TiFL: A Tier-based Federated Learning System

Zheng Chai*[1], Ahsan Ali*[2], Syed Zawad*[2], Stacey Treux[3], Ali Anwar[4], Nathalie Barcaldo[4], Yi Zhou[4], Heiko Ludwig[4], Feng Yan[2], Yue Cheng[1]
[1]Department of CS, George Mason University, Farifax, VA, USA {zchai2,yuecheng}@gmu.edu
[2]Department of CS, University of Nevada, Reno, NV, USA {aali,szawad,fyan}@nevada.unr.edu
[3]Department of CS, Georgia Institute of Technology, Atlanta, GA, USA staceytruex@gatech.edu
[4]IBM Research - Almaden, San Jose, CA, USA {ali.anwar2,baracald,yi.zhou,hludwig}@ibm.com

ABSTRACT

Federated Learning (FL) enables learning a shared model across many clients without violating the privacy requirements. One of the key attributes in FL is the heterogeneity that exists in both resource and data due to the differences in computation and communication capacity, as well as the quantity and content of data among different clients. We conduct a case study to show that heterogeneity in resource and data has a significant impact on training time and model accuracy in conventional FL systems. To this end, we propose TiFL, a Tier-based Federated Learning System, which divides clients into tiers based on their training performance and selects clients from the same tier in each training round to mitigate the straggler problem caused by heterogeneity in resource and data quantity. To further tame the heterogeneity caused by non-IID (Independent and Identical Distribution) data and resources, TiFL employs an *adaptive* tier selection approach to update the tiering on-the-fly based on the observed training performance and accuracy. We prototype TiFL in a FL testbed following Google's FL architecture and evaluate it using the state-of-the-art FL benchmarks. Experimental evaluation shows that TiFL outperforms the conventional FL in various heterogeneous conditions. With the proposed adaptive tier selection policy, we demonstrate that TiFL achieves much faster training performance while achieving the same or better test accuracy across the board.

KEYWORDS

federated learning; stragglers; resource heterogeneity; data heterogeneity; edge computing

ACM Reference Format:
Zheng Chai*[1], Ahsan Ali*[2], Syed Zawad*[2], Stacey Treux[3], Ali Anwar[4], Nathalie Barcaldo[4], and Yi Zhou[4], Heiko Ludwig[4], Feng Yan[2], Yue Cheng[1]. 2020. TiFL: A Tier-based Federated Learning System. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '20), June 23–26, 2020, Stockholm, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3369583.3392686

HPDC '20, June 23–26, 2020, Stockholm, Sweden

ACM ISBN 978-1-4503-7052-3/20/06…$15.00
https://doi.org/10.1145/3369583.3392686

1 INTRODUCTION

Modern mobile and IoT devices are generating massive amount of data every day, which provides opportunities for crafting sophisticated machine learning (ML) models to solve challenging AI tasks [11]. In conventional high-performance computing (HPC), all the data is collected and centralized in one location and proceed by supercomputers with hundreds to thousands of computing nodes. However, security and privacy concerns have led to new legislation such as the General Data Protection Regulation (GDPR) [27] and the Health Insurance Portability and Accountability Act (HIPAA) [24] that prevent transmitting data to a centralized location, thus making conventional high performance computing difficult to be applied for collecting and processing the decentralized data. Federated Learning (FL) [15] shines light on a new emerging high performance computing paradigm by addressing the security and privacy challenges through utilizing decentralized data that is training local models on the local data of each client (data parties) and using a central aggregator to accumulate the learned gradients of local models to train a global model. Though the computing resource of individual clients may be far less powerful than the computing nodes in conventional supercomputers, the computing power from the massive number of clients can accumulate to form a very powerful "decentralized virtual supercomputer". Federated learning has demonstrated its success in a range of applications ranging from user-end devices to medical analysis systems. There has also been a rise of FL tools and framework development, such as Tensorflow Federated [13], LEAF [7], PaddleFL [12] and PySyft [25] to facilitate these demands. Depending on the usage scenarios, FL is usually categorized into *cross-silo* FL and *cross-device* FL [14]. In cross-device FL, the clients are usually a massive number of mobile or IoT devices with various computing and communication capacities [14, 15, 20] while in cross-silo FL, the clients are a small number of organizations with ample computing power and reliable communications [14, 29]. In this paper, we focus on the cross-device FL (for simplicity, we call it FL in the following), which intrinsically pushes the heterogeneity of computing and communication resources to a level that is rarely found in datacenter distributed learning and cross-silo FL. More importantly, the data in FL is also owned by clients where the quantity and content can be quite different from each other, causing severe heterogeneity in data that usually does not appear in datacenter distributed learning, where data distribution is well controlled.

We conduct a case study to quantify how data and resource heterogeneity in clients impacts the training performance and model accuracy of FL. The key findings are below: (1) training throughput

* Equal Contribution and Co-First Authors.

is usually bounded by slow clients (a.k.a. stragglers) with less computational capacity and/or slower communication, which we name as the *resource heterogeneity*. Asynchronous training is often employed to mitigate this problem in datacenter distributed learning. However, in FL, almost all the existing privacy methods [2, 5, 21] are built with the assumption of synchronous training, which makes asynchronous training difficult to be applied here.

(2) Different clients may train on different quantity of samples per training round and results in different round time (similar to straggler effect), which impacts the training time and potentially also the accuracy. We name this observation the *data quantity heterogeneity*. (3) In datacenter distributed learning, the classes and features of the training data are uniformly distributed among all clients, namely Independent Identical Distribution (IID). However, in FL, the distribution of data classes and features depends on the data owners, thus resulting in a non-uniform data distribution, known as non-Identical Independent Distribution (*non-IID data heterogeneity*). Our experiments show that such heterogeneity can significantly impact the training time and accuracy.

Driven by the above observations, we propose TiFL, a Tier-based Federated Learning System. The key idea here is adaptively selecting clients with similar per round training time so that the heterogeneity problem can be mitigated without impacting the model accuracy. Specifically, we first employ a lightweight profiler to measure the training time of each client and group them into different logical data pools based on the measured latency, called *tiers*. During each training round, clients are selected uniform randomly from the same tier based on the adaptive client selection algorithm of TiFL. In this way, the heterogeneity problem is mitigated as clients belonging to the same tier have similar training time. In addition to heterogeneity mitigation, such tiered design and adaptive client selection algorithm also allows controlling the training throughput and accuracy by adjusting the tier selection intelligently, e.g., selecting tiers such that the model accuracy is maintained while prioritizing selection of faster tiers.

While *resource heterogeneity* and *data quantity heterogeneity* information can be reflected in the measured training time, the *non-IID data heterogeneity* information is difficult to capture. This is because any attempt to measure the class and feature distribution violates the privacy-preserving requirements. To solve this challenge, TiFL offers an *adaptive* client selection algorithm that uses the accuracy as indirect measure to infer the *non-IID data heterogeneity* information and adjust the tiering algorithm on-the-fly to minimize the training time and accuracy impact. Such approach also serves as an online version to be used in an environment where the characteristics of heterogeneity change over time.

We prototype TiFL in a FL testbed that follows the architecture design of Google's FL system [4] and perform extensive experimental evaluation to verify its effectiveness and robustness using popular ML benchmarks such as LEAF [7]. The experimental results show that in the *resource heterogeneity* case, TiFL can improve the training time by a magnitude of 6× without affecting the accuracy. In the *data quantity heterogeneity* case, a 3× speedup is observed in training time with comparable accuracy to the conventional FL. Overall, TiFL outperforms the conventional FL with 3× improvement in training time and 8% improvement in accuracy in CIFAR10 [16] and 3× improvement in training time using FEMINIST[7] under LEAF.

2 RELATED WORK

The straggler problem is not new in FL, and it has been well studied in datacenter distributed learning. However, the privacy requirement and significantly higher heterogeneity level in FL impose new challenges. [10] proposes to use P2P communication among workers to detect slowed workers, performs work re-assignment, and exploits iteration knowledge to further reduce how much data needs to be preloaded on helpers. However, migrating data between users is strictly restricted in FL. SpecSync is proposed in [30], where each worker speculates about the parameter updates from others, and if necessary, it aborts the ongoing computation, pulls fresher parameters to start over, so as to opportunistically improve the training quality. However, information sharing between clients is not allowed in FL.

