Timely Asynchronous Hierarchical Federated Learning: Age of Convergence

Purbesh Mitra Sennur Ulukus
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20742
pmitra@umd.edu *ulukus@umd.edu*

***Abstract*—We consider an asynchronous hierarchical federated learning (AHFL) setting with a client-edge-cloud framework. The clients exchange the trained parameters with their corresponding edge servers, which update the locally aggregated model. This model is then transmitted to all the clients in the local cluster. The edge servers communicate to the central cloud server for global model aggregation. The goal of each client is to converge to the global model, while maintaining timeliness of the clients, i.e., having optimum training iteration time. We investigate the convergence criteria for such a system with dense clusters. Our analysis shows that for a system of n clients with fixed average timeliness, the convergence in finite time is probabilistically guaranteed, if the nodes are divided into $O(1)$ number of clusters, that is, if the system is built as a sparse set of edge servers with dense client bases each.**

I. Introduction

Federated learning (FL), since its introduction in [1], has been studied extensively in distributed learning frameworks. Distributed learning (DL), as opposed to centralized learning, is often a useful tool in various applications where factors such as distributed data storage and data privacy are of concern, see e.g., [1]–[4]. The FL model used in these applications consists of a central parameter server (PS) and multiple clients which have the distributed data. The PS has a global model which has to be trained on all the client data. In the FL settings, the PS sends copies of the global model to individual clients, which are then trained locally. After training, the clients send back their individual local models or their gradients, which are securely aggregated at the PS by weighted averaging. In [1], the authors showed that this training method yields convergence to a local optimum.

Due to the distributed nature of FL, its convergence is affected by factors such as data heterogeneity and communication overhead [5], [6]. The convergence result in [1] relies on idealistic assumptions, such as, that the data is distributed in i.i.d. manner across all clients, and that the client devices participate in the federated training with regularity. In reality, the distributed data is often non-i.i.d. and the client devices are often unavailable in training cycles due to inefficient communication channels, energy availability of wireless devices, and the presence of Byzantine stragglers. To circumvent these issues, several solutions have been proposed in the literature, such as, priority-based client selection [7], clustering [8], [9], adaptive quantization [10], multi-task learning [11]–[17], asynchronous FL [18], and hierarchical FL [19].

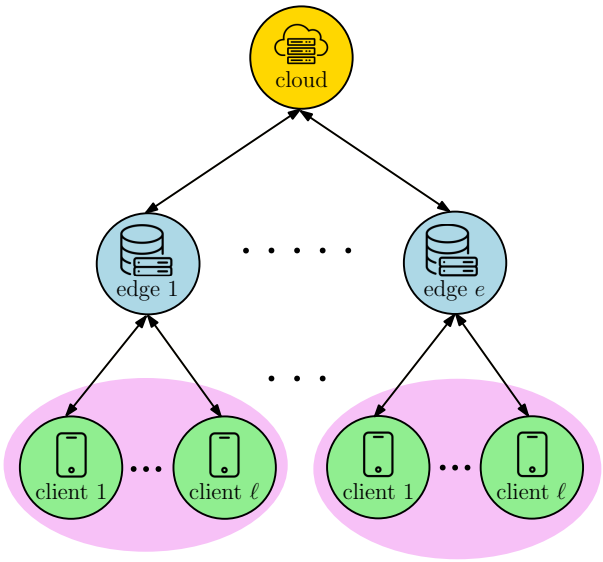


Fig. 1. Hierarchical federated learning (HFL) with client-edge-cloud setting.

In particular, the client-edge-cloud based hierarchical FL (HFL) setting, introduced in [19], is used for circumventing the problem of heterogeneity with provable guarantees [20]. In this HFL setting, there are multiple clusters of clients which are connected to their corresponding edge servers, which themselves are connected to a central cloud server. The edge servers perform local model aggregation from their associated client devices, which leads to a convergence of the local models. For global convergence, the edge servers update the cloud server, typically less frequently than the clients, to get back a globally aggregated model, which is then passed on to the client devices. Several variants of this model, such as, asynchronous hierarchical FL (AHFL) [21], [22], have been proposed in the literature which enable the training process to converge without any synchronicity requirement.

In all of these FL settings, better communication between clients and the PS is necessary. Works in [7], [23]–[39] have proposed various schemes, which yield faster and reliable communication. Another criteria for efficient communication is better timeliness of the system, which makes the overall training process faster. This aspect is investigated in [40], where the authors have shown that selecting any random k out of n clients as in [1] does not necessarily yield a fast convergence in FL. [40] uses the age of information metric

