HACCS: Heterogeneity-Aware Clustered Client Selection for Accelerated Federated Learning

Joel Wolfrath, Nikhil Sreekumar, Dhruv Kumar, Yuanli Wang, and Abhishek Chandra
Department of Computer Science and Engineering
University of Minnesota, Minneapolis, USA
Email: {wolfr046, sreek012, dhruv, wang8662, chandra}@umn.edu

***Abstract*—Federated Learning is a machine learning paradigm where a global model is trained in-situ across a large number of distributed edge devices. While this technique avoids the cost of transferring data to a central location and achieves a strong degree of privacy, it presents additional challenges due to the heterogeneous hardware resources available for training. Furthermore, data is not independent and identically distributed (IID) across all edge devices, resulting in statistical heterogeneity across devices. Due to these constraints, client selection strategies play an important role for timely convergence during model training. Existing strategies ensure that each individual device is included, at least periodically, in the training process. In this work, we propose HACCS, a Heterogeneity-Aware Clustered Client Selection system that identifies and exploits the statistical heterogeneity by representing all distinguishable data distributions instead of individual devices in the training process. HACCS is robust to individual device dropout, provided other devices in the system have similar data distributions. We propose privacy-preserving methods for estimating these client distributions and clustering them. We also propose strategies for leveraging these clusters to make scheduling decisions in a federated learning system. Our evaluation on real-world datasets suggests that our framework can provide 18%-38% reduction in time to convergence compared to the state of the art without any compromise in accuracy.**

***Index Terms*—Federated Learning, Non-IID data, Clustering, Scheduling**

I. Introduction

The proliferation of IoT sensors, smart phones, tablets, and laptops has led to vast amounts of data being generated at the edge of the network. This data can be utilized to perform a wide variety of analytics such as SQL-based analytics, graph analytics and machine learning based analytics. Given the expense and scarcity associated with transferring data over the wide area network (WAN), it is often infeasible to transfer all of this data to a centralized location for analysis or persistence. Furthermore, transferring this data may have privacy implications for the user or even the application owner, due to legal requirements such as the General Data Protection Regulation (GDPR) [1].

Federated learning (FL) is a technique designed to address these challenges for machine learning analytics. It performs model training in-situ across a large number of distributed devices. In doing so, it utilizes the computational resources available at each device and avoids the cost and privacy implications associated with data transfers to a central location. FL techniques have been successfully applied to many modeling problems, especially in applications for object detection [2], search suggestion [3], and healthcare [4]. Models are usually trained via *Federated Averaging* (or its variants) [5]. In each training epoch, a global set of model parameters are pushed to each participating client and updated using local data. Clients forward their model updates to a central server, where they are averaged with the updates received from other devices.

FL has two key challenges: (1) **System heterogeneity:** FL is performed on devices which may differ in processing capacity, network bandwidth, and power. Additionally, devices can join or leave the network during training. (2) **Statistical heterogeneity:** Data residing on participating devices may have different probability distributions. This violates the well-known IID assumption in machine learning, which assumes that all training samples are *independent* and come from the same joint probability distribution (*identically distributed*).

The total number of devices participating in federated learning can be in the millions [6]. Since it is not possible for all devices to be available for training simultaneously, federated learning selects a small subset of these devices in each training epoch. System heterogeneity and data heterogeneity make device selection crucial for timely model convergence and sufficient model accuracy [7], [8]. Existing device selection strategies have mostly focus on incorporating system heterogeneity during training. Statistical heterogeneity has been considered [9], [10], *but only to the degree that it is captured by the loss function used by model training*. In this work, we propose HACCS, a **H**eterogeneity-**A**ware **C**lustered **C**lient **S**election system which attempts to leverage the actual data distribution present on the client devices to mitigate statistical heterogeneity. We propose techniques for identifying client devices with similar data distributions, while still providing reasonable guarantees on user privacy. This involves communicating privacy preserving data summaries to the central server. The central server can then cluster similar devices and select clients in a way that ensures representation from each data distribution rather than ensuring representation of each device. Ensuring that each unique data distribution is represented during training, rather than every individual device, increases robustness during model training in a federated setting.

Research Contributions. We make the following research contributions in this work:

- We identify privacy-preserving summaries which can be

used to cluster client devices based on the discernible differences in their data distributions.
- We develop a scheduling algorithm that focuses on scheduling clusters rather than individual devices. The scheduling algorithm attempts to leverage system heterogeneity as well.
- We evaluate HACCS, our proposed framework, using real-world datasets. Our evaluation suggests that HACCS can provide 18%-38% reduction in model training time in comparison to the state of the art.

II. Background

A. System Model

We consider a two-tier topology comprising multiple client devices connected to a central server. The client devices communicate with the central server via the wide area network (WAN). The data is continuously or periodically generated at each client device. During training, each participating client device computes a locally optimal model update and propagates it to the central server. The central server systematically combines the updates from all the devices to compute a global update. This global update is then communicated back to the participating devices to further improve their local models. This sharing of model updates between central server and client devices happens over multiple iterations until model convergence is achieved.

B. Challenges in Federated Learning

Federated learning has two key challenges:

1) System Heterogeneity: In general, variation in system resources can have a large effect on training performance. This also occurs in general distributed machine learning environments, and is not unique to federated learning [11]. For example, differences in computational capacity and network connectivity can have a large impact on performance, even within a single data center. However, the problem is exacerbated in a federated environment due to further disparities in computation, memory, and bandwidth. Additional challenges are posed due to energy constraints and the possibility of devices dynamically joining or leaving the system [12]. Another factor is the amount of data present on each device; some devices may have much more data available for training compared to their peers [9]. These differences highlight the importance of client selection and workload distribution to mitigate the straggler effect in federated settings.

2) Statistical Heterogeneity: Machine learning modeling commonly requires that the input data points are independent and identically distributed (IID), which may or may not hold in practice. For example, independence violations may occur when spatial or temporal dependencies exist in the data [13]. When devices are geographically distributed, they may follow unique probability distributions depending on their location. This would also violate the IID assumption. For example, the distribution of alphanumeric characters that are used on a mobile phone will vary heavily by geographical region. The distribution of transportation vehicles and diversity of plant and animal species will also change based on location. This kind of skewed data can have a substantial, negative impact on model training and performance. Furthermore, higher degrees of skew often correspond to increased degradation in the system [14].

