Multi-layer Coordination for High-Performance Energy-Efficient Federated Learning

Li Li[1], Jun Wang[2], Xu Chen[3], and Cheng-Zhong Xu[4]

[1]ShenZhen Institutes of Advanced Technology, Chinese Academy of Sciences
[2]Futurewei Technology
[3]Sun Yat-sen University
[4]State Key Lab of IoTSC, University of Macau

***Abstract*—Federated Learning is designed for multiple mobile devices to collaboratively train an artificial intelligence model while preserving data privacy. Instead of collecting the raw training data from mobile devices to the cloud, Federated Learning coordinates a group of devices to train a shared model in a distributed manner with the training data located on the devices. However, in order to effectively deploy Federated Learning on resource-constrained mobile devices, several critical issues including convergence rate, scalability and energy efficiency should be well addressed.**

In this paper, we propose MCFL, a multi-layer online coordination framework for high-performance energy efficient federated learning. MCFL consists of two layers: a macro-layer on the central server and a micro-layer on each participating device. In each training round, the macro coordinator performs two tasks, namely, selecting the right devices to participate, and estimating a time limit, such that the overall training time is significantly reduced while still guaranteeing the model accuracy. Unlike existing systems, MCFL removes the restriction that participating devices must be connected to power sources, thus allowing more timely and ubiquitous training. This clearly requires on-device training to be highly energy-efficient. To this end, the micro coordinator determines optimal schedules for hardware resources in order to meet the time limit set by the macro coordinator with the least amount of energy consumption. Tested on real devices as well as simulation testbed, MCFL has shown to be able to effectively balance the convergence rate, model accuracy and energy efficiency. Compared with existing systems, MCFL can achieve a speedup up to 8.66× and reduce energy consumption by up to 76.5% during the training process.

I. Introduction

Mobile devices have improved exponentially not only in terms of processing speed and memory capacity but also in their support of diverse I/O components such as camera, audio, GPS, sensors and so on. With these equipped I/O components, mobile devices are collecting large amounts of data from multiple dimensions, since they are usually carried anytime and everywhere. Different kinds of artificial intelligence models can be learned with those collected data, which can make the devices intelligent and greatly improve user experience [1]. Traditionally, such data are collected and stored in the cloud for model training. However, gathering the sensitive user data to a central place can cause serious privacy issues.

Federated Learning. In order to effectively leverage the data on mobile devices while preserving data privacy, Federated Learning [2] is proposed to train a shared model collaboratively based on multiple mobile devices. In a Federated Learning system, the participating devices first train the model locally with their private data. Then, the local gradients are sent to the central server. After receiving the updated gradients, the central server conducts model aggregation based on the received gradients and updates the shared model. The updated model is then sent back to the participating devices. This process iterates until the model converges. We can notice that the raw training data never leave the mobile devices during the whole training process. Thus, data privacy is well preserved in this manner. Specifically, Federated Learning has been widely utilized to support different mobile-based applications such as on-device item ranking, content suggestions for on-device keyboards and next word prediction (e.g., Gboard) [3].

Limitations of existing solutions. The following issues affect large-scale deployment of Federated Learning on mobile devices and severely limit its application scope. First, in order to guarantee the model convergence, current Federated Learning systems adopt a synchronous model aggregation approach [2] which means that the central server only conducts model averaging when the gradients from *all* the participating devices in the current training round have been received and then enters the next training round. This, however, means that the time it takes to complete a training round is limited by the slowest device in the participants. Even a single straggler could significantly prolong the training process. Moreover, different random behaviors (e.g., smartphone out of battery and poor network connection) can take place in the highly dynamic training environment which easily block the whole training system. Thus, the training efficiency and the scalability of the system are severely impacted. Second, local training is only conducted when the participants are in charge and with good WiFi connection in order to reduce the impact on battery lifetime of mobile devices (e.g., smartphone, wearable devices) [2]. This highly contradicts the ubiquitous characteristics of mobile computing and limits the application scope of Federated Learning. More and more scenarios such as autonomous vehicles and online business analytics have requirements for processing the fresh data in a timely manner [4].

Key Observations and Challenges. The participants in a Federated Learning system can consist of mobile devices

978-1-7281-6887-6/20/$31.00 © 2020 IEEE

with diverse hardware configurations. In addition, the amount of training data on each participant highly depends on the user interaction which can be highly unbalanced. Different computing capacities coupled with various amounts of training data lead to totally different local training completion time. It is easy to see that the completion time of a training round can be severely bottle-necked by the participants that require the longest time to complete local training. Thus, one important question is whether the gradient updates from all the participants are required to guarantee the model accuracy. How to improve the scalability and robustness of a Federated Learning system is a critical challenge in practice. On the other hand, on-device training is computing intensive which can cause high energy consumption and seriously hurt the battery lifetime. Considering the requirement of processing the fresh data in a timely manner, energy efficiency is an important concern for battery-powered mobile devices. Then, another critical challenge is how to improve the energy efficiency while guaranteeing the training progress in a mobile-based Federated Learning system. *Thus, a framework that can intelligently conduct the training process in a timely, robust and energy-efficient manner is of great importance for effectively deploying Federated Learning on mobile devices in large scale.*

