Heterogeneity-Aware Adaptive Federated Learning Scheduling

Jingoo Han[‡], Ahmad Faraz Khan[‡], Syed Zawad[§],
Ali Anwar[||], Nathalie Baracaldo Angel[¶], Yi Zhou[¶], Feng Yan[§], Ali R. Butt[‡]
[‡]Virginia Tech, [§]University of Nevada, Reno,
[||]University of Minnesota, [¶]IBM Research–Almaden
[‡]{jingoo, ahmadfk, butta}@vt.edu, [§]{szawad, fyan}@nevada.unr.edu,
[||]aanwar@umn.edu, [¶]{baracald, yi.zhou}@ibm.com

***Abstract*—Federated learning (FL) is becoming an important distributed machine learning approach that considers privacy and security concerns while training a shared model across various clients with localized data. One of the key challenges in FL is heterogeneity in both hardware resources and local datasets due to the nature of incorporating diverse clients. Given the resource heterogeneity, the availability of participating clients is not stable over time and their resource usage patterns become dynamic. This leads to resource wastage and straggler issues. Additional challenges are introduced due to data heterogeneity, causing model biasness and poor model performance. However, most existing FL systems are not well suited to heterogeneous environments because those approaches are not adaptive to various and dynamically changing resource usage patterns and accuracy trends during training process. To this end, we propose a heterogeneity-aware scheduling which is adaptive to the accuracy trends and various resource usage patterns. Our proposed scheduling provides different scheduling knobs for achieving different goals such as resource-efficient fast training, resource fairness, accuracy fairness, and high model performance. To the best of our knowledge, this is the first effort to mitigate effects of resource and data heterogeneity while providing adaptive scheduling based on dynamically changing resource usage patterns and accuracy trends.**

***Index Terms*—Federated learning, Privacy-aware and secure deep learning, Federated learning scheduling, Resource management and scheduling, Distributed deep learning**

I. Introduction

Recently, a plethora of IoT devices are being used as sources to generate an unprecedented amount of data in our daily life, which means that machine learning (ML) models are trained using more abundant and sophisticated data. However, the existing distributed training methods [1]–[3] require local data for training to be moved from various local locations into a central location, leading to cybersecurity risks [4]. Furthermore, privacy laws and regulations [5], [6] prohibit data movement between multiple locations. To address this issue, federated learning (FL) [4] has emerged as a collaborative training approach without sharing raw data between other devices and locations. The main idea here is that each client[1] or device sends only local updates to a central server, while the local raw data stays within the premises. Each client (a data party) trains local model with its own local data, and sends learned gradients into a central place (an aggregator) instead of sending local data.

Deploying FL is not a trivial task. The largest challenges lie in heterogeneity, more specifically, resource, data and availability heterogeneity [4]. Resource heterogeneity is not avoidable because each client or device has different computing capability and connection bandwidth. A simple FL system randomly selects multiple devices among available devices per round. The training time per round is bounded by the slowest device [7], [8]. As a result, fast devices should wait until slow devices to complete their training process. To address such a straggler issue, several methods [9], [10] have been proposed. FedCS [9] addresses straggler issues by dropping slow devices. Google [10] proposes large-scaled FL systems, which selects more clients (e.g., 130%) and prunes slow ones. However, these approaches lead to biased models because information of data are missed when always dropping slow devices. Moreover, dropping devices results in resource wastage, making device availability one of the unique issues in FL. Due to various users' patterns at different local times, availability of devices also shows different status about battery charging, wifi connection, idle mode and so on [10]–[12]. Therefore, we need to incorporate all of these heterogeneity-related issues holistically into a single solution for heterogeneity-aware FL.

On top of resource heterogeneity in FL systems, each device may have different data distribution, which imposes challenges to obtain models with good performances trained on these datasets in a federated way (data heterogeneity). Unlike existing distributed training systems [1]–[3], in FL, it is highly possible that dataset is not uniformly distributed [13], namely non-Identical Independent Distribution (non-IID). It is well-known that models trained on non-IID party data distribution may have poorer model performance than those of models trained on Identical Independent Distribution (IID) party data distribution [13], [14]. To mitigate the above issues, the recent work TiFL [7] proposes an adaptive tier-based approach where parties are grouped by training latency to minimize straggler effect due to resource heterogeneity, while introduces credits to prevent biasness. Oort [15] introduces client selection based

[||]Work performed while at IBM Research–Almaden.

[1]The terms 'client' and 'participant' are used interchangeably in the paper.

on training latency and importance sampling, while providing trade-off between system and statistical efficiency. FedProx [16] addresses data heterogeneity by adding a proximal term to stabilize the convergence of model training. However, these approaches do not provide scheduling flexibility to prioritize different goals such as resource-efficient fast training, resource fairness, accuracy fairness, and high model performance or do not contain mechanisms to accommodate client availability.

Additionally, FL with numerous devices leads to a new challenge, availability of clients. In the real-world scenarios, it is observed that up to 10,000 mobile devices can be possibly connected into a single learning system [10]. Each device is located geographically at different places with different time zones, which means that the number of participating devices is seriously discrepant regarding local time and user's usage pattern [11]. Many existing approaches try to select unavailable devices without considering the availability of devices, leading to wastes of resources.

To the best of our knowledge, existing works [7], [9], [10], [15], [16] do not integrate availability, resource heterogeneity, and non-IID party data distribution into a holistic approach. Also, these approaches are not well suited to heterogeneous environment where resource usage patterns and model performances are dynamically changing during the training process. To address the aforementioned issues, in this paper, we propose a heterogeneity-aware adaptive scheduling method that can be configured to achieve different goals such as resource-efficient fast training, resource fairness, accuracy fairness, and model performance, where selecting parties is conducted based on resource usage pattern, accuracy measurement, and availability of each party. By doing so, our work jointly considers heterogeneity and fairness at the same time. To support a variety of different scheduling objectives, we extend our scheduling to incorporate fairness metrics like resource fairness and accuracy fairness, as well as performance metrics such as total training time and final model accuracy. For example, according to scheduling policy, the scheduling selects clients that spend less time on CPU for resource-efficient fast training, whereas the scheduling chooses clients that spend similar time on CPU for resource fairness.

