semi-asynchronous FL framework (*SEAFL*), to effectively and efficiently learn from devices with heterogeneous system characteristics. Based on empirical insights, *SEAFL* strikes for an optimal balance between involving more devices to contribute to global aggregations and reducing aggregation overheads. Moreover, having identified that the contribution of each device's local updates on the global model varies with their staleness across rounds, *SEAFL* dynamically assigns weights to local updates during aggregation to ensure efficient collaborative learning in heterogeneous environments. The essence of *SEAFL* lies in an adaptive weight aggregation mechanism to address the stale model problem by considering both the staleness of the received model updates and their similarity to the current global model. In contrast to prior work, our method emphasizes the importance of local updates according to the current global model which effectively accelerates model convergence. Additionally, to enhance training efficiency, we introduce a variant, *SEAFL*^{2}, which further reduces the training time by enabling partial training on straggler devices. To demonstrate the efficacy of *SEAFL*, we conduct extensive experiments on three benchmark datasets, comparing our approach with existing state-of-the-art (SOTA) FL methods. Results demonstrate that *SEAFL* significantly outperforms the SOTA FL approaches, especially its closest counterpart, *FedBuff*, by reducing the wall-clock training time required to achieve target accuracy, for up to $\sim$22%.

Our key contributions are summarized as follows:

- We investigate the impacts of local update staleness, buffer size, and importance of local updates in asynchronous FL training.
- We introduce *SEAFL*, a novel staleness-aware semi-asynchronous FL framework that adaptively assigns weights to local updates while considering their staleness and importance to the current global model.
- We empirically show the effectiveness of *SEAFL*, which outperforms other SOTA FL methods in terms of reducing the wall-clock training time required to achieve target accuracy.
- We present a theoretical analysis of the convergence behavior of the proposed *SEAFL* algorithm.
- In addition, to improve training efficiency, we propose a variant of *SEAFL*, *SEAFL*^{2}, that enables partial training on slower devices, allowing them to contribute to the global aggregation process and to reduce the excessive waiting time.

The rest of this paper is structured as follows. We review related work in Section II. Section III provides our preliminary insights. Section IV outlines the problem formulation and the design of our proposed *SEAFL* framework. A theoretical convergence analysis is presented in Section V. The experimental settings and results are given in Section VI. Finally, Section VII concludes the paper.

II. Related Work

A. Synchronous Federated Learning

A plethora of FL approaches [2] have been proposed to jointly train a global model by leveraging distributed user data. Many of them [2], [18] rely on a synchronous mechanism for aggregating models on the server. However, this approach requires the server to wait for all selected devices to transmit their model updates before performing aggregation, which has proven inefficient due to the presence of stragglers. As the number of devices increases and system heterogeneity grows, the probability of encountering straggler effects also rises. This issue significantly impedes the scalability of synchronous FL. Existing work tackles system heterogeneity and statistical heterogeneity separately. Several approaches, including regularization [18], personalization [19], [20], clustering [21], [22], and device selection [8], [23], have been proposed

in the literature to tackle statistical heterogeneity. However, these approaches lack the capability to dynamically adjust the significance of diverse models and instead focus solely on the synchronous mechanism.

Three different strategies are introduced in the literature to tackle system heterogeneity within the synchronous mechanism. Firstly, some methods focus on scheduling appropriate devices for local training while considering their computational and communication capabilities to achieve load balance and mitigate inefficiencies caused by stragglers [8], [24]. However, this type of approaches may reduce the participation frequency of less powerful devices, leading to decreased accuracy. Secondly, techniques such as pruning [25] or dropout [26] are leveraged during training, resulting in lossy compression and reduced accuracy. Thirdly, the clustering approach [27] groups devices with similar capacities into clusters and utilizes a hierarchical architecture [28] for model aggregation. Although these approaches aim to optimize the synchronous mechanism, they often suffer from low efficiency and may lead to significant accuracy degradation due to statistical heterogeneity.

B. Asynchronous and Semi-asynchronous FL

To address the system heterogeneity, AFL [5], [11] facilitates global model aggregation without the need to wait for all devices. In AFL, aggregation can be performed immediately upon receiving an update from any device [11], [17] or when multiple updates are buffered [5], [15], [29]. In *FedAsync* [11], the server employs a mixing hyperparameter α to determine the weight allocated to the newly arrived model update based on that of the fastest device during the aggregation. In fully AFL [11], [17], the aggregation process is no longer delayed by slower devices. Upon finishing their local training, their model updates may be based on an earlier version of the global model compared to those of faster devices. However, outdated uploaded models from stale devices may revert the global model to a previous state, significantly reducing accuracy [17]. Furthermore, it incurs excessive computation overhead due to frequent aggregation on the server.

Hence, the semi-asynchronous FL was introduced as a trade-off between synchronous and asynchronous FL. It alleviates the excessive computation overhead and privacy concerns by buffering a certain number of local updates instead of aggregating them immediately. Wu *et al.* proposed *SAFA* [10], which categorizes devices according to their training status to enhance convergence performance. It discards stale model updates based on a hyperparameter called lag tolerance. *FedSA* [30] introduced a two-phase FL training process, employing a large number of epochs during the initial training phase, and then switching to a reduced number of local epochs in the convergence phase. It adjusted the number of local training epochs in each round according to the device's staleness. *Fedbuff* [5] enables secure aggregation by keeping a predefined number of local updates in a secure buffer before aggregation. Liu *et al.* proposed *FedASMU*, [15] a reinforcement learning approach to dynamically choose a time slot for triggering server-side aggregation. However, it incurs additional computation overhead on both the device and server side.

Recent work, *EAFL* [14], introduced gradient similarity-based clustering and a two-stage aggregation strategy to address data and system heterogeneity issues in asynchronous FL. Nevertheless, it relies on a predefined number of clusters, which is challenging to determine without knowing the actual data distributions across devices, thus limiting flexibility and adaptability. Most of prior pursuits [5], [11] did not impose any staleness limitations on device updates, resulting in stale model updates that hinder the convergence of the final model. Furthermore, it does not perform well in cases of low data heterogeneity while incurring additional computation overhead. Unlike existing approaches, we introduce a semi-asynchronous FL framework, i.e., *SEAFL*, to tackle system heterogeneity. *SEAFL* dynamically adjusts weights to the received model updates according to their staleness and importance during global aggregation to minimize loss and improve accuracy. Moreover, our approach facilitates partial training on slower devices, enabling them to contribute to the global aggregation.