Our Contribution. In this paper, we propose MCFL, a multi-layer coordination system for high-performance energy-efficient federated learning. MCFL mainly consists of two layers: a macro layer and a micro layer. For the macro coordinator, we first select the *right* devices to participate in the current training round through jointly considering the amount of training data, computing capability and runtime training behavior of each device. In addition, the macro coordinator well estimates a time limit for the current training round in order to ensure that sufficient gradient updates can be collected within this time window in order to guarantee the model accuracy. For the micro layer, we observe that the current energy management mechanism on mobile devices is not energy optimal for a Federated Learning system. Thus, during the on-device training process, the micro coordinator intelligently coordinates different system components in order to complete the local training process within the time limit using the least amount of energy. Specifically, this paper makes the following contributions:

- We propose MCFL, a multi-layer coordination framework that intelligently balances the training completion time, model accuracy and energy efficiency in a Federated Learning system.
- To achieve high performance, we observe that a certain amount of gradient updates from the devices is sufficient to achieve the target model accuracy. Consequently, we design 1) a participant selection mechanism to dynamically determine the *right* mobile devices to participate in each training round, and 2) a time limit estimator that sets an appropriate time limit to help balance training time and model accuracy.
- We design an on-device coordinator to effectively coordinate different system components in order to complete the on-device training process in an energy efficient manner.
- We prototype MCFL on a Federated Learning system consisting of commercial mobile devices with heterogeneous hardware configurations.

The rest of the paper is organized as follows. Section II introduces the background of a typical Federated Learning system. Section III describes the key observations that motivate our design of MCFL. Sections IV-VI discuss the system design of our multi-layer coordination framework. Section VII evaluates the effectiveness of MCFL from different perspectives. Finally, Section VIII concludes the paper.

II. Background

A. Federated Learning

Federated Learning (FL) is a type of distributed learning that attempts to leverage the data being generated on mobile devices while protecting data privacy. A typical FL system comprises a central server and multiple mobile devices with heterogeneous hardware configurations. It works according to the following procedure:

- **Initialization:** At the initialization stage of each training round, the central server broadcasts the model to be trained to each participating device.
- **On-device Training:** The participating devices train the received model with their own training data.
- **Model Aggregation:** All the participating devices upload their gradient updates to the central server. The central server then aggregates the received local gradients and updates (e.g., model averaging) the global shared model. After that, the central server broadcasts the updated model to the participating devices and the whole system enters a new training round. This procedure iterates until the model converges.

We can notice that the raw training data (e.g., sensitive usage data) never leave the devices during the whole training procedure. In this way, the data privacy is well protected.

B. Related Work

Previous work has tried to improve the efficiency and effectiveness of federated learning from different perspectives [3], [5]–[8]. Wang et al. [5] aim to identify the irrelevant updates made by clients and preclude them from being uploaded for reduced network footprint. Konecny et al. [6] propose two approaches (structured updates and sketched updates) to reduce the uplink communication costs. Lin et al. [7] propose Deep Gradient Compression that integrate techniques including momentum correction, local gradient clipping, momentum factor masking and warm-up training in order to reduce the communication bandwidth. Verma et al. [8] design approaches that can result in effective machine learning models in the environments where the data may be highly skewed, and investigate their performance under different scenarios. We can find that the previous research can be mainly divided into two categories: 1) reducing the communication overhead and 2) improving the model accuracy in different scenarios. However, MCFL is proposed to intelligently trade off the convergence rate, model accuracy and energy efficiency (on-device learning) from a new perspective.

III. Motivation

In this section, we discuss the key observations that motivate the design of MCFL.

Obs 1: Model accuracy does not increase linearly with the amount of training data in Federated Learning.

We conduct the following experiments in order to investigate the relationship between model accuracy and training data size in Federated Learning. In the experiment, we train different models (e.g., 2-Layer CNN, Lenet-5 [9], AlexNet [10]) with the Mnist data set [11] which contains 60000 images in total. We distribute the training data in two different ways, random (each device is given a random amount of data) and uniform (evenly distributed) among 200 participating devices. Figure 1 shows the corresponding results. The X-axis shows the fraction (e.g., 0.1 means $60000 \times 0.1 = 6000$ images are distributed) of training data. Y-axis shows the corresponding model accuracy. We find that the model accuracy does not increase linearly with the amount of training data in a Federated Learning setting. The model accuracy becomes stable when certain amount of data are successfully trained, which is consistent with the previous work [12]. Thus, adding more data would not significantly improve model accuracy. Moreover, the distribution of training data on mobile devices does not highly impact the model performance. Furthermore, Figure 2 shows impact of the amount of training data on the model convergence rate. The x-axis represents the number of training round while the y-axis represents the model accuracy. We compare the relationship between the model accuracy and the training rounds when different amounts (e.g., 80%, 90% and 100%) of training data are utilized during the training process. We can find that the convergence rate in a Federated Learning system is not impacted when the training data reaches a certain amount (e.g., 80% of Mnist which is 48000 images in this case). It is important to note that the amount of training data required to achieve target performance can be obtained through fitting an inverse power law model [13].

TABLE I: Workload Profiling

Name	IPS	L2 Cache Miss
AngryBirds	0.143G	0.025M
On-Device Training	5.65G	1.317M

Obs 2: On-device training is computing intensive and current energy management mechanism is not energy optimal for Federated Learning.

