# Optimize Scheduling of Federated Learning on Battery-powered Mobile Devices

Cong Wang[1,*], Xin Wei[1], and Pengzhan Zhou[2]
[1]*Dept. of Computer Science, Old Dominion University, Norfolk, VA 23529, USA*
[2]*Dept. of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA*

***Abstract*—Federated learning learns a collaborative model by aggregating locally-computed updates from mobile devices for privacy preservation. While current research typically prioritizing the minimization of communication overhead, we demonstrate from an empirical study, that computation heterogeneity is a more pronounced bottleneck on battery-powered mobile devices. Moreover, if class is unbalanced among the mobile devices, inappropriate selection of participants may adversely cause gradient divergence and accuracy loss. In this paper, we utilize data as a tunable knob to schedule training and achieve near-optimal solutions of computation time and accuracy loss. Based on the offline profiling, we formulate optimization problems and propose polynomial-time algorithms when data is class-balanced or unbalanced. We evaluate the optimization framework extensively on a mobile testbed with two datasets. Compared with common benchmarks of federated learning, our algorithms achieve 2-10$\times$ speedups with negligible accuracy loss. They also mitigate the impact from mobile stragglers and improve parallelism for federated learning.**

***Keywords*-Federated learning; on-device deep learning; scheduling optimization; non-IID data.**

## I. Introduction

The past few years have witnessed an increasing migration of data-driven applications from the centralized cloud to mobile devices due to the rising privacy concerns. Originated from distributed learning [1], *Federated Learning* (FL) learns a centralized model where the training data is held privately by end users [2]–[10]. They compute local models in parallel and aggregate their updates towards a centralized *parameter server*. The server takes the average from the users, pushes the averaged model back to all the users as the initial point for the next iteration.

Though promising to ease the tension between data utility and privacy, existing research mainly focuses on addressing prominent problems left from distributed learning such as improving the communication efficiency [6]–[9] and security robustness [13], [14]. Their proof-of-concept implementations are largely based on cloud/edge servers with stable, external power and proprietary GPUs. This still leaves a gap to what FL was initially proposed as a learner of abstraction at the mobile data source for privacy preservation.

Fortunately, the dramatic increase of mobile processing power has enabled not only inference, but also moderate training (backpropagation) [5], [16], thus providing a basis to launch FL on mobile devices. However, the vast heterogeneity of mobile processing power has yet to be addressed in [2]–[10]. From an empirical study, we first validate that the bottleneck has actually shifted from communication back to computation on mobile devices. The runtime behavior depends on a complex combination of CPU architectures, memory speed, power management policies and computation intensity. Further, the high variance among user data adds another layer of statistical heterogeneity [10]. E.g., in activity recognition, some users may perform only a few actions (e.g., sitting for a long time), thus leading to highly skewed local distributions in a small subset of categories, which breaks the independent and identically distributed (IID) assumptions held as a basis for machine learning. When averaged into the global model, these skewed gradients may have a damaging impact on the collaborative model. Thus, the selection of participants should rely on both computation and data distribution, whereas the previous works typically discuss non-IIDness independently [2]–[4].

To tackle these challenges, we propose an optimization framework to schedule training using the workloads (amount of training data) as a tunable knob and achieve near-optimal staleness in synchronous gradient aggregation. We first build a performance profiler offline to characterize the relations between training time and data size/model parameters using a multiple linear regressor for the mobile devices. Then we start with the base case when data is IID (class-balanced), and formulate the problem into a min-max optimization problem to seek optimal partitioning of data that achieves minimum makespan. We propose an efficient $\mathcal{O}(n^2 \log n)$ algorithm ($n$ is the number of users) [24]. For non-IID data, we introduce a new cost of accuracy and re-formulate the problem to minimize the average sum of computation time and accuracy cost, balanced by a weight parameter. We develop an $\mathcal{O}(mn)$ algorithm to assign workloads with the minimum average cost in each step based on a variant of the bin packing problem [27] ($m$ is the number of data shards). The algorithm can also actively include unseen class samples during assignment to improve gradient diversity and model generalization [21]. The proposed algorithms are evaluated on MNIST and CIFAR10 datasets with a mobile testbed of various smartphone models. The main contributions are summarized below.

- We perform an empirical study by launching backpropagation on Android and discover that the processing time is dominated by computation with the stragglers having twice time delay than the mean time of completion.
- We formulate the optimization problems for both IID and non-IID data, and propose polynomial-time algorithms.
- We conduct extensive evaluations on MNIST and CIFAR10 datasets under a mobile testbed with three device combinations. Compared to the benchmarks, the results show an average of 5-10$\times$ speedups for IID data without accuracy loss and 2-5$\times$ speedups for non-IID data with no accuracy loss on MNIST and 0.01-0.02 accuracy loss on CIFAR10. The algorithms also demonstrate advantages of avoiding worse-case stragglers and improve parallelism.

*Corresponding author: Cong Wang, c1wang@odu.edu

1530-2075/20/$31.00 ©2020 IEEE
DOI 10.1109/IPDPS47924.2020.00031

The rest of the paper is organized as follows. Section II describes the background and related works. Section III motivates this work with an empirical study. Section V and VI formulate the problems and propose algorithms for IID and non-IID data. Section VII evaluates the framework on real testbed, dataset and Section VIII concludes this work.

## II. Background and Related Works

### A. Deep Learning on Mobile Devices

The rapid increase of processing power, battery capacity and improvement of power management these years make mobile devices capable of handling complex learning tasks not only limited to logistical regression or supported vector machine, but also the resource-intensive deep neural networks. The previous efforts mainly focus on optimizing the one-shot *inference* computation via compression [15]. Recently, there are new efforts to incorporate the entire *training* process on mobile devices for better privacy preservation and adaptation to the shifts in data distribution [16]. Their implementation demonstrates the feasibility of conducting backpropagation for deep neural networks on battery-powered mobile devices, which has laid the foundations of this paper.

The collaboration of mobile devices exhibits more heterogeneity to their counterparts in distributed cloud and edge computing. Like any other applications in the userspace, the learning process is also handled by the Linux kernel of Android, which controls `cpufreq` by the CPU governor in response to the workload. For example, the default `interactive` governor in Android 8.0 scales the clockspeed over the course of a timer regarding the workload. For better energy-efficiency and performance, smartphones on the market are embedded with asymmetric multiprocessing architectures, i.e., ARM's big.LITTLE [17]. For instance, Nexus 6P is powered by octa-core CPUs with four big cores running at 2.0 GHz and four little ones at 1.53 GHz. The design intuition is to handle the bursty nature of user interactions with the device by placing low-intensity tasks on the small cores, and vice versa. However, the behavior of such subsystem facing intensive, constant workload such as backpropagation remains underexplored. Further, since vendors usually extend over the vanilla task scheduler through proprietary designs of task migration/packing, load tracking and frequency scaling, the same learning task would have heterogenous processing time depending on the hardware and system-level implementation. Our goal is to mitigate such impact on FL while still using the default governor and scheduler for applications in the userspace.

### B. Federated Learning

A promising way of addressing staleness in distributed learning is using asynchronous updates [11], [12], which resumes computation on those faster nodes without waiting for the stragglers. However, inconsistent gradients could easily lead to divergence and amortize the savings in computation time. Therefore, most of the FL frameworks nowadays advocate the synchronous approach [2]–[8]. McMahan et al. introduce the algorithm of FedAvg that averages on aggregated parameters and demonstrate its effectiveness to learn from unbalanced, non-IID data [2]. Non-IIDness is further discussed in [3], [4], which propose remedies to either pre-share a subset of global data or utilize generative models to restore class balance, but at non-negligible computation, communication and coordination efforts. Scalable FL is approached in [5] from a system design perspective; however, it simply adopts a hard dropout of the stragglers if they fail to catch up with the schedule, while not attempting to make best use from their data.