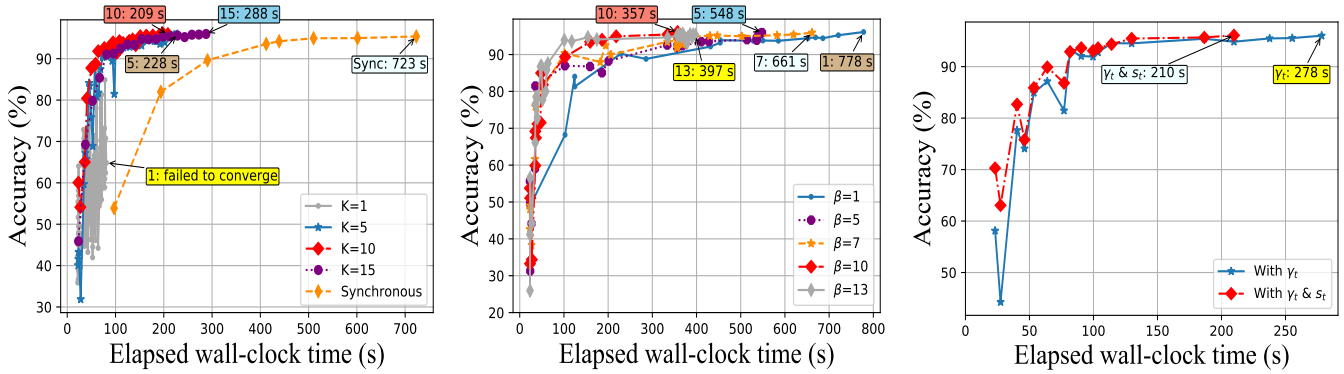
III. Preliminary Insights

In this section, we conduct preliminary experiments to analyze the impact of buffer size, model staleness, and the significance of local model updates to the global aggregation on semi-asynchronous FL training.

Our experimental setup involves 100 devices utilizing the MNIST dataset to train a LeNet-5 model. We simulate a non-IID distribution using the Dirichlet distribution [31] with a concentration parameter 0.3. Each device trains the model using 600 training samples. AFL is most suitable for scenarios where a few devices exhibit significantly slower training speeds, leading to a heavy-tailed distribution of local training speeds. To simulate this scenario in our testbed, we randomly generate idle period durations for each device after completing an epoch. These durations are sampled from a Zipf distribution [32] with parameter $s = 1.7$ and a maximum length of 60 seconds. In the synchronous mode, the server chooses 20 devices for training in each round. We vary the values of buffer size (K) and staleness limit (β) in a semi-asynchronous FL setting, and plot the results in Fig. 2. We have the following observations.

Buffer size. In AFL, the buffer size refers to the number of updates the server will wait before triggering the aggregation process. When the buffer size is set to 1, the server operates in fully asynchronous mode and immediately performs aggregation upon receiving any update. For instance, *FedAsync* [11] and *ASO-Fed* [17] are designed to work in this fully asynchronous mode. On the contrary, when the buffer size is equal to the number of devices selected for each round, the training reverts to the synchronous mode, similar to traditional *FedAvg* [2]. In this scenario, the server waits for all the chosen devices to upload their updates before starting the aggregation process.

We measure the elapsed wall-clock time as a performance metric while varying the number of updates needed before



(a) Varying the buffer size (K) (b) Varying the limit of staleness (β) (c) Effects of local updates importance

Fig. 2: Illustration of the impacts of buffer size, staleness limit, and importance of local updates on asynchronous FL, where γ_t indicates staleness factors, and s_t denotes the importance of updates.

the server starts aggregation. Fig. 2a clearly illustrates that the fully asynchronous approach, which immediately commences aggregation upon the arrival of a single update, failed to achieve convergence. This is because each device uses only 600 samples, and the server frequently aggregates model updates from faster devices. Consequently, when updates from considerably slower devices finally arrive, they are based on significantly outdated models. Moreover, the presence of a non-IID data distribution further exacerbates this issue.

On the other hand, synchronous FL did achieve convergence; however, it required a significantly longer wall-clock time. This indicates the well-known straggler issue, where the server is forced to wait for slower devices in each round. In our experiments, aggregating a minimum of 10 device updates yields the optimal outcome, taking only 209 seconds to reach a target accuracy of 96%.

Model staleness. The staleness of devices refers to the number of rounds that have passed since the device last received the global model from the server. We vary the staleness limit to observe its impact on the wall-clock time required to reach convergence. Intuitively, it may not be ideal to impose excessive restrictions when waiting for devices that are only slightly behind. Conversely, we must also be cautious not to incorporate devices that are excessively stale, as their models may be significantly out of sync with the majority. As depicted in Fig. 2b, the results of our preliminary experiments conducted on the MNIST dataset with $K = 10$ appear to confirm our intuition, indicating that the staleness limit of 10 provides the best performance. The notable difference in performance reveals that achieving a target accuracy of 96% required 778 seconds with a staleness limit of 1, whereas it only took 357 seconds with a staleness limit of 10. The choice of staleness limit significantly impacts the wall-clock time required to reach a target accuracy in asynchronous FL.

Importance of uploaded models. In AFL, multiple devices are training simultaneously using different versions of the global model. Their updates, when used for aggregation, may not equally contribute to or even be beneficial for the global model convergence. We measure the importance of each received update relative to the current global model, to be outlined in Section IV. Intuitively, if the aggregation weight of each update is set to be proportional to the contribution of the device, the performance can be further improved. Fig. 2c illustrates that incorporating the significance of local updates reduces the wall-clock time to achieve the target accuracy to 210 seconds, compared to 278 seconds without this consideration.

TABLE I: Notations and Descriptions

Symbol	Description
D	the complete dataset
$\mathcal{D}_k$	the local dataset of device k
N	the number of devices
K	the buffer size of device updates
E	the number of local training epoch
t	the current communication round
S_k	the staleness of device k's update
α	staleness weight
γ_t^k	the staleness factor for device k's update at round t
s_t^k	the importance of device k's update at round t
μ	similarity weight
β	staleness limit
Θ	the cosine similarity between two vectors
p_k	the weight assign to device k updates during aggregation

Insight: The performance of semi-asynchronous FL is greatly impacted by the buffer size and the stateless limit. In addition, not all local updates contribute equally for global optimization, calling for a weighting scheme that adaptively assigns weights to received updates based on their significance degrees for global aggregation.

Inspired by the aforementioned observations and insights, we have designed a staleness-aware semi-asynchronous FL framework with an adaptive weight aggregation mechanism called *SEAFL*. It dynamically assigns weights to received updates based on their staleness and importance degrees. The specifics of *SEAFL* will be described in the next section.