In addition, our proposed scheduling method provides scheduling knobs for achieving above mentioned various goals. To improve the performance and fairness of the obtained model, party selection is scheduled based on local accuracy and global accuracy of each party. To achieve efficiency and fairness of resource, parties are chosen based on consumption and biasness of resource usage of each party. To the best of our knowledge, our work is the first effort to handle heterogeneity of FL by scheduling parties based on availability, resource usage, and accuracy trend, while providing dynamically adaptive scheduling to accomplish various scheduling goals such as performance and fairness.

We summarize our major contributions as follows:

- We present a novel adaptive FL scheduling that profiles the availability and resource usage for each client over time, measures the model biasness and model performance, calculates possible combinations of client selection over time, and then determines the best client combination weighted by scheduling knobs that consider resource heterogeneity, fairness, and model performance.
- We design a method to measure the biasness of the global model by quantifying clients' benefit from the global model and each clients' contribution to the global model for achieving better global model accuracy.
- We propose a method to determine the priority of a client based on its local training data and schedule clients dynamically based on their resource usage.
- We implement and evaluate the proposed FL scheduling to demonstrate that the proposed scheduling provides configurable scheduling policies for achieving better model performance, faster training time, accuracy fairness, and resource fairness.

II. Background and Motivation

A. Federated Learning with Resource Heterogeneity

Due to the nature of FL systems, various devices process data with extremely heterogeneous resources [8]. This high resource heterogeneity has been known that it has negative effects such as the straggler issue on performance of FL in terms of training time and resource efficiency. According to [7], average training time of a single round shows a difference of nearly 8x between fastest and slowest clients. This observational study implies that overall training time can be increased by up to 8x when slow clients are frequently selected. FL aggregates local updates in a synchronous way where training throughput is bounded by slow devices [7], [8]. The single-round training time of slow devices is longer than other devices due to their low computing capabilities and slow communication bandwidth. Google [10] suggests over-selection to mitigate straggler effects, but this approach leads to a waste of resource utilization because devices are unnecessarily allocated per round.

B. Federated Learning with Non-IID Data

Unlike traditional distributed DL systems where the whole dataset is collected at a central location and redistributed in a controlled way, the dataset is generally not identically distributed among devices [13] in FL. As a result, each device participating in FL should handle highly skewed dataset, leading to bias according to discrepancy of dataset distribution [14]. In non-FL systems, classes of dataset can be uniformly distributed to each device, called Independent Identical Distribution (IID), since the ML engineers have complete control over the system. On the contrary, in FL, classes of dataset are not identically distributed depending on users' experience and preference, known as non-Identical Independent Distribution (non-IID), with additional restrictions imposed upon them (such as inability to view or move them) due to privacy constraints.

C. Federated Learning with Availability Heterogeneity

One of unique characteristics of FL is that availability of participants is very diverse according to users' usage pattern or status of devices [11], [12]. Devices need to meet requirements

to participate in training process for FL. For example, devices are not available when battery life is not enough or devices are connected to metered network. In this case, it is possible that devices participating in a round drop out during training time due to their unreliability. It is also known that device availability shows daily patterns and depends on its local time [10], [11]. Normally, devices are more available at night than day because devices are charging, idle, and connected on wifi at night. Google observed that difference between low and high numbers of participants is four times within 24 hours [10]. Even, it is also possible that some devices are in charging on wifi connection at day according to users' behavior. As a result, the availability heterogeneity causes participation bias and resource inefficiency (i.e., dropout problem), which has negative effects on accuracy of global model [17].

D. Clients Selection

The default FL randomly selects clients per round without considering resource situation, leading to inefficient resource utilization, straggler effect, and accuracy reductions. To address these problems, clients per round need to be chosen carefully based on resource condition, heterogeneity of dataset, and availability. FedCS [9] chooses clients as many as possible within specific time frame, considering resource information (i.e., training time, energy consumption). TiFL [7] chooses a group and then randomly selects clients within the selected group for minimizing straggler effects. Oort [15] also uses careful client sampling to prioritize selection such that the training completes faster with increased model performance. However, all of these approaches do not consider fluctuations in availability for selecting clients and scheduling training. Additionally, these approaches do not handle fairness, which means that a global model is biased either against or towards specific clients. To improve fairness, bias-aware metrics needs to be measured and clients should be selected based on them.

E. Various Resource Scenarios

Beyond heterogeneity at device level, it is possible that FL environment is highly resource heterogeneous. As a result, there are various resource scenarios when building FL systems. Let's assume that FL is processed with *ample resources*. A myriad of powerful machines may be connected with super-fast network connectors and there is no time limit on any machines. In this case, it is reasonable to give high priority to accuracy of the trained model. However, in the case of FL systems with *limited resources*, training with low resource consumption is more important than achieving high accuracy. Waste of resource usage should be minimized for improving resource utilization within limited resource environment. It is also possible that FL is conducted with *time constraints*. For example, in the case of HPC systems such as supercomputers, dedicated machines are allocated exclusively with maximum walltime [18]. FL should be finished within the walltime limit. Thus, overall training time is important and final model accuracy can be sacrificed. As can be seen in the above-mentioned cases, scheduling and client selection in FL systems need to consider different conditions.

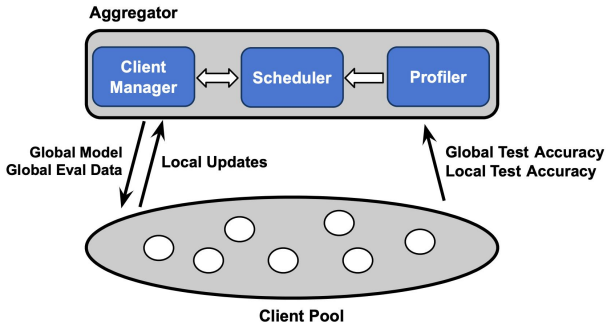


Fig. 1: Overview of heterogeneity-aware adaptive scheduling.

However, existing solutions [7], [15] on FL do not provide features to achieve various goals.