For general background of FL, we recommend readers to read these papers [14, 17]. Some recent research efforts in FL focus on the functionality [15] and privacy [5, 21], where they assume no heterogeneity in resource and data. [9] proposes a general statistical model for Byzantine machines and clients with data heterogeneity and uses it to cluster edge devices such that their datasets are similar. However, the authors do not consider the impact of clustering on training time or accuracy. FedCS [23] proposes to solve client selection issue via a deadline-based approach that filters out slowly-responding clients. However, FedCS does not consider how this approach effects the contributing factors of straggler clients in model training. [18] takes into account the resource heterogeneity in FL. The proposed approach assumes only two types of clients - stragglers and non-stragglers. But in a real FL environment there is a wide range of heterogeneity levels. In addition, their proposed solution involves partial training on stragglers which can lead to biasness in the trained model. [4] proposes a simple approach to handle stragglers problem in FL, where the aggregator selects 130% of the target number of devices to initially participate, and discards stragglers during training process. However, simply dropping the slower clients might exclude certain data distributions on the slower clients from contributing towards training the global model.

Asynchronous training is a common approach for mitigating the straggler problem in datacenter distributed learning, but it is difficult to be applied in FL as almost all the existing privacy methods [2, 5, 21] are built with the assumption of synchronous model weight updates.For instance, Differential Privacy [2] applies noise to each client's weights and the noise is determined by the variations of weights between other client's weights in that round. Similarly, Secure Aggregation [5] depends on Secret Sharing, where clients share secret keys to each other's encryption, thus requiring all secret sharing clients to be present during aggregation.

3 HETEROGENEITY IMPACT STUDY

Compared with datacenter distributed learning and cross-silo FL, one of the key features of cross-device FL is the significant resource and data heterogeneity among clients, which can potentially impact both the training throughput and the model accuracy. Resource

Algorithm 1 Federated Averaging Training Algorithm

Aggregator: initialize weight w_0
for each round $r = 0$ **to** $N - 1$ **do**
  C_r = (random set of $|C|$ clients)
  for each client c $\in C_r$ **in parallel do**
    $w_{r+1}^c = TrainClient(c)$
    s_c = (training size of c)
  end for
  $w_{r+1} = \sum_{c=1}^{|C|} w_{r+1}^c * \frac{s_c}{\sum_{c=1}^{|C|} s_c}$
end for

heterogeneity arises as a result of vast number of computational devices with varying computational and communication capabilities involved in the training process. The data heterogeneity arises as a result of two main reasons - (1) the varying number of training data samples available at each client and (2) the non-uniform distribution of classes and features among the clients.

3.1 Formulating Vanilla Federated Learning

FL is performed as an iterative process whereby the model is trained over a series of global training rounds, and the trained model is shared by all the involved clients. We define K as the total pool of clients available to select from for each global training round, and C as the set of clients selected per round. In every global training round, the aggregator selects a random fraction of clients C_r from K. The vanilla FL algorithm is briefly summarized in Alg. 1. The aggregator first randomly initializes weights of the global model denoted by ω_0. At the beginning of each round, the aggregator sends the current model weights to a subset of randomly selected clients. Each selected client then trains its local model with its local data and sends back the updated weights to the aggregator after local training. At each round, the aggregator waits until all selected clients respond with their corresponding trained weights. This iterative process keeps on updating the global model until a certain number of rounds are completed or a desired accuracy is reached.

The state-of-the-art FL system proposed in [5] adopts a client selection policy where clients are selected randomly. A coordinator is responsible for creating and deploying a master aggregator and multiple child aggregators for achieving scalability as the real world FL system can involve up to tens of thousands of clients [5, 14, 17]. At each round, the master aggregator collects the weights from all the child aggregators to update the global model.

3.2 Heterogeneity Impact Analysis

The resource and data heterogeneity among involved clients may lead to varying response latencies (i.e., the time between a client receives the training task and returns the results) in the FL process, which is usually referred as the straggler problem.

We denote the response latency of a client c_i as L_i, and the latency of a global training round is defined as

$$L_r = Max\Big(L_1, L_2, L_3, L_4 ... L_{|C|}\Big). \tag{1}$$

where L_r is the latency of round r. From Equation (1), we can see the latency of a global training round is bounded by the maximum training latency of clients in C, i.e., the slowest client.

We define τ levels of clients, i.e., within the same level, the clients have similar response latencies. Assume that the total number of levels is m and τ_m is the slowest level with $|\tau_m|$ clients inside. In the baseline case (Alg. 1), the aggregator selects the clients randomly, resulting in a group of selected clients with composition spanning multiple client levels.

We formulate the probability of selecting $|C|$ clients from all client levels except the slowest level τ_m as follows:

$$Pr = \frac{\binom{|K|-|\tau_m|}{|C|}}{\binom{|K|}{|C|}}. \tag{2}$$

Accordingly, the probability of at least one client in C comes from τ_m can be formulated as:

$$Pr_s = 1 - Pr. \tag{3}$$

Because $\frac{a-1}{b-1} < \frac{a}{b},\ while\ 1 < a < b$, we have

$$\begin{aligned} Pr_s &= 1 - \frac{\binom{|K|-|\tau_m|}{|C|}}{\binom{|K|}{|C|}} \\ &= 1 - \frac{(|K| - |\tau_m|)...(|K| - |\tau_m| - |C| + 1)}{|K|...(|K| - |C| + 1)} \\ &= 1 - \frac{|K| - |\tau_m|}{|K|} ... \frac{|K| - |\tau_m| - |C| + 1}{|K| - |C| + 1}. \end{aligned}$$

where Pr_s is probability of at least one client in C comes from τ_m. By applying the above proof, we get:

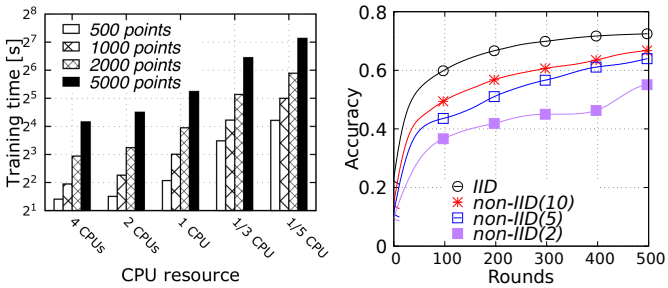
$$\begin{aligned} Pr_s &> 1 - \frac{|K| - |\tau_m|}{|K|} ... \frac{|K| - |\tau_m|}{|K|} \\ &= 1 - (\frac{|K| - |\tau_m|}{|K|})^{|C|} \end{aligned} \tag{4}$$

In real-world scenarios, large number of clients can be selected at each round, which makes $|K|$ extremely large. As a subset of K, the size of C can also be sufficiently large. Since $\frac{|K|-|\tau_m|}{|K|} < 1$, we get $(\frac{|K|-|\tau_m|}{|K|})^{|C|} \approx 0$, which makes $Pr_s \approx 1$, meaning in a vanilla FL training process, the probability of selecting at least one client from the slowest level is reasonably high for each round. According to Equation (1), the random selection strategy adopted by state-of-the-art FL system may suffer from a slow training performance.

3.3 Experimental Study

To experimentally verify the above analysis and demonstrate the impact of resource heterogeneity and data quantity heterogeneity, we conduct a study with a setup similar to the paper [8]. The testbed is briefly summarized as follows:

- We use a total of 20 clients and each client is further divided into 5 groups with 4 client per group.
- We allocate 4 CPUs, 2 CPUs, 1 CPU, 1/3 CPU, 1/5 CPU for every client from group 1 through 5 respectively to emulate the resource heterogeneity.
- The model is trained on the image classification dataset CIFAR10 [16] using the vanilla FL process 3.1 (model and learning parameters are detailed in Section 5).



(a) Average training time per round

(b) Accuracy across rounds

Figure 1: (a) Training time per round (logscale) for one client with varying amount of resource and training data quantity (number of training points) ; (b) accuracy under varying number of classes per client (non-IID) with fixed amoumd of computational resources.