Another thread of works address the communication efficiency in FL [6]–[10]. The full model is compressed and cast into a low-dimensional space for bandwidth saving in [6]. Local updates that diverge from the global model are identified and excluded to avoid unnecessary communication [7]. Decentralized approaches and convergence are discussed in [8] when users only exchange gradients with their neighbors. Evolutionary algorithm is explored to minimize communication cost and test error in a multi-objective optimization [9]. The challenges from system, statistics and fault tolerance are jointly formulated into a unified multi-task learning framework [10]. Our work complements these efforts from the computation and system optimization side to reduce major heterogeneity on mobile devices.

Our study has fundamental difference from a large body of works in scheduling paralleled machines [18]–[20]. First, rather than targeting at jobs submitted by cloud users, we delve into a more microcosmic level and jointly consider partitioning a learning task and makespan minimization, where the cost function is characterized from real experimental traces. Second, FL calls for the scheduling algorithm to be aware of non-IIDness and model accuracy when workloads are partitioned. Hence, our work is among the first to address computational and statistical heterogeneity on mobile devices.

TABLE I: Hardware configurations of benchmarking testbed.

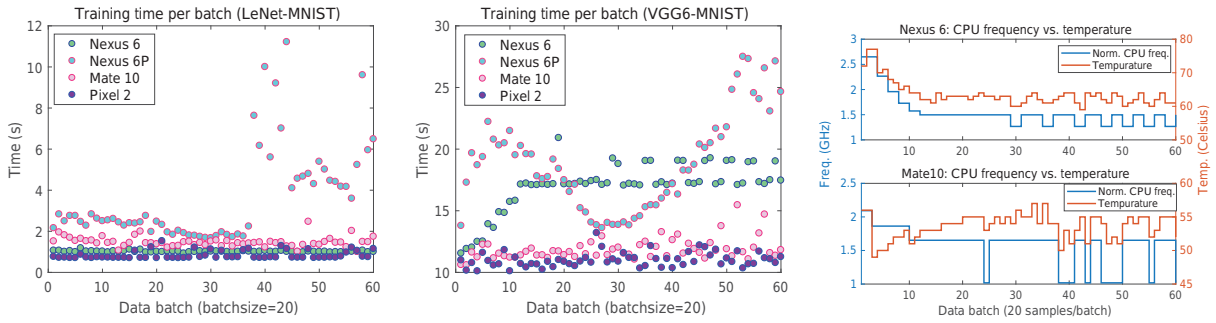
| model | SoC | CPU | big.LITTLE |
|---|---|---|---|
| Nexus 6 | Snapdragon 805 | 4×2.7GHz | ✗ |
| Nexus 6P | Snapdragon 810 | 4×1.55 GHz<br>4×2.0 GHz | ✓ |
| Mate10 | Kirin 970 | 4×2.36GHz<br>4×1.8GHz | ✓ |
| Pixel2 | Snapdragon 835 | 4×2.35 GHz<br>4×1.9 GHz | ✓ |

## III. Motivation and Measurement

### A. Processing Time

Most of the existing mechanisms equally partition data among the workers for load balance [2]. We motivate by an empirical study using the testbed shown in Table I and see how large the gap is between the stragglers and mean execution time. We develop a training App with DL4J [30] (see Section VII for more development details) and implement both LeNet [25] with 205K parameters and VGG6 [26] with 5.45M parameters.

**Computation Time.** We trace the per-batch training time on different devices as shown in Figs. 1(a-b), as well as the CPU clock speed every 5s vs. the temperature in Fig. 1(c). Though the CPU can switch frequencies much faster, this experiment




(a) (b) (c)

Fig. 1: Benchmark training performance on the mobile testbed using the MNIST dataset (a) LeNet (b) VGG6 (c) average CPU frequency vs. temperature.

| | | 3K samples | | 6K samples | |
|---|---|---|---|---|---|
| | | WiFi | LTE | WiFi | LTE |
| LeNet | Nexus6 | 31(1.5%) | 32(6.7%) | 62(0.8%) | 63(3.4%) |
| | Nexus6P | 69(0.7%) | 71(3%) | 220(0.2%) | 222(1.0%) |
| | Mate10 | 45(1.0%) | 47(4.6%) | 89(0.5%) | 91(2.4%) |
| | Pixel2 | 25(1.8%) | 27(7.9%) | 51(0.9%) | 53(4.0%) |
| VGG6 | Nexus6 | 495(2.5%) | 539(10.4%) | 1021(1.2%) | 1065(5.3%) |
| | Nexus6P | 540(2.3%) | 584(9.6%) | 1134(1.1%) | 1178(4.8%) |
| | Mate10 | 359(0.1%) | 403(0.5%) | 712(7.9%) | 756(7.4%) |
| | Pixel2 | 339(3.6%) | 383(14.7%) | 661(1.9%) | 705(8.0%) |

TABLE II: Training time of MNIST samples per epoch (s) with percentage representing the network communication overhead.

shows how the frequency and temperature interact over time to reach stability under the power management policy.

**Communication Time.** The server pushes (pulls) the model to (from) the devices in each epoch. We measure the transmission time of the LeNet (2.5MB) and VGG6 (65.4MB) model under 1 Gbps wireless link and T-mobile 4G LTE (-94 dBm), to simulate different networking environments. The WiFi uplink/downlink speed achieves around 80-90 Mbps on our campus network and LTE reaches about 60 Mbps and 11 Mbps for the uplink and downlink respectively. We upload/download the model from an AWS server between Washington D.C. and Norfolk, VA. By adding up the time of communication and computation, the makespan for each device is summarized in Table II. Based on the measurements, we have the following observations.

**Observation 1.** The computation time is generally governed by the processing power of the CPU and the mobile architecture, but with a few exceptions.

The actual speed is also affected by the system-level implementation from different vendors. For example, one may expect the newer smartphone generations to be more powerful. This is not always true: Nexus 6 back in 2014 were not designed for intensive workload like neural computations; surprisingly, Huawei Mate10 with octa cores still lags behind Nexus 6 running LeNet as observed in Fig. 1(a), though their average CPU frequency stabilizes around the same region in Fig. 1(c). Nexus 6 offers over 3× speed than Mate10. This observation suggests that old generations may still have opportunities to outperform the new ones depending on the computation intensity.

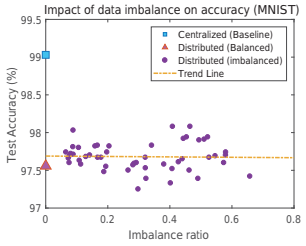
**Observation 2.** The continuous neural computation leads to heating and the governor quickly reacts to reduce the `cpufreq`, or even shuts down some cores, thereby causing a performance hit with large variance in the subsequent batch iterations (especially running heavy-weight networks like VGG6 on Nexus6/6P).

An extreme case is Nexus 6P with the controversial Snapdragon 810 chipset [22]. We find that the little cores are running at 70-80% and the big cores are below 50% utilization to their maximum frequency. The big cores quickly go offline and switch to the little cores after a moderate temperature surge, that occurs fairly often during the testing. The big cores never stay around their maximum frequency at 2.0 GHz, thus making Nexus 6P even much slower than Nexus 6.