III. Heterogeneity-Aware Adaptive Scheduling

A. Overall Design

The overall architecture of the proposed heterogeneity-aware adaptive scheduling is presented in Figure 1. Our proposed system includes additional modules, realized at the aggregator side: *client manager*, *profiler*, and *scheduler*. The main role of *client manager* is to interact with clients: it requests local training, distributes global model and global evaluation data, and aggregates local update from clients. The *profiler* monitors accuracy of clients to keep track of accuracy performance and accuracy bias, including other resource usage patterns such as training throughput. The profiler asks all registered clients to report both *global test accuracy* and *local test accuracy*. The global test accuracy can be measured by using the shared global model and clients' local test data, whereas the local test accuracy can be measured by using clients' trained local model and the global evaluation data. Then, each client sends these accuracy results to profiler, and profiler stores the collected accuracy results. The details of global test accuracy and local test accuracy will be explained in subsection III-C. In each training round, the *scheduler* sorts clients in descending order based on the collected accuracy records and resource usage patterns, and then ranks clients according to scheduling priority weights. Scheduler selects list of clients and forwards it to the client manager for training execution. In the following subsections, we discuss the details of clients selection.

B. Proposed Scheduling: Heterogeneity-Aware Adaptive Scheduling

We propose a scheduling algorithm for FL, named *Heterogeneity-Aware Adaptive Scheduling*. The main idea is to select participating clients per round based on their availability and training throughput, and dynamically adjust the client selection according to the accuracy measurements at both global and local levels, which mitigates straggler problems and biasness of trained model. Furthermore, scheduling policy is dynamically configured by scheduling knobs that prioritize different objectives such as resource efficiency, resource fairness, accuracy fairness, and model performance. The details of our proposed scheduling algorithm is shown in Algorithm 1. Initially, availability and training latency of each client

Algorithm 1: Heterogeneity-Aware Adaptive Scheduling.

Input: r: current round to be scheduled, R: total number of rounds, I: the interval of measuring accuracy, K: total number of clients, Acc^k_{global}: global test accuracy of client k, Acc^k_{local}: local test accuracy of client k

```
1  begin
2      Profile availability of each client over time;
3      Profile training latency of each client;
4      for each round r = 0 to R - 1 do
5          if r%I == 0 and r >= I then
6              for each client k = 0 to K - 1 do
7                  Acc^k_global,
                     Acc^k_local ← measure_accuracy();
8              end
9          end
10         Calculate objective function using availability,
             training latency, Acc^k_global, and Acc^k_local of
             each client;
11         Rank all clients based on objective function;
12         Select subset of clients per round;
13         Train selected clients;
14     end
15 end
16 Function measure_accuracy()
17     Sends global evaluation data and global model to all
         clients;
18     Train global model on local data;
19     Measure Acc^k_global by testing local model on global
         evaluation data;
20     Measure Acc^k_local by testing global model on local
         test data;
```

is profiled (lines 2–3). Per intervals of I rounds, global test accuracy and local test accuracy are measured and collected from each client (lines 5–9). Then, the scheduler calculates objective function, sorts all clients, and selects subset of clients to be trained per round (lines 10–12). The client manager trains the selected clients (line 13). The *measure_accuracy* is a function to return global test accuracy and local test accuracy of each client. The details of how to measure them will be explained in subsection III-C. The objective function used in the algorithm will be discussed as below.

1) Objective Function: Let's suppose that we have four clients, such as A, B, C, and D. These clients have different computing performances and show changing patterns of client availability. Training latency of each client is 1, 2, 3, and 4. Availability over rounds is shown in Table I.

We want to run for a total of 5 rounds, with 2 clients selected per round. For round 1, the client A is not available, while other clients can be chosen. We can come up with possible combination of clients for round 1 (i.e., BC, CD, and BD). For each possible combination, variance of training latency is calculated via population variance (i.e., $\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$) : 1.69, 3.19, 2.75. Thus, the combination of BC shows the least variance. We find the least variance per round, so that the least amount of difference between resource usage can be minimized by finding an optimal combination per round. Thus, objective function can be as below:

TABLE I: Availability of 4 clients over rounds.

Client \ Round	1	2	3	4	5
A	0	1	1	0	1
B	1	1	0	1	0
C	1	0	1	1	1
D	1	0	1	0	1

$$ObjectiveFunction = \arg\min_{k \in K}\{var(R_k)\} \quad (1)$$

R_k indicates resource usage for client k. Additionally, we can consider total resource usage. Sum of training latency of possible combination is 5, 7, and 6 for BC, CD, BD, respectively. Therefore, BC is the best combination for minimizing total usage of resources. In this case, objective function for sum of resource usage can be as below:

$$ObjectiveFunction = \arg\min_{k \in K}\{sum(R_k)\} \quad (2)$$

In case of the above example, the combination of BC is the best option in terms of both resource variance and resource consumption. However, unlike the above-mentioned example, it is possible that total resource usage and resource fairness do not lead to the same scheduling decision. To handle this case, the following multi-objective optimization can be considered with weight priorities (i.e., w_1, and w_2).

$$ObjectiveFunction = \arg\min_{k \in K}\{W_1 * sum(R_k) + W_2 * var(R_k)\} \quad (3)$$

Now, we have a scheduler that focuses on resource consumption while reducing drop-outs. Additionally, model performance and biasness can be considered for fast and fair convergence of trained global model. Our proposed method to measure the model accuracy and biasness is explained in detail in subsection III-C. If we have values of model accuracy and accuracy fairness, we can consider these values for our new objective function. Thus, final objective function can be presented as:

$$ObjectiveFunction = \arg\min_{k \in K}\{W_1 * sum(R_k) + W_2 * var(R_k) + W_3 * sum(B_k) + W_4 * sum(P_k)\} \quad (4)$$

B_k and P_k indicate bias for client k and model performance for client k, respectively. B_k and P_k can be obtained by measuring *global test accuracy* and *local test accuracy*, respectively. The details of how to measure global test accuracy and local test accuracy will be discussed in the following subsection. Each weight (i.e., w_1, w_2, w_3, and w_4) is user-defined and can be optimized according to user's preference.