C. Problem Statement

Our objective is to minimize the training time of a global federated machine learning model, in the presence of system and statistical heterogeneity. More specifically, we are required to train a supervised classification[1] model across a set of n distributed devices, with local training data $\mathcal{Z}_i$ at device i defined as:

$$\mathcal{Z}_i = \{(X_{i,1}, y_{i,1}),\ (X_{i,2}, y_{i,2}),\ \dots,\ (X_{i,m_i}, y_{i,m_i})\} \quad (1)$$

where X_i is the matrix of input feature vectors and y_i is the associated vector of class labels. We are tasked with identifying a client selection strategy which minimizes the time to convergence during model training. The convergence must be with respect to all devices in the system, not just the devices selected for training in each epoch, that is, the loss function must be evaluated over $\mathcal{Z}_i$ for all $i \in \{1,\ \dots,\ n\}$.

III. Impact of Statistical Heterogeneity

Statistical heterogeneity in FL implies the local data at each client may follow a different probability distribution. Our hypothesis is that aggregating information about the client data distributions can help improve scheduling decisions and reduce training time. For example, if the local data on two client devices, C_1 and C_2, follow a *sufficiently similar* probability distribution, we may choose to perform model training on only one of the two clients, while still getting a good representation of the other client's data distribution in the training process. If C_1 has more computational power, or better network connectivity than C_2, we may prefer to use C_1 during most of the training process. Furthermore, if C_1 drops out of the system in the middle of training, we can ensure C_2 is included during those epochs, so that their data distribution is always represented.

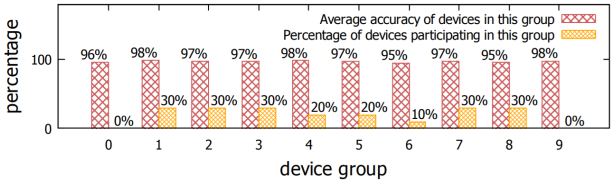
To evaluate these hypotheses, we conduct an initial experiment to determine the impact of distribution representation on training performance using the MNIST dataset [15] and the LEAF framework [16]. The dataset consists of 28x28 images of hand-written numbers, labeled from [0-9]. We use the same convolutional neural network model that is presented in LEAF. The overall accuracy is the average test accuracy on all devices across 1000 epochs. We simulated 100 clients and selected 20 of them for each training epoch. We adopt the setting from Zhao et al. [17] to partition 100 clients into 10 groups. Each group contains ten clients and is assigned two classes. We ensure that devices in a group will only have training data from the two classes assigned to the group (see Table I).
We implement two dropping policies: 1) randomly pre-select some clients to drop 2) pre-select an entire group of devices

[1]Our technique is also applicable to supervised regression models, without loss of generality.

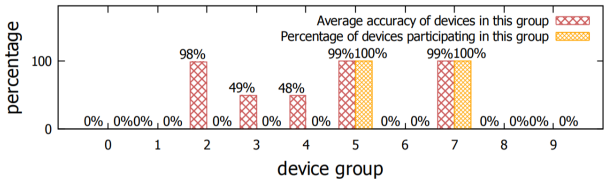
TABLE I: Partition of training data on 100 devices

Device Group No.	0	1	2	3	4
Classes	6,7	1,4	5,9	2,3	0,4
Device Group No.	5	6	7	8	9
Classes	2,5	6,8	0,9	7,8	1,3

to drop. For each policy, we drop 80 out of 100 devices and measure the trained global model's accuracy on the local test dataset of each device. The results are shown in Fig. 1.



(a) Devices are randomly dropped permanently



(b) Drop entire groups of devices permanently

Fig. 1: Dropout experiment with skewed labels

In Fig. 1a there is no drop in accuracy for any group, since there are always other clients available to represent the missing labels. We conclude that if some clients from each group participate in training, the accuracy of the group as a whole will not be impacted. In Fig. 1b we see that the groups which have been dropped completely experience a significant drop in accuracy. The accuracy drop for dropped groups that have some of their class labels in participating groups is less than the groups whose class labels are not present in any of the participating groups. These results show that the training accuracy depends on capturing all the unique data distributions rather than all the clients. Based on these insights, we propose a scheduling framework which attempts to identify data distributions on devices and ensure representation from each data distribution during model training.

IV. Proposed Approach: HACCS

Our proposed framework requires the scheduler to distinguish different distributions across clients and identify similarities. At the beginning of training, clients can send a compact summary of their local data to a central server, where it is used to identify clients with similar data and make scheduling decisions. Figure 2 shows an overview of HACCS. In the beginning, a client joins the system (❶) and sends a summary of its local data to the central server (❷). Once the central server is ready to begin model training, it compares the client summaries and clusters them to identify groups with similar

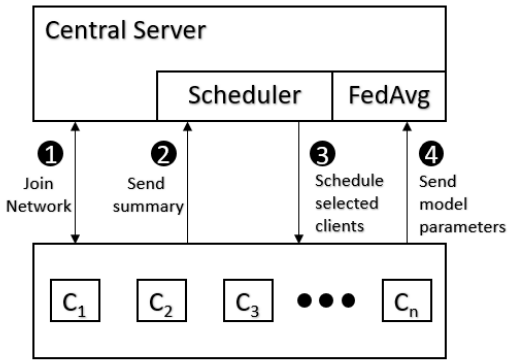


Fig. 2: Our proposed system.

data. These clusters are then used to schedule clients for training each epoch, e.g. by selecting the fastest devices within each cluster for training (❸). The selected devices perform model training and send their updates back to the central server to generate the next global model via federated averaging (❹). Given this proposal, we now explore methods for summarizing distributions (§IV-A and §IV-B), clustering them (§IV-C), and using the information for scheduling (§IV-D).

A. Identifying Similar Data Distributions

A clustered federated learning system must define and implement the following components:

1. A function $\mathcal{S}(\mathcal{Z}_i)$ which is run on each client device. This function takes the local dataset $\mathcal{Z}_i$ as input and produces a summary or distribution over the dataset, which will be sent to the central server.
2. A distance function $d(\mathcal{S}(\mathcal{Z}_a),\ \mathcal{S}(\mathcal{Z}_b))$ which computes how different the summaries of $\mathcal{Z}_a$ and $\mathcal{Z}_b$ are.

The central server is then able to perform clustering of clients based on the computed pairwise distances. The clustering quality will depend on the selections of $\mathcal{S}$ and d, while the degree of privacy depends solely on the choice of summary, $\mathcal{S}$. Our framework proceeds by having each client device compute $\mathcal{S}(\mathcal{Z}_i)$ and send it to the central server.

The distribution summaries we consider are motivated by techniques for identifying IID violations. Standard distributed machine learning models assume that the response variables y_i and the predictor variables X_i at each device are drawn IID from a shared joint distribution, $P(X, y)$. This joint distribution can be factored as follows:

$$P(X, y) = P(y)\ P(X \mid y) \tag{2}$$

Therefore, if $P(y)$ (the marginal distribution of the response labels) or $P(X \mid y)$ (the data distribution conditioned on the response) differs at any device, we have a violation of the IID assumption. When IID violations occur, it follows that *one of these distributions must differ across one or more devices* [18]. Based on this insight, we propose $P(y)$ and $P(X \mid y)$ as distribution summaries for our framework.