IV. System Design

A. Problem Formulation

In this section, we present the formulation of the FL training problem in a simplified setting. For clarity, we provide a

list of key notations frequently used throughout this paper in Table I. Consider a group of N devices collaborating in a federated learning process to train a shared model and determine an optimal set of parameters that minimize the global loss function:

$$\min_{w} F(w) \triangleq \sum_{k=1}^{N} p_k F_k(w) \tag{1}$$

Here, device k has a local dataset $\mathcal{D}_k$, $D = \sum_{k=1}^{N} \mathcal{D}_k$, and $p_k = \frac{|\mathcal{D}_k|}{|D|}$. Subsequently, the local objective function of device k is defined as the empirical loss computed over its local dataset, $\mathcal{D}_k$:

$$F_k(w) = \frac{1}{|\mathcal{D}_k|} \sum_{j_k=1}^{\mathcal{D}_k} f_{j_k}(w; x_{j_k}, y_{j_k}) \tag{2}$$

where $|\mathcal{D}_k|$ represents the number of local samples on each device. Each device trains the model independently on its local dataset and transmits the model updates back to the server. The most widely used synchronous FL algorithm *FedAvg* [2] aggregates received local model updates after each round to produce the new global model as:

$$w_{t+1}^{g} \leftarrow \sum_{k=1}^{M} p_k F_k(w) \tag{3}$$

Here, M is the number of devices selected for training in each round. It naively assigns weights (p_k) to device updates while aggregating based on the percentage of each device samples among the total number of samples in each round. However, this straightforward weight allocation scheme fails to account for the staleness and significance of model updates in asynchronous FL training scenarios. Consequently, it leads to a degradation in the accuracy of the global model, especially when dealing with stragglers and non-IID data distributions.

B. Adaptive Weight Aggregation

In this section, we present the proposed *SEAFL* with an adaptive weighted aggregation mechanism that dynamically allocates weight to each received update based on their staleness, and the importance of each update compared to the current global model. The primary design objective of *SEAFL* is to optimize the wall-clock training time required for an FL task to achieve a target accuracy, rather than focusing on the total number of communication rounds. We have identified key influential factors affecting AFL training through preliminary experiments and subsequently use these factors to assign weights to received updates adaptively during aggregation.

Staleness factor. In AFL training, slower devices that obtained the global model from the server several rounds earlier are prone to have outdated updates. As a result, their model updates may not significantly contribute to the aggregation process in terms of quality, resulting in slower convergence of the global model. Therefore, the weight allocated to these outdated updates should be reduced during aggregation. As the staleness of an update increases, its aggregation weight should also be correspondingly diminished. Since *SEAFL* synchronously waits for devices that exceed the staleness threshold, their staleness will always remain below that threshold. More specifically, let t denote the ongoing round at the server, and t_k represent the round in which device k last obtained its model from the server. The staleness of device k's update is computed as $t - t_k$. We measure the staleness of each update using the following staleness function, which will be used for adjusting the aggregation weights:

$$\gamma_t^k = \alpha \cdot \frac{\beta}{(t - t_k) + \beta} \tag{4}$$

Here, $t - t_k = S_k$ represents the staleness of device k's update, β is the staleness limit which follows $S_k \leq \beta$, and α serves as a hyperparameter controlling the significance of the staleness factor in the aggregation process.

Algorithm 1: *SEAFL*

Input: N: Number of available clients, K: buffer size, β: staleness limit, E: local training epochs, η: local learning rate, B: local mini-batch size, α: staleness factor , μ: similarity factor.
Output: w_T: The global model at Round T;
Server Initializes: Initialize t = 0, w_0^g;
Server Executes:
for t = *0, 1, 2,...,* $T-1$ **do**
  Server chooses a subset $\mathcal{S}_t$ of N devices at random;
  Broadcast w_t^g to all selected clients;
  flag = 0;
  while $flag \leq K$ **do**
    Server receives local model updates from clients w_t^k;
    Server stores received updates into the buffer;
    $flag$ +=1;
  end
  Server evaluates γ_t^k by Eq. (4);
  Server calculates s_t^k by Eq. (5);
  Server determines p_t^k by Eq. (6);
  Server aggregates parameters in K:
  $w_{t+1}^g \leftarrow \sum_{k=1}^{K} p_t^k w_t^k$;
  Server updates the global model:
  $w_{t+1}^g \leftarrow (1-\vartheta) w_t^g + \vartheta w_t^{new}$;
  Server sends w_{t+1}^g to the K newly updated clients;
end
ClientUpdate:
  Client k receives global model parameter w_t^g;
  $w_t^k \leftarrow w_t^g$;
  for *each client* $k \in \mathcal{S}_t$ *in parallel* **do**
    for *each local epoch* $l = 1, 2, \ldots, E$ **do**
      for *each batch* b *in* B_k **do**
        $w_{t+1}^k = w_t^k - \eta \Delta f(w_t^k; b)$;
      end
    end
  end
  Upload w_{t+1}^k to the server;

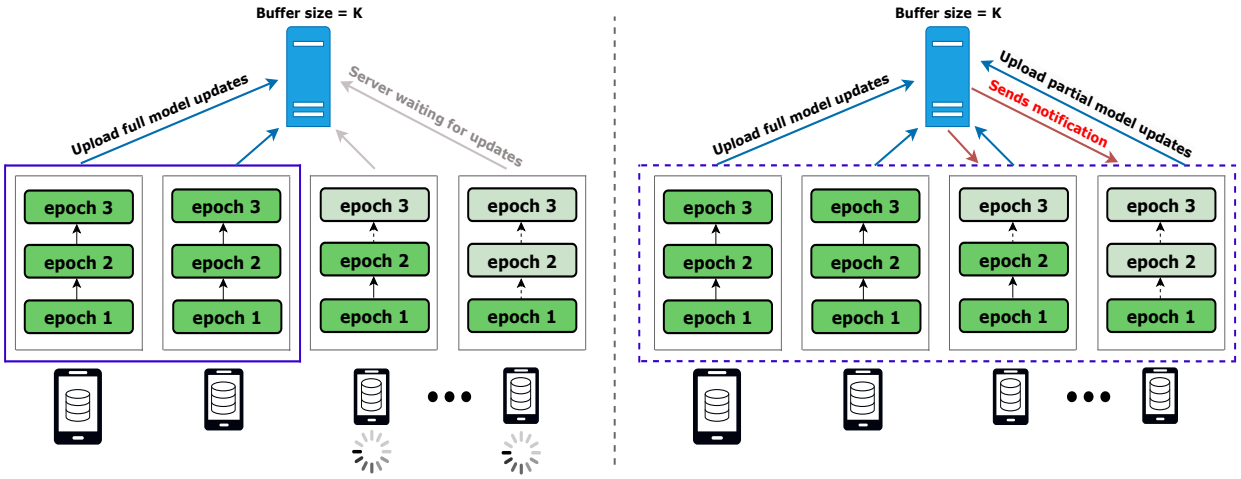


Fig. 3: **Left**: The traditional AsyncFL architecture where the server initiates aggregation upon receiving the required number of local updates. **Right**: The proposed *SEAFL*^{2} allows partial training on slower devices, enabling them to contribute to global aggregation. The server will notify slower devices to send their updates immediately after exceeding the staleness limit.