C. Accuracy Performance and Fairness

Due to data heterogeneity (i.e., non-IID), only focusing on resource consumption may lead to slow down accuracy

convergence although training time is reduced. In terms of model performance, all clients do *not* contribute equally to training global model (as has been pointed out by many works such as [7], [15]), which indicates that better contributing participants should be selected with higher priority to achieve better model performance. Also, we notice that all clients do *not* benefit equally from global model, which means that we need to check whether global model performs well on all clients to achieve accuracy fairness. Accuracy fairness means that trained model benefits fairly to all participating clients in terms of model performance, without being biased to specific clients. However, both contribution and biasness of each participant are unknown a-priori. This leads us to the following questions - *How can we measure performance contribution of a participant? How can we measure performance benefit to a participant?* To answer these questions, we propose the two following metrics: (1) *global test accuracy*, and (2) *local test accuracy*. The global test accuracy is the global accuracy of local model to measure *contribution* of a participant through IID dataset, while the local test accuracy is the local accuracy of global model to measure *biasness* of a participant through non-IID dataset. Both of the two metrics mentioned above (i.e., global test accuracy and local test accuracy), are measured per interval and clients are selected dynamically considering both model accuracy and fairness. Currently, these metrics are measured by local clients to keep privacy and security.

1) Global Test Accuracy: The *global test accuracy* is used to evaluate contributions of clients for achieving high model performance. Scheduler sends small unbiased evaluation dataset (IID) to each participant, and each local participant performs training locally on the evaluation dataset to measure global test accuracy of *its own local model*. To prevent communication overhead, the evaluation dataset is transferred only once at beginning and is reused when calculating the global test accuracy. If some participants show higher global test accuracy than others, we can define these participants as high performance participants. Thus, the global test accuracy on local model shows how well the locally trained model performs compared to other clients.

2) Local Test Accuracy: The *local test accuracy* is used to evaluate biasness of clients for achieving accuracy fairness. If performance of global model is poor on a specific participant, the participant needs to be chosen more frequently for better fairness. However, it is not easy to check fairness of the global model, even the fairness is changing over rounds. To evaluate biasness of each client, local test accuracy of *global model* can be measured on test dataset (non-IID) sampled from local participant data. If global model shows higher local test accuracy on a participant, global model is biased towards the participant. Thus, the local test accuracy of global model shows how much global model is biased towards a client.

3) Ranking Model Performance: For scheduling clients in each round, all clients participating in FL are given ranks on the basis of three metrics, and then three rank vectors are formed. The three rank vectors $\overrightarrow{R}$, $\overrightarrow{Acc_{global}}$, and $\overrightarrow{Acc_{local}}$ represent ranks of all clients on the basis of resources, global test accuracy, and local test accuracy, respectively. The ranks are calculated according to user preferences set via the scheduling knobs. For example, for achieving higher global accuracy, $w_4 > w_1, w_2, w_3$ ranks are given such that clients with higher global accuracy are given higher ranks.

$$\overrightarrow{R} = sort_{asc}\{\forall r \in R\} \tag{5}$$

$$\overrightarrow{Acc_{global}} = sort_{desc}\{\forall A \in Acc_{global}\} \tag{6}$$

$$\overrightarrow{Acc_{local}} = sort_{asc}\{\forall A \in Acc_{local}\} \tag{7}$$

After the calculation of ranks, R_k, B_k, and P_k of the equation 4 can be calculated as below:

$$R_k = \frac{\overrightarrow{R}}{n(n-1)/2} \tag{8}$$

$$P_k = \frac{\overrightarrow{Acc_{global}}}{n(n-1)/2} \tag{9}$$

$$B_k = \frac{\overrightarrow{Acc_{local}}}{n(n-1)/2} \tag{10}$$

In equations 8, 9, and 10, n means the total number of clients. Scores of all possible combinations are calculated using the objective function (Equation 4), and then a subset of clients is selected based on an optimal score among all possible subsets.

D. Resource-fair Scheduling

Resource usage biasness happens when resource heterogeneity exists between participating clients. It leads to waste of resource when grouping clients randomly. To address this problem, resource usage biasness needs to be considered when selecting participants. In our proposed approach, profiling of resource usage is performed for each participant. Then, with availability pattern and resource usage, all possible combinations of participants selection are ranked. The scheduler chooses a combination that requires minimum usage bias (Equation 1). Furthermore, resource usage itself needs to be considered to minimize total resource consumption (Equation 2). Thus, the scheduler chooses a combination according to both resource consumption and resource bias. However, in some cases, a combination might show a little variance between resource usage, while it consumes more resources than other combinations. It means that total resource usage and resource fairness do not lead to the same scheduling decision. To handle such a situation, multi-object optimization for least resource consumption and fair resource allocation is conducted by scheduler using multiple weight priorities (Equation 3).

E. Availability Awareness

Each participant has its own different usage pattern and shows discrepancy in availability over time. Previous approaches choose participants randomly or selectively, but does not focus on availability of each participant. As a result, existing approaches have possibility of dropout issues or resource waste. For example, Google's approach [10] selects more participants than a target number because some of them are dropped due to changing client status. However, this approach leads to waste

of resources because it requests training to participants that will not be aggregated into global model. Unlike most of existing approach, we consider heterogeneous availability to reduce dropout issues. First, availability of each client is profiled during specific times, and collected by profiler. Then, scheduler checks the profiled availability pattern of each participant and selects participants that are available on a per-round basis. This scheduling approach prevents unavailable clients to be chosen and mitigates dropout problems. Our proposed approach selects participants to be available for specific rounds so that resource wastes of participating clients can be prohibited. We assume that availability of most clients shows regular pattern, but if some clients begin to show different patterns, the profiler updates availability of the clients.

F. Priority-based Clients Selection

We design the objective function (Equation 4) where four factors (i.e., resource consumption, resource variance, model performance, and accuracy fairness) are considered to make a scheduling decision. Simply, the goal of scheduling is typically to minimize an output of the objective function (Equation 4). Each factor has different weights (i.e., w_1, w_2, w_3, and w_4) that decide how to prioritize decision making and are set by a user. Before calculating the objective function, clients are sorted according to accuracies (i.e., global test accuracy, local test accuracy), and resource usage (e.g., CPU consumption). All possible combinations are listed based on availability of all clients. Then, an optimal combination per round is chosen through calculation using multiple weights. Additionally, we introduce heuristic to reduce calculation overhead that comes from a large number of combinations. Instead of selecting the target number of participants at the same time, the heuristic divides calculation into several portions. It selects partial number of participants because its possible cases of partial participants are much less than the original combinations. This process is repeated until the number of selected participants matches the target number.