Response Distribution. For our first summary, we consider the distribution of response labels, $P(y)$. For instance, in an object detection application, this would be the distribution of labels attached to each image, e.g. "cat" or "dog". We use a histogram representation of $P(y)$ as our choice for $\mathcal{S}$. To perform clustering, we need a mechanism for measuring the distance between these histograms. We define our distance function d to be the *Hellinger distance* [19], given by:

$$H(\mathcal{S}(\mathcal{Z}_a),\ \mathcal{S}(\mathcal{Z}_b)) = \frac{1}{\sqrt{2}}\|\sqrt{\mathcal{S}(\mathcal{Z}_a)} - \sqrt{\mathcal{S}(\mathcal{Z}_b)}\|_2 \quad (3)$$

This distance function is useful for measuring the distance between histograms, since it can tolerate zero entries. It also produces a nice bounded output over our inputs, more specifically:

$$0 \leq H(\mathcal{S}(\mathcal{Z}_a),\ \mathcal{S}(\mathcal{Z}_b)) \leq 1 \quad (4)$$

If $c_i < \infty$ is the number of class labels that exist on a given device, then the data size required to send this histogram summary over the network is $\Theta(c_i)$.

Conditional Data Distribution. For the $P(X \mid y)$ summary, we need to use a set of histograms as our choice for $\mathcal{S}$, one for each unique class label c_i. Taking the example of the object detection application again, this would be the distribution of pixels associated with each response label. Therefore, the data size required to send this summary over the network is $\Theta(c_i p)$, where p is the number of bins in the histogram used to estimate the local X_i values. For the distance function, we use the *average* Hellinger distance between the two sets of histograms. Clearly, the $P(X \mid y)$ summaries are more expensive than the $P(y)$ summaries to send over the network. However, both summaries have the added benefit of being relatively consistent over time (assuming data isn't being generated very rapidly).

Gradients of the loss function or model weights could also be leveraged as a summary in our framework. The intuition is that after each epoch, some devices may have gradients that point in similar directions. However, these patterns may change rapidly, since gradients change every training epoch. This requires that summaries be communicated to the central server frequently and clustering be performed each epoch, which may not be optimal in practice.

B. Privacy Considerations

Federated learning frameworks ensure the privacy of user data, to the degree that model updates do not provide information about the data distribution [5]. Aggregating summaries of local data has the potential to weaken these privacy guarantees. To address this issue, we utilize techniques from *differential privacy* to provide stronger guarantees with respect to the privacy of each user. This allows us to explicitly quantify the degree to which a user's privacy is violated by our distribution summary.

As discussed in section IV-A, we use histograms to represent distributions in our framework. Since histograms have the potential to leak information about the client's data, we need to apply differential privacy techniques to ensure they meet the privacy requirements of the application. More formally, given a tolerable privacy loss ϵ, our histograms satisfy (ϵ, 0)-differential privacy if for each histogram bin, we add a random noise value λ_i drawn independently from a Laplace$(0, 1/\epsilon)$ distribution. This technique is known as the *Laplace mechanism* [20]. The second parameter for the Laplace distribution controls the variability, given by:

$$\text{Var}[\lambda_i] = 2\left(\frac{1}{\epsilon}\right)^2 \quad (5)$$

Therefore, smaller values of the privacy loss ϵ correspond to a larger variance in the noise distribution. This increases the difficulty of identifying the true distribution that was originally represented by the histogram, as is illustrated in figure 3. Thus, smaller values of the privacy loss provide stronger privacy but reduce accuracy of identifying similar distributions, establishing a fundamental trade-off between privacy and accuracy. We explore this trade-off in section V.

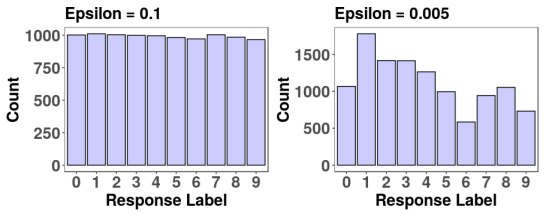


Fig. 3: Example histograms for a client with 1000 training points for each of 10 labels, using ϵ = 0.1 and ϵ = 0.005.

C. Clustering Mechanisms

There are many different clustering techniques in the literature which could be used for our proposed data summaries. For our implementation of HACCS, we use density-based algorithms, such as DBSCAN [21] and OPTICS [22], to identify similarities between clients. There are some key properties of these algorithms that make it useful for our application [23], including:

- The ability to identify clusters of arbitrary shape
- A relatively low number of hyperparameters to specify
- The ability to classify points as noise, rather than forcing them to be assigned a cluster. This is important, since our scheduling algorithm assumes a good degree of statistical similarity between clients in the same cluster.

We selected the OPTICS algorithm for our evaluation, since it satisfies these properties and has one less hyperparameter compared to DBSCAN. In practice, the data distribution at a given client device could change over time. While it seems unlikely that the distribution could change substantially during model training, our framework can adapt in real time to shifts in data distribution. If clients are allowed to asynchronously send updated data summaries to the central node, new cluster assignments could be generated while training is in progress. The same technique can be applied for devices joining the system during model training, which allows our framework to adapt to new information in real time.

D. Scheduling Policies

Given a set of client devices and their corresponding cluster assignments, we are required to schedule a subset of clients in each epoch during training. Assuming our clusters capture some of the statistical heterogeneity, our scheduler now must account for the difference in training loss across the devices, in addition to any system heterogeneity that exists. We propose a scheduling algorithm which leverages weighted random sampling of our identified clusters. We employ a selection strategy loosely based on the algorithm proposed by Xu et. al. [24] which systematically selects clients with higher loss and lower training latency (or time to completion). We adapt this technique to sample clusters rather than individual devices, using the average loss and latency in each cluster as inputs. We define the latency for a given client to be the expected time required to transfer the model parameters to and from the client, plus the time required to perform a single epoch.

Once a cluster has been sampled, we then select devices within a cluster based primarily on their estimated latency, since the devices within each cluster should have similar data distributions. Let ACL_i be the average loss per client in cluster i and define $\tau_i \in [0, 1]$ be the expected reduction in latency:

$$\tau_i = 1 - \frac{\text{Latency}_i}{\text{Latency}_{max}} \tag{6}$$

where Latency_{max} is the maximum latency across all clusters. Then, we are assigning a sampling weight θ_i to each cluster, which is a convex combination of the reduction in latency and the (normalized) average cluster loss:

$$\theta_i = \rho\tau_i + (1 - \rho)\frac{\text{ACL}_i}{\sum_{j=1}^{k} \text{ACL}_j} \tag{7}$$

where $\rho \in [0, 1]$ specifies the trade-off between loss and latency optimization. We use these weights for simple random sampling with replacement (Weighted-SRSWR) to select clusters each training epoch. The probability of selecting a given cluster increases if the expected reduction in latency is large or if the average loss in the cluster is large. Figure 4 depicts the scheduling algorithm at a high level. The full client selection proposal is outlined in Algorithm 1.