Table I shows the comparison of instructions per second and L2 cache misses of on-device training (train the Lenet-5 model with Mnist dataset) and the well-known mobile game Angrybirds. We see that on-device training is both CPU and memory intensive. Figures 3a and 3b show the histograms of CPU frequency and memory bandwidth selected by the default power management governors [14] at runtime when on-device training is conducted on a Nexus 6 smartphone. We see that the default governors usually select higher frequency levels and memory bandwidths during the training time. This is because the default governors of Android are utility-based – they select frequencies only based on how busy the hardware component is. Therefore, higher frequencies will be selected when the load is high, as is the case in Table I. While this type of algorithms allows the on-device training to complete as soon as possible, it may not be energy efficient for Federated Learning. Though a certain device can complete the local training process as soon as possible, the whole system still need to wait for other devices that require more training time (e.g., low-end devices with more training data) in order to get sufficient data to be trained before entering the next training round. Thus, unnecessarily high CPU frequency and memory bandwidth only lead to high energy consumption and will not accelerate the overall training process.

IV. System Overview of MCFL

Figure 4 shows the system architecture and workflow of MCFL. It contains two main components in order to intelligently coordinate a Federated Learning system: 1) a macro-coordinator located in the central server and 2) a micro-coordinator located on each mobile device. The entire system works as follows.

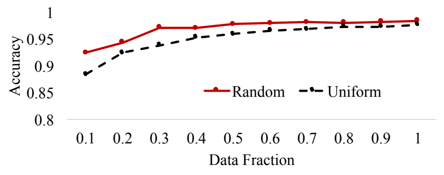
Step 1: At the initialization stage, the participating devices send the following local information to the macro-coordinator: 1) hardware information (e.g., available CPU frequency range, memory bandwidth range) and 2) the amount of data objects to be trained.

Step 2: After receiving the local information from the participating devices, the macro coordinator selects the *right* devices to participate in the current training round through jointly considering the training ability, amount of training data and runtime training behavior. In addition, it estimates a time limit (first time limit) of the current training round to balance the overall training progress and model accuracy. The macro coordinator then broadcasts the time limit to all the selected participants.

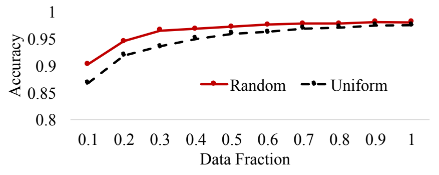
Step 3: The micro coordinator located on each mobile device then intelligently coordinates different system components to ensure that the on-device training process can be completed within the time limit with the least amount of energy.

Step 4: When the first time limit arrives, the macro-coordinator checks the progress of the current training round (i.e., the amount of data objects that have been trained and the corresponding weights have been successfully updated). If the central server has already received sufficient weight updates, the whole system directly enters the next training round. Otherwise, the macro coordinator checks the training progress of each device and conducts the participant selection and estimates the second time limit and broadcasts it to the selected devices.

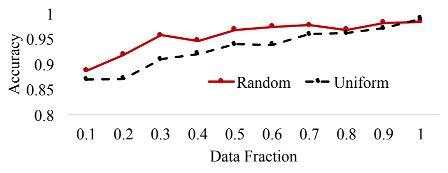
Step 5: After receiving the second time limit, the micro coordinator coordinates the local training process in the same way as that in Step 3. When the second time limit arrives, the macro-coordinator checks the current progress. If sufficient data objects have already been trained, the whole training system can directly enter the next training round. Otherwise,



(a) 2D CNN.

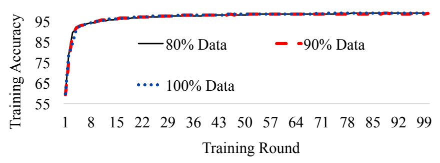


(b) Lenet5.

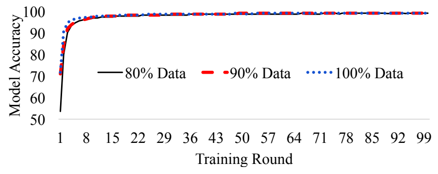


(c) AlexNet.

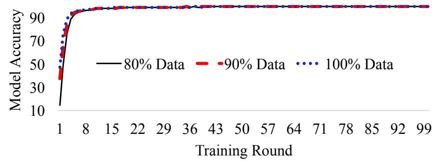
Fig. 1: Impact of training data distribution on model accuracy.



(a) 2D CNN.

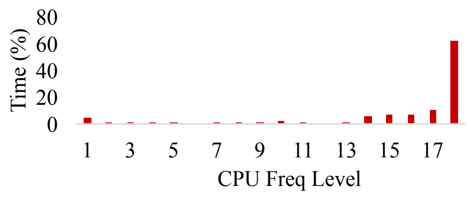


(b) Lenet5.

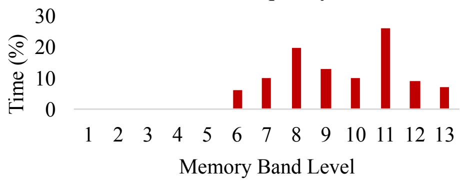


(c) AlexNet.