IV. Evaluation

We evaluate our scheduling on a CPU cluster using different scheduling options. We compare our scheduling option with the default random selection. The highlights of our evaluation are as follows:

- *Accuracy* scheduling option improves reached final accuracy by up to 57.4%, compared to the random selection.
- *Fast* scheduling option reduces overall training time by up to 39.5%, compared to the random selection.
- *Fair accuracy* scheduling option shows the lowest variance (1.0%) of clients' local accuracy for achieving accuracy fairness.
- *Fair resource* scheduling option shows the lowest variance (0.045%) of clients' resource consumption for achieving resource fairness.

A. Experimental Setup

1) Testbed: Our testbed is built by deploying 200 clients on a CPU cluster as a proof of concept case study, where each client has its own exclusive CPU core. We implement our scheduling on top of the IBM FL framework [19] version 1.0.4, using Keras [20] and TensorFlow [1] as a back-end library.

2) Model: We use a simple CNN [21] model for evaluating our proposed scheduling algorithm. The CNN model consists of 3 layers: a 5x5 convolution layers with 32 channels and ReLu activation followed by a MaxPooling layer of size 2x2, a 5x5 convolution layers with 64 channels and ReLu activation followed by a MaxPooling layer of size 2x2, and one fully-connected layer with 2,048 units and ReLu activation.

3) Datasets: We use three image classification datasets for evaluating our scheduling algorithm: FEMNIST [22], CIFAR10 [23], and MNIST [24].

- FEMNIST: FEMNIST dataset is a handwritten image dataset of alphabet letters and numbers, which are 28x28 gray-scale images consisting of 62 different classes.
- CIFAR10: CIFAR10 dataset contains 60,000 RGB images, which are 32x32 resolution images of 10 classes. Each class corresponds to different categories of objects.
- MNIST: MNIST is composed of handwritten digit images of 10 classes (a digit from 0 to 9).

4) Training hyperparameters: SGD [25] is used as the local optimizer for training local data in each local device. As the default training parameters of the LEAF [22] framework, initial learning late and batch size are set to 0.004 and 10, respectively. We train a CNN model on FEMNIST dataset for 500 rounds, while we train the model on CIFAR and MNIST dataset for 100 rounds because accuracy saturated at around 100 rounds. In each round, 10 clients are selected among 200 clients to train the CNN model and sends updated weights to an aggregator. We set the number of local epochs to 1.

5) Heterogeneous setup: We split all clients into 6 groups with different percentages. As the previous work [7], we configure different CPU frequency and allocate different numbers of CPUs to each group as Table II. The percentage of two lower groups (i.e., E, F) is 70 % in total, because most of participating devices are very slow in real-world FL scenarios. For heterogeneous data distribution, we use the non-IID distribution of FEMNIST dataset in LEAF. In a similar way, we make the non-IID training dataset of CIFAR10. For evaluating highly heterogeneous distribution of MNIST dataset, we divide the MNIST dataset unevenly and limit the number of classes to either 2 or 3 classes. Each client has either 2 or 3 classes, instead of including total 10 classes of MNIST. For

TABLE II: Heterogeneous resource setup.

Group	CPU per client	CPU frequency (GHz)	Clients
A	1.0	3.2	20
B	1.0	2.5	20
C	1.0	2.3	10
D	1.0	2.0	10
E	0.8	1.4	70
F	0.4	1.4	70

all three datasets, we set the number of data points to 200 per client.

6) Scheduling policies: We evaluate four different scheduling policies of our proposed heterogeneity-aware adaptive scheduling approach: (1) *fast*, (2) *fair resource*, (3) *fair accuracy*, and (4) *accuracy*. These policies are defined by selection weights ratios as Table III. Each weight prioritizes training throughput, resource fairness, accuracy fairness, and model performance, respectively. We name the default random selection as *vanilla* that has been used by the default FedAvg [4]. *Fast* policy selects only fastest clients in each round according to measurement of training latency. *Fair resource* policy considers variance of resource usages (i.e., CPU consumption per round) and selects clients that consume similar CPU resources. *Fair accuracy* policy minimizes difference of model performances among all participated clients, which provides fair model performance. *Accuracy* policy concerns only model performance, which selects clients showing high accuracy results.

TABLE III: Scheduling policy configurations.

Policy	Selection weights ratio			
	W 1	W 2	W 3	W 4
vanilla	N/A	N/A	N/A	N/A
fast	1.0	0.0	0.0	0.0
fair resource	0.0	1.0	0.0	0.0
fair accuracy	0.0	0.0	1.0	0.0
accuracy	0.0	0.0	0.0	1.0

7) Metrics: We use a variety of metrics to evaluate our scheduling approach.

Performance. To evaluate performance improvement, we measure total training time and final model accuracy, which are the same metrics used by other studies [7], [15]. We measure the overall training completion time to finish federated training to reach a predefined target round. After the training is completed, we measure both the final model accuracy of global model and local accuracy of each client.

Accuracy fairness. For accuracy fairness, we measure averaged variance of each client's local accuracy. This metric suggests that how much the trained model is biased to clients. Thus, low variance means the trained model provides better accuracy fairness.

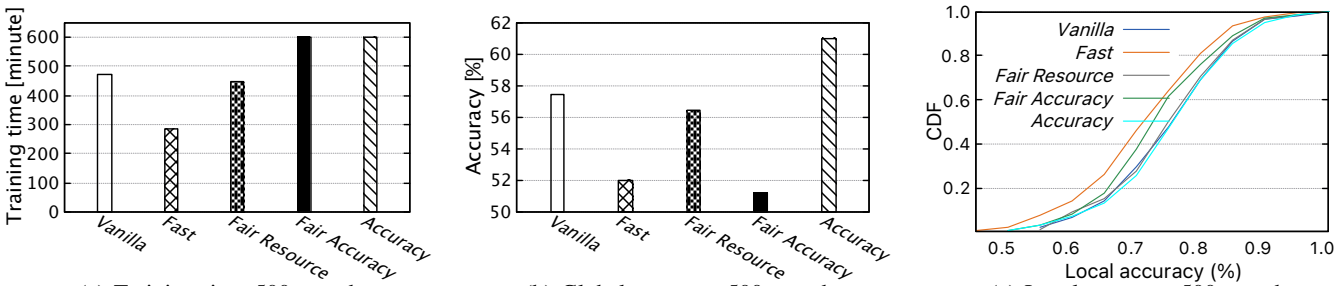
Resource fairness. For resource fairness, our target metric is resource variance that is measured by the variance of each client's CPU resource consumption. The each client's CPU resource consumption can be calculated as: total number of cores $\times$ core frequency. When the scheduler chooses clients that show similar resource usage patterns, difference of training time between clients is reduced, improving fairness of resource consumption. We also measure total CPU consumption, the total sum of each client's CPU resource consumption. It shows how much resource-fair scheduling prevents a waste of resource.