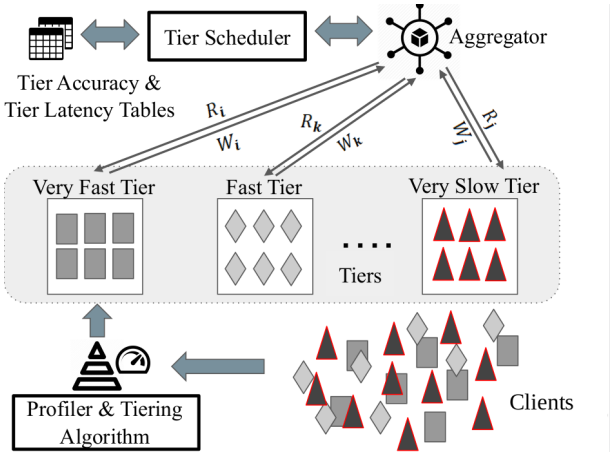


Figure 2: Overview of TiFL.

- Experiments with different data size and non-IIDness level for clients are conducted to produce data heterogeneity results.

As shown in Fig. 1 (a), with the same amount of CPU resource, increasing the data size from 500 to 5000 results in a near-linear increase in training time per round. As the amount of CPU resources allocated to each client increases, the training time gets shorter. Additionally, the training time increases as the number of data points increase with the same number of CPUs. These preliminary results imply that the straggler issues can be severe under a complicated and heterogeneous FL environment. To evaluate the impact of data distribution heterogeneity, we keep the same CPU resources for every client (i.e., 2 CPUs) and generate a biased class and feature distribution following [31]. Specifically, we distribute the dataset in such a way that every client has equal number of images from 2 (non-IID(2)), 5 (non-IID(5)) and 10 (non-IID(10)) classes, respectively. We train the model on Cifar10 dataset using the vanilla FL system as described in Section 3.1 with the model and training parameters detailed in Section 5. As seen in Fig. 1 (b), there is a clear difference in the accuracy with different non-IID distributions. The best accuracy is given by the IID since it represents a uniform class and feature distribution. As the number of classes per client is reduced, we observe a corresponding decrease in accuracy. Using 10 classes per client reduces the final accuracy by around 6% compared to IID (it is worth noting that non-IID(10) is not the same as IID as the feature distribution in non-IID(10) is skewed compare to IID). In the case of 5 classes per client, the accuracy is further reduced by 8%. The lowest accuracy is observed in the 2 classes per client case, which has a significant 18% drop in accuracy.

These studies demonstrate that the data and resource heterogeneity can cause significant impact on training time and training accuracy in FL. To tackle these problems, we propose TiFL— a tier-based FL system which introduces a heterogeneity-aware client selection methodology that selects the most profitable clients during each round of the training to minimize the heterogeneity impact while preserving the FL privacy proprieties, thus improving the overall training performance of FL.

4 TIFL: A TIER-BASED FEDERATED LEARNING SYSTEM

In this section, we present the design of the proposed tier-based federated learning system TiFL. The key idea of a tier-based system is that given the global training time of a round is bounded by the slowest client selected in that round (see Equation 1), selecting clients with similar response latency in each round can significantly reduce the training time. We first give an overview of the architecture and the main flow of TiFL system. Then we introduce the profiling and tiering approach. Based on the profiling and tiering results, we explain how a tier selection algorithm can potentially mitigate the heterogeneity impact through a straw-man proposal as well as the limitations of such static selection approach. To this end, we propose an adaptive tier selection algorithm to address the limitations of the straw-man proposal. Finally, we propose an analytical model through which one can estimate the expected training time using selection probabilities of tiers and the total number of training rounds.

4.1 System Overview

The overall system architecture of TiFL is present in Fig. 2. TiFL follows the system design to the state-of-the-art FL system [5] and adds two new components: a *tiering module* (a profiler & tiering algorithms) and a *tier scheduler*. These newly added components can be incorporated into the coordinator of the existing FL system [4]. It is worth to note that in Fig. 2, we only show a single aggregator rather than the hierarchical master-child aggregator design for a clean presentation purpose. For large scale system in practice, TiFL supports master-child aggregator design for scalability and fault tolerance.

In TiFL, the first step is to collect the latency metrics of all the available clients through a lightweight profiling as detailed in Section 4.2. The profiled data is further utilized by our tiering algorithm. This groups the clients into separate logical pools called *tiers*. Once the scheduler has the tiering information (i.e., tiers that the clients belong to and the tiers' average response latencies), the training process begins. Different from vanilla FL that employs a random client selection policy, in TiFL the scheduler selects a tier and then randomly selects targeted number of clients from that tier. After

the selection of clients, the training proceeds as state-of-the-art FL system does. By design, TiFL is non-intrusive and can be easily plugged into any existing FL system in that the tiering and scheduler module simply regulate client selection without intervening the underlying training process.

4.2 Profiling and Tiering

Given the global training time of a round is bounded by the slowest client selected in that round (see Equation 1), if we can select clients with similar response latency in each round, the training time can be improved. However, in FL, the response latency is unknown a priori, which makes it challenging to carry out the above idea. To solve this challenge, we introduce a process through which the clients are *tiered* (grouped) by the *Profiling and Tiering* module as shown in Fig. 2.

Offline Profiling and Tiering. As the first step, all available clients are initialized with a response latency L_i of 0. The profiling and tiering module then assigns all the available clients the profiling tasks. The profiling tasks execute for *sync_rounds* rounds and in each profiling round, the aggregator asks every client to train on the local data and waits for their acknowledgement for T_{max} seconds. All clients that respond within T_{max} have their response latency value RT_i incremented with the actual training time, while the ones that have timed out are incremented by T_{max}. After *sync_rounds* rounds are completed, the clients with $L_i >= sync\_rounds * T_{max}$ are considered *dropouts* and excluded from the rest of the calculation. The collected training latencies through profiling of clients creates a histogram, which is split into m groups and the clients that fall into the same group forms a tier. The response latency of each client are then stored by the scheduler and recorded persistently which is used later for scheduling and selecting tiers. The total overhead incurred by the offline profiling would be $sync\_rounds * T_{max}$.

Online Profiling and Tiering. The profiling and tiering can be conducted online to reflect the dynamic computation and communication performance so that clients can be adaptively grouped into the right tiers. It is important for online profiling to be lightweight, thus our online profiling method only measures the training throughput of a client instead of running a complete iteration. The training throughput can be paired with the quantity of training data to estimate the training latency. We also utilize the latency of the already participated clients to reduce the number of clients that need to be profiled to further reduce the profiling overhead.

4.3 Straw-man Proposal: Static Tier Selection Algorithm

In this section, we present a naive static tier-based client selection policy and discuss its limitations, which motivates us to develop an advanced adaptive tier selection algorithm in the next section. While the profiling and tiering module introduced in Section 4.2 groups clients into m tiers based on response latencies, the tier selection algorithm focuses on how to select clients from the proper tiers in the FL process to improve the training performance. The natural way to improve training time is to prioritize towards faster tiers, rather than selecting clients randomly from all tiers (i.e., the full K pool). However, such selection approach reduces the training time without taking into consideration of the model accuracy and privacy properties. To make the selection more general, one can specify each tier n_j is selected based on a predefined probability, which sums to 1 across all tiers. Within each tier, $\mid C \mid$ clients are selected according to inner-tier selection policy. While a sophisticated inner-tier selection policy can further optimize the performance, it needs to be carefully co-designed with the privacy method (detailed in Section 4.6). In this paper, we present a simple uniform selection policy to illustrate our approach and defer more sophisticated inner-tier selection policy as our future work.

In a real-world FL scenarios, there can be a large number of clients involved in the FL process [5, 14, 17]. Thus in our tiering-based approach, the number of tiers is set such that $m << |K|$ and number of clients per tier n_j is always greater than $|C|$. Another consideration for the number of clients per tier is that too few clients in a tier may introduce training bias as these clients can be selected too often, causing overfitting on the data for these clients. One way to solve this issue is by adjusting the tier selection probabilities.

However, adjusting the tier selection probabilities results in different trade-offs. If the users' objective is to reduce the overall training time, they may increase the chances of selecting the faster tiers. However, drawing clients only from the fastest tier may inevitably introduce training bias due to the fact that different clients may own a diverse set of heterogeneous training data spread across different tiers; as a result, such bias may end up affecting the accuracy of the global model. To avoid such undesired behavior, it is preferable to involve clients from different tiers so as to cover a diverse set of training datasets. We perform an empirical analysis on the latency-accuracy trade-off in Section 5.