E. Bias Considerations

Our scheduler prefers more performant devices within a cluster, which has the potential to bias the model by neglecting stragglers. If we assume the data across devices within a cluster is IID, then no bias is introduced since model updates for low latency devices are guaranteed to benefit all devices in that cluster. However, our summaries only consider part of the joint distribution and not the full joint (eq. 2). Therefore, in practice, it is possible for some amount of skew to exist within a cluster. For example, if we configure HACCS with the $P(X \mid y)$ summary, it is still possible for the label distribution, $P(y)$, to vary across devices within a cluster. This bias could be mitigated by clustering based on joint distributions. However, requiring the clients to send these distributions as a data summary will not scale well in practice. Other FL frameworks give the user a parameter which can control the preference for performance (and therefore the amount of bias) [9], [10]. The analogous parameter in our framework is ρ, which specifies the preference for low latency vs. bias in the system. Smaller values of ρ will cause us to assign larger weights to clusters with high loss. This increases the probability of sampling these clusters, and therefore increases the probability of including slow devices in training. We examine these device inclusion frequencies in section V.

Changes in the system can also help mitigate bias in our framework, since the ordering of clients in a given cluster can change during training. For example, re-clustering can occur when clients join or leave the system (dropout). Furthermore, the performance of any given client can change over time, e.g. in the case of mobile phones, a user might move to a Wi-Fi connection or charge the battery on the device. In these cases, devices could be reordered within a cluster, since the latencies have changed. If the relative performance of a device in a

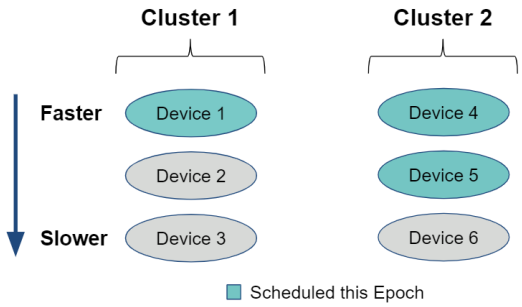


Fig. 4: Six clients are grouped into 2 clusters based on their data similarity. If we are allowed to schedule 3 devices, we sample the set of clusters 3 times and take the lowest latency device available. This illustrates one possible outcome.

Algorithm 1: Centralized Client Selection

Input: Device summaries $\mathcal{S}(\mathcal{Z}_i)$, Distance function d, Target number of clients k
Result: Devices for training in the current epoch

```
/* Computed at the start of training          */
distMatrix ← Pairwise differences, d(S(Z_i), S(Z_j))
clusters ← OPTICS(distMatrix)

/* Computed each epoch                        */
θ_i ← Output from eq. 7
sample ← Weighted-SRSWR(k, clusters, θ_i)
clients ← empty list
for cluster in sample do
    bestClient ← < min latency client in cluster >
    insert(clients, bestClient)
    remove(cluster, bestClient)
end
return clients
```

cluster increases, its selection probability will also increase, since we include the fastest devices first for model training.

F. Implementation

Our implementation of HACCS is built in Python and uses gRPC [25] for communication between clients and the central server. When a client is initialized, it connects to the server and provides some basic information, including a summary of its local data for scheduling as well as estimates of its available computational resources. The client can optionally add noise to its data summary prior to sending, which enforces any differential privacy requirements for the system.

We use the PySyft [26] framework for the model training and communication of model weights between the clients and server. Each training epoch, the central server pushes the global model parameters to each of the clients scheduled for training. PySyft communicates these parameters to each client and computes a model update using the local data of each client. Once all client updates are received, the server averages the model updates to produce a new set of global parameters.

V. Evaluation

We evaluate the performance of HACCS compared to other techniques in the literature using real-world datasets. We use the time-to-accuracy (TTA) metric to evaluate performance, which measures how long it takes to obtain a pre-specified model accuracy on the selected dataset.

A. Setup

Datasets. We consider two datasets for our evaluation:

- FEMNIST [16]: This dataset is part of the LEAF framework and is an extended version of MNIST [15]. It consists of 28x28 black and white handwritten alphanumeric characters (lowercase and uppercase).
- CIFAR-10 [27]: This dataset consists of 32x32 color images used for classification with 10 classes. Example response labels include cars, airplanes, cats, and dogs.

Testbed. Our experimentation setup considers two 24 core hyperthreaded Intel Xenon CPU E5-2620 machines with 64GB RAM each to emulate 25 clients per machine, to get 50 clients in total[2]. We train a convolutional neural network based upon the LeNet architecture [28]. The data distribution per client consists of a majority label (75%) and three noise labels (12% / 7% / 6%) unless otherwise specified. The amount of data available in each client varies.

To address system heterogeneity, we introduced time-based delays to simulate differences in computation, bandwidth, and network latency. For each attribute, we assigned a performance category: fast, medium, slow, and very slow, with the probability of assignment being 60%, 20%, 15%, and 5% respectively. Table II outlines the differences between each category. Where there are intervals listed in the table, we randomly generated a number uniformly over that interval for each client. We used these delays and distributions to simulate performance differences across clients in the system.

Attribute	Fast	Medium	Slow	Very Slow
Compute	No Delay	1.5 - 2.0x Delay	2.0 - 2.5x Delay	2.5 - 3.0x Delay
Bandwidth	75-100 Mbps	50-75 Mbps	25-50 Mbps	1-25 Mbps
NW Latency	20-200 ms	20-200 ms	20-200 ms	20-200 ms

TABLE II: Categories for modeling system heterogeneity.

Client Selection Strategies. We evaluate the following baseline and proposed strategies:

- Random Selection: Out of the available clients, we randomly select k clients per epoch for training.
- TiFL: Clients are first grouped into tiers based on their computational performance [9]. Each epoch, a tier is randomly selected based on the average loss in each tier and how often tiers have been sampled in past epochs.
- Oort: Each client is assigned a utility based on the current loss and the estimated latency [10]. In each epoch, we recompute the utility of each client available for training and select k clients with the highest utility.
- HACCS using $P(y)$: Clients are clustered based on the marginal distribution of response labels. Clusters and clients are selected as outlined in §IV-D.
- HACCS using $P(X \mid y)$: Clustering is based on data conditioned on the response labels as discussed in §IV-A.

