Parameter-less Asynchronous Federated Learning under Computation and Communication Constraints

Mengfan Wu*†, Mate Boban*, and Falko Dressler†
*Munich Research Center, Huawei Technologies
†School of Electrical Engineering and Computer Science, TU Berlin
Email: {mengfan.wu, mate.boban}@huawei.com, dressler@ccs-labs.org

***Abstract*—Federated Learning is a fast-developing distributed learning scheme that has promising applications in vertical domains such as industrial automation and connected automated driving. In this paper we address the heterogeneity of the participation of devices in federated learning caused by: i) non-uniform distribution of local data; ii) uneven and varying computational resources across the devices; and iii) dynamic communication link. We propose a quasi-dynamic simulation scheme allowing realistic approximation of these three factors of heterogeneity. Aggregation schemes at the server based on the clients' work status are implemented. We show that the new asynchronous aggregation algorithm does not require tuning of hyper-parameters such as the round time in synchronous federated learning and the aggregation weight in classic asynchronous aggregation, while providing better or comparable performance in terms of accuracy and convergence speed.**

***Index Terms*—Federated learning, heterogeneous devices, wireless communications**

I. Introduction

Along with the growing computing power of edge devices and concerns on sharing sensitive data, federated learning (FL) emerged as a promising distributed learning scheme that utilizes the computational resources of devices while preserving user privacy, since only learned models are shared with the server or across users [1]. In particular, there is a strong interest in deploying FL in mobile and dynamic environments such as industrial automation and connected automated driving [2], [3] and modern edge computing [4]. While the devices in those environments are typically powerful and do not have severe energy constraints, there exists a set of challenges to the classic synchronized FL scheme, since some devices become stragglers due to limited computational power or slow update resulting from poor link conditions [5].

For example, in connected automated driving, some vehicles might experience unstable link connection (e.g., due to signal blockage by large objects), thus interrupting the transmission of models in FL. Moreover, since the priority of training is lower than other critical tasks for vehicles, the computational resources available for FL are likely to be dynamic and heterogeneous across devices, thus leading to slow participants in the FL task. The above aspects slow down the learning progress and result in bias of the learned models. In the following paragraphs, we use participants and clients interchangeably to denote devices taking part in the learning task.

Asynchronous FL is proposed in [6] to address the heterogeneity of devices, where the authors point out the following heterogeneous properties of participants: 1) non-uniform computational resources of the devices; 2) unstable or uneven resources of communication between the server and clients; and 3) heterogeneous data distribution, in terms of the number of data samples as well as the features of the data.

To address the issues of stragglers and heterogeneity, researchers have worked in reducing communication overhead, allowing flexible participation of devices, and integrating advanced optimization algorithms, etc. However, most of the work investigates the heterogeneity of participants from a simplistic perspective (e.g., modelling the local training time and/or the success of participation in probability [7], [8]).

In this paper, we propose a scheme for FL that models the heterogeneity of devices explicitly. We propose an aggregation algorithm that uses client information and requires fewer hyper-parameters to be tuned compared to classic approaches [6], [9]. Our work contributes as follows to the deployment of FL in dynamic environments:[1]

1) We design and implement a step-wise scheme that allows realistic modelling of computational resources and quality of communication link for participants;
2) We propose strategies for model aggregation with considerations of devices' local progress, weight in the overall task, and the latency of their model updates; and
3) We show that the new aggregation scheme is versatile in various scenarios and easy to set up with fewer hyper-parameters to tune compared to existing approaches.

II. Related Work

A. Modelling Heterogeneity

In FedAvg [9], heterogeneity of data distribution and possible communication constraints have been highlighted. The authors experimented with different tasks with independent and identically distributed (IID) data and non-IID data. The communication between the server and clients is assumed to be steady. However, there is a client-selection step which can be interpreted as modelling the availability of clients, which is caused by limitations of either computation or communication. Client selection has also been explored in [8], [10] as well as in FedProx [7], where a portion of clients are simulated to be

[1]The code of implementation can be found at: https://github.com/mengfanwu96/StepFedSim.

underperforming. However, the actual cause of slow workers are not modelled with detail.

To avoid bias towards fast clients, allowing partial progress of slow clients has been proposed to apply in the scenario of heterogeneous computational resources. For example, in FedProx [7], a portion of the clients is assumed to be only capable of completing less than required number of optimization epochs in limited time.