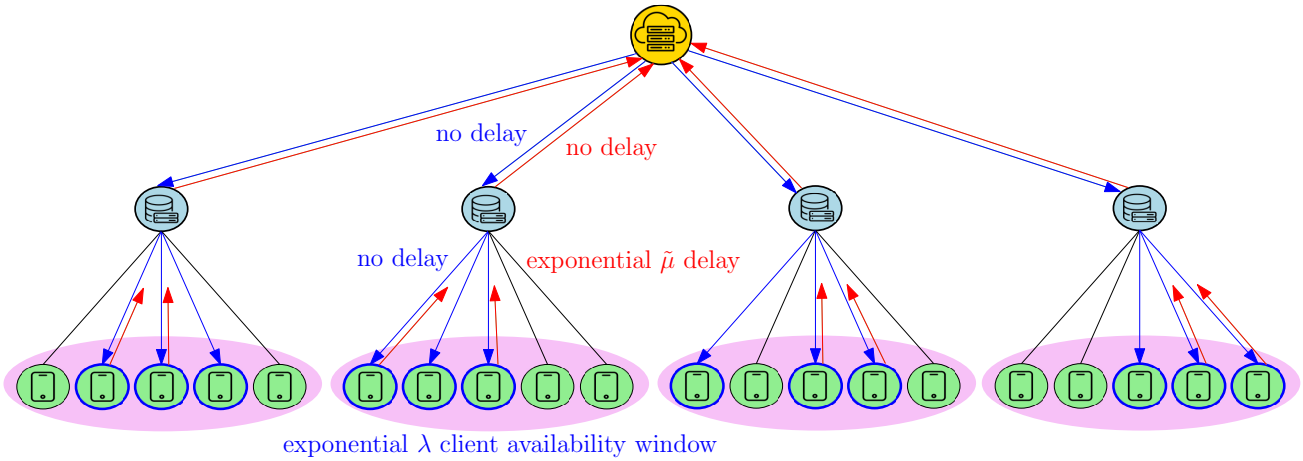


Fig. 2. Timely asynchronous hierarchical federated learning with client-edge-cloud setting. The number of edge servers $e = 4$. Each edge server has $\ell = 5$ clients associated with it. At each training iteration, edge server waits for $m = 3$ clients to be available, and only the first $k = 2$ earliest finishers are aggregated at the edge. The local aggregation is then globally aggregated at the cloud asynchronously.

[41]–[45], a widely used metric in the communications and networking literature, to characterize the timeliness of the individual clients. [40] also proposes a timely FL scheme, where the PS waits for m out of the n clients to be available and then sends the updated model. Then, for model aggregation, only the first k responses out of the m clients are chosen. [40] shows that this scheme results in faster training cycles, and hence faster convergence. This gain in performance is due to the fact that, in the proposed scheme in [40], the initial waiting period ensures that all the clients are available in the training as opposed to the random selection scheme in [1].

Motivated by this approach, in this work, we focus on the timeliness of a more realistic FL setting, namely AHFL. In particular, we consider the client-edge-cloud setting, where the local client-edge training happens in the same way as in [40] and the edge-cloud aggregation takes place asynchronously. The cloud model updates its version number by one whenever it performs aggregation. On the client's side, if a client manages to send its trained model to the edge, and therefore, to the cloud, it catches up with the global version of the cloud; else, its version does not change. A client with its model being successfully aggregated in a training cycle will have the most recent version number, while a client trained on an older version incurs some *staleness* in the system, which is denoted as the difference of versions between the cloud and the client. The analysis in [18] shows that the global convergence of an asynchronous FL system is dominated by this staleness factor, i.e., the smaller is the staleness, the faster the system will converge. Since we are not considering a traditional FL setting, rather an AHFL system, we have to incorporate both the timeliness and the bounded staleness in the design of the system for better performance.

With this backdrop, in this work, we investigate the convergence criteria of a timely AHFL system consisting of dense clients. We show that there can be a probabilistically guaranteed global convergence if the number of clusters of the hierarchical system is $O(1)$, while maintaining timeliness. That is, a good operating setting for AHFL is to have sparse edge devices with dense clients.

II. System Model

In a FL setting, the parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ of a model are trained in n different client devices with distributed training data. The goal of the system is to minimize a loss function,

$$L(\boldsymbol{\theta}; D) = \frac{1}{n} \sum_{i=1}^{n} L(\boldsymbol{\theta}; D_i), \tag{1}$$

where D_i is the training data available to the ith client and $D = \bigcup_{i=1}^{n} D_i$ is the total data set. In the beginning of the tth cycle, the PS sends a global model $\boldsymbol{\theta}_t$ to the clients. At each cycle, the clients perform $\tau \in \mathbb{N}$ step stochastic gradient descent (SGD) on their local objective function as follows

$$\boldsymbol{\theta}_{j+1}^i = \boldsymbol{\theta}_j^i - \eta_j^i \nabla L(\boldsymbol{\theta}_j^i; D_i), \quad j \in [\tau], i \in [n], \tag{2}$$

where η_ℓ^i is the learning rate corresponding to the jth local step of the ith client and $\boldsymbol{\theta}_1^i = \boldsymbol{\theta}_t$. After training completion, each client sends the PS its model, which are aggregated as

$$\boldsymbol{\theta}_{t+1} = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \boldsymbol{\theta}_{\tau+1}^i. \tag{3}$$

This updated model is then used for the next training cycle and the process continues until the learning model converges.

In a HFL, the most commonly used model is the client-edge-cloud model, introduced in [19]. In this model, shown in Fig. 1, there are total n clients distributed under e edge servers. For simplicity, we assume that the clusters are symmetric and there are ℓ clients corresponding to each edge server, i.e., $n = e\ \ell$. For this work, we consider that the edge-client network is performing local update in a timely manner as described in [40]. This model is shown in Fig. 2. The clients are assumed to be available at each training cycle after an exponentially distributed time window with parameter λ. The first m available clients among ℓ clients receive the model from the edge servers without any downlink communication delay.