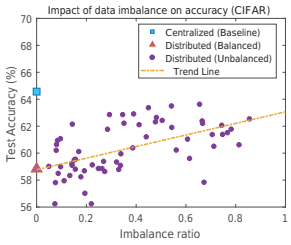
**Observation 3.** In contrast to the conjectures in [2], communication only takes a small portion of the training time about 5% on average (maximum at 15% with VGG6 using LTE). This confirms that with today's networking speed and the upcoming 5G, the bottleneck of FL remains to be computation on mobile devices. Part of the reason is also because the mobile cannot host heavy-weight model architectures that may overwhelm the memory capacity. This implicitly alleviates the communication overhead of transmitting these models.

**Observation 4.** To process the same amount of data, the mobile devices exhibit substantial diversity in their completion time. For example, the straggler requires an additional 62% and 109% time running LeNet and VGG6 compared to the mean completion time (maximum time minus the mean time in Table II), and this deviation is expected to get larger with more complex models or data iterations.

As observed in Table II, for only processing 5% and 10% of MNIST, 10 epoches lead to 30 mins and 1 hour delay due to stragglers. Therefore, an appropriate scheduling mechanism is needed. In cloud environment, stragglers may be caused by resource contention, unbalanced workload or displacement of workers on different parameter servers [19], which are typically handled by *load balancing*. In mobile environment, they are caused by the fundamental disparity in device's processing power, so can we do the opposite and leverage *load unbalancing* to offset the speed of those stragglers? Since each epoch requires a full pass of the local data, among a variety of tunable knobs, workload is directly proportional to the amount of training data. However, distributed learning often assumes a balanced data partition among the workers [2],

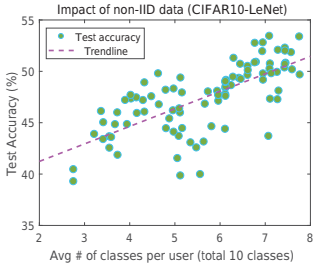



(a)

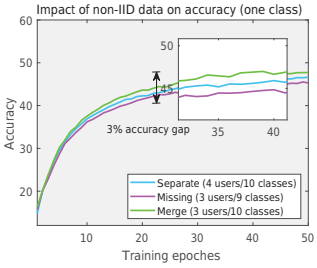



(b)

Fig. 2: Impact of data imbalance (still IID) on FL accuracy (a) MNIST (b) CIFAR10




(a)




(b)

Fig. 3: Impact of non-IID data on model accuracy in CIFAR10 (a) relation between the degree of non-IID class distribution and accuracy (b) influence from individual outliers.

would data imbalance lead to accuracy loss? To validate, we conduct further experiments.

### B. Impact of Data Imbalance to IID Data

We partition the datasets of MNIST and CIFAR10 among 20 users. For MNIST, the training set of 60K images results an average of 3K images per user. Then we utilize a Gaussian distribution to sample around the mean and adjust the standard deviation to generate data imbalance among users. The ratio between different classes is maintained to be uniform so no class dominates the local set. We utilize an index of *imbalance ratio* as the ratio between the standard deviation and the mean as the x-axis (larger ratio means more extreme), and benchmark the accuracy against the centralized and distributed learning with balanced data in Fig.2. The results indicate that as long as the data remains to be IID, data imbalance does not lead to accuracy loss (the accuracy even trends up a little for CIFAR10). This provides the basis to launch new performance optimization discussed in the following sections.

### C. Impact of non-IID Data

Nevertheless, non-IIDness can be detrimental to the collaborative convergence [2], [3]. We identify the dominant factors that have an influence on accuracy based on the experiments in Fig. 3. First, we show the relation between class-wise non-IIDness and accuracy in Fig. 3(a). The severity is measured by the number of classes that each user has, or simply referred as $n$-class non-IIDness [3]. We iterate $n$ from 2-8 (out of 10 classes) plus a standard deviation of samples among the existing classes as the $x$-axis. The result matches with our intuition that more missing classes cause higher accuracy degradation with a substantial loss of 10-15% on CIFAR10.

When users with different distributions are mixed, a straightforward approach is to exclude users with higher gradient divergence (*individual outliers*) to avoid accuracy loss [7]. To gain more insights of how individual outlier may affect global convergence, we construct the following scenario. We set 3 users and each user randomly picks 3 classes (out of the total 10 classes). This leaves one remaining class for a potential fourth user (one-class outlier). Then we designate the remaining one-class outlier in three ways: a) *Missing*, only 3 users with a total 9 classes are trained. The outlier is missing from the training set; b) *Separate*, involve the outlier as a separate user (4 users); c) *Merge*, merge the missing class back into the 3rd user so the last user has 4 classes.

The result in Fig. 3(b) displays a 3% accuracy gap with the case of missing class ranked the lowest. The result suggests that although individual outliers may lead to local bias and undermine global convergence (gap between Merge and Separate), simple exclusion of the outliers need to exercise with more precaution. It should be further conditioning on whether those outliers contain samples that are not yet included in the training set. If not, they still contribute to the diversity for unseen classes and would be beneficial to the generalization of the global model. Therefore, inclusion or exclusion not only depends on computation speed, but also the actual non-IID distributions. We follow these guidelines in the algorithm designs.

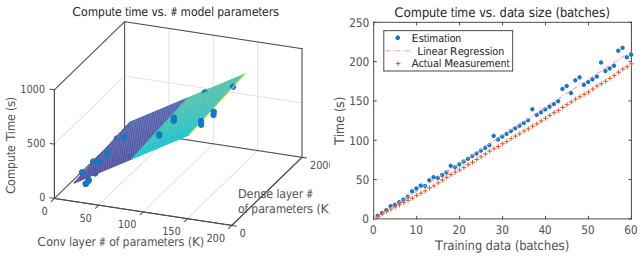
## IV. System Model and Performance Profiling

### A. System Model

We use *shards* to represent the minimum granularity of samples (e.g. 100 samples/shard). The parameter server has sufficient bandwidth so simultaneous transmissions do not cause network congestion or performance saturation [5]. Our framework mainly tackles heterogeneity from computation and data distribution, and is amenable to decentralized topologies without a parameter server [8]. We delegate the role of management to the server to gather users' meta data such as smartphone model and information of non-IID class distribution. For simplicity, we assume the server is honest and does not attempt to infer user privacy from the collaborative model as we can always resort to security protocols to protect the intermediate gradients, model and differentially-private class information [13].

### B. Performance Profiling

For effective scheduling, the server builds performance profiles for the participants. This can be done either online through a bootstrapping phase or offline measured by a collection of devices. The objective is to estimate the training time given the model architecture and data size, which holds a linear relationship in general. For better characterization, we take a two-step approach to first profile the computation time regarding model parameters given the data size. Since the convolutional layers have higher computation intensity, we separate them from the densely connected layers. We test a number of $k$ different model architectures and their training time of $d$ data, denoted by, $\mathbf{y}^{(d)} = [y_1, y_2, \cdots, y_k]^{(d)}$. $\mathbf{x}_i^{(d)} = [x_{i,1}, x_{i,2}]^{(d)}$ are the number of parameters for convolution and dense layers




(a) (b)

Fig. 4: Example: profile training time on Mate10 using linear regression (a) Step 1, characterize training time vs. model parameters (b) Step 2, predict training time vs. data size.

of different models. We employ a multiple linear regression model (for each data size $d$),

$$y_i = \beta_0 + \sum_{j=1}^{2} \beta_j x_{i,j} + e_i, \tag{1}$$

where $e_i$ is a noise vector to compensate measurement error. The parameters are found by solving the least square problem, $\hat{\beta} = \mathbf{y} \cdot \mathbf{X}^{-1}$, which is computed by $\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \|\mathbf{y} - \beta\mathbf{X}\|_2^2$. The output of the first step is $\{\beta_0, \beta_1, \beta_2\}^{(d)}$ for different $d$. Shown in Fig.4(a), we find the hyperplane that predicts training time given the model parameters. The linear relation is emphasized by convolutions, which coincides with the initial step of separation.