Besides modelling the effect of heterogeneity, we also see works modelling the actual limitations of clients' resources. Wu et al. [11] defines a performance variable as the number of batches that a device processes per second, which follows an exponential distribution. In FedCS [12], the uplink throughput for clients to upload models is simulated in an LTE environment where clients share time-division resource blocks.

B. Approaches to Address Heterogeneity

1) Data Compression and Partial Transmission: This approach is to trade off model accuracy for light communication overhead, which is plausible due to the redundancy in machine learning models, especially in deep neural networks. According to [13], neural networks can be pruned with up to 60% sparseness while achieving comparable performance to unpruned models. One naïve method is to convert parameters in machine learning models from high-precision data types to approximated ones. Another idea is to design quantization schemes to approximate the values with less binary bits (i.e., linear/exponential scaling [14]). For partial transmission, part of the parameters in models with greater numerical weights is selected, e.g., matrix sparsification [15], [16]. This way, the communication overhead can be significantly reduced. However, the computational overhead added in the (de)compression/(de)sparsification process, together with the increased latency due to these procedures, need to be weighted against the benefits and will depend from one scenario to the other.

2) Flexible Strategies for Client Participation: A lot of work has been proposed to allow more flexible client participation, e.g., allowing incomplete or out-of-synchronization updates from clients and target-driven selection of participants.

a) Flexible Local Progress: By modelling the effect of limited resources as incomplete progress, various strategies are proposed to integrate partially-trained model updates into the global model. Li et al. [7] propose to use a parameter of inexactness to allow early stopping of optimization, which is customizable according to the computational capacity of participants. However, the model updates with different progress are treated homogeneously. The instability caused by aggregating less-trained models is not addressed.

b) Asynchronous Federated Learning: Asynchronous FL is beneficial for dealing with straggler effect, by allowing fast participants to achieve high efficiency and accepting the contribution from slow participants. We often use the term staleness, usually related to the interval between the two consecutive updates of a client, to distinguish fast and slow clients in the task. Based on the availability of a global clock to define rounds of optimization, asynchronous FL can be further classified into semi-synchronous (with global clock) and fully asynchronous (without global clock). The criteria for the global clock to step forward can be time-based [11], [17], or filling a buffer [17], [18]. Authors in [17] designed a weighting scheme when aggregating updates with different staleness. Their goal is to match each clients' overall contribution proportionally to the size of their local data. We note that in these method, adapting the aggregation weights for stale updates usually entails an attenuation function to be carefully tuned in trial runs so as to achieve good performance.

c) Target-Driven Selection of Participants: In both synchronous and asynchronous FL, it is possible to select a subset of fast or reliable clients to achieve fast convergence or fairness. For example, Nishio and Yonetani [12] aim to maximize the number of participants in one round of aggregation by assigning early time-division resource blocks to fast participants. Imteaj and Amini [19] propose a scoring scheme to evaluate the participants' activity and contribution and select the most reliable ones. In selection algorithm taking clients' status of resource as inputs, clients are often required to exchange their system information, thus creating additional communication overhead. Moreover, such exchange might not even be successful in extreme communication conditions.

In general, despite the challenges posed by application conditions, allowing flexible participation suits the characteristics of FL in dynamic environments. Our work follows this direction and aims to build a system that maximizes flexibility and also address the heterogeneity issue resulting from it.

III. System Design

In this section, we elaborate on the learning algorithms, the functions of system components, and the aggregation schemes in the step-wise simulation system of FL.

A. Learning Objective

We consider a machine learning task performed by N computing devices together with a model aggregator. Assume the data distribution follows the vector $\mathbf{D} = \langle D^1, \ldots, D^N \rangle$, with local dataset D^i only visible to device i. The global and local learning tasks are the same minimization optimization problem, aiming to reduce the user-defined loss on certain datasets. Mathematically, with the loss function $\mathcal{L}$, we define the original centralized global task as finding the optimal model x^* which results in the minimum global loss:

$$x^* = arg\min_x \mathcal{L}(x, \cup_{i=1}^N D^i) \tag{1}$$

Due to the privacy concern in federated learning task, devices perform local training task aiming to minimize the local loss $x_i^* = arg\min_x \mathcal{L}(x, D^i)$ via gradient-descent method: $x' \leftarrow x - \eta \cdot \nabla_x \mathcal{L}(x, D_B)$, where η is the local learning rate and D_B is the sampled data in each optimization. The parameter server then receives trained models from devices and aims to find a global model $x^{*'}$ that minimize the sum of local losses:

$$x^{*'} = arg\min_x \sum_{i=1}^N \mathcal{L}(x, D^i) \tag{2}$$

B. Learning Systems

We simulate a learning task in T time steps as in Algorithm 1. The system starts by defining the global parameters and initializing learning models on clients. At each step, the scheduler, the clients, and the server take actions sequentially. Functions in Algorithm 1, line 2 assign computation token s_t^i and communication tokens c_t^i to client i at time step t to determine how many batch operations the client performs and how much of the model can be transmitted at this time step. Both s_t^i and c_t^i are currently drawn from offline simulation, which can be extended to using online simulation of mobile computing systems as well as real traces of computation and communication. We simplified the modelling of communication and consider the uplink (from client to server) only. If needed, the downlink can be modelled in the same way.

Algorithm 1 Learning System Simulation

Input: $N, T, \eta, B, E, \mathbf{D}, m$ // Input parameters: number of clients, total steps, local learning rate, batch size, epochs to optimize, dataset distribution, size of model
Output: x_T^g

Initialization

Initialize local models x_0^i, $\forall i \in \{1, 2, \ldots, N\}$
Set learning parameters: η, B, E on each client
Assign local dataset D^i to client i, $\forall i \in \{1, 2, \ldots, N\}$

Optimization

1: **for** $t = 1, \ldots, T$ **do**
2:   `ToAggregate` $\leftarrow \emptyset$
3:   **for** $i = 1, \ldots, N$ **do**
4:     $s_t^i \leftarrow$ sample from computational resource profile
5:     $c_t^i \leftarrow$ sample from communitation resource profile
6:     `state`, `model` $\leftarrow$ `Client[i].step()`
7:     **if** `state` == uploaded **then**
8:       `ToAggregate` $\leftarrow$ `ToAggregate` $\cup$ `model`
9:     **end if**
10:   **end for**
11:   `Server.step(ToAggregate,t)`
12: **end for**
13: **return** $x_T^g \leftarrow$ `Server.GlobalModel()`

C. System Components

1) Server: The server maintains a global model and receives model updates from clients. It tracks clients' information, such as the size of their local dataset (`LocalDataSize`), the number of batch optimizations performed since their last update (`ClientOwnPrg`), and the updating time (`LastUpdateTime`) and interval (`LastUpdateIntv`). In fully asynchronous setting, the server performs updates of the global model as soon as it receives an update from clients. The actions performed by the server are described in Algorithm 2.

At each time step, if model updates are received from clients, the server first updates the information vectors. We design the matrix `OthersPrg` to keep track of peer clients' contribution to the global model. The entry `OthersPrg[i][j]` denotes client j's contribution in the time interval starting from client i's last update to the current time step, and is therefore incremented when server receives no upload from client i and client j finishes uploading at the time step. All progress is measured as the summed batch optimizations performed to yield the model updates.

Algorithm 2 Server Actions

Information Vectors of Clients

Vectors $\in \mathbb{R}_+^{1\times N}$: `LastUpdateTime`, `LastUpdateIntv`, `LocalDataSize`, `ClientOwnPrg`
Matrix $\in \mathbb{R}_+^{N\times N}$: `OthersPrg`
All entries initialized as 0

Optimization

Input: $t, C, \{x_t^i, \forall i \in C\}$, `ProgressV`, `DataSizeV`
// current time step, list of clients that finish uploading, their model updates, a vector of their performed number of batch optimizations, a vector of sizes of their datasets.
Output: x_t^g

for $i = 1, \ldots, N$ **do**
  if $i \in C$ **then**
    `LastUpdateIntv[i]` $\leftarrow t -$ `LastUpdateTime[i]`
    `LastUpdateTime[i]` $\leftarrow t$
    `LocalDataSize[i]` $\leftarrow$ `DataSizeV[i]`
    `ClientOwnPrg[i]` $\leftarrow$ `ProgressV[i]`
  else
    `OthersPrg[i][j] += ProgressV[j]`, $\forall j \in C$
  end if
end for
$\mathbf{w} \leftarrow$ `ComputeAggregationWeights` (C)
$x_t^g \leftarrow (1 - \sum_{i\in C} \mathbf{w}^i) \cdot x_{t-1}^g + \sum_{i \in C} \mathbf{w}^i \cdot x_t^i$
Distribute x_t^g to all clients in C
$\forall i \in C$, reset `OthersPrg[i]`
return x_t^g

---

2) Client: Clients are modelled as deterministic finite-state machines (cf. Algorithm 3). In each time step, a client performs actions such as training and uploading, or it stays idle. If one time step starts at the states of `distributed` or `training`, the client i performs s_t^i number of batch optimizations assigned by the external controller. Once the designated number of epochs E is reached, the client transitions to the `uploading` state, where it can transmit data of size c_{t+1}^i in the next time step. Upon finishing transmission, the client enters `wait` mode. The distribution of global model from the server triggers `distributed` mode.