Algorithm 1 Timely AHFL algorithm

Initialize $\boldsymbol{\theta}_0$ and send to all the edges and the clients.
for $t \in [T]$ **do**
  procedure CLOUDUPDATE
    Receive $\boldsymbol{\theta}^{t'}$ from the edge server s.
    Perform global aggregation as

$$\boldsymbol{\theta}^{t} = (1 - \sigma(t - t^{t'}))\boldsymbol{\theta}^{t-1} + \sigma(t - t^{t'})\boldsymbol{\theta}^{t'}.$$

    Send $\boldsymbol{\theta}^{t}$ to the edge server s.
  for $s \in [e]$ **do**
    procedure EDGEUPDATE(edge s)
      Receive $\boldsymbol{\theta}_t$ from the cloud server.
      Start the client training cycle.
      Wait till m among the ℓ clients to become available.
      Send $\boldsymbol{\theta}_t$ to the clients.
      Wait until the first k clients (set $\mathcal{K}$) respond.
      Aggregate as $\boldsymbol{\theta}^{t'} = \sum_{i \in \mathcal{K}} \frac{|\mathcal{D}_i|}{|\mathcal{D}_\mathcal{K}|}\boldsymbol{\theta}^i_{t+1}$.
      Send $\boldsymbol{\theta}^{t'}$ to the cloud server.
    for $i \in [\ell]$ **do**
      procedure CLIENTTRAINING(client i)
        Become available after exponential (λ) time.
        Receive $\boldsymbol{\theta}_t$ from the edge.
        $\boldsymbol{\theta}^i_1 \leftarrow \boldsymbol{\theta}_t$.
        for $j \in [\tilde{t}]$ **do**
          $\boldsymbol{\theta}^i_{j+1} = \boldsymbol{\theta}^i_j - \eta^i_j \nabla \mathcal{L}(\boldsymbol{\theta}^i_j; D_i)$.
        Send $\boldsymbol{\theta}^i_{\tilde{t}+1}$ to the edge in exponential (μ) time.

For any client, the training takes c units of time, which is a deterministic quantity. The uplink communication to the edge is modeled as an exponential random variable with parameter μ. For aggregation, only the first k responding clients are considered. Analysis of [40] shows that choosing the (k, m) pair with a linear dependency on n, i.e., $k = \alpha m$ and $m = \beta\ell$ are sufficient for maintaining an average timeliness independent of ℓ, where $\alpha, \beta \in (0, 1)$ are constants, which can be chosen optimally.

In the timely AHFL system, the edge-cloud network, on the other hand, is doing global updates asynchronously with negligible delay due to their high data rate. To penalize the staleness, instead of doing SGD on $L(\boldsymbol{\theta}; D_i)$, each client performs SGD on the modified loss function

$$\mathcal{L}(\boldsymbol{\theta}; D_i) = L(\boldsymbol{\theta}; D_i) + \frac{\rho}{2}\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|^2 \tag{4}$$

and sends it to the edge, where the regularizing parameter ρ controls the allowed deviations of the local models. Each time an edge server generates a local update, it immediately sends it to the cloud, which updates the global model by taking a weighted sum of the arrived local model $\boldsymbol{\theta}^{t'}$ and the existing global model $\boldsymbol{\theta}^t$,

$$\boldsymbol{\theta}^{t+1} = (1 - \sigma(t - t^{t'}))\boldsymbol{\theta}^t + \sigma(t - t^{t'})\boldsymbol{\theta}^{t'}, \tag{5}$$

where $t^{t'}$ is the last time-stamp of the trained model from the particular edge and $\sigma(\cdot) \in (0, 1)$ is a decreasing function of the staleness, chosen optimally. This new model is then sent back to the edge servers, which are then used for client training in the subsequent cycle. The AHFL procedure is shown in Algorithm 1.

We denote the training time duration of the tth cycle as Y_t and the time duration of the trained model at the ith node to be aggregated as Y^i_j, where $j \in \mathbb{N}$ denotes the index of successful participation of node i. From the analysis of [40], we obtain the expression of an average training cycle as

$$\mathbb{E}[Y_t] = \mathbb{E}[Z_{m:\ell}] + c + \mathbb{E}[X_{k:m}], \tag{6}$$

where $Z_{m:\ell}$ denotes the time it takes for m out of ℓ clients to be available and $X_{k:m}$ denotes the time for the first k out of m clients to send their model to the PS. Using the order statistics formulation in [46], we obtain

$$\mathbb{E}[Z_{m:\ell}] = \frac{1}{\lambda}(H_\ell - H_{\ell-m}), \tag{7}$$

where $H_\ell = \sum_{j=1}^{\ell} \frac{1}{j}$. Similarly, we can also write

$$\mathbb{E}[X_{k:m}] = \frac{1}{\mu}(H_m - H_{m-k}). \tag{8}$$

Due to the symmetry of the network, the probability of a particular client being available for training and successfully sending its trained model to the PS is $\frac{k}{\ell}$. Therefore, we obtain the following expectation

$$\mathbb{E}[Y^i_j] = \frac{\ell}{k}\mathbb{E}[Y_t]. \tag{9}$$

Since the edge-cloud communication is asynchronous and instantaneous, the expectation in (9) also implies the expected time of the ith client's version number to be updated.