B. Scheduling Performance

In this experiment, we compare the time-to-accuracy (TTA) for different client selection strategies. The number of clients participating in the training process is set to 10 (20% of 50 clients) and the total number of labels available is set to 10 across both CIFAR-10 and FEMNIST[3].

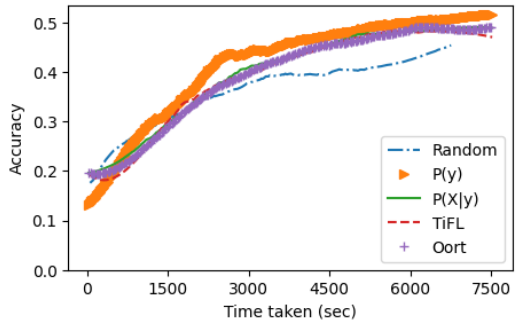
Figure 5 shows a smoothed curve for the training time across both datasets. For CIFAR-10 (Figure 5a), $P(y)$ achieves the highest accuracy and lowest convergence time followed by $P(X \mid y)$, TiFL, Oort and Random. For example, to reach an accuracy of 50%, both $P(y)$ and $P(X \mid y)$ take less than 4300 seconds compared to the 5600 seconds taken by TiFL and Oort, a 23% improvement. $P(y)$ and $P(X \mid y)$ take advantage of the data summaries of response labels and input features respectively to cluster clients with similar data distributions. TiFL and Oort eventually obtain comparable model performance to $P(X \mid y)$; however, they take more time to converge in the absence of methods which take advantage of data heterogeneity.

For FEMNIST (Figure 5b), $P(y)$ achieves higher accuracy with low convergence time when compared to $P(X \mid y)$, TiFL, Oort and Random. $P(y)$ takes 18%, 27%, 40% and 74% less time than $P(X \mid y)$, TiFL, Random and Oort respectively to reach 80% accuracy. All models eventually obtain comparable accuracy on this dataset. The Oort scheduler ensures that it

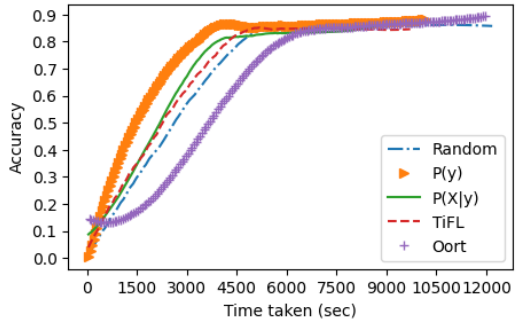
[2]Since real world client devices may not necessarily have GPUs, we emulate the client devices using commodity CPU hardware. Nevertheless, we expect similar results with GPUs.

[3]Wherever applicable, we consider only a portion of labels so as to have a tangible overlap of labels between clients.

always selects the highest utility clients per epoch. However, it may fail to accommodate all the data distributions during training, leading to a higher convergence time.



(a) Performance with the CIFAR-10 dataset



(b) Performance with the FEMNIST dataset

Fig. 5: Training convergence performance of different client selection strategies for CIFAR10 and FEMNIST.

C. Dropout Performance

Federated learning systems can experience a large amount of volatility during the model training process. For example, the owners of the client devices may move from one location to another, resulting in a switch or loss of internet connectivity. In this section, we investigate the effect of device dropout across the client selection strategies. To simulate dropout, we randomly mark 10% of the clients as unavailable at the beginning of each epoch. At the end of each epoch, we recover the failed devices and mark them as available for the next epoch. We seed the random number generators to ensure that the same set of devices are dropped in each epoch across all the client selection strategies. We use 20 classes of the FEMNIST dataset for this experiment, where labels are distributed across the devices in the same proportion (75% / 12% / 7% / 6%) as used in the scheduling performance experiments.

Figure 6 depicts the model accuracy over time with device dropout. $P(X \mid y)$ achieves higher accuracy and lower training time, followed by TiFL, $P(y)$, Random and Oort. To reach an accuracy of 50%, TiFL, $P(y)$, and Random take 18%, 29% and 60% extra time as compared to $P(X \mid y)$. The clustering of client devices based on data and system heterogeneity allows $P(y)$ and $P(X \mid y)$ to replace dropped devices with next best performing devices in the same cluster if one is

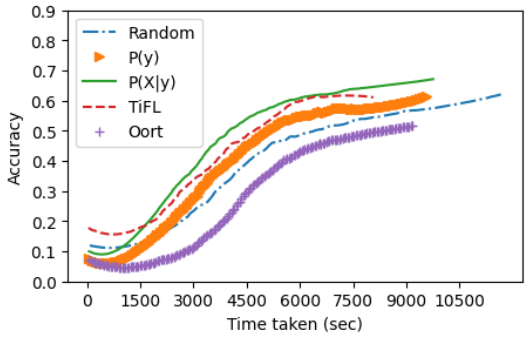


Fig. 6: Performance with 10% dropout on FEMNIST dataset.

available. The availability of more features ensures better clustering of client devices for $P(X \mid y)$, causing it to perform better compared to $P(y)$. As for TiFL, the dropped devices can be replaced by devices from the same tier. $P(y)$ and TiFL have a similar trajectory, so we believe the devices present in tiers for the experiment had similar data distributions. However, for the Oort strategy, which takes into account the loss for client selection, if a client with unique distribution drops frequently, this can cause oscillation of accuracy values leading to more time to converge.

D. Sensitivity Analysis

We now evaluate the degree to which our framework's performance depends on the various tuning parameters in HACCS, in addition to the degree of skew in the data distribution using the CIFAR-10 dataset.

1) Degree of Label Skew: This work proposes techniques for leveraging skewed data (via clustering) to accelerate model training. However, the degree of skew can vary substantially in practice. If the data across federated devices is close to IID, our clustering could result in one large cluster. We now evaluate HACCS against data distributions with various degrees of skew to characterize the worst-case performance. We compare three levels of skew: the IID case (no skew) with 10 labels per client, skewed data with 5 randomly selected labels per client, and highly skewed data, which has one main label per client, plus a few randomly selected noise labels.