D. Aggregation Weights

Classic synchronized FL performs a weighted average over all received updates based on the sizes of clients' local datasets, as in FedAvg [9]. When asynchronous FL was firstly proposed in [6], the aggregation weight is set as a hyper-parameter to be tuned, while scaled by a function that adapts the weight w.r.t. the staleness of the updates. The literature does not agree on the direction of weight adaptation for stale

Algorithm 3 Client i Actions

Initialization

`ToUpload, state` $\leftarrow m$, initialized

`OptimizedEpochs, OptimizedSteps` $\leftarrow 0, 0$

Step function when `state` $\in$ {`distributed`, `training`}

Input: t, s_t^i

for $k = \{1, \ldots, s_t^i\}$ **do**

  Take B samples from local dataset and train the model $x \leftarrow x - \eta \cdot \nabla_x \mathcal{L}(x, D_B)$

  `OptimizedEpochs +=` 1 if dataset iterated

  `OptimizedSteps +=` 1

  if `OptimizedEpochs` $\geq E$ **then**

    `state` $\leftarrow$ `uploading`

    `ToUpload` $\leftarrow m$

    break

  end if

end for

Step function when `state == uploading`

Input: t, c_t^i

`ToUpload` $\leftarrow$ `ToUpload` $- c_t^i$

if `ToUpload` ≤ 0 **then**

  `state` $\leftarrow$ `wait`

  return `OptimizedSteps`, x (marked as x_t^i)

end if

updates. To this end, we did experiments with IID data and varing link status and found that reducing the aggregation weights for stale updates results in a more stable performance and faster convergence. Therefore, we follow this direction and propose three basic types of aggregation weights:

$$
\begin{aligned}
w_D^i &= \frac{|D^i|}{\|\langle |D^1|, \ldots, |D^N| \rangle\|_2} \\
w_P^i &= \frac{P^i}{\|\langle OP^{i,1}, \ldots, OP^{i,N}, P^i \rangle\|_2} \\
w_S^i &= \frac{Q^i}{\|\langle Q^1, \ldots, Q^N \rangle\|_2}, \text{with } Q^i = \frac{\sum_{j=1,\ldots,N} \mathtt{Intv}^j}{\mathtt{Intv}^i}
\end{aligned}
\tag{3}
$$

$|D^i|$ corresponds to the local data size of client i in Algorithm 2, P^i to the `ClientOwnPrg` of client i, $OP^{i,j}$ to the `OthersPrg[i][j]`, and $\mathtt{Intv}^i$ to the `LastUpdateIntv` of client i. We interpret Q as quickness, which is computed as the ratio of summed updating intervals to a client's own updating interval. We use the norm of vectors as the denominator rather than the sum of vector elements because the former yields larger weights and is beneficial for quicker evolving of global model. It is possible that multiple weights, computed for multiple model updates at the same time step as in Line 13 in Algorithm Algorithm 2, sum up to more than 1. In this case, we divide the weights by their sum to keep the aggregation stable.

The data size weight w_D in our setup follows FedAvg [9], as it prefers clients with greater importance when testing accuracy. The progress weight w_P assigns greater importance to updates with more optimization performed, implicitly with more computing resources spent, which tends to be better-trained and beneficial to the global model. For staleness weight w_S, we assign small weights to updates with long updating intervals, so as to avoid model divergence since they are from clients not frequently synchronized with the server. There are contradictions among the three weights. For example, for a client with a large local dataset, the data size weight assigns its updates greater importance, while the client is also likely to have longer training time and thus gets smaller weights in terms of staleness. To this end, **by using the average of the three types of weights**, we hope to provide a trade-off among the three factors. The aggregation weights are deployed only after the server receives updates from each client for at least once.