Importance of updates. From our preliminary experiments, it is evident that considering staleness alone does not yield optimal results. To improve the *SEAFL* performance further, we introduce the concept of incorporating the importance of local updates relative to the current global model. Specifically, we prioritize local updates that demonstrate higher similarity to the current global model and consequently assign them a higher weight. Mathematically, two methods can be utilized to assess the similarity between two vectors quantitatively. The first method involves computing the dot product, which considers both the magnitude and the angle between the vectors. In contrast, cosine similarity offers an alternative by focusing exclusively on the angle between the vectors. In *SEAFL*, we utilize cosine similarity, represented as s_t^k, to measure the similarity between two vectors quantitatively. A lower value of s_t^k indicates less similarity between the two vectors. We normalize s_t^k values to [0, 1] by computing $(\Theta + 1)/2$ instead. The significance of the update received from device k at global round t is therefore defined as:

$$s_t^k = \mu \cdot \frac{\Theta(\Delta_t^k, w_t^g) + 1}{2} \tag{5}$$

Similar to the staleness factor, we introduce another hyperparameter μ, serving as another tuning knob to control the importance of each update during aggregation. After computing both influential factors, and considering that each device k executes E training epochs on its local dataset $\mathcal{D}_k$, the aggregation weight for each device can be determined as follows:

$$p_t^k = \frac{|\mathcal{D}_k|}{|D|}(\gamma_t^k + s_t^k) \tag{6}$$

in which D represents the collection of all data samples utilized by the participating devices K in the current round. The server normalizes all p_t^k so that their sum equals 1, and then aggregates the K parameters from the buffer as follows:

$$w_t^{new} \leftarrow \sum_{k=1}^{K} p_t^k w_t^k \tag{7}$$

After acquiring w_t^{new}, the server employs a weighted averaging strategy to update the global model:

$$w_{t+1}^g \leftarrow (1 - \vartheta) w_t^g + \vartheta w_t^{new} \tag{8}$$

where the hyper-parameter $\vartheta \in (0, 1)$. The server then transmits the updated global model w_{t+1}^g to the newly updated device for the upcoming round of local training. The pseudocode of *SEAFL* is shown in Algorithm 1.

C. Partial Training

In the design of *SEAFL*, we acknowledge that the adaptive weighted aggregation mechanism by itself may not provide optimal outcomes in terms of wall-clock training time, especially when compared to the total number of rounds. This issue arises when a few significantly slower devices exceed the staleness threshold, potentially becoming stragglers and resulting in an extended training time needed to reach the target accuracy. Unlike existing works [5], [15] addressing the AFL challenges, which require all devices to complete an equal number of local epochs for each update regardless of device heterogeneity, the proposed *SEAFL*^{2} enables partial training on slower devices. Moreover, some existing approaches [10], [15] discard local updates that exceed a predefined staleness limit, resulting in wasted training efforts and slower devices being unable to contribute to the global aggregation which may impede the model's convergence.

To alleviate the negative impacts of these stragglers, the server in *SEAFL*^{2} notifies all devices that exceed the staleness limit. Upon receiving such a notification, devices refrain from advancing to the next epoch of local training. Instead, they

immediately transmit their local model updates upon completing the ongoing training epoch. This proves beneficial for the server when dealing with slower devices, as it eliminates the need to wait for the completion of all local epochs on those devices. Rather, the server only needs to wait for the ongoing epoch to finish.

Fig. 3 illustrates how the server notifies stale devices to send their updates immediately. The buffer size and staleness limit discussed in Section III remain applicable in *SEAFL*^{2}. The server continuously waits to receive a requisite number of device updates. However, with the partial training strategy, it also monitors whether any devices exceed the staleness limit. If such scenarios arise, the server will send notifications to these devices. These notifications introduce an additional round trip between the server and devices with stale updates. After receiving the notification, these devices will transmit their model updates immediately upon finishing their current training epoch.

V. Theoretical Analysis

We consider the following theoretical context to analyze *SEAFL*'s convergence behavior. In each round $t \in T$, the server chooses *M* devices from a pool of *N* devices. Each device k executes *E* epochs of training on its local dataset $\mathcal{D}_k$, utilizing the model $w_{t_k}^k$ received from the server in round t_k. During each local training epoch $i \in [0, E]$, the local model $w_{t_k,i+1}^k$ is updated using an SGD optimizer with a learning rate η_l^i and a batch size *B*. This process can be expressed as $w_{t_k,i+1}^k = w_{t_k}^k - \eta_l^i g(w_{t_k,i}^k)$, where the gradient $g(w_{t_k,i}^k) = \nabla f_k(w_{t_k,i}^k, \mathcal{D}_k)$. The server commences the aggregation process once *K* devices have reported. We first outline the key assumptions necessary to present our theoretical analysis on the convergence of *SEAFL*, listed in the following.

Assumption 1. *(Lipschitz gradient) The objective function of each device f_k is L-smooth. Thus f_k has Lipschitz continuous gradients with constant $L > 0$, i.e., $\|\nabla f_k(w) - \nabla f_k(w')\| \leq L\|w - w'\|$.*

Assumption 2. *(Unbiased local gradient) For each device the stochastic gradient $\nabla f_k(w;\xi)$ is unbiased, i.e., $\mathbb{E}[f_k(w;\xi] = \nabla f_k(w)$.*

Assumption 3. *(Uniformly bounded local gradient) The expected squared norm of stochastic gradients is uniformly bounded, i.e., $\mathbb{E}\|\nabla f_k(w;\xi\|^2 \leq G^2$ for all $k = 1, \cdots, K$ and $t = 1, \cdots, T-1$.*