4.4 Adaptive Tier Selection Algorithm

While the above naive static selection method is intuitive, it does not provide a method to automatically tune the trade-off to optimize the training performance nor adjust the selection based on changes in the system. In this section, we propose an adaptive tier selection algorithm that can automatically strike a balance between training time and accuracy, and adapt the selection probabilities adaptively over training rounds based on the changing system conditions. The observation here is that heavily selecting certain tiers (e.g., faster tiers) may eventually lead to a biased model, TiFL needs to balance the client selection from other tiers (e.g., slower tiers). The question being which metric should be used to balance the selection. Given the goal here is to minimize the bias of the trained model, we can monitor the accuracy of each tier throughout the training process. A lower accuracy value of a tier t typically indicates that the model has been trained with less involvement of this tier, therefore tier t should contribute more in the next training rounds. To achieve this, we can increase the selection probabilities for tiers with lower accuracy. To achieve good training time, we also need to limit the selection of slower tiers across training rounds. Therefore, we introduce $Credits_t$, a constraint that defines how many times a certain tier can be selected.

Specifically, a tier is initialized randomly with equal selection probability. After the weights are received and the global model is updated, the global model is evaluated on every client for every tier on their respective $TestData$ and their resulting accuracies are stored as the corresponding tier t's accuracy for that round r. This is stored in A_t^r, which is the mean accuracy for all the clients in tier t in training round r. In the subsequent training rounds,

Algorithm 2 Adaptive Tier Selection Algorithm. $Credits_t$: the credits of Tier t, I: the interval of changing probabilities, $TestData_t$: evaluation dataset specific to that tier t, A_t^r: test accuracy of tier t at round r, τ: set of Tiers.

Aggregator: initialize weight w_0, $currentTier = 1$, $TestData_t$, $Credits_t$, equal probability with $\frac{1}{T}$, for each tier t.
for each round $r = 0$ **to** $N-1$ **do**
  if $r\%I == 0$ and r $\geq$ I **then**
    if $A^r_{currentTier} \leq A^{r-I}_{currentTier}$ **then**
      $NewProbs = ChangeProbs(A_1^r, A_2^r ... A_T^r)$
    end if
  end if
  while $True$ **do**
    $currentTier$ = (select one tier from T tiers with $NewProbs$)
    if $Credits_{currentTier} > 0$ **then**
      $Credits_{currentTier} = Credits_{currentTier} - 1$
      break
    end if
  end while
  C_r = (random set of $|C|$ clients from $currentTier$)
  for each client c $\in C_r$ **in parallel do**
    $w_r^c = TrainClient(c)$
    s_c = (training size of c)
  end for
  $w_r = \sum_{c=1}^{|C|} w_{r+1}^c * \frac{s_c}{\sum_{c=1}^{|C|} s_c}$
  for each t in τ **do**
    $A_t^r = Eval(w_r, TestData_t)$
  end for
end for
function ChangeProbs($AccuraciesByTier$)
  $A = SortAscending(AccuraciesByTier)$
  $D = n * (n-1)/2$ where n = # of tiers with $Credits_t > 0$
  $NewProbs = []$
  for each Index i, Tier t in A **do**
    $NewProbs[t] = (n-i)/D$
  end for
  return $NewProbs$
end function

the adaptive algorithm updates the probability of each tier based on that tier's test accuracy at every I rounds. This is done in the function *ChangeProbs*, which adjusts the probabilities such that the lower accuracy tiers get higher probabilities to be selected for training. With the new tier-wise selection probabilities ($NewProbs$), a tier that has remaining $Credits_t$ is selected from all available tiers τ. The selected tier will have its $Credits_t$ decremented. As clients from a particular tier gets selected over and over throughout the training rounds, the $Credits_t$ for that tier ultimately reduces down to zero, meaning that it will not be selected again in the future. This is a way of limiting the number of times a tier can be selected so as to control the training time by controlling the maximum number of times the slower tiers are selected. This serves as a control knob for the number of times a tier is selected and by setting this upper-bound, we can limit the amount of times a slower tier contributes to the training, thereby effectively gaining some control over setting a soft upper-bound on the total training time. For the straw-man implementation, we used a skewed probability of selection to manipulate training time. Since we now wish to adaptively change the probabilities, we add the $Credits_t$ to gain control over limiting training time.

On one hand, the tier-wise accuracy A_r^t essentially makes TiFL's adaptive tier selection algorithm *data heterogeneity aware*; as such, TiFL makes the tier selection decision by taking into account the underlying dataset selection biasness, and automatically adapt the tier selection probabilities over time. On the other hand, $Credits_t$ is introduced to intervene the training time by enforcing a constraint over the selection of the relatively slower tiers. While $Credits_t$ and A_t^r mechanisms optimize towards two different and sometimes contradictory objectives — training time and accuracy, TiFL cohesively synergizes the two mechanisms to strike a balance for the training time-accuracy trade-off. More importantly, with TiFL, the decision making process is automated, thus relieving the users from intensive manual effort. The adaptive algorithm is summarized in Algo. 2. One potential problem is that the uneven selection probability might impact the overall accuracy due to overfitting on data from particular tiers or training too long on tiers with "bad" data. We denote such tiers as "bad tiers". "Bad tier(s)" can result in lower accuracy on other tiers due to overfitting. In Algo. 2, we use the function ChangeProbs to change the selection probability of tiers. ChangeProbs sorts the tiers in ascending order based on their accuracies and assigns higher tier selection probabilities to the lower accuracy tiers and vice versa. Given "bad tier(s)" have higher accuracy, in the next rounds other tiers would get selected more often, and thus mitigate the overfitting impact of "bad tier(s)".

4.5 Training Time Estimation Model

In real-life scenarios, the training time and resource budget is typically finite. As a result, FL users may need to compromise between training time and accuracy. A training time estimation model would facilitate users to navigate the training time-accuracy trade-off curve to effectively achieve desired training goals.

Therefore, we build a training time estimation model that can estimate the overall training time based on the given latency values and the selection probability of each tier:

$$L_{all} = \sum_{i=1}^{n} (max(L_{tier\_i}) * P_i) * R. \quad (5)$$

where L_{all} is the total training time, L_{tier_i} is the response latency of all the clients in tier i, P_i is the probability of tier i, and R is the total number of training rounds. The model is a sum of products of the tier and maximum latency of each tier, which gives the latency expectation per round. This is multiplied by the total number of training rounds to get the total training time.

4.6 Discussion: Compatibility with Privacy-Preserving Federated Learning

FL has been used together with privacy preserving approaches such as *differential privacy* to prevent attacks that aim to extract private information [22, 26].

Privacy-preserving FL is based on client-level differential privacy, where the privacy guarantee is defined at each individual client. This can be accomplished by each client implementing a centralized private learning algorithm as their local training approach. For example, with neural networks this would be running one or more epochs using the approach proposed in [2], wherein each client

adds the appropriate noise into their local learning to protect the privacy of their individual datasets. Here we demonstrate that TiFL is compatible with such privacy preserving approaches.

Assume that for client c_i, one round of local training using a differentially private algorithm is (ϵ, δ)-differentially private, where ϵ bounds the impact any individual may have on the algorithm's output and δ defines the probability that this bound is violated. Smaller ϵ values therefore signify tighter bounds and provide a stronger privacy guarantee. Enforcing smaller values of ϵ requires more noise to be added to the model updates sent by clients to the FL server which leads to less accurate models. Selecting clients at each round of FL has distinct privacy and accuracy implications for client-level privacy-preserving FL approaches. Let us consider the scenario wherein a tier is chosen randomly each round according to a pre-defined probability distribution. Recall that each client adds differential privacy noise to its model updates every time it replies to a query and that this noise prevents any leakage of private information. In our approach, clients are queried according to a tier-based random process, where the number of times a client gets selected depends both on the tier selection policy and the inner-tier selection policy. Because of the composition property of differential privacy and the fact that each client adds its own noise, each client can monitor and control how their differential privacy budget is spent and thus their differential privacy guarantee. Therefore, it is possible for each client to meet their privacy requirements by stopping their replies if their budget has been consumed. In this fashion, it is possible to combine our approach and differential privacy. While additional optimizations to further reduce the amount of differential private noise (e,g. by incorporating techniques that would enable random sampling amplification [3]) are interesting research directions, these optimizations are orthogonal to the scope of this paper.