B. Experimental Results

In this subsection, we evaluate and demonstrate the efficiency of our proposed scheduling method. The performance of the proposed approach is tested against the default FedAvg [4] that is one of the most widely used FL algorithms [26]. The FedAvg randomly selects clients without considering either heterogeneity or availability.

1) Overall performance: We evaluate performance of proposed scheduling approach in terms of training time and model accuracy in environments with resource and data heterogeneity. Figure 2a shows overall training time of each scheduling policy. As we expected, *fast* takes shorter training time (285 minutes) than other options (446$\sim$600 minutes), achieving an improvement of 39.5%, compared to *vanilla*. *Vanilla* selects clients randomly, whereas *fast* chooses fast clients on the purpose. On the contrary, *fair accuracy* and *accuracy* policies take longer times than *vanilla* and *fast*. This is because it is possible that these policies (i.e., *fair accuracy* and *accuracy*) select slower clients for either accuracy fairness or model performance, regardless of training throughput. We evaluate the final model accuracy of 500-round training. We measure two different accuracy types: (1) global accuracy, and (2) local accuracy. For global accuracy, aggregator evaluates global model using test data set located on aggregator. For local accuracy, each client evaluates global model using its own local test data and sends its accuracy result to aggregator. In terms of global accuracy, figure 2b shows that *accuracy* policy reaches the highest (61.0%). *Accuracy* shows 3.6% higher accuracy than *vanilla*. *Fast* and *fair accuracy* options show low accuracy results (i.e., 52.0% and 51.2%). These options select fast clients and low-accuracy clients, respectively, without considering model accuracy. According to our measurement of local accuracy, *accuracy* policy reaches the highest of averaged local accuracy (77.4%), while *fast* reaches the lowest of averaged local accuracy (73.5%). As can be seen in Figure 2c, in *accuracy* case, there are more clients showing lower local accuracy (50%$\sim$55%), compared to *fair accuracy* policy. This is because these low performance clients are selected less frequently than high performance clients. It means that local datasets of these low performance clients are less contributed to training the global model. As we expected, *fair accuracy* policy shows more fair distribution compared to other policies, while showing lower averaged local accuracy (75.6%) than *accuracy* policy (77.4%). *Fast* policy shows limited improvement (73.5%) among the policies because it does not consider either model performance or bias. Unlike *vanilla*, proposed scheduling approach considers availability and schedules clients to prevent drop-outs. Thus, it helps achieving higher accuracy than *vanilla*'s random selection. Evaluation on availability awareness is shown in Subsection IV-B4.

2) Accuracy fairness: FL trains model using non-IID data tightly coupled with each client. Trained model cannot guarantee fair performance to all participating clients. It is possible that trained model shows biased accuracy when evaluating the model on local dataset of each client. Thus, it is one of important factors to provide stable model accuracy without bias. Figure 3 shows that *fair accuracy* option provides lowest variance (1.0%). The *fair accuracy* policy aims to provide bias-resistant training where bias is measured by accuracy of global model on local test data (i.e., local test accuracy). On the other hand, *fast* and *accuracy* options show high variance



(a) Training time 500 rounds (b) Global accuracy 500 rounds (c) Local accuracy 500 rounds

Fig. 2: Comparison results for different scheduling policies on FEMNIST with resource and data heterogeneity.

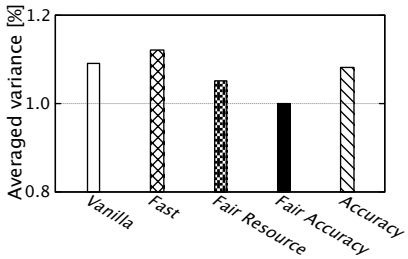
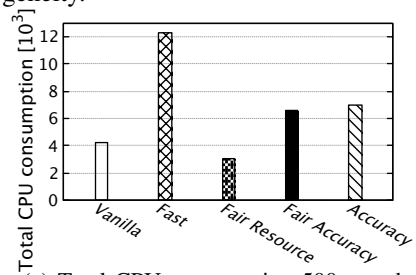
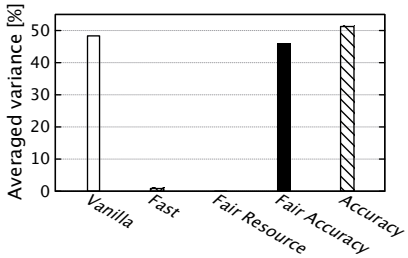


Fig. 3: Averaged variance of accuracy at 200 rounds for different scheduling policies on FEMNIST with resource and data heterogeneity.



(a) Total CPU consumption 500 rounds



(b) Averaged variance 500 rounds

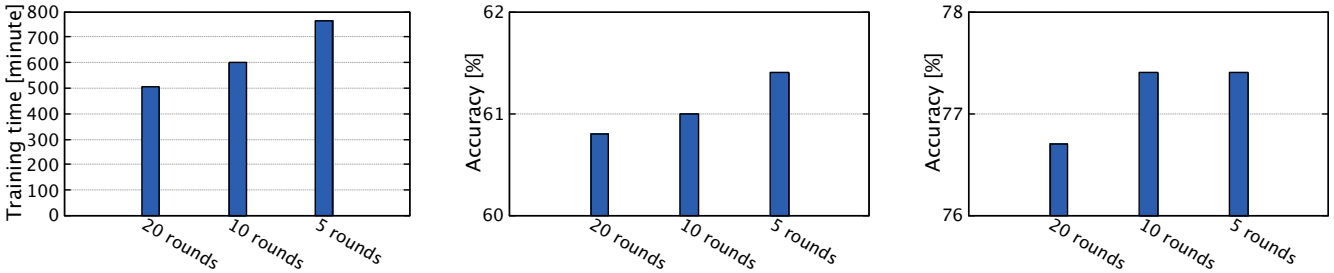
Fig. 4: CPU resource consumption for different scheduling policies on FEMNIST with resource and data heterogeneity.

because these options consider performance of throughput and accuracy, instead of fairness. Scheduling decision for reducing bias is adaptively changed per specific rounds according to the measurement of local test accuracy.