Assumption 4. *(Bounded local gradients) Let ξ be a sample drawn uniformly at random from the local data of the k-th device. The variance of the stochastic gradients for each device is constrained as follows: $\mathbb{E}_\xi\|f_k(w;\xi) - f_k(w)\|^2 \leq \sigma_k^2$ for $k = 1, \cdots, K$. We then define $\sigma_l^2 := \sum_{k=1}^K \frac{|\mathcal{D}_k|}{|D|}\sigma_k^2$.*

Algorithm 2: *SEAFL*^{2}

Input: *N*: Number of available clients, *K*: buffer size, β: staleness limit, *E*: local training epochs, η: local learning rate, *B*: local mini-batch size, α: staleness factor , μ: similarity factor.
Output: w_T: The global model at Round *T*;
Server Initializes: Initialize *t* = 0, w_0^g;
Server Executes:
for *t = 0, 1, 2,..., T − 1* **do**
  Server chooses a subset $\mathcal{S}_t$ of *N* devices at random;
  Broadcast w_t^g to all selected clients;
  flag = 0;
  while $flag \leq K$ **do**
    Server receives local model updates from clients w_t^k;
    Server stores received updates into the buffer;
    $flag$ +=1;
  end
  for *each client* $k \in \mathcal{S}_t$ **do**
    if *client k's update exceed* β **then**
      Send a notification to client k;
  end
  Server evaluates γ_t^k by Eq. (4);
  Server calculates s_t^k by Eq. (5);
  Server determines p_t^k by Eq. (6);
  Server aggregates parameters in *K*:
  $w_{t+1}^g \leftarrow \sum_{k=1}^K p_t^k w_t^k$;
  Server updates the global model:
  $w_{t+1}^g \leftarrow (1-\vartheta)w_t^g + \vartheta w_t^{new}$;
  Server sends w_{t+1}^g to the *K* newly updated clients;
end
ClientUpdate:
  Client *k* receives global model parameter w_t^g;
  $w_t^k \leftarrow w_t^g$;
  for *each client* $k \in \mathcal{S}_t$ *in parallel* **do**
    for *each local epoch* $l = 1, 2, \ldots, E$ **do**
      for *each batch b in* B_k **do**
        $w_{t+1}^k = w_t^k - \eta\Delta f(w_t^k; b)$;
      end
      if *k receives a notification* **then**
        Finish the current epoch;
        Send w_{t+1}^k to the server immediately;
      else
        Continue training remaining epochs;
    end
  end
  Upload w_{t+1}^k to the server;

Assumption 5. *(Bounded gradient dissimilarity) For any device k and parameter w, we denote δ_k as the upper bound for $\|f_k(w) - f(w)\|^2$, i.e., $\|f_k(w) - f(w)\|^2 \leq \delta_k^2$. We then define $\delta_g^2 := \sum_{k=1}^K \frac{|\mathcal{D}_k|}{|D|}\delta_k^2$.*

SEAFL can be characterized as an asynchronous aggregation

problem that incorporates buffered updates, a concept previously addressed in *FedBuff* [5]. Additionally, *SEAFL*'s partial training strategy also guarantees a staleness limit to device updates as mentioned in Section IV. Furthermore, we mathematically define a staleness factor that leverages the devices's staleness to modify the weights assigned to each gradient. By incorporating the importance factor, we establish Lemma 1 regarding the weights assigned to each gradient.

Lemma 1. *Given the hyperparameters associated with the staleness factor and importance factor, α and μ, the aggregation weight p_t^k for each gradient can be bounded within the interval $p_t^k \in [\frac{\alpha}{2}d_k, (\alpha+\mu)d_k]$ where $d_k = \frac{|\mathcal{D}_k|}{|D|}$.*

We can simplify Lemma 1 by ignoring the denominator term since it does not influence the convergence proof. Consequently, we can derive the convergence rate of *SEAFL* as follows:

Theorem 1 (Convergence rate) *Based on Assumptions 1 to 4 and Lemma 1, the convergence rate of SEAFL can be formulated as follows:*

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(w_t)\|^2 \leq \frac{2(f(w_0)-f(w^*))}{\Omega(E)TK} \\ + 6K(\alpha+\mu)^2\lambda(d)L^2Q\phi(E)(K^2\beta^2+1)\sigma^2 \\ + \frac{\phi(E)L}{K\Omega(E)}(\alpha+\mu)\sigma_l^2\end{aligned} \tag{9}$$

where $\Omega(E) = \sum_{i=1}^{E} n_l^i$, $\lambda(d) = \sum_{j=1}^{K} d_j^2$, $\phi(E) = \sum_{i=1}^{E}(\eta_l^i)^2$, *and* $\sigma^2 = (\alpha+\mu)\sigma_l^2 + (\alpha+\mu)\sigma_g^2 + G^2$.

To achieve the upper bound on convergence, the relationship between K and n_l must satisfy the following condition:

$$\frac{4(\alpha+\mu)}{\alpha^2\lambda(d)}K\eta_l^i \leq \frac{1}{L} \tag{10}$$

Proof. Following the conventional method for proving convergence in federated learning algorithms, such as in [5], which handles non-convex objective function, our proof begins by applying the smoothness Assumption 1. This allows us to establish an upper bound for $f(w_{t+1})$ as follows:

$$\begin{aligned}f(w_{t+1}) \leq f(w_t) - \sum_{k\in K} p_t^k(\nabla f(w_t), \Delta_{t_k}) \\ + \frac{L}{2}\|\sum_{k\in K} p_t^k \Delta_{t_k}\|^2\end{aligned} \tag{11}$$

where $\Delta_{t_k} = \sum_{i=1}^{E} n_l^i \nabla f_k\left(w_{t_k,i}^k\right)$.

Then, as presented in Eq. (7), *SEAFL* evaluates the staleness factor for each gradient. It incorporates a new aggregation mechanism that generalizes the scenario considered in *FedBuff* [5], where equal weights are assigned during aggregation. Specifically, we outline our proof in three parts.

Initially, we derive the upper bound for three crucial components. According to Assumptions 3, 4, 5, and Lemma 1, we establish a bound on the expectation of the stochastic gradient $\mathbb{E}\|\nabla f_k(w_{t_k,i}^k, \mathcal{D}_k)\|$ of device k by $\sigma^2 = (\alpha+\mu)\sigma_l^2 + (\alpha+\mu)\sigma_g^2 + G^2$. Next, we prove the upper bound for staleness-aware gradient divergence by utilizing Assumption 1 and including a zero term in the decomposition $\mathbb{E}\|\sum_{k=1}^{K} p_t^k(\nabla f_k(w_t) - f_k(w_{t_k}^k))\|^2$ is $6K\sum_{k=1}^{K}(p_t^k)^2\sum_{k=1}^{K} L^2Q\phi(E)(K^2\beta^2+1)\sigma^2$. Finally, we establish a bound on $\mathbb{E}\|\sum_{k\in K} p_t^k\Delta_{t_k}\|^2$ by employing Lemma 1 and Assumption 5.