With an unknown model architecture, the first step provides $d$ estimates $[y_1, y_2, \cdots, y_d]$ of compute time. The second step generalizes this for unknown data sizes by applying linear regression again to fit the predictions from the first step in the least-square sense. Fig.4(b) shows the final curve versus its actual measurement from experiment and a small gap is observed. Such profiles can be constructed offline for the scheduling algorithms discussed next.

## V. Assignment with IID Data Distribution

We first consider the case that the local dataset contains data from all classes (i.e., IID) and optimize the training time per epoch. IID is the basic assumption for machine learning and exemplifies the case when users have compiled a class-balanced dataset in sufficiently long time.

### A. Problem Formulation

In each epoch, a set of $n = |\mathcal{N}|$ users are chosen to run $m = |\mathcal{M}|$ distributed training tasks. Each participant conducts one training task ($m = n$). As experimented in Section III-B, unbalanced local data leads to negligible accuracy loss when the data is IID so this gives enough latitude for the task assignments. To be clear, we use subscripts $i$ for tasks and $j$ for users. When user $j$ conducts training task $i$, the *computation time* is a function of data size $D_i$ and model $M$, $T_j^c(D_i)$. Depending on the networking environments, the uplink and downlink network latency for user $j$ depends on the model size $M$, $T_j^u(M) + T_j^d(M)$. The goal is to find an optimal assignment of training data so that the maximum processing time is minimized per epoch. The problem is formalized below.

$$\mathbf{P1}: \quad \min_{X \in \phi} \max_{j \in \mathcal{N}} \left(T_j^c(D_i) + T_j^u(M) + T_j^d(M)\right) x_{ij} \tag{2}$$

**s.t.**

$$\sum_{i \in \mathcal{M}} D_i = D, i \in \mathcal{M} \tag{3}$$

$$\sum_{j \in \mathcal{N}} x_{ij} = 1, i \in \mathcal{M} \tag{4}$$

$$\sum_{i \in \mathcal{M}} x_{ij} = 1, j \in \mathcal{N} \tag{5}$$

The objective is to minimize the makespan given all possible data partitions and the assignment of training task $i$ with $D_i$ data to $j$. Eq. (3) states that the sum of local data should be equal to $D$. Eqs. (4) and (5) impose that each task is assigned to only one user, and vice versa. Decision variable $x_{ij}$ is 0-1 valued. $X$ is an assignment matrix with elements $x_{ij}$ and $\phi$ is the set of all permutations of the assignment matrix $X$.

**P1** can be viewed as a combination of a *partitioning problem* and a variant of the *linear bottleneck assignment problem* (LBAP) [23]. The classic assignment problem finds an optimal assignment of workers to tasks with minimum sum of cost. LBAP is its min-max version. It assigns tasks to parallel workers and ensures the latest worker uses minimum time. We adopt the same analogy here to ensure each training epoch is finished in minimum time. The problem is different from both the classic assignment problem and LBAP. The number of *potential tasks* is not equal to the number of workers (mobile devices), but rather, a much wider potential range to choose from due to the combinatorial partitions of the dataset. The final choice would be determined by the set of constraints that optimizes Eq. (2). A naive solution is to list all the partitions of $D$ in brute force, construct cost values per user for all the potential permutations, solve an LBAP and find the assignment with the minimum makespan. For a total number of $s$ shards, the possible permutations are in the order of $s^n$, which makes it intractable even for small $n$.

### B. Joint Partitioning and Assignment

Though the naive method turns out to be intractable in polynomial time, the following property of mobile devices helps simplify the problem.

**Property 1.** For data $D_i$, $T_j^c(D_i) + T_j^u(M) + T_j^d(M)$ is a non-decreasing function.

Then it is not necessary to test a large number of potential partitions, if a partition of smaller size has already satisfied Eq. (3) with less computation time. For example, consider possible permutations of $\sum_{i=1}^{3} D_i = 13$ among three users. If the first or the second user is the straggler in partition $(4, 4, 5)$, then partitions such as $(5, 5, 3)$, $(6, 6, 1)$ definitely leads to more running time. This allows us to potentially skip a large number of sub-optimal solutions.

The classic LBAP has a polynomial-time thresholding algorithm in $\mathcal{O}(n^{\frac{5}{2}} \log n)$ [23]. This algorithm checks whether a *perfect matching* exists in each iteration using the Hopcroft-Karp algorithm, that takes $\mathcal{O}(n^{\frac{5}{2}})$. Here, when $D$ is divided into $s$ shards, perfect matching between user and data shard is no longer needed as introduced in the following property.

**Property 2.** A bipartite graph $\mathcal{G} = (\mathcal{U}, \mathcal{V}; \mathcal{E})$ is constructed with $|\mathcal{U}| = n$, $|\mathcal{V}| = s$ and edges $(j, k) \in \mathcal{E}$. Each vertex in $\mathcal{U}$

**Algorithm 1:** Fed-LBAP (for IID data)

**Input:** Total data size $D$, cost matrix $C = \{C_{jk}\}$, number of users $n$.
**Output:** The assignments of tasks $\{A_j\}$ for each user $j$.
$\overline{C} \leftarrow C$ sorted in the ascending order.
min $\leftarrow 0$, max $\leftarrow |\overline{C}|$, median $\leftarrow \lfloor \frac{\text{min}+\text{max}}{2} \rfloor$; $D' \leftarrow 0$
**while** min $<$ max **do**
  $C^* \leftarrow \overline{C}$(median)
  **for** $j = 1$ *to* $m$ **do**
    $A_j \leftarrow \arg\max_j \{C_{jk} | C_{jk} \leq C^*\}$
    $D' \leftarrow D' + A_j$
  **if** $\forall j, A_j = 0$ *or* $D' < D$ **then**
    min $\leftarrow$ median
  **else**
    max $\leftarrow$ median

should have degree of 1 and vertices in $\mathcal{V}$ can have degree of 0 (as long as the sum of vertices having degree 1 equals $D$).

**Fed-LBAP Algorithm.** Based on Properties 1 and 2, we can further reduce the time complexity by extending [23]. We propose a joint partitioning and assignment algorithm to solve the problem in polynomial time. For the $n$ users, we define a cost matrix $C = \{c_{jk}\}$ of dimension $n \times s$ (i.e., the matrix represents the cost to assign a task of $k$ shards to user $j$). A *thresholding matrix* $\overline{C}$ with the same setting is also initiated. We sort all the elements from the cost matrix in ascending order and perform binary search by utilizing a threshold $c^*$: if $c_{jk} > c^*$, $\overline{c_{jk}} = 0$; otherwise, $\overline{c_{jk}} = 1$. The sum of all cost values found in each iteration is compared to $D$. If larger, find a new median for the left half; otherwise, find a new median for the right half until the optimal median value is reached. In short, our algorithm first performs a sorting of all the cost values and conducts a binary search for the minimal threshold $c^*$ such that *Property 2* and Eq. (3) hold. The procedures are summarized in Algorithm 1.

The time complexity is analyzed below. In the worse case, binary search takes $\mathcal{O}(\log ns)$ iterations. We need to check whether $\overline{c_{jk}} = 0$ during the iterations. This takes $\mathcal{O}(s)$ time for one user and is repeated for $n$ times. The time complexity is $\mathcal{O}(ns \log ns)$ with $s \geq n$. To be consistent with [23], when $s = n$, our algorithm is $\mathcal{O}(n^2 \log n)$.

## VI. Assignment with Non-IID Data Distribution

Due to the inherent difference among users in their behaviors and interests, non-IIDness is quite common in mobile applications. This section investigates the scheduling strategy when local data is non-IID.