5 EXPERIMENTAL EVALUATION

We prototype TiFL with both the naive and our adaptive selection approach and perform extensive testbed experiments under three scenarios: resource heterogeneity, data heterogeneity, and resource plus data heterogeneity.

5.1 Experimental Setup

Testbed. As a proof of concept case study, we build a FL testbed for the syntehtic datasets by deploying 50 clients on a CPU cluster where each client has its own exclusive CPU(s) using Tensorflow [1]. In each training round, 5 clients are selected to train on their own data and send the trained weights to the server which aggregates them and updates the global model similar to [4, 20]. [4] introduces multiple levels of server aggregators in order to achieve scalability and fault tolerance in extreme scale situations, i.e., with millions of clients. In our prototype, we simplify the system to use a powerful single aggragator as it is sufficient for our purpose here, i.e., our system does not suffer from scalabiltiy and fault tolerance issues, though multiple layers of aggregator can be easily integrated into TiFL.

We also extend the widely adopted large scale distributed FL framework LEAF [7] in the same way. LEAF provides inherently non-IID with data quantity and class distributions heterogeneity. LEAF framework does not provide the resource heterogeneity among the clients, which is one of the key properties of any real-world FL system. The current implementation of the LEAF framework is a simulation of a FL system where the clients and server are running on the same machine. To incorporate the resource heterogeneity we first extend LEAF to support the distributed FL where every client and the aggregator can run on separate machines, making it a real distributed system. Next, we deploy the aggregator and clients on their own dedicated hardware. This resource assignment for every client is done through uniform random distribution resulting in equal number of clients per hardware type. By adding the resource heterogeneity and deploying them to separate hardware, each client mimics a real-world edge-device. Since we do not assume specific data distribution within each tier, we randomly distribute data across clients (i.e., clients can have very similar or widely different datasets per tier). Specifically, for LEAF, we use the data distribution provided by the framework. For Cifar10/MNIST/FMNIST, we set the level of Non-IIDness per client and distribute the data following [20]. The clients hosting these datasets are uniform randomly assigned to a node (for generating resource heterogeneity). Given that LEAF already provides non-IIDness, with the newly added resource heterogeneity feature the new framework provides a real world FL system which supports data quantity, quality and resource heterogeneity. For our setup, we use exactly the same sampling size used by the LEAF [7] paper (0.05) resulting in a total of 182 clients, each with a variety of image quantities. The test sets for all the datasets are generated through sampling 10% of the total data per client. As such, the test distribution is representative of the distribution of the training set.

Table 1: Scheduling Policy Configurations.

DataSet	Policy	Selection probabilities				
		Tier 1	Tier 2	Tier 3	Tier 4	Tier 5
Cifar10 / FEMNIST	*vanilla*	N/A	N/A	N/A	N/A	N/A
	slow	0.0	0.0	0.0	0.0	1.0
	uniform	0.2	0.2	0.2	0.2	0.2
	random	0.7	0.1	0.1	0.05	0.05
	fast	1.0	0.0	0.0	0.0	0.0
MNIST FMNIST	*vanilla*	N/A	N/A	N/A	N/A	N/A
	uniform	0.2	0.2	0.2	0.2	0.2
	fast1	0.225	F0.225	0.225	0.225	0.1
	fast2	0.2375	0.2375	0.2375	0.2375	0.05
	fast3	0.25	0.25	0.25	0.25	0.0

Table 2: Estimated VS Actual Training Time.

Policy	Estimated [s]	Actual [s]	MAPE [%]
slow	46242	44977	2.76
uniform	12693	12643	0.4
random	5143	5053	1.8
fast	1837	1750	5.01

5.2 Experimental Results

Models and Datasets. We use four image classification applications for evaluating TiFL. We use *MNIST and Fashion-MNIST [28]*, where each contains 60,000 training images and 10,000 test images, where each image is 28x28 pixels. We use a CNN model for both datasets, which starts with a 3x3 convolution layer with 32 channels

http://yann.lecun.com/exdb/mnist/

and ReLu activation, followed by a 3x3 convolution layer with 64 channels and ReLu activation, a MaxPooling layer of size 2x2, a fully connected layer with 128 units and ReLu activation, and a fully connected layer with 10 units and ReLu activation. Dropout 0.25 is added after the MaxPooling layer, dropout 0.5 is added before the last fully connected layer. We use *Cifar10* [16], which contains richer features compared to MNIST and Fashion-MNIST. There is a total of 60,000 colour images, where each image has 32x32 pixels. The full dataset is split evenly between 10 classes, and partitioned into 50,000 training and 10,000 test images. The model is a four-layer convolution network ending with two fully-connected layers before the softmax layer. It was trained with a dropout of 0.25. Lastly we also use the FEMNIST data set from LEAF framework [7]. This is an image classification dataset which consists of 62 classes and the dataset is inherently non-IID with data quantity and class distributions heterogeneity. We use the standard model architecture as provided in LEAF [6].

Training Hyperparameters. We use RMSprop as the optimizer in local training and set the initial learning rate (η) as 0.01 and decay as 0.995. Local batch size of each client is 10, and local epochs is 1. For CIFAR10 the total number of clients ($|K|$) is 50 and the number of participated clients ($|C|$) at each round is 5. For FEMNIST the number of total clients is 182, clients per round is 18 and default training parameters provided by the LEAF Framework (SGD with lr 0.004, batch size 10). We train for a total of 2000 rounds for FEMNIST and 500 rounds for the synthetic datasets. Every experiment is run 5 times to produce average values.

Heterogeneous Resource Setup. Among all the clients, we split them into 5 groups with equal clients per group. For MNIST and Fashion-MNIST, each group is assigned with 2 CPUs, 1 CPU, 0.75 CPU, 0.5 CPU, and 0.25 CPU per part respectively. For the larger Cifar10 and FEMINIST model, each group is assigned with 4 CPUs, 2 CPUs, 1 CPU, 0.5 CPU, and 0.1 CPU per part respectively. This leads to varying training time for clients belong to different groups. By using the tiering algorithm of TiFL, there are 5 tiers

Heterogeneous Data Distribution. FL differs from the datacenter distributed learning in that the clients involved in the training process may have non-uniform data distribution in terms of amount of data per client and the non-IID data distribution. • For *data quantity heterogeneity*, the training data sample distribution is 10%, 15%, 20%, 25%, 30% of total dataset for difference groups, respectively, unless otherwise specifically defined. • For *non-IID heterogeneity*, we use different non-IID strategies for different datasets. For MNIST and Fashion-MNIST, we adopt the setting in [20], where we sort the labels by value first, divide into 100 shards evenly, and then assign each client two shards so that each client holds data samples from at most two classes. For Cifar10, we shard the dataset unevenly in a similar way and limit the number of classes to 5 per client (non-IID(5)) following [31], [19] unless explicitly mentioned otherwise. In the case of FEMINIST we use its default non-IID-ness.

Scheduling Policies. We evaluate several different naive scheduling policies of the proposed tier-based selection approach, defined by the selection probability from each tier, and compare it with the state-of-the-practice policy (or no policy) that existing FL works adopt, i.e., randomly select 5 clients from all clients in each round [4, 20], agnostic to any heterogeneity in the system. We name it as *vanilla*. *fast* is a policy that TiFL only selects the fastest clients in each round. *random* demonstrates the case where the selection of the fastest tier is prioritized over slower ones. *uniform* is a base case for our tier-based naive selection policy where every tier has an equal probability of being selected. *slow* is the worst policy that TiFL only selects clients from the slowest tiers and we only include it here for reference purpose so that we can see a performance range between the best case and the worst case scenarios for static tier-based selection approach. We use the above policies for CIFAR-10 and FEMINIST training. For MNIST and Fashion-MNIST, given it is a much more lightweight workload, we focus on demonstrating the sensitivity analysis when the policy prioritizes more aggressively towards the fast tier, i.e., from *fast1* to *fast3*, the slowest tier's selection probability has reduced from 0.1 to 0 while all other tiers got equal probability. We also include the *uniform* policy for comparison, which is the same as in CIFAR-10. Table 1 summarizes all these scheduling policies by showing their selection probabilities.