III. CONVERGENCE GUARANTEE OF AHFL

In this section, we formulate the expression for the version difference between the cloud and a client, and derive an upper bound. We denote $N(\tau)$ and $N_i(\tau)$ as the counting processes of the cloud version and the ith node version, respectively, as shown in Fig. 3. The process $N(\tau)$ is increasing at every $T_j, j \in \mathbb{N}$, with Y_j being the inter-update times, whereas the process $N_i(\tau)$ gets updated at every $T^i_j, j \in \mathbb{N}$, with mean inter-arrival time $\mathbb{E}[Y^i_j]$. Using a similar formulation as [47], we define the version difference for the ith node as

$$X_i(\tau) = N(\tau) - N_i(\tau). \tag{10}$$

Since $N(\tau)$ always leads ahead of $N_i(\tau)$, $X_i(\tau)$ is always non-negative. From (10), we can write the staleness of the jth successful training of the ith node as

$$S_i[j] = X_i(T^i_{j-1}), \quad \forall j \in \mathbb{N}. \tag{11}$$

Now, in the convergence analysis of [18], the authors have shown that for asynchronous federated optimization with T training cycles and the condition that $S_i[j] \le M, \forall j \in [T]$,

$$\min_{t=0,\dots,T-1} \mathbb{E}\left[\|\nabla L(\boldsymbol{\theta}_t; D)\|^2\right] \le O(M^2). \tag{12}$$

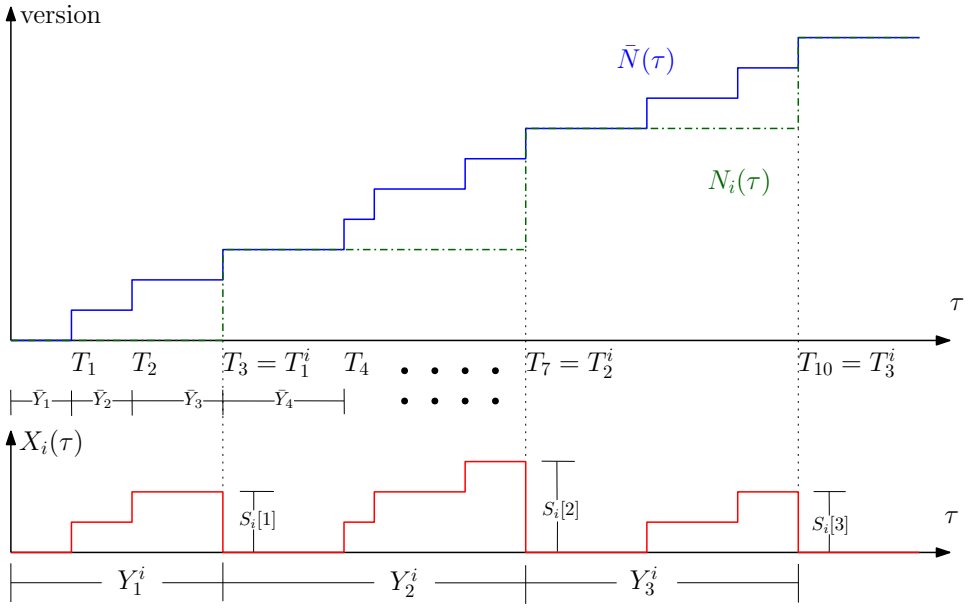


Fig. 3. Sample paths of version evolution in the cloud server and the ith node and the corresponding staleness.

The same criterion also holds true for our AHFL system. Therefore, to have faster convergence, it is necessary to bound the staleness tightly. Since we have defined $\{S_i[j]\}_{j \in \mathbb{N}}$ as a random process in (11), we use the Markov inequality to obtain the following bound

$$\mathsf{P}\big(S_i[j] \quad M\big) \quad 1 \quad \frac{\mathsf{E}[S_i[j]]}{M}. \tag{13}$$

From (13), we conclude that minimizing $\mathsf{E}[S_i[j]]$, would result in tighter upper bound. In the following lemma, we derive the expression of the expectation of staleness.

Lemma 1 *The expected staleness of an AHFL system in steady state is given by*

$$\lim_{j \to 1} \mathsf{E}[S_i[j]] = \frac{\mathsf{E}[Y_j^i]}{\mathsf{E}[Y]} \quad 1. \tag{14}$$

Proof: We can write the staleness as difference of the counting processes as following

$$S_i[j] = N(T_j^i \;) \quad N_i(T_j^i \;) \tag{15}$$

Since the version of i does not change for Y_j^i duration, we have $N(T_j^i \;) = N(T_{j\ 1}^i + Y_j^i) \quad 1$ and $N_i(T_j^i \;) = N(T_{j\ 1}^i)$. Substituting these values in (15), we obtain

$$S_i[j] = N(T_{j\ 1}^i + Y_j^i) \quad N(T_{j\ 1}^i) \quad 1. \tag{16}$$

Asymptotically, for large training time, we have $T_{j\ 1}^i \; ! \; 1$ as $j \; ! \; 1$. Now, since Y_j^i is a random variable, we use the result from [48, Lem. 2] (see also [49]) and apply the modified Blackwell renewal theorem to evaluate the expectation of $S_i[j]$ from (16) as follows

$$\lim_{j \to 1} S_i[j] = \lim_{T_{j\ 1}^i \to 1} N(T_{j\ 1}^i + Y_j^i) \quad N(T_{j\ 1}^i) \quad 1 \tag{17}$$

$$= \frac{\lim_{j \to 1} \mathsf{E}[Y_j^i]}{\mathsf{E}[Y]} \quad 1 = \frac{\mathsf{E}[Y_j^i]}{\mathsf{E}[Y]} \quad 1. \tag{18}$$

This concludes the proof of the lemma.

We have the expression of $\mathsf{E}[Y_j^i]$ from Section II. We evaluate the expectation of $\mathsf{E}[Y]$ in the following lemma.