Fig. 2: Impact of the amount of training data on convergence rate.



(a) CPU Frequency.



(b) Memory Bandwidth.

Fig. 3: CPU frequency and memory bandwidth selected by the default governor during the training process.

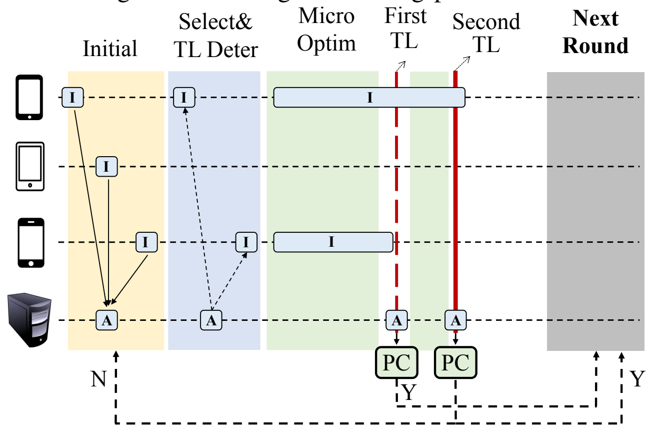


Fig. 4: Workflow of MCFL per training round (A: MACRO Coordinator, I: MICRO Coordinator, PC: Progress Check, TL: Time Limit.)

the current training round restarts to prevent the whole system from blocking indefinitely.

In the following sections, we will discuss the specific design of the macro and micro coordinators in details.

V. Macro-coordinator

Figure 5 shows the architecture of the macro coordinator, which mainly contains the following three components: Training Ability Predictor, Participant Selector and Time Limit Determinator. It conducts the following three tasks: 1) intelligently estimates the training ability of each mobile device and 2) selects the *right* devices to participate in the current training round and 3) determines a time limit to balance the system robustness, efficiency and model accuracy.

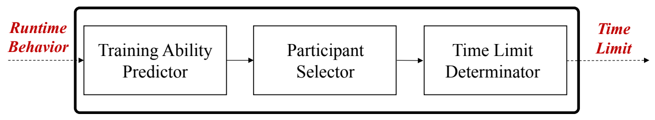


Fig. 5: System Architecture of the Macro Coordinator.

A. Training Ability Predictor

On-device training is both CPU and memory intensive. Thus, CPU frequency and memory bandwidth can jointly determine the training ability of a device. We define the training ability A_i of device i as the number of instructions of the training task that can be processed in a unit time, which can be represented as follows:

$$A_i = f_i^{\alpha_i} m_i^{\beta_i} \tag{1}$$

where f_i represents CPU frequency and m_i represents memory bandwidth on device i, while α_i and β_i are the corresponding coefficients. Thus, the time required to train a certain amount of data objects on device i can be represented as follows:

$$t_i = \frac{c_i D_i}{A_i} \tag{2}$$

where c_i represents the number of instructions required to process one training data object which can be obtained through offline profiling, while D_i represents the number of data objects. We determine the parameters α_i and β_i in Eq. (1) through a system identification process [15]. Specifically, we collect data on real mobile devices and establish a statistical model based on the measured data. It is important to note that the system identification process is conducted offline before the whole learning process starts.

Apart from hardware configuration, resource contention caused by user interaction with the foreground app (concurrent

TABLE II: Training Process Impacted by the Concurrent Apps

Con-App	No	Badminton	AngryBird	BasketBall
IPS(Train)	4.49G	3.16G	2.69G	1.37G
IPS(Con-App)	0	0.1G	0.144G	0.448G
Train CompTime	3.072s	5.5s	6.52s	8.268s

app) can also highly impact the background training process, and must be considered if we were to allow a device to participate in federated learning even when it is being used for other purposes. Table II shows the IPS and the local training completion time under different scenarios. We find that the higher the IPS of the foreground app is, the lower the IPS of of the background training process is, which leads to a longer time to complete the local training. Moreover, the training completion time is highly correlated with the IPS of the local training process. Thus, we introduce another coefficient (u_i) to characterize the impact of resource contention on the training ability of a certain device at runtime. Specifically, u_i is defined as follows:

$$u_i = \frac{I_i^{train}}{I_i^{train\_only}} \tag{3}$$

where I_i^{train} represents the average IPS of the background training process in a training round, while $I_i^{train\_only}$ represents the IPS of the training process when there is no concurrent app running. Thus, the range of u_i is between 0 and 1. The higher the resource contention is, the lower the training ability of a device is and then u_i is closer to 0. Thus, the training ability of a device i at runtime can be represented as follows:

$$A_i^{runtime} = u_i * A_i \tag{4}$$

We can see that when $u_i = 0$ (e.g., mobile device runs out of battery), the training ability of device i is 0, while the runtime training ability only depends on the hardware configuration when $u_i = 1$ (e.g., there are no concurrently running processes). In a mobile device, u_i usually dynamically changes according to different user interaction patterns across different training rounds. Effectively predicting the runtime training ability in the upcoming round is critical to participant selection and time limit estimation.