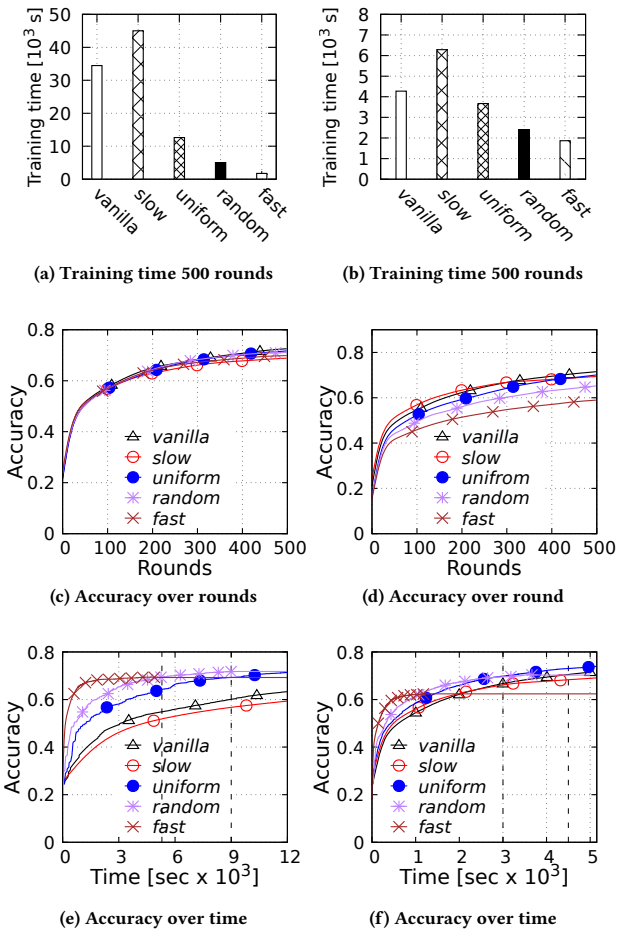
5.2.1 Training Time Estimation via Analytical Model. In this section, we evaluate the accuracy of our training time estimation model on different naive tier selection policies by comparing the estimation results of the model with the measurements obtained from test-bed experiments. The estimation model takes as input of the profiled average latency of each tier, the selection probabilities, and total number of training rounds to estimate the training time. We use mean average prediction error (MAPE) as the evaluation metric, which is defined as follows:

$$\text{MAPE} = \frac{|L_{all}^{est} - L_{all}^{act}|}{L_{all}^{act}} * 100, \tag{6}$$

where L_{all}^{est} is the estimated training time calculated by the estimation model and L_{all}^{act} is the actual training time measured during the training process. Table 2 demonstrates the comparison results. The results suggest the analytical model is very accurate as the estimation error never exceeds more than 6 % with slight variations occurring due to system randomness.

5.2.2 Resource Heterogeneity. In this sections, we evaluate the performance of TiFL with static selection policies in terms of training time and model accuracy in a resource heterogeneous environment as depicted in 5.1 and we assume there is no data heterogeneity. We evaluated TiFL with adaptive selection policy in section 5.2.5. In practice, data heterogeneity is a norm in FL, we evaluate this scenario to demonstrate how TiFL tame resource heterogeneity alone and we evaluate the scenario with both resource and data heterogeneity in Section 5.2.4.

In the interest of space, we only present the Cifar10 results here as MNIST and Fashion-MNIST share the similar observations. The results are organized in Fig. 3 (column 1), which clearly indicate that when we prioritize towards the fast tiers, the training time reduces significantly. Compared with *vanilla*, *fast* achieves almost 11 times improvement in training time, see Fig. 3 (a). One interesting observation is that even *uniform* has an improvement of over 6 times over the *vanilla*. This is because the training time is always bounded by the slowest client selected in each training round. In



(a) Training time 500 rounds (b) Training time 500 rounds

(c) Accuracy over rounds (d) Accuracy over round

(e) Accuracy over time (f) Accuracy over time

Figure 3: Comparison results for different selection policies on Cifar10 with resource heterogeneity (0.5 to 4 CPUs) and homogenous data quantity (Column 1), and data quantity heterogeneity with with homogenous resources (2 CPUs per client) (Column 2).

TiFL, selecting clients from the same tier minimizes the straggler issue in each round, and thus greatly improves the training time. While there can be variation in training time among the clients, it can be mitigated by tuning the number of tiers used for client binning. For accuracy comparison, Fig. 3 (c) shows that the difference between polices are very small, i.e., less than 3.71% after 500 rounds. However, if we look at the accuracy over wall-clock time, TiFL achieves much better accuracy compared to *vanilla*, i.e., up to 6.19% better if training time is constraint, thanks to the much faster per round training time brought by TiFL, see Fig. 3 (e). Note here that different policies may take very different amount of wall-clock time to finish 500 rounds. It is worth pointing out that the accuracy of FL is expected to be lower than traditional distributed machine learning due to the skewed data distribution among the clients as well as the different batching and gradients aggregation methods [15].

5.2.3 Data Heterogeneity. In this section, we evaluate data heterogeneity due to both *data quantity heterogeneity* and *non-IID heterogeneity* as depicted in Section 5.1. To demonstrate only the impact from data heterogeneity, we allocate homogeneous resource to each client, i.e., 2 CPUs per client.

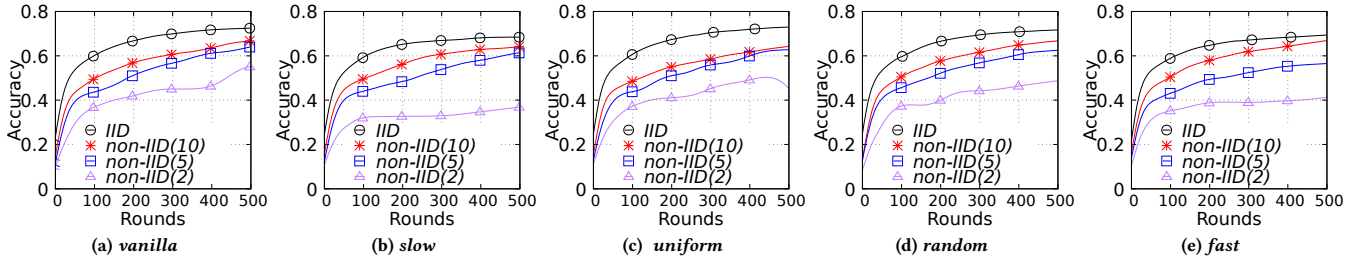
• *Data quantity heterogeneity.* The training time and accuracy results are show in Fig. 3 (column 2). In the interest of space, we only show Cifar10 results here. From the training time comparison in Fig. 3 (b), it is interesting that TiFL also helps in data heterogeneity only case and achieves up to 3 times speedup. The reason is that *data quantity heterogeneity* may also result in different round time, which shares the similar effect as resource heterogeneity. Fig. 3 (d) and (f) show the accuracy comparison, where we can see *fast* has relatively obvious drop compared to others because Tier 1 only contains 10% of the data, which is a significant reduction in volume of the training data. *slow* is also a heavily biased policy towards only one tier, but Tier 5 contains 30% of the data thus *slow* maintains good accuracy while worst training time. These results imply that like resource heterogeneity only, data heterogeneity only can also benefit from TiFL. However, policies that are too aggressive toward faster tier needs to be used very carefully as clients in fast tier achieve faster round time due to using less samples. It is also worth pointing out that in our experiments the total amount of data is relatively limited. In a practical case where data is significantly more, the accuracy drop of *fast* is expected to be less pronounced.

• *non-IID heterogeneity.* We observe that non-IID heterogeneity does not impact the training time. Hence, we omit the results here. However, non-IID heterogeneity effects the accuracy. Fig. 4 shows the accuracy over rounds given 2, 5, and 10 classes per client in a non-IID setting. We also show the IID results in plot for comparison. These results show that as the heterogeneity level in non-IID heterogeneity increases, the accuracy impact also increases for all policies due to the strongly biased training data. Another important observation is that *vanilla* case and *uniform* have a better resilience than other policies, thanks to the unbiased selection behavior, which helps minimize further bias introduced during the client selection process.