Figure 7 shows the time-to-accuracy for each strategy and degree of skew. First, we examine the IID case, where data points across all devices come from the same joint distribution. More specifically, we ensure that data from each label is present on every client device. We also ensure that the same number of training samples exist on each client, which is unlikely to occur in practice [14], [29]. All of the techniques outperform the random scheduler, which selects slow clients unnecessarily. Our $P(y)$ method and the Oort scheduler both achieve the fastest TTA in this setting. The clustering for $P(y)$ groups all of the clients into a single cluster, which allows us to simply select the fastest clients each epoch. Our $P(X \mid y)$ scheduler was only better than the random scheduler. The higher dimensional data summaries used for clustering caused it to identify a few clusters, even though the data was

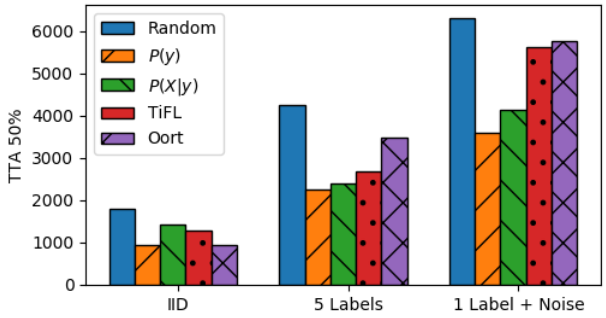


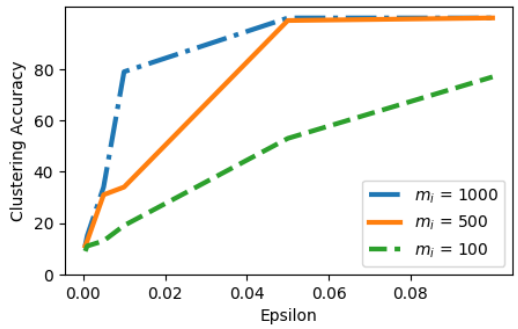
Fig. 7: Time to obtain 50% accuracy across various degrees of skew in the data distribution.

IID. Therefore, some slower devices were selected for training unnecessarily. Tier selection in TiFL is probabilistic in nature, so it will not necessarily select the fastest tier each epoch. All of the experiments with IID data outperform the experiments using skewed data, which is expected.

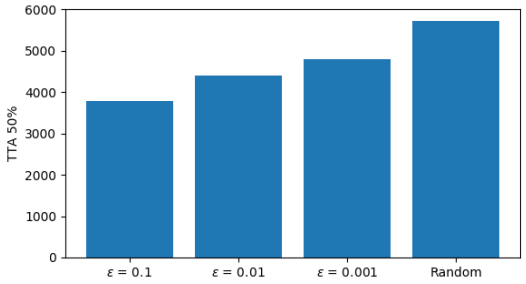
We now introduce skew in the data distribution, which is more representative of a real-world dataset. We randomly assign 5 labels to each client; each individual label now exists only on a subset of the clients. This puts more pressure on the scheduling algorithm to select clients that ensure representation from each data distribution. In this case, both of our proposed methods outperform the baseline techniques (TiFL and Oort). $P(y)$ gives 16% and 35% reduction in TTA when compared to TiFL and Oort respectively. We observe that TiFL outperforms the Oort scheduler and both outperform the random scheduler. In the final experiment, we increase the skew further to a single label per device, plus a few noise labels. We observe a similar result to the 5 label experiment when comparing the schedulers. More specifically, our proposed approaches outperform the baselines (TiFL and Oort). $P(y)$ gives 36% and 38% reduction in TTA when compared to TiFL and Oort respectively. We conclude that our techniques are most beneficial when the data distribution is skewed and our $P(y)$ technique is comparable to existing techniques when the underlying data is IID.

2) Privacy Parameter: As previously discussed, we apply differential privacy techniques to ensure that our histograms do not reveal information about the client data. This technique requires a privacy budget, epsilon (ϵ), which is specific to each application. First, we evaluate the impact of epsilon on our clustering accuracy. Then, we evaluate its impact on scheduling and training time.

First, we evaluate the impact of privacy loss ϵ on our clustering accuracy using the $P(y)$ summary. We use 20 clients for this experiment, where exactly 2 clients are assigned to each response label from the CIFAR-10 dataset. This will ideally generate 10 clusters, each containing two clients. The clustering accuracy will be based on the number of clusters we correctly identify. The results are plotted in figure 8a. Each device has m_i data points following a 70% / 10% /10 % / 10% distribution. We vary the amount of data, since the noise



(a) Epsilon values vs Clustering Accuracy for the $P(y)$ summary. Small values of epsilon (< 0.01) can heavily impact our ability to identify clusters for all three data sizes.



(b) Impact of the ϵ parameter on TTA.

Fig. 8: Differential privacy experiments using CIFAR-10

added to each histogram bin will have a larger impact on small data sizes. For each value of ϵ, we performed clustering 10 times and averaged the observed accuracy values (all margins of error for a 95% CI are less than 0.1). We observe that when there are more than 500 data points on each client, the clustering accuracy remains very high for $\epsilon \geq 0.05$. When there are only 100 data points per device, we see a much smoother drop in accuracy across the different values of ϵ.

Our second experiment evaluates the effect of the privacy loss ϵ on the model training performance. Figure 8b shows the results of this experiment. We observe that $\epsilon = 0.1$ achieves a 34% reduction in TTA as compared to the random scheduler. For $\epsilon = 0.01$ and $\epsilon = 0.001$, we observe reductions of 23% and 16% respectively. Therefore, increasing the privacy requirements on the system reduces the effectiveness of our clustering technique, which is expected. When data summaries fail to accurately reflect the local data on each client, our ability to leverage it in scheduling is also reduced.

3) Effect of ρ on Scheduling: We now systematically vary the ρ parameter to identify its impact on training performance. This parameter is used to specify the trade-off between latency and loss when scheduling clusters in each epoch (see Eq. 7, low values of ρ favor clients with high loss, while high values of ρ favor clients with low latency). We used the CIFAR-10 dataset for this experiment and the same skewed distribution of labels used in the performance experiments.

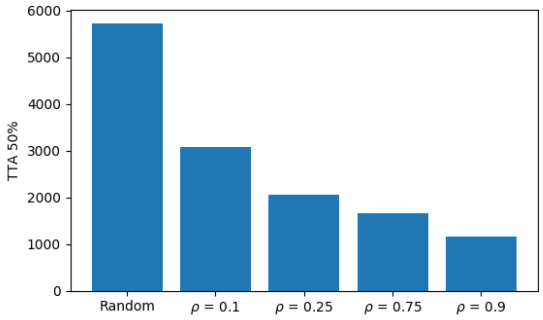


Fig. 9: Effect of the ρ parameter during model training.

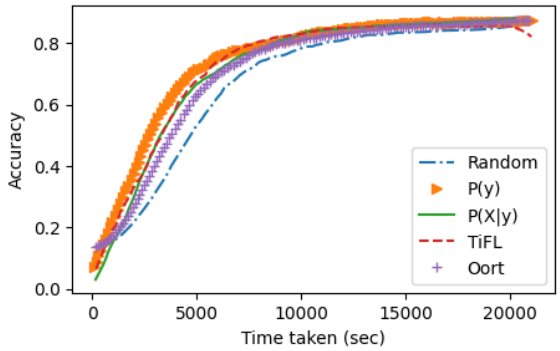


Fig. 10: Performance with both label and feature skew.