Then, incorporating these derived components into Eq. (11), we manipulate the equation to derive the specific upper bound for $\mathbb{E}[f(w_t)]$. To eliminate the term containing $\mathbb{E}\|\nabla f_k(w_{t_k}^k)\|^2$, we aim to make the upper bound of its coefficient to 0, i.e., $-\frac{K}{2}\left(\sum_{k=1}^{K}(p_t^k)^2\right) + \frac{LK^2E(n_l^i)^2}{2}p_t^k \leq 0$. Therefore, based on Lemma 1, we obtain Eq. (10). Finally, with the simplified right-hand side of Eq. (11), We compute the summation from 1 to T and reorganize the equation to obtain Eq. (9).

We present Corollary 1, which is based on Theorem 1:

Corollary 1. *In accordance with the convergence rate established in Theorem 1, when n_l is a constant value and satisfies the conditions in Eq. (10), i.e., $n_l = \frac{1}{\sqrt{TKE}}$, then, we derive for a sufficiently large T:*

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(w_t)\|^2 \leq \mathcal{O}\left(\frac{(f(w_0)-f(w^*))}{\sqrt{TKE}}\right) \\ + \mathcal{O}\left(\frac{EK^2\beta^2\sigma^2}{T}\right) + \mathcal{O}\left(\frac{E\sigma^2}{T}\right) \\ + \mathcal{O}\left(\frac{\sigma_l^2}{K\sqrt{TKE}}\right)\end{aligned} \tag{12}$$

where $\sigma^2 = (\alpha+\mu)\sigma_l^2 + (\alpha+\mu)\sigma_g^2 + G^2$.

The proof of this corollary is excluded due to space limitations. Our theoretical analysis allows us to highlight several critical observations regarding the key factors influencing convergence.

The staleness limit β. The effect of the staleness limit on convergence diminishes at a rate of $1/T$, as indicated by the second term in Eq. (12). A large staleness limit is not preferable due to its contribution to an increase in the second term. Nevertheless, we can modify the buffer size to mitigate its influence on the convergence rate.

The buffer size K. The first term of Eq. (12) indicates that as the buffer size *K* increases, there is a rapid decrease in the loss towards the optimal value. However, the impact of the variance σ^2 can be increased, thus enhancing the gradient drift during training. Furthermore, a large staleness limit β results in server aggregation incorporating outdated updates while waiting for more devices, which negatively affect convergence. As a result, we anticipate $K \in (1, M]$, where *M* should not be excessively large.

In contrast to related approaches such as *FedBuff* [5], *SEAFL* represents a more generalized framework of semi-

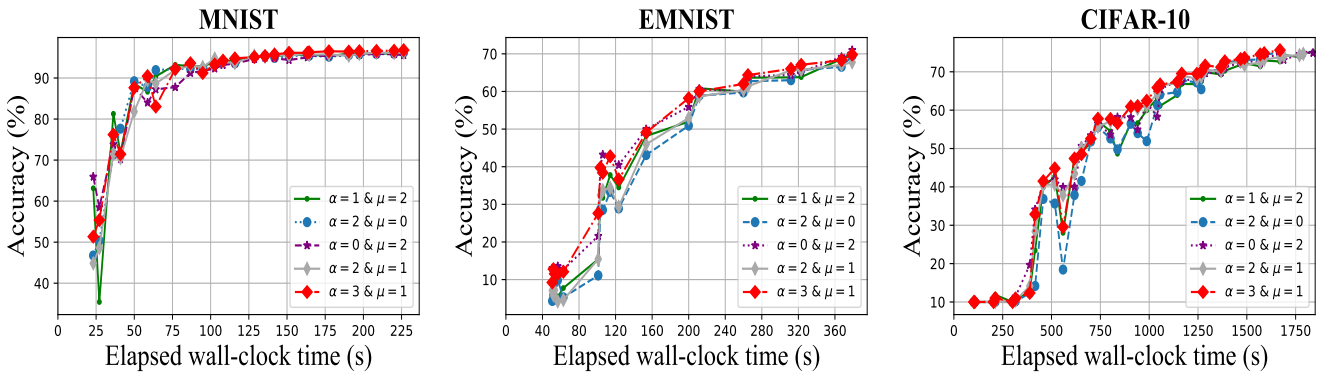


Fig. 4: Elapsed wall-clock time required to reach target accuracy for different combinations of α and μ.

asynchronous aggregation techniques with buffered updates. *SEAFL*'s convergence rate naturally degenerates into Theorem 1 in [5] by setting consistent weights $p_t^k = \frac{1}{K}$ for gradients in the server aggregation process.

VI. Performance Evaluation

In this section, we present the experimental comparison of *SEAFL* with three state-of-the-art approaches across three commonly used datasets to validate its effectiveness.

A. Experimental Setup

Datasets and Models. The experiments are conducted over different image classification tasks using three popular benchmark datasets i.e., EMNIST [33], CIFAR-10 [34], and CINIC-10 [35]. The data distributed across each device exhibits a non-IID pattern, generated from a Dirichlet distribution with a concentration parameter of 5 for all datasets. In our experiments, we consider LeNet-5 [36] model for EMNIST, ResNet-18 [37] for CIFAR-10, and VGG-16 [38] for CINIC-10.

Baselines Methods. To demonstrate the performance of *SEAFL*, we compare it against three state-of-the-art (SOTA) FL approaches: (i) *FedAvg* [2], which is a synchronous federated learning approach; (ii) *FedBuff* [5], a semi-asynchronous method for federated learning; and (iii) *FedAsync* [11], the standard asynchronous federated learning approach.

Implementation. We have implemented *SEAFL* using the open-source research framework PLATO [32]. This framework supports the emulation of asynchronous FL training and provides the ability to measure the elapsed wall-clock time during an FL training session. We assume 100 devices are available for all experiments and up to 20% of them are sampled randomly in each communication round for the synchronous mode. In all experiments, we set $\vartheta = 0.8$, and epochs $E = 5$. All experiments are performed on a server equipped with an NVIDIA GeForce RTX 3080 Ti GPU. We used Pareto distribution to simulate heavy-tailed client speed.