IV. Experiments

With functions `SetComputationCapacity(t)` and `SetLinkStatus(t)` in Algorithm 1, we can now experiment with various application scenarios. Moreover, we also experiment with different data distribution on devices. The size of the local dataset, the computational capacity, and the link throughput have intertwined impacts on the progress and staleness of model updates. We perform a set of control-variable experiments to evaluate the performance of the newly proposed asynchronous learning scheme against FedAvg [9] and asynchronous FL based on weight attenuation. Synthetic IID data as in FedProx [7] is used in the following three groups of experiments. The task is a convex classification task and has 10 target classes and a 1-dimensional feature space with length 60. To this end, a single-layer perceptron is used as the learning model, which suffices to have a satisfactory local accuracy, with potentially better performance achievable if more complex learning models are deployed. The loss function used here is cross entropy loss for the softmax layer output (the output layer of the perceptron), which is commonly adopted for classification tasks.

For synchronous FL benchmark, we implement FedAvg with the same external control of computation and communication, while the server has a fixed round time (40, 60, 80, or 100) to receive updates and perform model aggregation.

For parameterized asynchronous FL benchmark, we implement the attenuation-based method which decreases the aggregation weight of model updates w.r.t. staleness:

$$
w_{att}^i = w_D^i \cdot (\mathtt{Intv}^i - t_{cut})^{\alpha} \tag{4}
$$

t_{cut} is the hyper-parameter as baseline to determine the staleness of the update, and α is the factor determining the scale of attenuation. In the following experiments, we use $\alpha = 0.9$ only and search for a suitable t_{cut} with interval 5, which is dependent on the computational speed and communication condition.

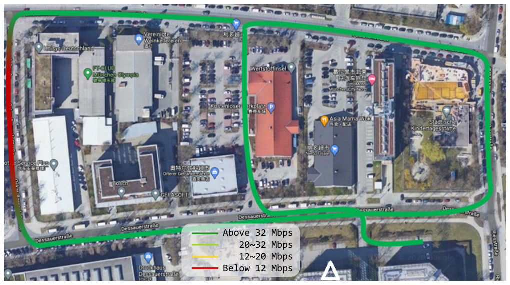


Fig. 1: Driving route and uplink throughput of Link Profile 5. The base station is located at ▲. The height of the base station antenna is 21 meters above ground level, whereas the vehicle antenna is at approximately 1.5 meter height, mounted on the vehicle roof. The test vehicle traversed the double loop shown by the overlay 10 times.

The evaluation is based on test accuracy and convergence time (defined as the the time step to achieve 85% of the highest final accuracy in the group of experiments). Local learning parameters are set as: learning rate $\eta = 0.02$, batch size for optimization: $B = 8$, number of training epochs per round $E = 40$.

A. Dynamic computational resources

In this group of experiments, we set the clients' local datasets to be of the same size (240). Communication links are set to be steady with transmission time being 5 steps. We simulate 3 scenarios with clients' computational resources varying in different ranges. The computation token s is changed every 32 steps. We show the settings and the corresponding results in Table I. For the first case where computational speed is fixed, all clients train and upload at identical time. Asynchronous FL finds an optimal round time for aggregation implicitly and outperforms all synchronous scenarios. For the second and third cases, the performances of parameter-less asynchronous FL are in between of the best and second best synchronous settings, and close to the best cases of attenuation-based asynchronous FL where t_{cut} is searched in a range.

B. Dynamic link resources

In this group of experiments, we keep the local samples to be evenly distributed (240 samples/client) and fix the computational resource stable at 30 batches/step. We simulate different communication scenarios by varying the link throughput and the size of models to be transmitted. Communication tokens of Profile 1 to 4 are sampled from uniform, poisson, poisson, and lognormal distributions accordingly. Link profile 5 is extracted from link measurements collected in a measurement campaign evaluating uplink throughput as shown in Figure 1 and described in detail in [20]. The transmission time distribution in different link profiles is shown in Figure 2.

The learning results of different communication scenarios are shown in Table II. Since the server does not distinguish the training time and uploading time of clients' updates, longer uploading time in this group is comparable to the cases in the last group where the computation is slow. We see similar results in Table II, where parameter-less asynchronous FL outperforms all FedAvg in cases with the most steady communication

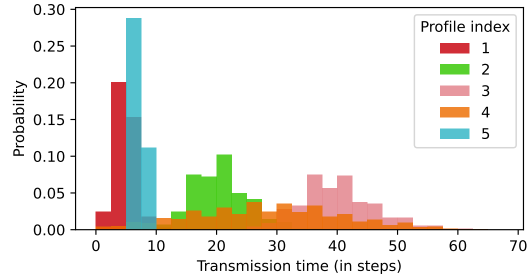


Fig. 2: Histogram of transmission time for communication scenarios.

link throughput (profile 5) and therefore stable uploading interval. For others with varying uploading time, performance of parameter-less asynchronous FL is further away from the best performing synchronous FL if the throughput is less stable (instability rank: profile $4 >$ profile $3 >$ profile $2 >$ profile 1). Moreover, parameter-less asynchronous FL achieves similar performances as the optimal case of attenuation-based method. We note that under different communication conditions, the optimal setting of t_{cut} in attenuation-based method varies greatly.