In order to characterize the long-term history of user interaction behavior with a smartphone, we train a Long Short-Term Memory (LSTM) model [16] (two hidden layers and 50 units in each layer) with the history training ability information (measured with the runtime monitor in each micro coordinator in each training round). LSTM introduces a memory cell and employs gate mechanism to control the information flow. It has shown powerful ability to model long range dependencies embedded in the sequential data. In particular, the user interaction trace from LiveLab [17] is adopted to train the LSTM predictor. In each training round, the measured u_i (from the micro coordinator) from device i is sent as input to the LSTM predictor in order to predict the u_i in the upcoming training round. The training is conducted at the same time with the prediction process.

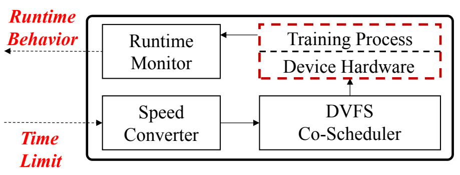


Fig. 6: System Architecture of the Micro Coordinator.

B. Participant Selector and Time Limit Determinator

The selection of participating devices not only impacts the model accuracy, but also highly impacts the whole training progress. Thus, participant selection becomes a critical issue for efficient deployment of Federated Learning system in practice. In this paper, we formulate participant selection as a constrained optimization problem jointly considering the amount of training data, hardware configuration and runtime behavior. We first define the notations as follows: N is the number available devices in the current training round. $D_{required}$ represents the amount of data required to be trained in order to achieve predefined model accuracy, which is obtained through fitting the inverse power law model [13]. D_{backup} represents the extra amount of data to be trained in the current round in case of uncertainty issues such as out of battery, poor network connection. θ_i is a binary variable used to determine whether device i is selected in the current round. C_k represents the time limit of the current training round.

The participant selection and time limit determination problem is formulated as follows:

$$Min\{C_k\} \tag{5}$$

Subject to

$$\sum_{i=1}^{i=N} \theta_i * D_i \geq D_{required} + D_{backup} \tag{6}$$

$$\frac{\theta_i * c_i * D_i}{u_i * A_i} \leq C_k, \quad i \in \{1, 2, ..., N\} \tag{7}$$

Equation 5 represents the optimization goal in which we aim to minimizing the training completion time of training round k, C_k. Equation 6 guarantees that sufficient amount of data can be trained in the current round to achieve the predefined model accuracy while Equation 7 guarantees that the selected devices can complete the on-device training within the time limit. The time limit (C_k) and the selected devices (θ_i) can be obtained through effectively solving the optimization problem.

VI. Micro-coordinator

Once the time limit is broadcast to the participants, the micro-coordinator on each selected mobile device conducts local optimization to complete the training within the time limit while minimizing energy consumption. Figure 6 shows the architecture of the micro coordinator which contains the following main components. 1) The *Speed Converter* converts the time limit (C_k) received from the macro-coordinator to an internal speed metric, namely, instructions per second (IPS).

This *target speed* is passed to the DVFS Co-Scheduler. 2) The *DVFS Co-Scheduler*, using the target speed as a basis, schedules the hardware resources, namely, the CPU frequency and memory bandwidth to achieve the speed requirement with minimal energy. 3) The Runtime Monitor measures the runtime impact of training ability (e.g., u_i defined in Equation 3) during the training process and feeds back to the macro coordinator for runtime training ability prediction of the next training round.

A. Speed Converter

The input and output of the Speed Converter are as follows:

Input: time limit (C_k), performance model (Equation 2) and power model (Equation 8). The performance model characterizes training time as a function of hardware settings and data size. To optimize energy, a power model [18] is built that takes into account both CPU frequency and memory bandwidth:

$$p = a * f^3 + b * m + c \tag{8}$$

where f and m represent the CPU frequency and memory bandwidth respectively. a, b and c are the corresponding coefficients of a specific device which are determined through the system identification process.

Output: target speed r.

For a training round k with time limit C_k mandated by the macro-coordinator, the Speed Converter calculates a target speed r measured in instructions per second (IPS) such that the training round can be completed in C_k seconds if it runs at r IPS. At the same time the energy consumption in the training period is minimized. The foundation for this is the fact that CPU and memory as well as other hardware components on contemporary mobile devices generally support dynamic voltage and frequency scaling (DVFS) [19]. By intelligently coordinating CPU frequency and memory bandwidth, the system can meet the time limit while optimizing energy, thereby conserving battery life. This problem can be formulated as a constrained optimization problem as follows:

$$\text{Find } \operatorname*{argmin}_{f,m} E(f,m) = t^{idle} * p^{idle} + t^{train} * p^{train}(f,m) \tag{9}$$

$$\text{subject to } \; t^{idle} + t^{train} = C_k, \; 0 \le t^{idle}, t^{train} \le C_k$$
$$f^{min} \le f \le f^{max}, \; m^{min} \le m \le m^{max} \tag{10}$$

where $E(f,m)$ is the energy consumption of the on-device training process in the C_k time period when the CPU frequency is set to f and memory bandwidth is set to m, t^{idle} and t^{train} are respectively the time spent in the idle state and running the training task, and p^{idle} and p^{train} are the corresponding power consumption.