Figure 9 shows the result of this experiment across multiple values of ρ. We observe that larger values of ρ are associated with a faster time to converge to 50% model accuracy. This suggests that assigning a larger sampling weight to faster clusters is preferable to assigning a larger weight to clusters with higher loss. There are a few factors that explain this relationship. First, our data distributions allowed for 25% of the data on each client to consist of "noise" labels. This provides some diversity in the data distributions of each cluster, which allows us to learn more than just a single label from each cluster. Second, even if smaller weights are assigned to clusters with higher loss, the law of large numbers implies they should still be sampled occasionally during training (provided the weights are not extremely small).

4) Feature Skew: This work mostly considers label skew, which is consistent with the evaluations in related works [9], [10]. However, since one of our data summaries explicitly considers feature skew, we include an experiment to briefly examine its effects. For this experiment, we use a modified MNIST dataset, where half of the images are randomly selected and rotated 45° prior to training [15]. Similar to previous experiments, we skew the label distribution and we also ensure that the major labels all have the same rotation angle (either 0° or 45°). This rotation ensures that features will be skewed, even when the majority labels match across devices.

Figure 10 shows the accuracy obtained over time for our methods and the baselines. For a target accuracy of 85%, the $P(X \mid y)$ method had the fastest TTA, followed by $P(y)$ and TiFL, which were both about 4% slower. The decrease in performance with the $P(y)$ summary is expected, since we have ensured there will be skew within the identified clusters. For example, if the $P(y)$ method puts all devices with the label "8" in a cluster, there will be some devices that have rotated images and some without.

5) Scheduling Bias: The ρ parameter controls our preference for low latency, at the expense of potentially introducing bias into the model. Since our summary functions do not represent the full joint distribution, it is possible for a slow device to have a unique skew which is not captured elsewhere in the cluster. To examine these effects, we ran the feature skew experiment with $\rho = 0.01$ (i.e. a strong preference for minimizing loss rather than latency). Table III shows the fraction of devices in each cluster that were included for training over 200 epochs. We observe that all clusters included at least 50% of their devices at some point during training over 200 epochs. Most of the clusters (8/10 for $P(y)$ and 30/31 for $P(X \mid y)$) included 75% or more of their devices at some point during training.

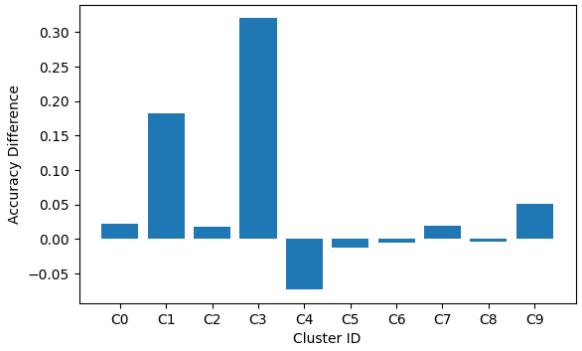
TABLE III: Device inclusion over 200 epochs

Devices Included	0-50%	50-75%	75-100%
$P(y)$ **Clusters**	0	2	8
$P(X \mid y)$ **Clusters**	0	1	30

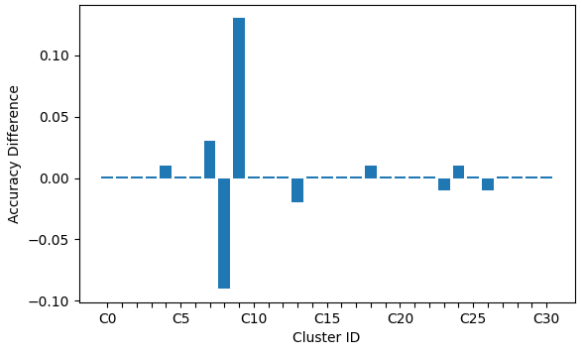
Figure 11 takes a deeper look at the difference in accuracy between the fastest and slowest devices in each cluster. We observe that for both clustering methods, the difference in accuracy between the fastest and slowest devices is close to zero and even negative in some cases. Most of the zero entries for the $P(X \mid y)$ summary indicate clusters that only had a single device as a member. Negative numbers indicate that the global model actually performed better on the slowest device rather than the fastest device. We also observe that the global model sometimes performs worse on the slowest devices within a cluster (which is more notable with the clusters using the $P(y)$ summary). This is caused by feature skew within the identified clusters. One possible way to ensure stragglers within a cluster are selected more frequently would be to perform sampling within a cluster, rather than simply using the current ordering based on latency.

E. Discussion

Both of our proposed methods, $P(y)$ and $P(X \mid y)$, perform well compared to the baseline techniques. However, we did not observe a consistent winner between the $P(y)$ and $P(X \mid y)$ method across all the experiments. In practice, we believe the $P(y)$ summary may be preferable since it is more compact and it provides less information about the user's private data. The $P(X \mid y)$ summary generates a representation of the distribution of pixels for each class of images on a device, which has more privacy and bandwidth implications. While we proposed and evaluated a scheduling strategy based on



(a) Accuracy difference for $P(y)$ clusters



(b) Accuracy difference for $P(X \mid y)$ clusters

Fig. 11: Difference in accuracy between fastest and slowest device in each cluster.

clusters, we believe exploring additional scheduling strategies will be an important future research direction. Different kinds of privacy-preserving data summaries could also affect performance in HACCS and could be a future topic of research.

VI. Related Work

Distributed ML Systems: Many of the existing general purpose distributed ML systems [30], [31] fall short of adapting model training in presence of system heterogeneity (compute and network) to reduce the training time. Although prior work has addressed compute and network heterogeneity [32]–[35], it led to trade-offs between training time and model accuracy. This trade-off is addressed by [11], however it does not take advantage of the data heterogeneity. Also, these systems focus on complex machine learning models that require stable CPU/GPU clusters. However, in case of FL, the heterogeneous nodes are volatile in nature. Our methods accelerate model training in this setting by utilizing the heterogeneous nodes.

System and Statistical Heterogeneity: Many systems exist that attempt to identify and mitigate system heterogeneity in federated learning [36]–[38]. The FedProx system [36] attempts to address these issues by dynamically tailoring the workloads to match the resources available on each client. For example, devices that are bandwidth constrained or have limited computational resources are allowed to perform a smaller amount of work each training epoch. This non-uniform distribution of work mitigates the straggler effect to some degree. Other systems attempt to address both system and statistical heterogeneity simultaneously. The Oort system dynamically prioritizes devices for scheduling based on their perceived utility [10]. The utility values are based on the empirical loss each epoch, where devices with high losses are given scheduling priority in the system. The TiFL system first groups clients into tiers based on their system performance capabilities and then selects one tier each epoch for model training [9]. Disparities in accuracy between tiers are considered when scheduling a specific tier. Both Oort and TiFL quantify statistical heterogeneity using only the empirical loss. Our evaluation results show that our proposed framework outperforms both TiFL and Oort in a variety of scenarios, especially when the degree of statistical heterogeneity is high. This is because our system attempts to leverage additional information about the client data to further mitigate the effects of statistical heterogeneity.