C. Imbalanced data distribution

In this group of experiments, we keep the computation and communication of clients steady. The computational capacity is fixed at 30 batches/step and the link throughput is set steady with fixed transmission time 5 steps. We create distributions of data samples across devices with different standard deviation. Moreover, we also set the number of target classes available at one device from 2 to 10, so as to investigate the effect of class imbalance. The total number of samples are fixed at 7200.

Table III shows simplified results in the form of the ranking of parameter-less asynchronous FL compared to synchronous FL with round time of 40, 60, 80, and 100 and attenuation-based asynchronous FL with t_{cut} in $\{30, 35, 40, 45\}$. The values range from 1, when parameter-less asynchronous FL outperforms all other schemes, to 5, when it performs the worst).

The new asynchronous FL has significant advantages over synchronous FL when the samples are relatively evenly-distributed across devices (std = 0, 50, 100) and class imbalance is mild (classes per device = 9, 10). When samples are evenly distributed (std = 0), asychronous FL finds the optimal round time for synchronized case implicitly. Moreover, for 49 out of 54 cases evaluating accuracy and 45 out of 54 cases evaluating convergence speed, the new asynchronous FL ranks 1 or 2, meaning that overall it has superior or comparable performance to the best performing synchronous FL scheme.

The advantages of parameter-less FL are further confirmed when compared with attenuation-based method, with parameter-less asynchronous FL performs the best in more than 95% of the cases . We note that due to the imbalance of data distribution, it is hard to determine a suitable t_{cut} for attenuation-based method.

TABLE I: Settings and results of experiments with dynamic computational resources.

`min`	`max`	Param.-less Async. FL		Sync. FL			Attenuation-based Async. FL		
s	s	acc.	T_{conv}	round T	acc.	T_{conv}	t_{cut}	acc.	T_{conv}
30	30	884	315	60	0.859	413	$[29, 44]$	$[0.884, 0.884]$	$[315, 315]$
				others	≤ 0.841	≥ 550	≤ 24	≤ 0.876	≥ 338
20	40	0.875	356	40	0.885	275			
				60	0.860	405	$[39, 44]$	$[0.878, 0.879]$	$[367, 378]$
				others	≤ 0.844	≥ 540	≤ 34	≤ 0.870	≥ 376
10	50	0.876	379	40	0.887	281			
				60	0.859	434	$[34, 44]$	$[0.880, 0.894]$	$[336, 393]$
				others	≤ 0.843	≥ 559	≤ 29	≤ 0.866	≥ 417

TABLE II: Settings and results of experiments with dynamic link throughput.

Profile	Param.-less Async. FL		Sync. FL			Attenuation-based Async. FL		
Index	acc.	T_{conv}	round T	acc.	T_{conv}	t_{cut}	acc.	T_{conv}
1	0.876	341	40	0.887	292			
			60	0.859	430	42	0.880	330
			others	≤ 0.841	≥ 574	≤ 37	≤ 0.874	≥ 349
2	0.853	379	60	0.861	316			
			80	0.841	423	55	0.853	381
			others	≤ 0.830	≥ 529	≤ 50	≤ 0.849	≥ 395
3	0.833	450	80	0.843	365	75	0.835	410
			100	0.830	458	70	0.830	446
			others	≤ 0.336	≥ 1920	≤ 65	≤ 0.820	≥ 525
4	0.845	457	60	0.861	334			
			80	0.841	427	$[55, 65]$	$[0.850, 0.856]$	$[406, 424]$
			others	≤ 0.831	≥ 532	≤ 50	≤ 0.844	≥ 477
5	0.875	294	60	0.859	361	44	0.876	294
			others	≤ 0.841	≥ 481	≤ 39	≤ 0.872	≥ 288