3) Resource fairness: We evaluate how resource fairness is affected by different scheduling polices. We first measure total CPU consumption of 500 rounds training (Figure 4a). As expected, *fair resource* consumes lower CPU resources than *vanilla* because it selects clients with similar patterns of CPU resource consumption. Training latency of each client is similar and it minimizes idle status of each client. In the case of random selection, fast clients should wait until slow devices are finished, which leads to unnecessary resource waste. To support the above result, we also check variance of CPU resources (Figure 4b). *Fair resource* shows lowest variance (0.045%), while *vanilla*, *fair accuracy*, and *accuracy* show much higher variance, such as 48.4%, 45.7%, and 51.3%, respectively. *Fast* shows higher variance (0.77%) than *fair resource*, but it also shows much lower variance, compared to *vanilla*. This is because *fast* chooses fast clients with high CPU consumption without selecting slow clients. As a result, it reduces variance of resource consumption between clients because it selects clients that shows similar resource usage patterns (i.e., fast devices). On the other hand, *vanilla* selects clients that show different resource usage patterns (i.e., fast and slow devices) at the same round, which shows high variance.

4) Availability awareness: As we mentioned in Section III-E, our scheduling design supports availability-aware participants selection. We evaluate our proposed availability-aware mechanism by comparing final model accuracy of the baseline and our availability-aware scheduling cases. Figure 6 shows the result of averaged local accuracy. While vanilla shows 71.9% of local accuracy, our proposed availability-aware scheduling shows 76.3% of local accuracy, resulting in 6.1% of improvement. The vanilla scheduling does not consider states of participants, and thus some of participants are absent during training, leading to accuracy drop. On the other hand, our availability-aware approach does selects participants that are highly available during training rounds.

5) Interval and training time: Next, we conduct experiment of *accuracy* scheduling option to evaluate the trade-off between training performance and scheduling interval (Figure 5). As we expected, training time is linearly increasing with the increase of intervals. In the Figure 5a, training times of 20-round, 10-round, and 5-round intervals are 504, 600, and 764 minutes, respectively. In the Figures 5b and 5c, global accuracy and local accuracy of 20-round interval are 60.8% and 76.7%, respectively, whereas global accuracy and local accuracy of 5-round interval are 61.4% and 77.4%, respectively. Thus, frequent scheduling interval leads to longer training time, while it improves global accuracy and averaged local accuracy. However, averaged local accuracy is saturated at interval value of 10 rounds.



(a) Training time (b) Global Accuracy (c) Averaged Local Accuracy

Fig. 5: Comparison results for different intervals of accuracy option on FEMNIST.

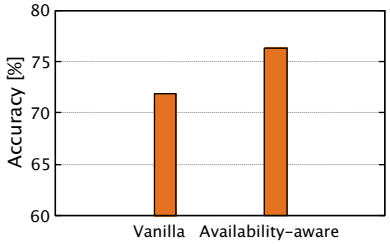


Fig. 6: Averaged local accuracy at 500 rounds for vanilla and availability-aware scheduling on FEMNIST.

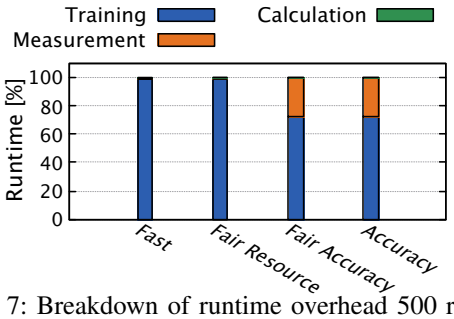


Fig. 7: Breakdown of runtime overhead 500 rounds.

6) Runtime overhead: We measure runtime routines such as scheduling calculation, measurement overhead of training latency and accuracy, and training computation. Figure 7 shows runtime overhead of each policy. Scheduling calculation is lower than 1% (0.4%∼0.9%). In cases of *fast* and *fair resource*, measurement overhead is very low (i.e., 0.6% and 0.9%) because it only measures training latency at the first round, not repeating measurement of accuracy per interval. On the other hand, *fair accuracy* and *accuracy* policies repeat measurement of accuracy per interval for adaptively scheduling clients selection based on accuracy information. As a result, it leads to overhead of 27.8%.

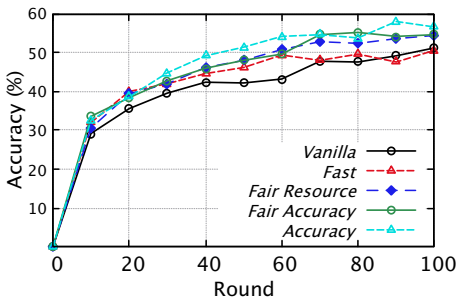


Fig. 8: Global accuracy at 100 rounds for different scheduling policies on CIFAR10.

7) Comparison for CIFAR10 dataset: We measure model accuracy using CIFAR10 dataset to check if the proposed scheduling approach can be applied to other datasets. Figure 8 shows global accuracy results of each scheduling policy. Compared to the result of FEMNIST, CIFAR10 also shows similar accuracy trends. As expected, *accuracy* option shows the best (56.6%), providing an improvement of 10.5%, compared to *vanilla* option. On the other hand, *Fast* option shows lower accuracy (50.6%) than other options. Compared to *vanilla* case, the *fast* option shows higher accuracy at early rounds, but it saturates because *fast* does not consider data quality of clients.