Evaluation Metrics. Existing works typically evaluate performance based solely on metrics such as the number of gradients, updates, or communication rounds required to reach a target accuracy. However, these metrics may not accurately reflect real-world training time due to the nature of asynchronous FL, where the communication round index can advance whenever a single device reports to the server. As a result, these metrics often fail to reflect the actual wall-clock time needed to achieve target accuracy. Hence, we consider the elapsed wall-clock time required to reach a target accuracy on the test set, rather than the number of rounds.

B. Results and Analysis

Effect of hyperparameters. Initially, we run a large number of experiments to figure out the optimal combination of hyperparameters α and μ, crucial for our adaptive weight aggregation mechanism. We explore values ranging from 0 to 10 for both α and μ. Fig. 4 illustrates the comparison of various representative pairs of values for α and μ. In general, the combination of $\alpha = 3$ and $\mu = 1$ provided a modest performance improvement compared to other value pairs.

***SEAFL* vs. baselines.** We compare the performance of *SEAFL* (without partial training) with baseline methods. In Fig. 5, we present the wall-clock training time required to achieve a target accuracy for each dataset. It is evident that *FedAsync* completely failed to converge in all cases. This is attributed to its aggressive strategy of immediately aggregating the fastest device update upon arrival, as well as the design of its own aggregation algorithm. *SEAFL* consistently outperformed the synchronous FL baseline, *FedAvg*, in terms of the wall-clock training times needed to reach target accuracy for all datasets.

As *FedBuff* does not impose any restriction on staleness, it effectively operates with an ∞ staleness limit. Therefore, we conducted experiments with all three scenarios: *SEAFL* with a staleness limit of 10, *SEAFL* with an ∞ staleness limit, and *FedBuff*. From Fig. 5, we can see that, *SEAFL* performed very similarly while having an ∞ staleness limit. This is attributed to the fact that the majority of the aggregated device updates are not too stale. However, both approaches suffer from accuracy degradation for a few rounds when stale devices eventually arrive. In contrast, *SEAFL* exhibits superior performance across all datasets with a staleness limit of 10, especially with EMNIST. These experiments support the intuition that imposing a reasonable staleness limit provides

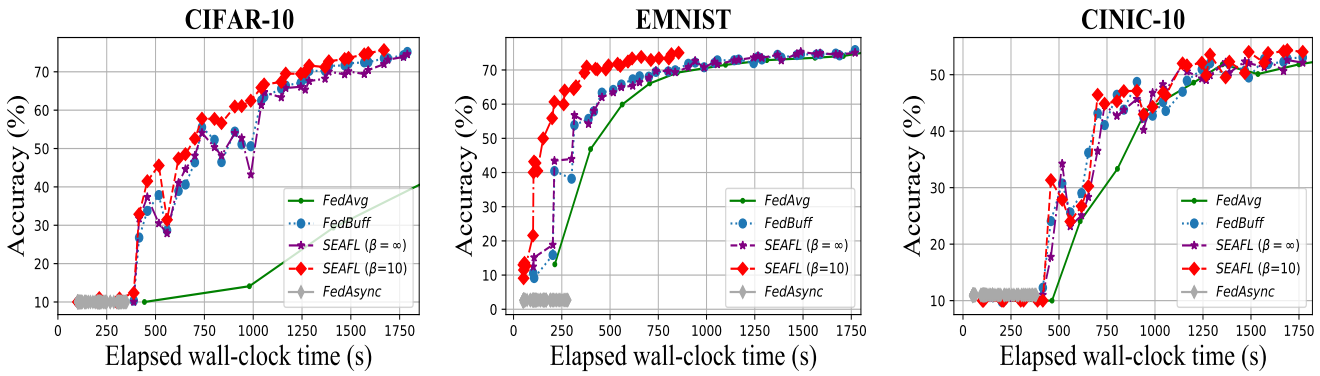
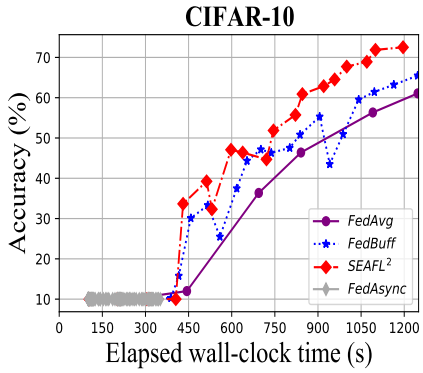
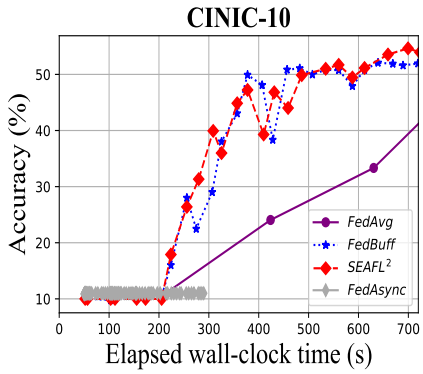


Fig. 5: Elapsed wall-clock time required to reach target accuracy for *SEAFL* (without partial training), *FedBuff*, *FedAsync*, and *FedAvg*. *SEAFL* converges faster to reach target accuracy and consistently outperforms other baselines.



(a) *SEAFL*^{2} with staleness limit of 3 vs. baselines



(b) *SEAFL*^{2} with staleness limit of 12 vs. baselines

Fig. 6: Performance comparison of *SEAFL*^{2} and baseline methods.

benefits in terms of wall-clock time, even while accommodating slower devices.

***SEAFL*^{2} vs. baselines.** In this series of experiments, we activate the partial training mechanism in *SEAFL* and measure the resulting training durations. Our findings are depicted in Fig. 6. With a low staleness limit of 3, *SEAFL*^{2} enabled the server to promptly notify devices upon reaching the staleness limit while working with the CIFAR-10 dataset. In this scenario, *SEAFL*^{2} has clearly shown its advantages: it achieved 50% accuracy in only 745 seconds, and reached 70% accuracy in 1105 seconds. In contrast, its closest rival, *FedBuff*, required 905 seconds to achieve 50% and 1341 seconds to achieve 70%. This indicates that *SEAFL*^{2} achieved a performance advantage of up to 22% compared to *FedBuff*.

We experimented with a higher staleness limit of 12 using the CINIC-10 dataset to observe the performance of *SEAFL*^{2}. Fig. 6b depicts that it initially progresses similarly to *FedBuff*, and only shows a slight advantage as convergence approaches completion. With the CINIC-10 dataset, where each device utilized only 3% of the total samples for training compared to 10% with CIFAR-10, this observation suggests that a staleness limit of ∞ may not be detrimental at all if local training finishes quickly and there is a high turnover rate to new devices. Consequently, in such scenarios, the performance benefit of *SEAFL*^{2} over *FedBuff* may decrease, as the impacts of partial training on stale devices become less impactful. However, in contrast to *FedAsync* and *FedAvg*, it is evident that *SEAFL*^{2} exhibits a significant performance advantage with both datasets.