Once the optimal CPU frequency and memory bandwidth in Equation 9 is found, the required speed in IPS can be found using the following equation:

$$r = \frac{c_i D_i}{t_i^{train}} \tag{11}$$

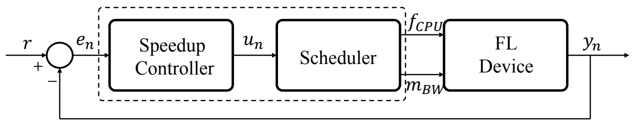


Fig. 7: Design of DVFS Co-Scheduler as a feedback control loop.

where c_i represents the number of instructions required to process one training data object, while D_i represents the number of data objects required to be trained.

B. DVFS Co-Scheduler

The input and output of the *DVFS Co-Scheduler* are as follows:

Input: target speed r.

Output: schedule S for hardware resources, i.e., CPU frequency and memory bandwidth in this work.

Given the target speed r, the simplest scheduling strategy is to just set the CPU frequency and memory bandwidth to the optimal value f_{opt}, m_{opt} found by the Speed Converter. However, there are a few issues with this strategy. First, available CPU frequencies and memory bandwidths on a mobile device have discreet values. Therefore, f_{opt} and m_{opt} may fall between two available values. Second, modeling errors are inevitable. Third, as the training process is running, performance variations may occur. To better deal with modeling errors and runtime performance fluctuations, we have adopted the feedback loop based resource scheduler from [20]. Figure 7 shows the overall structure of the *DVFS Co-Scheduler*. As can be seen, in this feedback control system, the mobile device is the plant [21] while the Speedup Controller and the Scheduler together serve as the controller.

Speedup Controller: The control system operates in cycles. In this context the length of a cycle is usually 1 to 3 seconds. Within a training round, there are usually multiple control cycles. As each cycle begins, the difference $e(n)$ between the target speed r and the actual speed $y(n)$ is calculated, and is fed to the Controller. The goal of the Controller is to calculate a control signal that minimizes the errors. We adopt an integral controller [21] to minimize the accumulated error $\Sigma e(n)$. The output of the Speedup Controller is as follows:

$$s(n) = s(n-1) + g \cdot \frac{e(n-1)}{b(n-1)} \tag{12}$$

The control signal $s(n)$ is its value in the previous cycle plus the most recent error multiplied by a scaling factor. The constant g ($0 < g < 1$) is the controller gain [21]. Although $s(n)$ could represent the desired absolute IPS value, here we use the relative speed instead — $s(n)$ represents the speedup with regard to the *base speed* $b(n)$, which is the speed obtained when CPU frequency and memory bandwidth are set to their lowest values. The major reason for decomposing the speed into the base speed and speedup is that $b(n)$ itself is dynamically estimated to deal with modeling errors between the performance model and actual performance.

Scheduler: Using the speedup $s(n)$ calculated by the Speedup Controller, the Scheduler then determines a resource schedule that meets the speedup with minimal energy. Since training a neural network on a mobile device is both CPU and memory intensive, in this work we consider controlling DVFS for both CPU and memory. Given a set of N selected configurations, each member being a pair <CPU frequency, memory bandwidth>, we can determine in advance the performance speedup s_i and power consumption p_i for each configuration in the set. The Scheduler uses these data to determine an optimal schedule $\tau = [\tau_1, \tau_2, ...\tau_N]$, where τ_i means applying the i-th configuration for τ_i time units. As it turns out, the scheduling problem is a linear programming problem as follows:

$$\text{Minimize energy consumption} \quad \sum_{i=1}^{N} \tau_i \cdot p_i \tag{13}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \tau_i \cdot s_i = s(n) \tag{14}$$

$$\text{and} \quad \sum_{i=1}^{N} \tau_i = T, \quad 0 \leq \tau_i \leq T \tag{15}$$

where T is the control period of the feedback controller. Applying linear programming theory [22], we can further simplify this problem to requiring only two elements of the vector τ to be non-zero.

C. Runtime Monitor

The *Runtime Monitor* monitors the average IPS during the local training process. This information is fed back to the macro coordinator at the end of each round so that it can update the prediction model for participant selection.

VII. Results

In this section, we first introduce the experimental methodology and then present the evaluation results.

A. Experimental Methodology

We evaluate the effectiveness of MCFL with both simulation and hardware testbeds. For simulation, we build a simulator following a server/client architecture, in which different processes are created to simulate the central server and participating devices. In addition, the hardware configuration, training completion time model and the power model are well integrated. The local learning process is implemented with PyTorch [23] and the Livelab trace [17] is adopted to emulate the corresponding user interaction. For the hardware testbed, we establish a Federated Learning system consisting of six smartphones with different hardware configurations including Honor, Lenovo, ZTE, Mi, Nexus and Samsung. They are configured with different Android verions (e.g, from 5.1.1 to 8.0), different number of CPU cores (e.g., from 4 to 8) and different available CPU frequency, memory bandwidth ranges. The on-device training process is implemented as a background service in DL4J [24]. A Monsoon power monitor [25] is utilized to measure the power consumption of different devices. In addition, we use the following baselines to evaluate the performance of MCFL.