Lemma 2 *For a symmetric AHFL system with e edge servers, the mean inter-update time of the cloud server is given by*

$$\frac{1}{\mathsf{E}[Y]} = \frac{e}{\mathsf{E}[Y_t]}. \tag{19}$$

Proof: We can write the renewal process $N(\tau)$ as a superposition of multiple renewal processes as follows

$$N(\tau) = \sum_{j=1}^{e} N_j(\tau), \tag{20}$$

where $N_j(\tau)$ is a renewal process which indicates the version number of the jth edge server s_j, which can be expressed as

$$N_j(\tau) = \max\{N_i(\tau) : i \in s_j\}. \tag{21}$$

The cluster version updates after completion of a training cycle, i.e., with inter-update time Y_t. For the steady state, i.e., for $\tau \; ! \; 1$, we can write the following relation

$$\lim_{\tau \to 1} \frac{N(\tau)}{\tau} = \sum_{j=1}^{e} \lim_{\tau \to 1} \frac{N_j(\tau)}{\tau}. \tag{22}$$

Using the renewal theorem, we obtain $\lim_{\tau \to 1} \frac{N(\tau)}{\tau} = \frac{1}{\mathbb{E}[Y]}$ and $\lim_{\tau \to 1} \frac{N_j(\tau)}{\tau} = \frac{1}{\mathbb{E}[Y_t]}$. Substituting these in (22) yields

$$\frac{1}{\mathsf{E}[Y]} = \sum_{j=1}^{e} \frac{1}{\mathsf{E}[Y_t]} = \frac{e}{\mathsf{E}[Y_t]}. \tag{23}$$

This concludes the proof of the lemma.

Now, using Lemmas 1 and 2, we find the criteria for a probabilistic upper bound of the staleness of a node being finite in the following theorem.

Theorem 1 *In a timely AHFL system with dense clients setting, if the number of clusters* $e = O(1)$*, then for any* $0 < \varepsilon < 1$*, there exists a constant* M*, such that the staleness of a node is bounded by* M *with probability* $\mathsf{P}(S_j^i \quad M) \quad 1 \quad \varepsilon$.

Proof: Using (9) and substituting the result of Lemma 1 into Lemma 2, we obtain the expression for the expected staleness in steady state as

$$\lim_{j \,!\, 1} \mathsf{E}[S_i[j]] = \frac{\ell}{k}\mathsf{E}[Y_t] \quad \frac{e}{\mathsf{E}[Y_t]} \quad 1 \tag{24}$$

$$= \frac{e \quad \ell}{k} \quad 1 = \frac{n}{k} \quad 1. \tag{25}$$

From the timeliness condition of [40], $k = \alpha\beta\ell$. Substituting this in (24), we obtain

$$\lim_{j \,!\, 1} \mathsf{E}[S_i[j]] = \frac{n}{\alpha\beta\ell} \quad 1 = \frac{e}{\alpha\beta} \quad 1. \tag{26}$$

Now, using (26) in (13), we obtain the following probabilistic bound

$$\mathsf{P}(S_i[j] \quad M) \quad 1 \quad \frac{1}{M}\left(\frac{e}{\alpha\beta} \quad 1\right). \tag{27}$$

Now, for a dense client setting, i.e., $n \;!\; 1$ and $e = O(1)$, the probabilistic bound in (27) can be written as

$$\lim_{n \,!\, 1} \mathsf{P}(S_i[j] \quad M) \quad 1 \quad \lim_{n \,!\, 1} \frac{O(1)}{M}. \tag{28}$$

Clearly, there exists a finite M for which the bound in (28) is satisfied asymptotically. Choosing any $M \quad \frac{1}{\varepsilon}\left(\frac{e}{\alpha\beta} \quad 1\right)$ yields the statement of the theorem.

IV. Numerical Results

In this section, we simulate the timely AHFL model for a simple linear regression task. In our experiment, we synthetically generate a dataset, in the same way as [40], [50], [51]; divide the dataset into equal parts, give to the clients and perform Algorithm 1 to numerically calculate the regression loss and the average staleness.

We generate the dataset $D = \{(\boldsymbol{x}_j, y_j)\}$, where $\boldsymbol{x}_j \;2\; \mathsf{R}^d$. In our case, $d = 100$ and the data points are from the Gaussian mixture distribution $\boldsymbol{x}_j \quad \frac{1}{2}\mathcal{N}\left(\frac{1.5}{d}\boldsymbol{w}\ , \boldsymbol{I}_d\right) + \frac{1}{2}\mathcal{N}\left(\quad\frac{1.5}{d}\boldsymbol{w}\ , \boldsymbol{I}_d\right)$, where each component of $\boldsymbol{w} \;2\; \mathsf{R}^d$ are chosen uniformly from the interval $[0, 1]$. The corresponding output is $y_j = \boldsymbol{x}_j^T\boldsymbol{w}$. Therefore, the overall loss function to minimize is

$$L(\boldsymbol{\theta}; D) = \frac{1}{|D|}\sum_{j2[|D|]}\left(\boldsymbol{x}_j^T\boldsymbol{\theta} \quad y_j\right)^2 \tag{29}$$

$$= \frac{1}{|D|}\|X\boldsymbol{\theta} \quad \boldsymbol{y}\|_2^2, \tag{30}$$

where $X = \left[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|D|}\right]^T$ and $\boldsymbol{y} = \left[y_1, \ldots, y_{|D|}\right]^T$ are data and corresponding outputs and $\| \quad \|_2$ represents the l_2 norm. The loss function achieves its minimum when $\boldsymbol{\theta} = \boldsymbol{w}$, since all the error terms in the summation of (29) will be 0. The convergence analysis of [18] shows that in order to achieve provable convergence via asynchronous training, this loss function L must satisfy the criteria of being L-smooth and μ-weakly convex. From the formulation in (29), since L is differentiable, we can write the gradient as