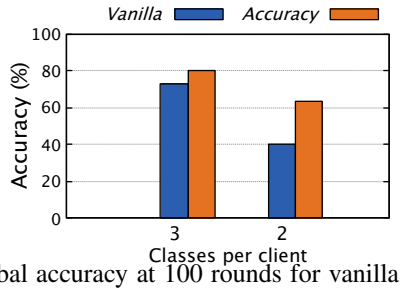


Fig. 9: Global accuracy at 100 rounds for vanilla and *accuracy* option on MNIST.

8) Data heterogeneity: To evaluate the effect of the proposed *accuracy* option on data heterogeneity, we measure global accuracy when data distribution is highly heterogeneous. For varying data heterogeneity, we use MNIST dataset as test dataset and vary the number of classes per each client between 2 and 3 classes, instead of 10 classes. Figure 9 shows the result. In the case of *vanilla*, global accuracy of 2 classes is much lower than global accuracy of 3 classes. This is because when data is more heterogeneous, it causes accuracy drop [7]. In both cases, *accuracy* option improves global model accuracy by 9.3% and 57.4%, respectively, compared with *vanilla*. Therefore, *accuracy* scheduling option is more effective on high heterogeneous data.

V. Related Work

There already exists a plethora of work on mitigating the effect of heterogeneity for federated learning (FL). FedCS [9] addresses straggler issues by dropping slow devices. [10] proposes large-scaled FL systems, which chooses more clients and throws out slow ones. However, these approaches lead to bias on achieved accuracy because information of slow devices is missed. Additionally, dropping devices makes wastes of resources. Some research starts to study on effects

of heterogeneity on FL. FedProx [16] addresses resource heterogeneity by allowing partial results, instead of dropping slow devices. TiFL [7] minimizes straggler effects by tiering parties based on training latency. Based on estimation models, [27] adjusts the number of local epochs that is globally applied to all parties. Oort [15] selects clients based on criteria such as training latency and importance sampling. [12] argues that different users' behavior leads to unreliable drop-out of devices. However, none of the existing works suggests a heterogeneity-aware adaptive method where scheduling policy is configured by scheduling knobs for achieving various FL goals, as well as considering availability of each device, resource heterogeneity, and data heterogeneity. Unlike the existing approach, our scheduling approach considers fairness of resource and accuracy for scheduling clients in FL systems. Also, our proposed approach addresses model bias issues by dynamically adjusting clients selection based on profiling measurement of local accuracy. Thus, our work is the first effort to handle heterogeneity in resource and data of FL by adaptively selecting clients while scheduling parties according to weighted scheduling options.

VI. Conclusion

In this paper, we have proposed a heterogeneity-aware adaptive scheduling algorithm that provides various scheduling policies to meet various performance and fairness objectives while dynamically addressing heterogeneity in both resource and data for efficient and fair FL systems. Unlike existing random selection leading to drop-off or straggler issues, our proposed approach minimizes drop-off of devices per round based on client availability, and chooses subset of devices for achieving different scheduling goals such as resource efficiency, resource fairness, accuracy fairness, and model performance. We introduce how to check accuracy performance and accuracy biasness among registered devices, which measures global test accuracy and local test accuracy, each meaning contribution of each local device and biasness against each local data, respectively. For achieving either resource efficiency or resource fairness, our scheduling algorithm selects subset of devices using resource usage patterns. For achieving either accuracy fairness or model performance, our proposed approach adaptively and dynamically schedules participating devices based on measurement of accuracy performance and accuracy biasness. The result shows that our approach achieves an improvement over existing random client selection by 57.4% in reached accuracy and by up to 39.5% in training time, according to scheduling policies.

Acknowledgment

This work is sponsored in part by the NSF under the grants: CSR-2106634, CCF-1919113, OAC-2004751, and CSR-1838271.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. USENIX OSDI*, 2016.

[2] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *Proc. USENIX OSDI*, 2014.

[3] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *NIPS*, vol. 25, pp. 1223–1231, 2012.

[4] 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*, 2017, pp. 1273–1282.

[5] G. D. P. Regulation, "General data protection regulation (gdpr)," *Intersoft Consulting, Accessed in October*, vol. 24, no. 1, 2018.

[6] A. Act, "Health insurance portability and accountability act of 1996," *Public law*, vol. 104, p. 191, 1996.

[7] Z. Chai, A. Ali, S. Zawad, S. Truex, A. Anwar, N. Baracaldo, Y. Zhou, H. Ludwig, F. Yan, and Y. Cheng, "Tifl: A tier-based federated learning system," in *Proc. ACM HPDC*, 2020.

[8] Z. Chai, Y. Chen, A. Anwar, L. Zhao, Y. Cheng, and H. Rangwala, "Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–16.

[9] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE ICC*, 2019.

[10] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[11] 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," *arXiv preprint arXiv:1912.04977*, 2019.

[12] C. Yang, Q. Wang, M. Xu, S. Wang, K. Bian, and X. Liu, "Heterogeneity-aware federated learning," *arXiv e-prints*, pp. arXiv–2006, 2020.

[13] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang, "On the convergence of fedavg on non-iid data," in *International Conference on Learning Representations*, 2020.

[14] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[15] F. Lai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Oort: Efficient federated learning via guided participant selection," in *Proc. USENIX OSDI*, 2021.

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

[17] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *Proc. ICML*, 2019.

[18] J. Han, M. M. Rafique, L. Xu, A. R. Butt, S.-H. Lim, and S. S. Vazhkudai, "Marble: A multi-gpu aware job scheduler for deep learning on hpc systems," in *Proc. IEEE/ACM CCGRID*, 2020.

[19] H. Ludwig, N. Baracaldo, G. Thomas, Y. Zhou, A. Anwar, S. Rajamoni, Y. Ong, J. Radhakrishnan, A. Verma, M. Sinn *et al.*, "Ibm federated learning: an enterprise framework white paper v0. 1," *arXiv preprint arXiv:2007.10987*, 2020.

[20] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[21] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *NIPS*, vol. 2, 1989.

[22] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[23] G. H. Alex Krizhevsky, Vinod Nair, "CIFAR-10 dataset," *https://www.cs.toronto.edu/ kriz/cifar.html*, 2014.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[26] J. Yao, Z. Dou, and J.-R. Wen, "Fedps: A privacy protection enhanced personalized search framework," in *Proceedings of the Web Conference*, 2021, pp. 3757–3766.

[27] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.