### A. Non-IID Formulation

Given the disparity of class distributions among users, which users are selected is vital to the computation time and accuracy. One may follow the previous algorithm to weigh more on those with higher processing power. But non-IIDness brings new challenges: if some users only have a single class, the gradient may adversely prolong global convergence. On the other hand, as indicated by Section III-C, if the class is not yet included in the training set, inviting the user into training would be beneficial to model generalization. To this end, we introduce an *accuracy cost* $F_j$ to choose user $j$. Denote the set of existing classes as $\mathcal{U}$ for the current training set. The testset has $K$ classes ($|\mathcal{U}| \leq K$). The cost of accuracy selecting user $j$ with $|\mathcal{U}_j|$ classes is,

$$F_j = \begin{cases} \frac{K}{|\mathcal{U}_j|}, & \mathcal{U} \cap \mathcal{U}_j \neq \emptyset \\ \frac{K}{|\mathcal{U}_j|} - \frac{\beta}{\alpha} \cdot D_u, & otherwise. \end{cases} \tag{6}$$

where $\alpha, \beta$ are two input parameters ($\alpha > \beta$). $D_u$ is the number of data shards in the current training set. For user $j$, the accuracy cost is inversely proportional to the number of classes he has, if the intersection between his class and the current set is not empty; otherwise, we deduct the cost by the size of training set times $\beta$ to make the user more appealing during selection. In practice, the users could truthfully report their accuracy cost instead of detailed $\mathcal{U}_j$ to reduce privacy leakage of class-level information. We balance the time and accuracy cost using the weight parameter $\alpha$ and our objective is to derive a schedule with the minimum average cost.

$$\mathbf{P2}: \quad \min \sum_{j \in \mathcal{N}} T_j^c \big( \sum_{i \in \mathcal{M}} x_{ij} \big) + \big( T_j^u(M) + T_j^d(M) + \alpha F_j \big) y_j \tag{7}$$

**s.t.**

$$\sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{N}} x_{ij} = D \tag{8}$$

$$\sum_{i \in \mathcal{M}} x_{ij} \leq C_j, j \in \mathcal{N} \tag{9}$$

$$0 \leq x_{ij} \leq D, \quad x_{ij} \in \mathbb{N} \tag{10}$$

$$y_j = \mathbb{1} \big( \sum_{i \in \mathcal{M}} x_{ij} > 0 \big) \tag{11}$$

We adopt most of the notations from $\mathbf{P1}$ and assume an initial set of tasks $\mathcal{M}$ (e.g., an equal partition, but divisible afterwards). The new objective is to determine the data shards $x_{ij}$ to be assigned from task $i$ to user $j$ such that the sum of computation/communication and cost of accuracy (scaled by $\alpha$) is minimized. We can consider the accuracy cost as a fixed cost when a user is involved, which gradually changes defined by Eq. (6). Constraint (8) ensures the tasks are fully packed and all the partitions sum up to $D$ data in total. Constraint (9) states that the size of data does not exceed user $j$'s capacity $C_j$, which can be quantified by the storage or battery energy. Constraint (10) bounds the proportion of $x_{ij}$ from zero to $D$ in integers. Constraint (11) makes $y_j$ equal to 1 if user $j$ is selected; otherwise, $y_j$ is 0.

The problem can be abstracted into a close analogy of the *bin packing problem with item fragmentation* [27], which finds an assignment of items to a fixed number of bins by splitting them into fragments. For each fragmentation, there is an associated unit cost. In our scenario, the items correspond to the learning tasks splittable into data shards and the users represent the bins. Unlike the original bin packing, the objective no longer minimizes the number of users (with unit cost); instead, it is characterized by a function of computation time and model accuracy. The fragmentation cost is also different from the unit cost in [27]. It actually depends on which destined user the fragments are assigned to. If the user has been already involved in training (bin/user is open), the cost mainly depends on the computation time of the new fragments; otherwise, the cost of accuracy in Eq. (6) is also considered. We propose an algorithm as described next.

**Algorithm 2:** Fed-MinAvg Algorithm (for Non-IID Data)

**Input:** Number of data shards $D$ and size $d$ per shard, $n = |\mathcal{N}|$ users, cost profiles $T(\cdot)$, set of class coverage $\mathcal{U}$ and user coverage $\mathcal{O}$, parameters $\alpha, \beta$, number of data shards $l_j$ for user $j$, number of classes $K$ in the testset.
**Output:** Data for each user $l_j$.
$\mathcal{U}, \mathcal{O} \leftarrow \emptyset$, $D_u \leftarrow 0$.
**while** $D_u < D$ **do**
  **if** $\mathcal{N} \setminus \mathcal{O} \neq \emptyset$ **then**
    $j \leftarrow \arg\min_{j \in \mathcal{O}, k \in \mathcal{N} \setminus \mathcal{O}} \{T_j((l_j+1)\cdot d) + \alpha F_j, T_k(d) + \alpha F_k\}$.
  **else**
    $j \leftarrow \arg\min_{j,k \in \mathcal{O}} \{T_j((l_j+1)\cdot d) + \alpha F_j, T_k((l_k+1)\cdot d) + \alpha F_k\}$.
  $l_j \leftarrow l_j + 1$.
  **if** $\mathcal{U} \cap \mathcal{U}_j \neq \emptyset$ **then**
    $\alpha F_j \leftarrow \alpha \cdot \frac{K}{|\mathcal{U}_j|}$
  **else**
    $\alpha F_j \leftarrow \alpha \cdot \frac{K}{|\mathcal{U}_j|} - \beta \cdot D_u$
  **if** $l_j \geq C_j$ **then**
    $F_j \leftarrow \infty$
  $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{U}_j$, $\mathcal{O} \leftarrow \mathcal{O} + j$, $\mathcal{N} \leftarrow \mathcal{N} - j$, $D_u \leftarrow D_u + 1$.

### B. Min Average Cost Algorithm

The main idea of the *Min Average Cost Algorithm* is to iteratively assign the data shards to the user with the minimum average cost in a greedy fashion. Consider the dataset of $D$ data shards and $n$ users. The initial cost is $T_j(d) + \alpha F_j$, if a user $j$ is open for training with $d$ data[1]. Starting from $j$ with the lowest initial cost, we assign $d_1 = d$ to $j$, and update the set of current class coverage as $\mathcal{U} \cup \mathcal{U}_j$ and the accuracy cost in Eq. (6). Denote the set of users that are already involved in training as $\mathcal{O} \subseteq \mathcal{N}$. For $d_2 = d$, we compare the cost by either assigning it to $j$ with cost $T_j(2d) + \alpha F_j$, or to $k$ with cost $T_k(d) + \alpha F_k$ ($k \in \mathcal{N} \setminus \mathcal{O}$), and select the one with less cost. For all the users $j \in \mathcal{O}$ and a potential user $k \in \mathcal{N} \setminus \mathcal{O}$, we assign $d$ according to,

$$j^* = \arg\min_{j \in \mathcal{O}, k \in \mathcal{N} \setminus \mathcal{O}} \{T_j((l_j+1)\cdot d) + \alpha F_j, T_k(d) + \alpha F_k\}, \quad (12)$$

where $l_j$ is the current number of data shards of user $j$. If all the users are involved ($j, k \in \mathcal{O}$), we compare $T_j((l_j+1)\cdot d) + \alpha F_j$ with $T_k((l_k+1)\cdot d) + \alpha F_k$ and select the one with less cost. If $j$ reaches the capacity that $l_j \cdot d \geq C_j$, it is excluded from further selections (bin is closed); otherwise, it remains to be open. The algorithm repeats until $D$ is exhausted and runs in $\mathcal{O}(mn)$ time, where $m$ is much larger than $n$. The procedure is summarized in Algorithm 2.