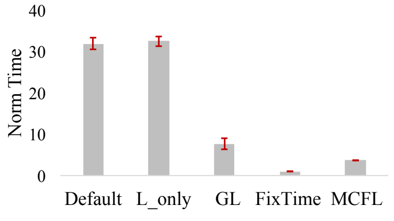
- **Default**: On the server side, model averaging is only conducted when gradient updates from all the clients are received [2]. In each mobile device, the default CPU governor (*interactive*) and *devfreq* governor ($cpubw\_hwmon$) manage DVFS for the respective components.
- **L_only** (Local optimization Only): The central server estimates a time limit within which all the participating devices are expected to complete the on-device training. The participating devices then minimize the energy consumption of the training process through intelligently scheduling CPU frequency.
- **GL** (Global and local optimization): GL conducts coordination at both macro and micro levels. On the server side, GL determines the time limit to guarantee that certain percentage of the participating devices can successfully submit their gradient updates (e.g., 80% is utilized in the experiment). On the device side, GL conducts energy optimization through dynamically adjusting CPU frequency.
- **FixTime**: In the initialization step, the central server selects the participating clients in order to guarantee that sufficient amount of data can be trained and determines the time limit according to the selected devices. After that, this time limit is fixed for every training round. On the device side, each device performs energy optimization through controlling CPU frequency.

Table III shows the summary of the characteristics of different schemes.

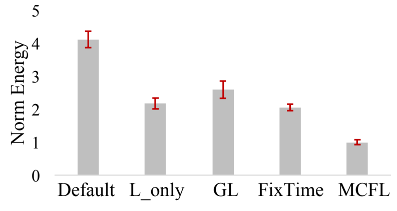
B. Evaluation with Uniform Distribution of Local Training Data

In this section, we evaluate the effectiveness of MCFL from the following perspectives: training completion time, energy consumption and data completion ratio. Specifically, data completion ratio is defined as the amount of data that are successfully trained (the corresponding gradient updates are successfully updated) divided by the amount of data required to be trained. In the experiment, we build a simulation testbed consisting of 1 central server and 200 mobile devices. The hardware configuration of each mobile device is randomly selected from the following six types including: Honor, Lenovo, ZTE, Mi, Nexus 6 and Samsung S3. In the experiment, we train a Lenet5 [9] model with the Mnist dataset [11] for the image classification task. Each device is allocated 500 images. It is important to note that although we use the well known image classification task as an example to demonstrate the effectiveness of MCFL, MCFL is designed for general Federated Learning tasks.

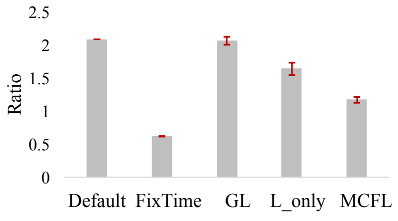
Figure 8 shows the corresponding results. For the training completion time and energy consumption in each round, we normalize the measured value to the smallest one in order to effectively present the characteristics of different methodologies. Figure 8a represents the average training completion time of each training round. We can find that *Default* and *L_only* have the longest training completion time. This is because these two schemes try to wait for all the participating devices to successfully submit their gradient updates in each training round. In those cases, the overall training progress can be severely bottle-necked by the devices with low-end



(a) Training completion time per round.



(b) Energy consumption per round.

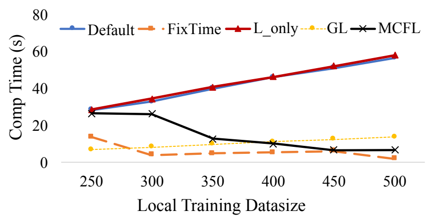


(c) Data completion ratio per round.

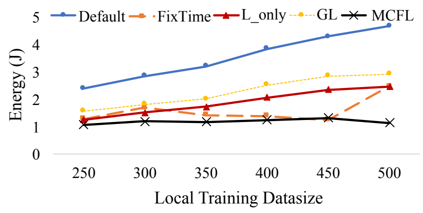
Fig. 8: Comparison of different schemes with fixed amount of local training data.

TABLE III: Summary and Comparison of Schemes

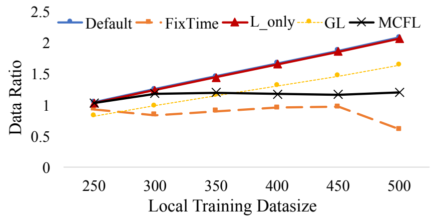
Name	Server Side	Client Side
Default	Wait for all devices to complete per round	Default governors
L_only	Estimate time for all devices to complete per round	Control CPU frequencies
GL	Estimate time for certain percentage of clients to complete per round	Control CPU frequencies
FixTime	Select devices; initial time limit for all training rounds	Control CPU frequencies
MCFL	Select devices; estimate time for certain amount of data to be trained	Control CPU frequencies and memory bandwidth



(a) Training completion time per round.



(b) Energy consumption per round.



(c) Data completion ratio per round.

Fig. 9: Comparison of different schemes with different amounts of local training data.