5.2.4 Resource and Data Heterogeneity. This section presents the most practical case study with static selection policies as we evaluate with both resource and data heterogeneity combined. We evaluated TiFL for both resource and data heterogeneity combined with adaptive selection policy in section 5.2.5.

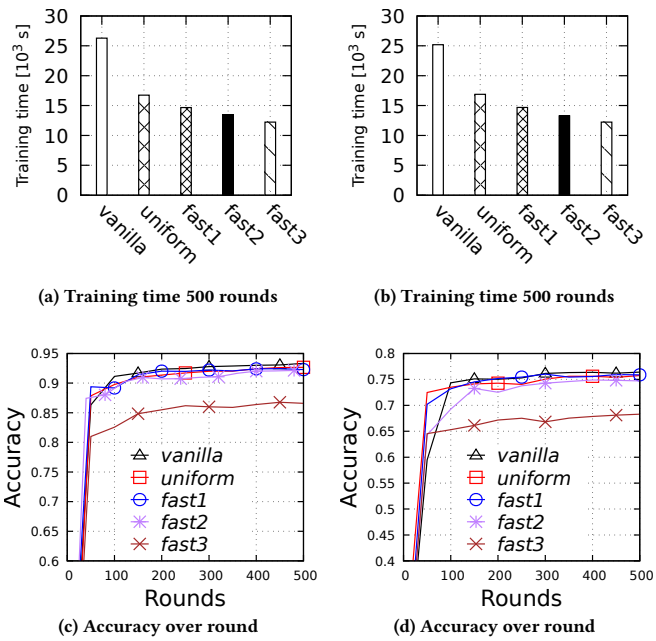
MNIST and **Fashion-MNIST (FMNIST)** results are shown in Fig. 5 columns 1 and 2 respectively. Overall, policies that are more aggressive towards the fast tiers bring more speedup in training time. For accuracy, all polices of TiFL are close to *vanilla*, except *fast3* falls short as it completely ignores the data in Tier 5.

Cifar10 results are shown in Fig. 6 column 1. It presents the case of resource heterogeneity plus non-IID data heterogeneity with equal data quantities per client and the results are similar to resource heterogeneity only since non-IID data with the same amount of data quantity per client results in a similar effect of resource heterogeneity in terms of training time. However, the accuracy degrades slightly more here as because of the non-IIDness the features are skewed, which results in more training bias among different classes.



(a) *vanilla* (b) *slow* (c) *uniform* (d) *random* (e) *fast*

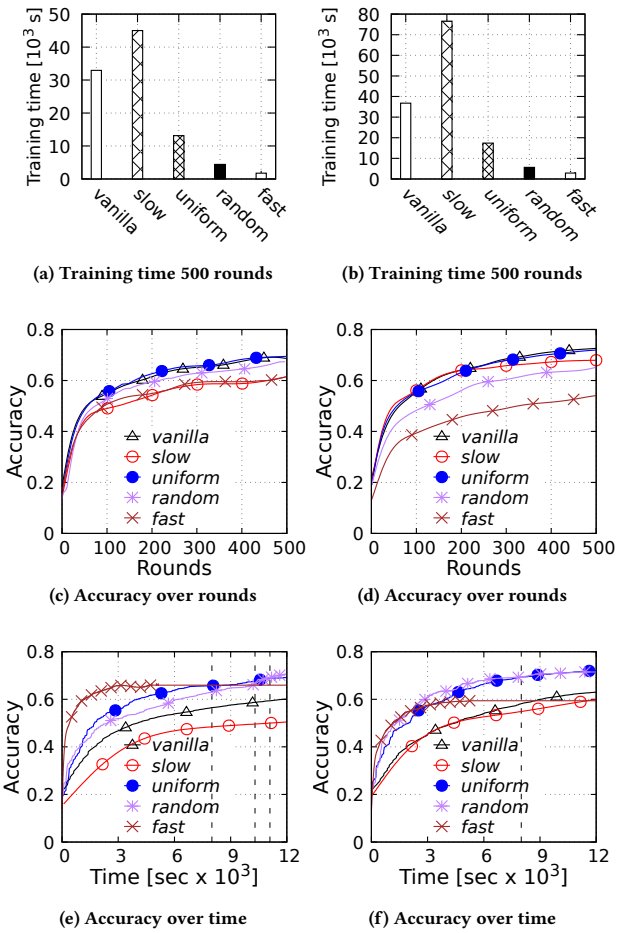
Figure 4: Comparison results for different selection policies on Cifar10 with different levels of non-IID heterogeneity (Class) and fixed resources.



(a) Training time 500 rounds (b) Training time 500 rounds

(c) Accuracy over round (d) Accuracy over round

Figure 5: Comparison results for different selection policies on MNIST (Column 1) and FMNIST (Column 2) with resource plus data heterogeneity.



(a) Training time 500 rounds (b) Training time 500 rounds

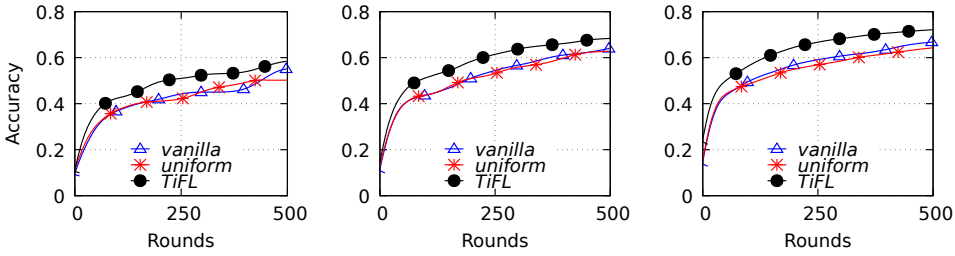
(c) Accuracy over rounds (d) Accuracy over rounds

(e) Accuracy over time (f) Accuracy over time

Figure 6: Comparison results for different selection policies on Cifar10 with resource plus non-IID heterogeneity heterogeneity (Column 1) and resource, data quantity, and non-IID heterogeneity heterogeneity (Column 2).

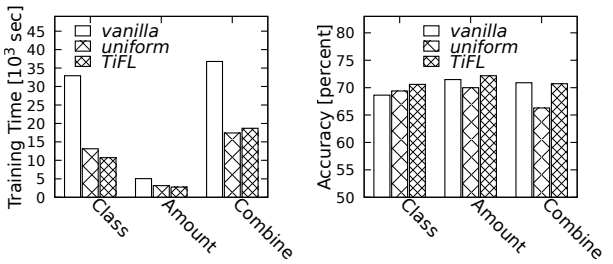
Fig. 6 column 2 shows the case of resource heterogeneity plus both the data quantity heterogeneity and non-IID heterogeneity. As expected, the training time shown in Fig. 6 (b) is similar to Fig. 6 (a) since the training time impact from different data amounts can be corrected by TiFL. However, the behaviors of round accuracy are quite different here as shown in Fig. 6 (d). The accuracy of *fast* has degraded a lot more due to the data quantity heterogeneity as it further amplifies the training class bias (i.e., the data of some classes become very little to none) in the already very biased data distribution caused by the non-IID heterogeneity. Similar reasons can explain for other policies The best performing policy in accuracy here is the *uniform* case and is almost the same as *vanilla*, thanks to the even selection nature which results in little increase in training class bias. Fig. 6 (f) shows the wall-clock time accuracy. As expected, the significantly improved per round time in TiFL shows its advantage here as within the same time budget, more iterations can be done with shorter round time and thus remedies the accuracy disadvantage per round. *fast* still falls short than *vanilla* in the long run as the limited and biased data limits the benefits of more iterations. *fast* also perform worse than *vanilla* as it has no training advantage.

5.2.5 Adaptive Selection Policy. The above evaluation demonstrate the naive selection approach in TiFL can significantly improve the training time, but sometimes can fall short in accuracy, especially



(a) 2-class per client (b) 5-class per client (c) 10-class per client

Figure 7: Comparison results of Cifar10 under non-IID heterogeneity (Class) for different client selection policies with fixed resources (2 CPUs) per client.