## VII. Evaluation

In this section, we evaluate the proposed algorithms on a mobile testbed using two datasets and compare with various baselines. The main goal is to evaluate the performance speedups of the proposed algorithms and the consequent impact on model accuracy when user data is either IID or non-IID.

[1]We omit the communication cost here for clarity.

**Mobile Development.** The mobile framework is developed in DL4J [30], a java-based deep learning framework that can be seamlessly integrated with Android. Training is conducted using multi-core CPUs enabled by OpenBLAS in Android 8.0.1. We use `AsyncTask` to launch the training process by the foreground thread with the default `interactive` governor and scheduler. To avoid memory error, we enlarge the heap size to 512 MB by setting `largeHeap` and use a batch size of 20 samples. This allows us to train VGG-like deep structures.

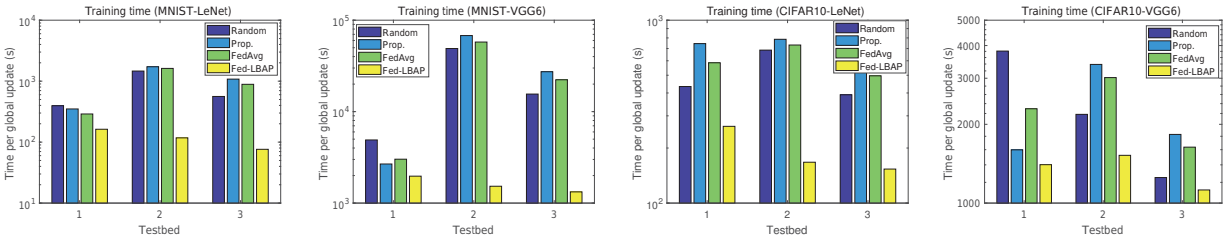
**Experiment Setting.** We use the collection of devices to construct three combinations of mobile testbeds: (I) $1\times$ Nexus6, $1\times$ Mate10 and $1\times$ Pixel2 of 3 devices; (II) $2\times$ Nexus6, $2\times$ Nexus6P, $1\times$ Mate10 and $1\times$ Pixel2 of 6 devices; (III) $4\times$ Nexus6, $2\times$ Nexus6P, $2\times$ Mate10 and $2\times$ Pixel2 of 10 devices. The experiment is conducted on two commonly used datasets: MNIST [28] and CIFAR10 [29] with 60K and 50K training samples. We fully charge all the devices, pre-load both datasets into the flash storage of the mobile devices and read them in mini-batches. Users perform one epoch of local training in each round and the global gradient averaging iterates 20 and 50 epoches for MNIST and CIFAR10 respectively.

To emulate the dynamics of mobile data, we generate random distributions among the users: 1) For IID data, each user retains all the classes and the ratio between samples from different classes is equivalent; 2) For non-IID data, each user has a random subset of classes and each class may also have different number of samples. Two fundamental networks of LeNet [25] and VGG6 [26] are evaluated and their efficiency has been proved to handle learning problems at sufficient scales. To meet the input dimensions, we tailor the original 16 layers of VGG16 by stacking five $3 \times 3$ convolutional layers with one densely connected layer. While scheduling with non-IID data, we search for the optimal cost parameter $\alpha$ in $[100, 5000]$ and set $\beta = 2$ unless stated otherwise. The uplink and downlink latencies are added to computation time according to the measurement in Table II.

**Benchmarks.** The proposed algorithms are compared with several benchmarks: 1) Proportional: a heuristic that assigns training data proportional to the processing power measured by the mean CPU frequencies per core; 2) Random: uniformly random data partitions among the users; 3) Equal: assign equal shares of data to users as adopted by FedAvg [2]. Since the model architecture is fixed, we mainly compare the computational time and treat the communication time as a constant. The results are averaged over 10 experimental runs. To facilitate the evaluation, we also adopt pytorch with GTX1080/K40 GPUs to evaluate different benchmarks.

### A. Scheduling with IID Data

**Computation Time.** We first evaluate the Fed-LBAP algorithm developed for IID data. Fig. 5 shows the training time per global update for all the combinations between the testbed, datasets and models (y-axis in log-scale). We can see that the mobile processing power (both individual and collective) and the workloads play key roles in the computational time, which sums up to nontrivial relations. First, unlike cloud settings




(a) (b) (c) (d)

Fig. 5: Comparison of computation time when data is IID (time in log-scale) (a) training MNIST with LeNet; (b) training MNIST with VGG6; (c) training CIFAR10 with LeNet; (d) training CIFAR10 with VGG6.

in which computation time scales well with the number of workers, mobile stragglers would easily degrade the overall performance even if more users are involved. For example, the time surge from Testbed 1 (3 users) to Testbed 2 (6 users) is due to the problematic Nexus6P, that the poor design of heat dissipation, frequency scaling and power management are the main cause of the overall slowdown. This drag is magnified with complex network architectures of higher computation intensity (VGG6 with more convolutional layers) and more training data (60K of MNIST vs. 50K CIFAR10). The 10K data addition exacerbates the compute time parabolically by 20 times (Testbed 2 between Figs. 5(b) and (d) running VGG6 on MNIST and CIFAR10), if the scheduling is done inappropriately. By bringing 2 additional devices with higher processing power, we observe a slight performance improvement.

With the ordinary schedules (Prop., Random and Equal), we hardly see any consistent parallelism when more users are involved. However, Fed-LBAP is capable of utilizing the additive computational resources by appropriately assigning workloads to the more efficient users, so the training time accomplishes a downtrend with more users, even worse-case stragglers are present. Among the benchmarks, Fed-LBAP achieves an overall 5-10$\times$ speedups, with the best speedup of almost 2 orders of magnitude for Testbed 2 in Fig. 5(b). Although naive schemes may look for an optimal scheduling that is proportional to device CPU frequency or equally assign workloads as [2], the runtime behavior may be drastically different due to complex system dynamics, and our experiment shows that such schemes perform no better than a purely randomized schedule.

**Accuracy Loss.** Table III summarizes the test accuracy under different scheduling. We can see the accuracy trends down slightly with LeNet while more users are involved. Since the data is IID, Fed-LBAP can be considered as one special permutation from the random partitions. Our results indicate that as long as the data remains to be IID, we can leverage load unbalancing without accuracy loss. We also notice that the random assignments tend to yield the highest accuracy. The reason could be attributed to the diversity of gradients from the random permutations, where high similarity between the gradients may not facilitate the learning process [21].