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}; D) = \frac{2}{|D|}X^T(X\boldsymbol{\theta} \quad \boldsymbol{y}). \tag{31}$$

Now, for $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, we have the following equation

$$\nabla L(\boldsymbol{\theta}_1; D) \quad \nabla L(\boldsymbol{\theta}_2; D) = \frac{2X^TX}{|D|}(\boldsymbol{\theta}_1 \quad \boldsymbol{\theta}_2). \tag{32}$$

Now, choosing $L = \frac{2}{|D|}\|X^TX\|_2$, we obtain from (32):

$$\|\nabla L(\boldsymbol{\theta}_1; D) \quad \nabla L(\boldsymbol{\theta}_2; D)\|_2 = L\|\boldsymbol{\theta}_1 \quad \boldsymbol{\theta}_2\|_2, \tag{33}$$

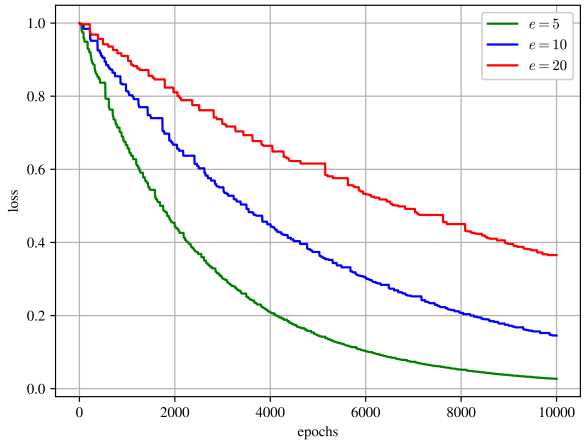
which implies that L is a L-smooth function. Also, since (29) is a l_2 norm, which is a convex function, it is also μ-weakly convex. Hence, the regression loss function satisfies both the provable convergence criteria.

In our experiment, $|D| = 10,000$. This data is equally distributed among n clients. Therefore, for the ith client, $|D_i| = |D|/n$. Each client tries to minimize the modified loss function, defined in (4), when they are included in a training cycle. For the simulations, we consider $\rho = 0.01$. Also, since the goal of our work is to show the effect of edge servers on convergence, we take both α and β as 0.5, i.e., in each training cycle corresponding to a particular edge server, the edge server waits for half of the clients associated with it to become available with exponential time parameter $\lambda = 1$ and half of those clients get to finish their $\bar{\tau}$ step gradient descent in time $c = 1$ and update the edge server with exponential delay parameter $\mu = 1$. In simulations, $\bar{\tau} = 10$. After aggregating the model at the edge, the cloud server is then updated as (5). We use the following decreasing function

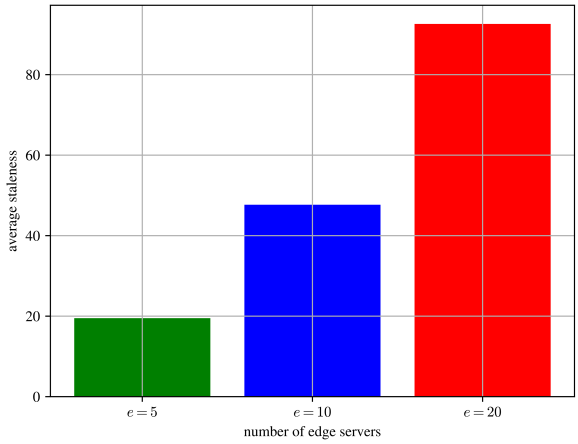
$$\sigma(t \quad t^0) = \frac{1}{(t \quad t^0)^{0.1}}, \quad t > t^0 \tag{34}$$

for taking staleness into account. We consider three different orders for the number of edge servers: $e \quad O(1)$, $O(\sqrt{n})$ and $O(n)$. The training process continues for $T = 10,000$ epochs, and the obtained results are shown in Fig. 4.

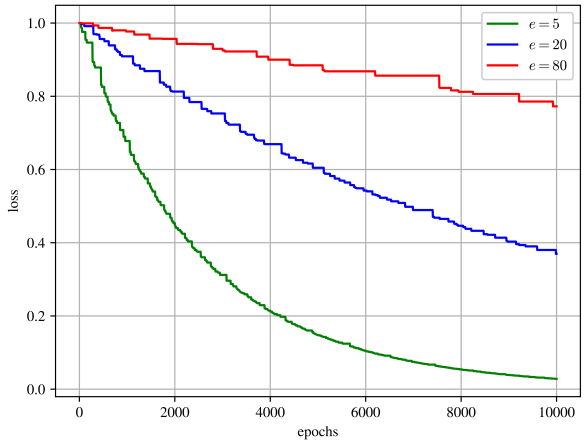
First, we consider $n = 100$ clients. Each client has 100 data points drawn from the Gaussian mixture distribution, and they train in the AHFL setting. The number of edge servers is considered as $e = 5$, $e = \sqrt{n} = 10$ and $e = 0.2 \quad n = 20$. From Fig. 4(a), we observe that for $e = 5$, the regression loss decreases rapidly, i.e., the client model converges with the fastest rate, whereas the convergence rate slows down as e increases. This is due to the increasing staleness of the system as e increases. For α and β as 0.5, the expected staleness, from (26), is $4e \quad 1$, i.e., 19, 39 and 79, respectively. This analytical result is close to the numerically evaluated average