Clustering: There is a growing literature on clustering for addressing statistical heterogeneity in federated learning. Clusters are identified based on the similarity of model updates or model parameters, which may change every epoch [39], [40]. In most of these works, separate models are then trained for each identified cluster as opposed to training a single global model. This literature is distinct from our approach, since our problem statement focuses on training a single global model across all devices and considers additional privacy-preserving summaries when performing clustering. Other works focus specifically on training models using stochastic gradient descent (SGD) which could be clustered based on descent directions for each device [41]. However, these clusters must be re-evaluated each epoch as the model weights change. In this work, we suggest some alternative data summaries that can be used for clustering and are not expected to change as frequently as data summaries used in the prior work.

Parallel/Distributed systems: Many of the existing parallel/distributed systems [42]–[44] take into account system heterogeneity to schedule tasks for improved resource utilization. In our methods, we also explore the system heterogeneity to identify the best performing node among each device clusters created using data heterogeneity factors.

VII. Conclusion

Both system and statistical heterogeneity can negatively impact model training in a federated setting. Efficient client selection strategies can have a large impact on model training time by mitigating the straggler effect. We proposed a novel client selection scheme which leverages skewed data by clustering devices with similar data distributions. In non-IID settings, our technique can substantially reduce the required model training time, up to 38%.

Acknowledgement

This research was supported in part by the NSF under grant CNS-1717834.

REFERENCES

[1] C. Tankard, "What the gdpr means for businesses," *Netw. Secur.*, vol. 2016, pp. 5–8, 2016.

[2] Y. Liu *et al.*, "Fedvision: An online visual object detection platform powered by federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 08, 2020, pp. 13 172–13 179.

[3] T. Yang *et al.*, "Applied federated learning: Improving google keyboard query suggestions," *CoRR*, vol. abs/1812.02903, 2018. [Online]. Available: http://arxiv.org/abs/1812.02903

[4] L. Huang *et al.*, "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," *Journal of Biomedical Informatics*, vol. 99, p. 103291, 2019.

[5] B. McMahan *et al.*, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282.

[6] K. A. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proceedings of the 2nd SysML Conference*, ser. SysML 2019, 2019.

[7] T. Li *et al.*, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[8] S. K. Lo *et al.*, "A systematic literature review on federated machine learning: From a software engineering perspective," *ACM Comput. Surv.*, vol. 54, no. 5, May 2021.

[9] Z. Chai *et al.*, "Tifl: A tier-based federated learning system," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 125–136.

[10] F. Lai *et al.*, "Oort: Efficient federated learning via guided participant selection," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 19–35.

[11] R. Hong and A. Chandra, "Dlion: Decentralized distributed deep learning in micro-clouds," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '21. New York, NY, USA: Association for Computing Machinery, 2020, p. 227–238.

[12] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 234–243.

[13] T. Darrell, M. Kloft, M. Pontil, G. Rätsch, and E. Rodner, "Machine learning with interdependent and non-identically distributed data," *Dagstuhl Reports*, vol. 5, p. 18–55, 2015.

[14] K. Hsieh *et al.*, "The non-IID data quagmire of decentralized machine learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 4387–4398.

[15] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] S. Caldas *et al.*, "LEAF: A benchmark for federated settings," *CoRR*, vol. abs/1812.01097, 2018. [Online]. Available: http://arxiv.org/abs/1812.01097

[17] Y. Zhao *et al.*, "Federated learning with non-iid data," 2018. [Online]. Available: https://arxiv.org/abs/1806.00582

[18] P. Kairouz *et al.*, *Advances and Open Problems in Federated Learning*, 2021.

[19] T. Kailath, "The divergence and bhattacharyya distance measures in signal selection," *IEEE Transactions on Communication Technology*, vol. 15, no. 1, pp. 52–60, 1967.

[20] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, no. 3–4, p. 211–407, Aug. 2014.

[21] M. Ester *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[22] M. Ankerst *et al.*, "Optics: Ordering points to identify the clustering structure," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 49–60.

[23] A. Nagpal, A. Jatain, and D. Gaur, "Review based on data clustering algorithms," in *2013 IEEE Conference on Information Communication Technologies*, 2013, pp. 298–303.

[24] B. Xu *et al.*, "Online client scheduling for fast federated learning," *IEEE Wireless Communications Letters*, vol. 10, no. 7, pp. 1434–1438, 2021.

[25] C. N. C. Foundation, "gRPC: gRPC Remote Procedure call," https://grpc.io/, 2021.

[26] T. Ryffel *et al.*, "A generic framework for privacy preserving deep learning," *CoRR*, vol. abs/1811.04017, 2018. [Online]. Available: http://arxiv.org/abs/1811.04017

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[28] 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.

[29] J. Luo *et al.*, "Real-world image datasets for federated learning," *CoRR*, vol. abs/1910.11089, 2019. [Online]. Available: http://arxiv.org/abs/1910.11089

[30] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[31] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[32] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 463–478.

[33] Q. Luo *et al.*, "Hop: Heterogeneity-aware decentralized training," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 893–907.

[34] P. Watcharapichat *et al.*, "Ako: Decentralised deep learning with partial gradient exchange," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 84–97.

[35] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching {LAN} speeds," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 629–647.

[36] T. Li *et al.*, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 429–450.

[37] A. M. Abdelmoniem and M. Canini, "Towards mitigating device heterogeneity in federated learning via adaptive model quantization," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, ser. EuroMLSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 96–103.

[38] E. Diao, J. Ding, and V. Tarokh, "Hetero{fl}: Computation and communication efficient federated learning for heterogeneous clients," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=TNkPBBYFkXg

[39] A. Ghosh *et al.*, "An efficient framework for clustered federated learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 19 586–19 597.

[40] C. Briggs *et al.*, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–9.

[41] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3710–3722, 2021.

[42] D. Cheng *et al.*, "Adaptive scheduling of parallel jobs in spark streaming," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[43] V. S. Marco *et al.*, "Improving spark application throughput via memory aware task co-location: A mixture of experts approach," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, 2017, pp. 95–108.

[44] L. Xu *et al.*, "A heterogeneity-aware task scheduler for spark," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2018, pp. 245–256.