### *B. Scheduling with Non-IID Data*

**Effectiveness of Parameters $\alpha$ and $\beta$.** The evaluation of non-IIDness calls for generating specific data distributions and matches them with the mobile devices. To enable analysis and gain more insights, we generate 3 representative data

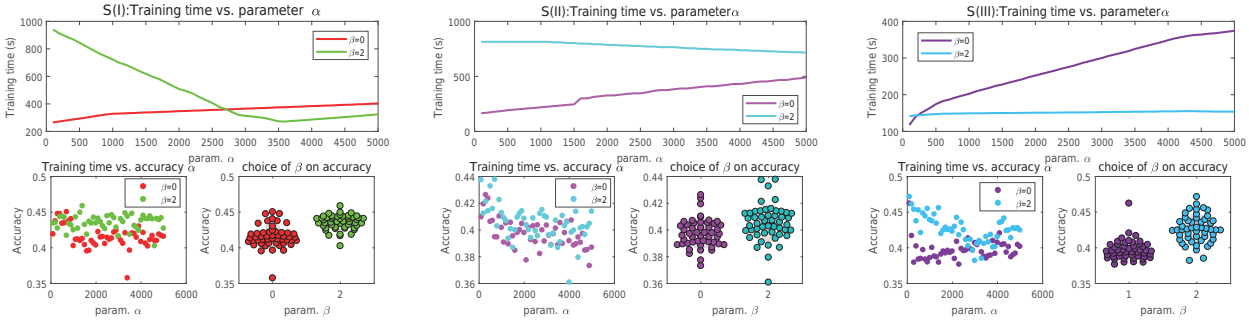
| | | | Prop. | Random | Equal | Fed-LBAP |
|---|---|---|---|---|---|---|
| MNIST | LeNet | (I) | 0.9908 | **0.9911** | 0.9907 | 0.9908 |
| | | (II) | 0.9898 | 0.9896 | 0.9896 | **0.9899** |
| | | (III) | 0.9882 | 0.9882 | 0.9883 | **0.9886** |
| | VGG6 | (I) | 0.9939 | **0.9939** | 0.9936 | 0.9936 |
| | | (II) | 0.9932 | 0.9933 | **0.9934** | 0.9932 |
| | | (III) | 0.9928 | 0.9906 | 0.9928 | **0.9929** |
| CIFAR10 | LeNet | (I) | **0.5966** | 0.5951 | 0.5921 | 0.5926 |
| | | (II) | **0.5853** | 0.5851 | 0.5831 | 0.5846 |
| | | (III) | 0.5622 | 0.5632 | 0.5625 | **0.5689** |
| | VGG6 | (I) | 0.7295 | **0.7330** | 0.7286 | 0.7285 |
| | | (II) | 0.7317 | **0.7328** | 0.7286 | 0.7308 |
| | | (III) | 0.7269 | **0.7353** | 0.7287 | 0.7321 |

TABLE III: Comparison of model accuracy with different benchmarks (IID data)

distributions shown in col.2-4 in Table IV. E.g., S(I) has Nexus6(a) with classes from 0 to 6, 9 and class 7 only comes from Pixel2(a). Then we adjust the parameters of $\alpha$ and $\beta$ to see how Fed-MinAvg reacts to these distributions. Recall that $\alpha$ increases the initial cost of the users with more missing classes and $\beta$ makes the process aware of the class coverage.

We illustrate the potential trade-offs in Fig. 6 by referencing to the workload assignments generated from the algorithms in Table IV(col.5-16). The top figure depicts the trace of training time when $\alpha$ increases from 100 to 5000. When $\beta = 0$, the training time generally trends up when $\alpha$ increases. This is because a large accuracy cost re-distributes the workloads to the devices with more classes and reduces parallelism. E.g., in S(I), the best performing device Pixel2(a) is unfortunate to have only two classes (higher initial accuracy cost). Large $\alpha$ makes it less likely to assign data to Pixel2(a), so the workloads are transferred to Nexus6(a) as observed in Table IV(col.5-6). Similar observations are made in S(II) and S(III) and the underperforming devices with higher skewness of non-IIDness would be excluded from scheduling. From the assignments in Table IV, when $\alpha$ increases, the algorithm favors the user with fewer missing classes (perceived from the gap between $p_1, p_2$ and $p_3, p_4$). When $\alpha = 5000$, slower devices with higher non-IIDness are assigned with zero data.

If we take a closer look into the class distributions in Table IV, some classes only belong to the outliers such as class 7 in S(I) and class 4 in S(II). Thus, the exclusion of these users in training has an adverse impact to accuracy as perceived in the bottom left Figs. 6(a-b) - as $\alpha$ increases, the accuracy trends down. In contrast, the corresponding plot in Fig. 6(c) indicates the opposite. This is because the outlier classes are also included by other users so their participation would interfere with the training process, i.e., their exclusion could lead to an accuracy gain. The assignment decision would have




(a) (b) (c)

Fig. 6: Effectiveness of $\alpha$ and $\beta$ and their relations with training time and accuracy (best view in color) (a) S(I); (b) S(II); (c) S(III).

| | class dist. | | | S(I) | | | | S(II) | | | | S(III) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S(I) | S(II) | S(III) | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
| Nexus6(a) | (0-6,9) | (1,2,5,7) | (2,6,8,9) | 23.8 | 35.4 | 13.5 | 28.7 | 16.1 | 2.3 | 8.7 | 5.8 | 9.7 | 0 | 3.7 | 0 |
| Nexus6(b) | – | (2,6,8) | (0,1,3,7-9) | – | – | – | – | 18.0 | 40.1 | 9.1 | 19.3 | 11.1 | 33.4 | 4 | 12.1 |
| Nexus6(c) | – | – | (9) | – | – | – | – | – | – | – | – | 0.4 | 0 | 1.5 | 0 |
| Nexus6(d) | – | – | (0,5) | – | – | – | – | – | – | – | – | 8.1 | 0 | 2.9 | 0 |
| Nexus6P(a) | – | (0,3,8,9) | (2) | – | – | – | – | 0.3 | 0 | 1.2 | 0 | 0 | 0 | 0 | 0 |
| Nexus6P(b) | – | (0) | (0-2,4,5) | – | – | – | – | 2.6 | 0 | 3.2 | 0 | 1.2 | 0.37 | 8 | 2.4 |
| Mate10(a) | (2-6,8) | (4,9) | (1,3,4,8) | 19.4 | 14.6 | 12.3 | 20.5 | 7.6 | 0 | 5.6 | 0 | 7.5 | 0 | 3.1 | 0 |
| Mate10(b) | – | – | (9) | – | – | – | – | – | – | – | – | 0.5 | 0 | 0.2 | 0 |
| Pixel2(a) | (7,8) | (0,1,2) | (1) | 6.8 | 0 | 24.2 | 0.8 | 5.4 | 7.6 | 22.2 | 24.9 | 1.3 | 0 | 0.9 | 0 |
| Pixel2(b) | – | – | (0-3,7,8) | – | – | – | – | – | – | – | – | 6.6 | 12.9 | 32.9 | 35.5 |

TABLE IV: Class distribution and schedules computed by *MinAvg algorithm* with $(\alpha, \beta)$ combinations, $p_1 = (100, 0), p_2 = (5000, 0), p_3 = (100, 2), p_4 = (5000, 2)$. The numbers are in $10^3$ data samples and evaluated using CIFAR10-LeNet.

a subtle impact on the accuracy and we leave more exploration to the future works.

When $\beta = 2$, the algorithm would continuously compensate a small benefit for outliers if their classes are not included in the overall class coverage. It might allocate some data to those outliers, but end up being far from time-optimal shown in the top Figs. 6(a-b). With the increase of $\alpha$, the cost of accuracy from outliers re-gains their leverage against benefit, and data is re-allocated back to other devices so we mainly see a decline of the training time thereafter. The advantage of incorporating $\beta$ on accuracy is measured by the bottom right of Figs. 6 as it further lifts the accuracy by $0.02$ to $0.03$.

**Computation Time.** Next, we perform large-scale experiments using random permutations of the class distributions among the devices and demonstrate the comparison in Fig. 7. We found the best $\alpha$ value over the interval of $[100, 5000]$ and set $\beta = 0$ to weigh more on the computation time. The average speedups of the Fed-MinAvg on the three testbeds are $1.3\times, 8\times, 6\times$ for MNIST and $1.92\times, 2.05\times, 1.67\times$ for CIFAR10. Though the numbers are less than the case of IID due to new considerations of non-IIDness, the algorithm manages to achieve an overall speedup, especially when worse-case stragglers are present in Testbed 2.