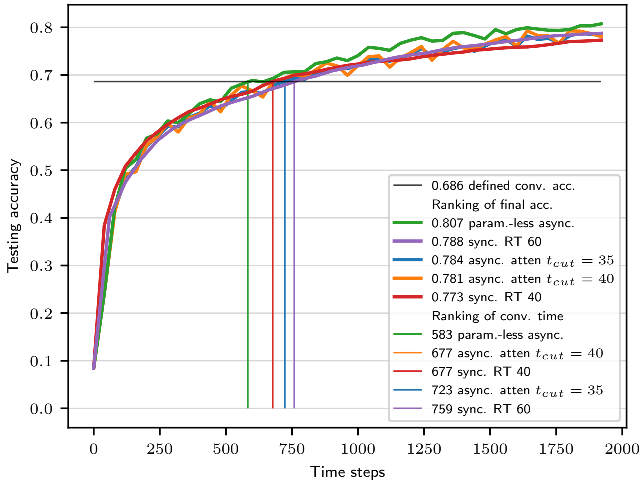
We also show two sample cases of the evolving accuracy of different learning algorithms in Figure 3. The horizontal line is plotted at 85% of the highest achieved accuracy after 1920 time steps. Then the intersection of the horizontal line with the curves are identified, from which vertical lines are then drawn to indicate the time reaching convergence. Two sets of legends are provided, one in descending order of final accuracy and another in ascending order of convergence time. We note that parameter-less asynchronous FL shows the best accuracy and second best convergence speed, while at the same time experiencing greater fluctuation as general asynchronous FL during the learning process in case of distribution imbalance.

V. Discussions and Conclusion

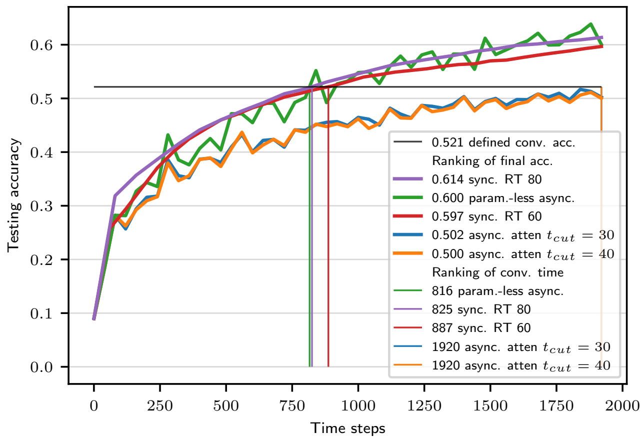
Our new asynchronous FL algorithm simplifies the process of deploying FL in an unknown condition by requiring no environment-related hyper-parameters. It keeps the advantages of asynchronous FL, collecting information from all clients, therefore making the global model more inclusive; the disadvantage of asynchronous FL is still visible in our solution, with frequent partial aggregation at the server lacking stability, especially when the updates are from non-representative devices / stale devices. The disadvantages are prominent when class imbalance is mild but updating intervals vary, and when class imbalance is extreme and updating intervals are relatively steady. The advantage of asynchronous FL in collecting more information is prominent in the cases when both class imbalance and sample imbalance are extreme, where FedAvg misses important updates from clients whereas asynchronous

TABLE III: Data and label distributions and corresponding ranking of the performance of parameter-less asynchronous FL.

	Number of classes per device								
	2	3	4	5	6	7	8	9	10
Compared with sync. FL with diff. round time									
Dist. std.	Ranking of final acc.								
0	1	1	1	1	1	1	1	1	1
50	5	3	2	2	2	1	1	1	1
100	1	1	1	1	1	1	2	1	1
200	2	1	1	1	1	1	1	3	3
300	1	2	1	1	2	2	2	2	3
400	2	1	2	1	2	2	2	2	2
Dist. std.	Ranking of convergence speed								
0	1	1	1	1	1	1	1	1	1
50	3	3	2	3	3	3	2	1	1
100	3	1	1	2	1	1	1	1	1
200	1	1	1	1	1	1	1	3	3
300	1	2	1	1	1	2	2	2	3
400	1	1	2	1	2	1	2	2	2
Compared with attenuation-based async. FL with diff. t_{cut}									
Dist. std.	Ranking of final acc.								
0	1	1	1	1	1	1	1	1	1
50	3	3	1	1	2	1	1	1	1
$\geq$100	1	1	1	1	1	1	1	1	1
Dist. std.	Ranking of convergence speed								
0	1	1	1	1	1	1	1	1	1
50	2	1	1	1	1	1	1	1	1
100, 200	1	1	1	1	1	1	1	1	1
300	3	1	1	1	1	1	1	1	1
400	1	1	1	1	1	1	1	1	1



(a) std.$= 50$, classes per device $= 9$



(b) std.$= 200$, classes per device $= 2$

Fig. 3: Sampled evolving accuracy of different algorithms under different data distributions

FL receives them all. We also note that for cases when FedAvg outperforms asynchronous FL, the settings of round time are different depending on computation and link resources.