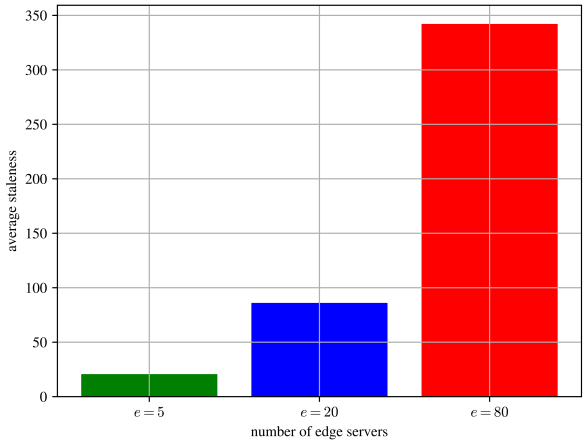
(a) Loss vs. epochs for n = 100.



(b) Staleness for n = 100.



(c) Loss vs. epochs for n = 400.



(d) Staleness for n = 400.

Fig. 4. Loss profiles and staleness comparison for $O(1), O(\sqrt{n})$ and $O(n)$ number of edge servers.

staleness, as shown in Fig. 4(b). The effect of this increasing staleness is also observed in Fig. 4(a) as the loss profile has more *staircase* decrements due to the individual nodes in the clusters being trained less frequently.

Similar trends are observed for n = 400. In this case, each client has 25 data points for training. The number of edge servers is 5, 20 and 80, respectively. Fig. 4(c) shows that the e = 5 is the fastest, as before, and the rate of convergence decreases as e increases. We also observe that for e = 5 and 20, the 100 client setting converges faster than 400 client setting. This might be due to the fact that with higher number of clients, the data becomes more distributed, which results in slower convergence rate. Finally, the staleness profile, as shown in Fig. 4(d), indicates that with higher number of edge servers, the models become more stale. The average values also match with the analytically calculated mean staleness, which are 19, 79 and 319, respectively.

V. Conclusion

We analyzed a timely updating scheme in an AHFL model. In this system model, the clients are divided into multiple clusters which are connected to the edge servers which perform local aggregation in a timely manner. In each training cycle, the edge servers send their models to a selected number of clients who become available first. The clients train the model with their locally available data. After completion of the training, the clients send back their trained model to the edge servers. Only a limited number of models, that arrive first, are used for the edge-aggregation. When an edge is updated, it sends its locally aggregated model to a cloud server, which performs asynchronous global aggregation and gives back the new global model to the edge servers with negligible time delay. This model is then used for the next training cycle of the clients. Since each client's trained model is not necessarily

aggregated in each training cycle, this model incurs some staleness into the system. With bounded staleness, this timely AHFL training model can converge. We showed that if the number of edge servers is $e = O(1)$, then for a system with dense clients, there is a probabilistic guarantee of convergence of the distributed learning model in a finite time.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, April 2017.

[2] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li. PMF: A privacy-preserving human mobility prediction framework via federated learning. *ACM Interact. Mob. Wearable Ubiq. Tech.*, 4(1):10–21, March 2020.

[3] G. Damaskinos, R. Guerraoui, A. Kermarrec, V. Nitu, R. Patra, and F. Taiani. Fleet: Online federated learning via staleness awareness and performance prediction. *ACM Trans. Intell. Syst. Tech.*, 13(5):1–30, September 2022.

[4] L. Li, Y. Fan, M. Tse, and K. Lin. A review of applications in federated learning. *Comp. & Indust. Engineering*, 149:1–15, November 2020.

[5] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, June 2021.

[6] D. Shome, O. Waqar, and W. U. Khan. Federated learning and next generation wireless communications: A survey on bidirectional relationship. *Trans. on Emer. Telecomm. Tech.*, 33(7):e4458, October 2022.

[7] J. Perazzone, S. Wang, M. Ji, and K. S. Chan. Communication-efficient device scheduling for federated learning using stochastic optimization. In *IEEE Infocom*, May 2022.

[8] C. Briggs, Z. Fan, and P. Andras. Federated learning with hierarchical clustering of local updates to improve training on non-iid data. In *IEEE IJCNN*, July 2020.

[9] J. Wang, S. Wang, R. Chen, and M. Ji. Demystifying why local aggregation helps: Convergence analysis of hierarchical SGD. In *AAAI Conf. on Artificial Intelligence*, March 2022.

[10] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, and W. Ding. Communication-efficient federated learning with adaptive quantization. *ACM Trans. Intell. Syst. Tech.*, 13(4):1–26, August 2022.

[11] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. In *MLSys*, June 2020.

[12] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *ICML*, July 2020.

[13] M. Mortaheb, C. Vahapoglu, and S. Ulukus. FedGradNorm: Personalized federated gradient-normalized multi-task learning. In *IEEE SPAWC*, July 2022.

[14] C. Vahapoglu, M. Mortaheb, and S. Ulukus. Hierarchical over-the-air FedGradNorm. In *Asilomar Conference*, October 2022.

[15] M. Mortaheb, C. Vahapoglu, and S. Ulukus. Personalized federated multi-task learning over wireless fading channels. *Algorithms*, 15(11):421, November 2022.

[16] A. Fallah, A. Mokhtari, and A. Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *NeurIPS*, October 2020.