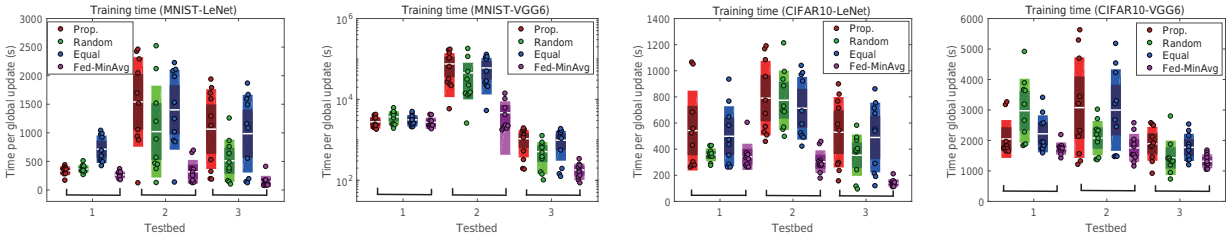
**Accuracy Loss.** Table V shows the potential accuracy loss due to scheduling and compares with the benchmarks. For MNIST, Fed-MinAvg results almost no accuracy loss and for CIFAR10, the accuracy loss is within $0.02$. It is interesting to see two phenomenons. First, by contrast to the IID case, in the vertical direction, the accuracy climbs up considerably when more users are involved. We conjecture that the reasons are due to dynamics learned from users with the entire class coverage but more dispersed distribution, that has actually improved the generalization of the global model. In other words, the contribution from such gradient diversity has surpassed the potential damage from the bias of local gradient. Second, horizontally, the advantage of random assignment is more visible than IID. However, most of these permutations would yield high computation time, thus not time-optimal respect to performance.

| | | | Prop. | Random | Equal | Fed-MinAvg |
|---|---|---|---|---|---|---|
| MNIST | LeNet | (I) | 0.9116 | **0.9177** | 0.8983 | 0.9059 |
| | | (II) | 0.9611 | 0.9581 | **0.9565** | 0.9541 |
| | | (III) | 0.9814 | 0.9778 | 0.9599 | **0.9817** |
| | VGG6 | (I) | 0.9147 | **0.9250** | 0.8991 | 0.9211 |
| | | (II) | 0.9601 | **0.9759** | 0.9538 | 0.9739 |
| | | (III) | 0.9805 | 0.9860 | 0.9562 | **0.9880** |
| CIFAR10 | LeNet | (I) | 0.4348 | **0.4558** | 0.4486 | 0.4285 |
| | | (II) | 0.4640 | **0.4793** | 0.4689 | 0.4745 |
| | | (III) | 0.5064 | **0.5070** | 0.5036 | 0.4892 |
| | VGG6 | (I) | 0.5913 | **0.6232** | 0.6183 | 0.5923 |
| | | (II) | 0.6553 | **0.6614** | 0.6545 | 0.6406 |
| | | (III) | 0.6833 | 0.6773 | **0.6852** | 0.6663 |

TABLE V: Comparison of model accuracy with different mechanisms (non-IID data)

## VIII. Conclusion

In this paper, we optimize the computation time of launching federated learning on battery-powered mobile devices. We first motivate this work by showing heterogenous processing time and non-IID challenges. Then we develop two efficient near-optimal algorithms to schedule workload assignments for




(a) (b) (c) (d)

Fig. 7: Comparison of computation time when data is Non-IID (time in log-scale) (a) training MNIST with LeNet; (b) training MNIST with VGG6; (c) training CIFAR10 with LeNet; (d) training CIFAR10 with VGG6.

both IID and non-IID data. Our extensive experiments on real mobile testbed and datasets demonstrate up to 2 orders of magnitude speedups and minimum accuracy loss when data is non-IID.

## IX. Acknowledgement

This work was supported in part by the U.S. National Science Foundation under grant number CCF-1850045.

## References

[1] J. Dean, et. al., "Large scale distributed deep networks", *NIPS*, 2012.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. Arcas, "Communication-efficient learning of deep networks from decentralized data", *AISTATS*, 2017.

[3] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin and V. Chandra, "Federated Learning with Non-IID Data", *arXiv:1806.00582*.

[4] E. Jeong, S. Oh, H. Kim, S. Kim, J. Park and M. Bennis, "Communication-efficient on-device machine learning: federated distillation and augmentation under non-IID private data", *NIPS-MLPCD*, 2018.

[5] K. Bonawitz, et. al., "Towards federated learning at scale: system design", *SysML* Conference, 2019.

[6] J. Konecny, et. al., "Federated learning: strategies for improving communication efficiency", *NIPS*, 2016.

[7] L. Wang, W. Wang and B. Li, "CMFL: Mitigating communication overhead for federated learning", *IEEE ICDCS*, 2019.

[8] X. Lian, et. al., "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent", *NIPS*, 2017.

[9] H. Zhu, Y. Jin, "Multi-objective evolutionary federated learning", arXiv preprint arXiv:1812.07478, 2018.

[10] V. Smith, C. Chiang, M. Sanjabi and A. Talwalkar, "Federated multi-task learning", *NIPS*, 2017.

[11] Q. Ho, J. Cipar, H. Cui, J. Kim, S. Lee, P. Gibbons, G. Gibson, G. Ganger and E. Xing, "More effective distributed ML via a stale synchronous parallel parameter server", *NIPS*, 2013.

[12] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z. Ma and T. Liu, "Asynchronous stochastic gradient descent with delay compensation", *ICML*, 2017.

[13] K. Bonawitz, et. al., "Practical secure aggregation for privacy-preserving machine learning", *ACM CCS*, 2017.

[14] P. Blanchard, E. Mhamdi, R. Guerraoui and J. Stainer, "Machine learning with adversaries: byzantine tolerant gradient descent", *NIPS*, 2017.

[15] S. Han, H. Mao and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding", *ICLR*, 2016.

[16] C. Wang, Y. Xiao, X. Gao, L. Li and J. Wang, "Close the gap between deep learning and mobile intelligence by incorporating training in the loop", *ACM Multimedia*, 2019.

[17] ARM's big.LITTLE, https://www.arm.com/why-arm/technologies/big-little

[18] L Epstein and J. Sgall, "Approximation Schemes for Scheduling on Uniformly Related and Identical Parallel Machines", *ESA*, pp. 151-162, 1999.

[19] Y. Peng, Y. Bao, Y. Chen, C. Wu and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters", *EuroSys*, 2018.

[20] Y. Bao, Y. Peng, C. Wu and Z. Li,"Online Job Scheduling in Distributed Machine Learning Clusters", *IEEE INFOCOM*, 2018.

[21] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran and P. Bartlett, "Gradient diversity: a key ingredient for scalable distributed learning", *AISTATS*, 2018.

[22] In-depth with the Snapdragon 810's heat problems, https://arstechnica.com/gadgets/2015/04/in-depth-with-the-snapdragon-810s-heat-problems

[23] R. Burkard, M. Dell'Amico and S. Martello, "Assignment problems", *SIAM*, 2012

[24] J. B. Mazzola and A. W. Neebe, "Bottleneck generalized assignment problems", *Engineering Costs and Production Economics*, vol. 14, no. 1, pp. 61-65, 1988.

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

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *ICLR*, 2015.

[27] B. LeCun, T. Mautor, F. Quessette and M. Weisser, "Bin packing with fragmentable items: Presentation and approximations", *Theo. Comp. Sci.*, vol. 602, 2015.

[28] MNIST dataset, http://yann.lecun.com/exdb/mnist/

[29] CIFAR10 dataset, https://www.cs.toronto.edu/ kriz/cifar.html

[30] Deep Learning for Java, https://deeplearning4j.org