To conclude, we identify the cause and effect of heterogeneity of device participation in FL. We propose a scheme that models the heterogeneity and can be extended to various application scenarios. We conduct a set of experiments to simulate heterogeneous user contribution and update intervals and show that the average of the three designed weights performs well in general application scenarios. The biggest practical benefit of the new scheme is that it does not require system information to tune hyper-parameters such as round time in synchronous FL and benchmark duration in classic asynchronous FL, while providing better or comparable performance. Therefore, we see a great potential in terms of flexibility for practical deployments of the proposed scheme. Further improvements in convergence speed and accuracy can be possibly achieved by exploring adaptive progress of users and dynamic resource management at the server side.

References

[1] W. Y. B. Lim, N. C. Luong, D. T. Hoang, et al., "Federated Learning in Mobile Edge Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[2] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated Learning for Internet of Things: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, Jul. 2021.

[3] M. K. Abdel-Aziz, C. Perfecto, S. Samarakoon, M. Bennis, and W. Saad, "Vehicular Cooperative Perception Through Action Branching and Federated Reinforcement Learning," arXiv, cs.LG 2012.03414, Dec. 2020.

[4] F. Dressler, C. F. Chiasserini, F. H. P. Fitzek, et al., "V-Edge: Virtual Edge Computing as an Enabler for Novel Microservices and Cooperative Computing," *IEEE Network*, vol. 36, no. 3, pp. 24–31, May 2022.

[5] F. Malandrino and C. F. Chiasserini, "Federated Learning at the Network Edge: When Not All Nodes Are Created Equal," *IEEE Communications Magazine*, vol. 59, no. 7, pp. 68–73, Jul. 2021.

[6] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," arXiv, cs.DC 1903.03934, Mar. 2019.

[7] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Conference on Machine Learning and Systems (MLSys 2020)*, Austin, TX, Mar. 2020.

[8] H. Jin, N. Yan, and M. Mortazavi, "Simulating Aggregation Algorithms for Empirical Verification of Resilient and Adaptive Federated Learning," in *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT 2020)*, Leicester, United Kingdom: IEEE, Dec. 2020.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, Fort Lauderdale, FL: PMLR, Apr. 2017, pp. 1273–1282.

[10] Z. Qu, K. Lin, J. Kalagnanam, Z. Li, J. Zhou, and Z. Zhou, "Federated Learning's Blessing: FedAvg has Linear Speedup," arXiv, cs.LG 2007.05690, Jul. 2020.

[11] 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, May 2021.

[12] T. Nishio and R. Yonetani, "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge," in *IEEE International Conference on Communications (ICC 2019)*, Shanghai, China: IEEE, May 2019.

[13] H. Guo, S. Li, B. Li, Y. Ma, and X. Ren, "A New Learning Automata-Based Pruning Method to Train Deep Neural Networks," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3263–3269, Oct. 2018.

[14] J. Mills, J. Hu, and G. Min, "Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.

[15] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, and D. B. Ananda Theertha Suresh and, "Federated Learning: Strategies for Improving Communication Efficiency," arXiv, cs.LG 1610.05492, Oct. 2016.

[16] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Sparse Binary Compression: Towards Distributed Deep Learning with minimal Communication," in *International Joint Conference on Neural Networks (IJCNN 2019)*, Budapest, Hungary: IEEE, Jul. 2019.

[17] Z. Wang, Z. Zhang, and J. Wang, "Asynchronous Federated Learning over Wireless Communication Networks," in *IEEE International Conference on Communications (ICC 2021)*, Virtual Conference: IEEE, Jun. 2021.

[18] 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, Dec. 2021.

[19] A. Imteaj and M. H. Amini, "FedPARL: Client Activity and Resource-Oriented Lightweight Federated Learning Model for Resource-Constrained Heterogeneous IoT Environment," *Frontiers in Communications and Networks*, vol. 2, Apr. 2021.

[20] M. Boban, C. Jiao, and M. Gharba, "Measurement-based Evaluation of Uplink Throughput Prediction," in *95th IEEE Vehicular Technology Conference (VTC 2022-Spring)*, Helsinki, Finland: IEEE, Jun. 2022.