(a) Training time for 500 rounds (b) Accuracy at 500 rounds

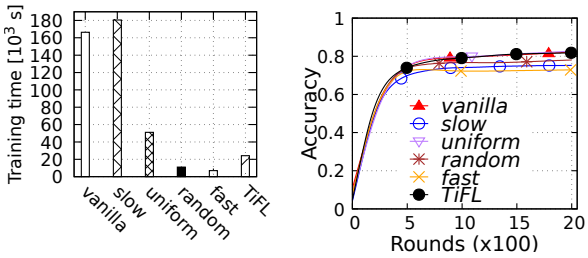
Figure 8: Comparison results for different selection policies on Cifar10 with data quantity heterogeneity (Amount), non-IID heterogeneity (Class), and resource plus data heterogeneity (Combine).

when strong data heterogeneity presents as such approach is data-heterogeneity agnostic. In this section, we evaluate the proposed *adaptive* tier selection approach of TiFL, which takes into consideration of both resource and data heterogeneity when making scheduling decisions without privacy violation. We compare *adaptive* with *vanilla* and *uniform*, and the later is the best accuracy performing static policy.

Fig. 8 shows *adaptive* outperforms *vanilla* and *uniform* in both training time and accuracy for resource heterogeneity with data quantity heterogeneity (Amount) and non-IID heterogeneity (Class), thanks to the data heterogeneity-aware schemes. In the combined resource and data heterogeneity case (Combine), *adaptive* achieves comparable accuracy with *vanilla* with almost half of the training time and a slightly higher training time compared to uniform. The time difference arises when the adaptive policy tries to balance training time and accuracy, i.e. the 10% difference in training time is for the tradeoff of achieving around 5% better accuracy. The other policy which achieves this accuracy is *vanilla*, which has almost 2x more training time. Considering this, we note that the training time difference is not significant, and performs similar as *uniform* in training time while improves significantly in accuracy.

The above robust performance of *adaptive* is credited to both the resource and data heterogeneity-aware schemes. To demonstrate the robustness of *adaptive*, we compare the accuracy over rounds for different policies under different non-IID heterogeneity in Fig. 7. It is clear that *adaptive* consistently outperforms *vanilla* and *uniform* in different level of non-IID heterogeneity.

5.2.6 Adaptive Selection Policy (LEAF). This section provides the evaluation of TiFL using a widely adopted large scale distributed FL dataset FEMINIST from the LEAF framework [7] . We use exactly the same configurations (data distribution, total number of clients, model and training hyperparameters) as mentioned in [7] resulting in total number of 182 clients, i.e. deploy-able edge devices. Since LEAF provides it's own data distribution among devices the addition of resource heterogeneity results in a range of training times thus generating a scenario where every edge device has a different training latency. We further incorporated TiFL's tiering module and selection policy to the extended LEAF framework. The profiling modules collects the training latency of each clients and creates a logical pool of tiers which is further utilized by the scheduler. The scheduler selects a tier and then the edge clients within the tier in each training round. For our experiments with LEAF we limit the total number of tiers to 5 and during each round we select 10 clients, with 1 local epoch per round.



(a) Training time for 2000 rounds (b) Accuracy over rounds

Figure 9: Comparison results for different selection policies on LEAF with default data heterogeneity (quantity, non-IID heterogeneity), and resource heterogeneity.

Figure 9 shows the training time and accuracy over rounds for LEAF with different client selection policies. Figure 9a shows the training time for different selection policies. The least training time is achieved by using the *fast* selection policy however, it impact the final model accuracy by almost 10% compared to *vanilla* selection policy. The reason for the least accuracy for *fast* is the result of less training point among the clients in tier 1. One interesting observation is *slow* out performs the selection policy *fast* in terms of accuracy even though each of these selection policies rely on data from only one tier. It must be noted that the slow tier is not only the reason of less computing resources but also the higher

quantity of training data points. These results are consistent with our observations from the results presented in Section 5.2.3.

Figure 9b shows the accuracy over-rounds for different selection policies. Our proposed *adaptive* selection policy achieves 82.1% accuracy and outperforms the *slow* and *fast* selection policies by 7% and 10% respectively. The *adaptive* policy is on par with the *vanilla* and *uniform* (82.4% and 82.6% respectively). when comparing the total training time for 2000 rounds *adaptive* achieves 7 × and 2 × improvement compare to *vanilla* and *uniform* respectively. *fast* and *random* both outperformed the *adaptive* in terms of training time however, even after convergence the accuracy for both of these selection policies show a noticeable impact on the final model accuracy. The results for FEMNIST using the extended LEAF framework for both accuracy as well as training time are also consistent with the results reported in Section 5.2.5.

6 CONCLUSION

In this paper, we investigate and quantify the heterogeneity impact on "decentralized virtual supercomputer" - FL systems. Based on the observations of our case study, we propose and prototype a Tier-based Federated Learning System called TiFL. Tackling the resource and data heterogeneity, TiFL employs a tier-based approach that groups clients in tiers by their training response latencies and selects clients from the same tier in each training round. To address the challenge that data heterogeneity information cannot be directly measured due to the privacy constraints, we further design an *adaptive* tier selection approach that enables TiFL be data heterogeneity aware and outperform conventional FL in various heterogeneous scenarios: *resource heterogeneity*, *data quantity heterogeneity*, *non-IID data heterogeneity*, and their combinations. Specifically, TiFL achieves an improvement over conventional FL by up to 3× speedup in overall training time and by 6% in accuracy.

ACKNOWLEDGMENTS

We are grateful to our shepherd, Prateek Sharma, as well as the anonymous reviewers, for their valuable comments and suggestions that significantly improved the paper. This work is sponsored in part by NSF under CCF-1756013, IIS-1838024, CCF-1919075, CCF-1919113, and AWS Cloud Research Grants.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[3] Amos Beimel, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. 2010. Bounds on the sample complexity for private learning and private data release. In *Theory of Cryptography Conference*. Springer, 437–454.

[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. 2019. Towards Federated Learning at Scale: System Design. *arXiv preprint arXiv:1902.01046* (2019).

[5] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1175–1191.

[6] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. 2018. Expanding the Reach of Federated Learning by Reducing Client Resource Requirements. *arXiv preprint arXiv:1812.07210* (2018).

[7] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2018. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097* (2018).

[8] Zheng Chai, Hannan Fayyaz, Zeshan Fayyaz, Ali Anwar, Yi Zhou, Nathalie Baracaldo, Heiko Ludwig, and Yue Cheng. 2019. Towards Taming the Resource and Data Heterogeneity in Federated Learning. In *2019 {USENIX} Conference on Operational Machine Learning (OpML 19)*. 19–21.

[9] Avishek Ghosh, Justin Hong, Dong Yin, and Kannan Ramchandran. 2019. Robust Federated Learning in a Heterogeneous Environment. *arXiv preprint arXiv:1906.06629* (2019).

[10] Aaron Harlap, Henggang Cui, Wei Dai, Jinliang Wei, Gregory R Ganger, Phillip B Gibbons, Garth A Gibson, and Eric P Xing. 2016. Addressing the straggler problem for iterative convergent parallel ML. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 98–111.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] https://github.com/PaddlePaddle/PaddleFL 2020. PaddleFL. [Online; accessed 21-January-2020].

[13] https://www.tensorflow.org/federated 2020. Tensorflow Federated. [Online; accessed 21-January-2020].

[14] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[15] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* 55 (2014).

[17] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2019. Federated learning: Challenges, methods, and future directions. *arXiv preprint arXiv:1908.07873* (2019).

[18] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2018. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127* (2018).

[19] Lumin Liu, Jun Zhang, SH Song, and Khaled B Letaief. 2019. Edge-Assisted Hierarchical Federated Learning with Non-IID Data. *arXiv preprint arXiv:1905.06641* (2019).

[20] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

[21] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

[22] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910* (2018).

[23] Takayuki Nishio and Ryo Yonetani. 2019. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[24] Jacquelyn K O'herrin, Norman Fost, and Kenneth A Kudsk. 2004. Health Insurance Portability Accountability Act (HIPAA) regulations: effect on medical record research. *Annals of surgery* 239, 6 (2004), 772.

[25] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. 2018. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017* (2018).

[26] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[27] Colin Tankard. 2016. What the GDPR means for businesses. *Network Security* 2016, 6 (2016), 5–8.

[28] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[29] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 12.

[30] Chengliang Zhang, Huangshi Tian, Wei Wang, and Feng Yan. 2018. Stay Fresh: Speculative Synchronization for Fast Distributed Machine Learning. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 99–109.

[31] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. 2018. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582* (2018).