[17] S. Wang, J. Perazzone, M. Ji, and K. S. Chan. Federated learning with flexible control. 2022. Available at arXiv:2212.08496.

[18] C. Xie, S. Koyejo, and I. Gupta. Asynchronous federated optimization. 2020. Available at arXiv:1903.03934.

[19] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief. Client-edge-cloud hierarchical federated learning. *IEEE ICC*, June 2020.

[20] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin. Hierarchical federated learning across heterogeneous cellular networks. In *IEEE ICASSP*, April 2020.

[21] X. Wang and Y. Wang. Asynchronous hierarchical federated learning. 2022. Available at arXiv:2206.00054.

[22] Q. Chen, Z. You, J. Wu, Y. Liu, and H. Jiang. Semi-asynchronous hierarchical federated learning over mobile edge networks. *IEEE Access*, 11:18887–18899, December 2023.

[23] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. 2022. Available at arXiv:1806.00582.

[24] T. Nishio and R. Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. *IEEE ICC*, May 2018.

[25] M. M. Amiri and D. Gunduz. Machine learning at the wireless edge: Distributed stochastic gradient descent over-the-air. In *IEEE ISIT*, July 2019.

[26] L. P. Barnes, H. A. Inan, B. Isik, and A. Ozgur. $r$top-k: A statistical estimation approach to distributed SGD. *IEEE Jour. Sel. Areas Info. Theory*, 1(3):897–907, November 2020.

[27] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat. Coded federated learning. In *IEEE Globecom*, December 2019.

[28] M. M. Amiri, D. Gündüz, S. R. Kulkarni, and H. V. Poor. Update aware device scheduling for federated learning at the wireless edge. In *IEEE ISIT*, June 2020.

[29] W. Chang and R. Tandon. Communication efficient federated learning over multiple access channels. 2020. Available at arXiv:2001.08737.

[30] J. Hamer, M. Mohri, and A. T. Suresh. Fedboost: A communication-efficient algorithm for federated learning. In *ICML*, July 2020.

[31] N. Hyeon-Woo, M. Ye-Bin, and T. Oh. FedPara: Low-rank hadamard product for communication-efficient federated learning. In *ICLR*, May 2022.

[32] Z. Qiao, X. Yu, J. Zhang, and K. B. Letaief. Communication-efficient federated learning with dual-side low-rank compression. 2021. Available at arXiv:2104.12416.

[33] S. Vithana and S. Ulukus. Efficient private federated submodel learning. In *IEEE ICC*, May 2022.

[34] C. Liu, T. J. Chua, and J. Zhao. Time minimization in hierarchical federated learning. In *IEEE/ACM SEC*, December 2022.

[35] A. Ali and A. Arafa. Delay sensitive hierarchical federated learning with stochastic local updates. 2023. Available at arXiv:2302.04851.

[36] D. Wen, K. Jeon, and K. Huang. Federated dropout—a simple approach for enabling federated learning on resource constrained devices. *IEEE Wireless Comm. Lett.*, 11(5):923–927, February 2022.

[37] N. Bouacida, J. Hou, H. Zang, and X. Liu. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In *IEEE Infocom*, May 2021.

[38] Y. Chen, Z. Chen, P. Wu, and H. Yu. FedOBD: Opportunistic block dropout for efficiently training large-scale neural networks through federated learning. 2022. Available at arXiv:2208.05174.

[39] H. H. Yang, A. Arafa, T. Q. S. Quek, and H. V. Poor. Age-based scheduling policy for federated learning in mobile edge networks. In *IEEE ICASSP*, May 2020.

[40] B. Buyukates and S. Ulukus. Timely communication in federated learning. *IEEE Infocom*, May 2021.

[41] A. Kosta, N. Pappas, and V. Angelakis. Age of information: A new concept, metric, and tool. In *Foundations and Trends in Networking*, volume 12, pages 162–259, November 2017.

[42] Y. Sun, I. Kadota, R. Talak, and E. H. Modiano. Age of information: A new metric for information freshness. In *Age of Information*, volume 12, pages 1–224, December 2019.

[43] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus. Age of information: An introduction and survey. *IEEE Jour. Sel. Areas in Comm.*, 39(5):1183–1210, May 2020.

[44] M. Bastopcu and S. Ulukus. Information freshness in cache updating systems. *IEEE Trans. on Wireless Communications*, 20(3):1861–1874, March 2021.

[45] P. Kaswan, M. Bastopcu, and S. Ulukus. Freshness based cache updating in parallel relay networks. In *IEEE ISIT*, July 2021.

[46] H. A. David and H. N. Nagaraja. *Order statistics*. John Wiley & Sons, New York, 3rd edition, 2003.

[47] M. Bastopcu and S. Ulukus. Who should Google Scholar update more often? In *IEEE Infocom*, July 2020.

[48] P. Kaswan and S. Ulukus. Timely tracking of a remote dynamic source via multi-hop renewal updates. 2023. Available at arXiv:2304.01989.

[49] P. Kaswan and S. Ulukus. Age of information with non-Poisson updates in cache-updating networks. In *IEEE ISIT*, June 2023.

[50] E. Ozfatura, B. Buyukates, D. Gunduz, and S. Ulukus. Age-based coded computation for bias reduction in distributed learning. In *IEEE Globecom*, December 2020.

[51] S. Li, S. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr. Polynomially coded regression: Optimal straggler mitigation via data encoding. 2018. Available at arXiv:1805.09934.