Finally, it is important to highlight that, unlike the fully asynchronous operation with an ∞ staleness limit employed in *FedBuff*, a finite staleness limit offers a well-established and appealing theoretical property: guaranteed convergence during training [39]. Despite the consistent convergence of *FedBuff* in our experiments, having a theoretical guarantee provides further assurance.

VII. Conclusion

In this work, we designed a novel staleness-aware semi-asynchronous FL framework, i.e., *SEAFL*, to address the straggler and excessive computation overhead issues of synchronous and asynchronous FL. *SEAFL* adaptively assigns weights to local updates during aggregation, considering both the staleness of received model updates and their similarity to the current global model. Additionally, we proposed *SEAFL*^{2} to further enhance the training efficiency, facilitating partial training on slower devices. *SEAFL*^{2} allows straggler devices to contribute to global aggregation and reduces the overall

waiting time. Experimental results demonstrated significant advantages of *SEAFL* over state-of-the-art synchronous and asynchronous counterparts in terms of the convergence time required to achieve target accuracy. Moreover, we provided the theoretical convergence analysis of our proposed approach. In the future, we plan to extend *SEAFL* by incorporating adaptive partial training to dynamically adjust local model sizes using sub-model extraction techniques tailored to devices' real-time resource capabilities.

REFERENCES

[1] EU. 2018. European Union's General Data Protection Regulation (GDPR). European Union. Accessed 2024-04. [Online]. Available: https://eugdpr.org/

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[3] M. Liu, S. Ho, M. Wang, L. Gao, Y. Jin, and H. Zhang, "Federated learning meets natural language processing: A survey," *arXiv preprint arXiv:2107.12603*, 2021.

[4] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang, "Fedvision: An online visual object detection platform powered by federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 08, 2020, pp. 13 172–13 179.

[5] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba, "Federated learning with buffered asynchronous aggregation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 3581–3607.

[6] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, "Clusterfl: a similarity-aware federated learning system for human activity recognition," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 54–66.

[7] Z. Jiang, W. Wang, B. Li, and B. Li, "Pisces: Efficient federated learning via guided asynchronous training," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 370–385.

[8] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2021, pp. 19–35.

[9] C. Yang, Q. Wang, M. Xu, Z. Chen, K. Bian, Y. Liu, and X. Liu, "Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data," in *Proceedings of the Web Conference 2021*, 2021, pp. 935–946.

[10] W. Wu, L. He, W. Lin, R. Mao, C. Maple, and S. Jarvis, "Safa: A semi-asynchronous protocol for fast federated learning with low overhead," *IEEE Transactions on Computers*, vol. 70, no. 5, pp. 655–668, 2020.

[11] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous federated optimization," *arXiv preprint arXiv:1903.03934*, 2019.

[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, 2023.

[13] C. Zhou, J. Liu, J. Jia, J. Zhou, Y. Zhou, H. Dai, and D. Dou, "Efficient device scheduling with multi-job federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, pp. 9971–9979.

[14] Y. Zhou, X. Pang, Z. Wang, J. Hu, P. Sun, and K. Ren, "Towards efficient asynchronous federated learning in heterogeneous edge environments," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, 2024.

[15] J. Liu, J. Jia, T. Che, C. Huo, J. Ren, Y. Zhou, H. Dai, and D. Dou, "Fedasmu: Efficient asynchronous federated learning with dynamic staleness-aware model update," *arXiv preprint arXiv:2312.05770*, 2023.

[16] Q. Wang, Q. Yang, S. He, Z. Shi, and J. Chen, "Asyncfeded: Asynchronous federated learning with euclidean distance based adaptive weight aggregation," *arXiv preprint arXiv:2205.13797*, 2022.

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

[18] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[19] T. Li, S. Hu, A. Beirami, and V. Smith, "Ditto: Fair and robust federated learning through personalization," in *Proceedings of International Conference on Machine Learning*. PMLR, 2021, pp. 6357–6368.

[20] J. Zhang, Y. Hua, H. Wang, T. Song, Z. Xue, R. Ma, and H. Guan, "Fedala: Adaptive local aggregation for personalized federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 9, 2023, pp. 11 237–11 244.

[21] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 586–19 597, 2020.

[22] M. S. Islam, S. Javaherian, F. Xu, X. Yuan, L. Chen, and N.-F. Tzeng, "Fedclust: Tackling data heterogeneity in federated learning through weight-driven client clustering," in *Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 474–483.

[23] S. Javaherian, S. Panta, S. Williams, M. S. Islam, and L. Chen, "Fedfair^3: Unlocking threefold fairness in federated learning," *in Proceedings of IEEE International Conference on Communications (ICC)*, pp. 1–7, 2024.

[24] C. Li, X. Zeng, M. Zhang, and Z. Cao, "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.

[25] H. Zhang, J. Liu, J. Jia, Y. Zhou, H. Dai, and D. Dou, "Fedduap: Federated learning with dynamic update and adaptive pruning using shared data on the server," *arXiv preprint arXiv:2204.11536*, 2022.

[26] S. Horvath, S. Laskaridis, M. Almeida, I. Leontiadis, S. Venieris, and N. Lane, "Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 876–12 889, 2021.

[27] G. Li, Y. Hu, M. Zhang, J. Liu, Q. Yin, Y. Peng, and D. Dou, "Fedhisyn: A hierarchical synchronous federated learning framework for resource and data heterogeneity," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[28] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin, "Hierarchical federated learning across heterogeneous cellular networks," in *Proceedings of ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8866–8870.

[29] T. Zhang, L. Gao, S. Lee, M. Zhang, and S. Avestimehr, "Timelyfl: Heterogeneity-aware asynchronous federated learning with adaptive partial training," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 5063–5072.

[30] 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.

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

[32] B. Li, N. Su, C. Ying, and F. Wang, "Plato: An open-source research framework for production federated learning," in *Proceedings of the ACM Turing Award Celebration Conference-China 2023*, 2023, pp. 1–2.

[33] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," in *Proceedings of International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.

[34] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[35] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "Cinic-10 is not imagenet or cifar-10," *arXiv preprint arXiv:1810.03505*, 2018.

[36] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[39] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, "More effective distributed ml via a stale synchronous parallel parameter server," *Proceedings of Advances in Neural Information Processing Systems*, vol. 26, 2013.