TABLE IV: Model Accuracy

	Default	L_Only	GL	FixTime	MCFL
Lenet5	98.23%	98.15%	98.13%	92.27%	98.1%

hardware configurations (e.g., ZTE, Lenovo) and low runtime training ability (e.g., highly impacted by the resource contention caused by the foreground app). Compared to *Default* and *L_only*, *GL* is more robust and leads to shorter training completion time, as it does not require *all* the participants but only requires *a certain percentage* of participants to complete gradient updates, which increases the robustness and scalability of the Federated Learning system. *MCFL* further reduces the training completion time for the reason that it only requires the predefined amount of data to be trained and reduces the number of required gradient updates even further. Moreover, *MCFL* jointly considers the amount of training data, hardware configuration and runtime training ability in order to select the *right* devices to conduct the training process in a more efficient manner. Compared with the *Default* scheme, *MCFL* accelerates the training completion time by 8.66×, while it accelerates by 2.06× compared with the *GL* methodology. Though the *FixTime* scheme maintains the lowest training completion time, it usually cannot meet the data ratio requirement (hence reduced model accuracy), which will be discussed in details in the following.

Figure 8b shows the normalized energy consumption with different schemes. We can see that *Default* leads to the highest energy consumption. This is because on-device training is both CPU and memory intensive. The default CPU and memory governors on mobile devices usually select the high-end CPU frequencies and memory bandwidths most of the time (see Figure 3), which leads to high energy consumption during the local training process. *L_only* consumes much lower energy, as it conducts energy optimization through adjusting the CPU frequency on the local device side. *GL* consumes more energy than *L_only* for the reason that *GL* maintains much lower training completion time which provides less optimization space for the energy optimizer. *MCFL* consumes the least amount of energy in the local training process. This results from the following two perspectives: 1) from the global perspective, *MCFL* intelligently selects the *right* devices to participate in the training process and 2) from the local perspective, *MCFL* coordinately adjusts *both* the CPU frequency and memory bandwidth to minimize the energy consumption within the time limit in each training round. In this experiment, *MCFL* achieves 75.6% energy savings compared with the *Default* scheme and 61.38% compared with *GL*.

Figure 8c shows the average data ratio per training round. We see that *FixTime* cannot successfully train sufficient amount of data in each training round (i.e., data ratio is less than 1). This is because *FixTime* adopts a fixed time limit for each training round and does not consider the impact of the variation of training ability at runtime of different devices. The other schemes can usually enable sufficient amount of data to be trained in each training round (i.e., data ratio ≥ 1). Table IV shows the model accuracy. We can see that *MCFL* can achieve the same level of model accuracy as *Default* and *GL* while *FixTime* has considerably lower accuracy.

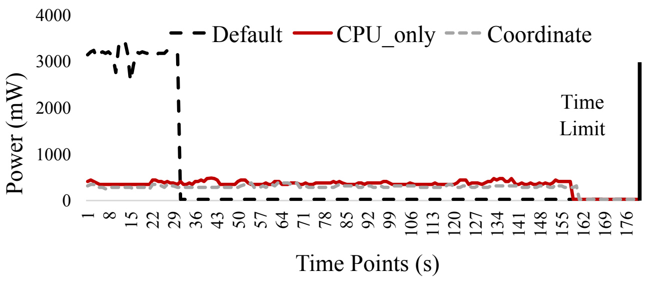
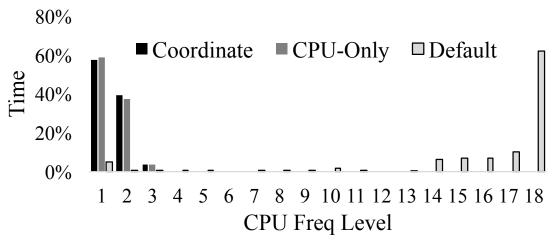


Fig. 10: Power consumption of the training process with different schemes.

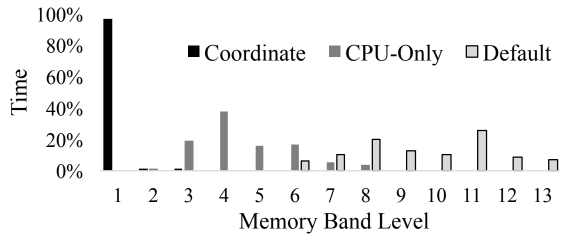
C. Evaluation with Different Amounts of Local Training Data

In this section, we evaluate the effectiveness of MCFL under different scenarios when the participating devices are allocated with different amounts of local training data. Figure 9 shows the evaluation results. The x-axis represents the number of images for training in each mobile device. It is important to note that the amounts of training data across different mobile devices are the same in each experiment scenario. Figure 9a shows the result of completion time. We see that the completion time of *Default* and *L_only* keeps increasing as the amount of local training data increases for the reason that these two schemes expect all the participating devices to complete their local training and larger amount of training data prolongs the local training completion time. On the other hand, the completion time of *MCFL* decreases as the local training data size increases. As discussed, MCFL requires a predefined amount of data to be trained to guarantee the model accuracy. The Participant Selector gives preference to the devices with high-end hardware configuration and higher runtime training ability. As long as these devices meet the data size requirement, the whole training process is not bottle-necked by the devices with low hardware configuration and low runtime training ability.

Figure 9b shows the results of energy consumption. We can notice that MCFL consumes the least amount of energy. Moreover, the total energy consumption does not increase with the local amount of training data on each device. This is due to the following two reasons. First, MCFL intelligently coordinates different system components (CPU frequency and memory bandwidth) to conduct the local training in an energy efficient way. Second, MCFL does not require all the devices to participate in the training process. Thus, as the amount of local training data increases, fewer mobile devices are selected to conduct on-device training which prevents significantly increase in the total amount of energy. Figure 9c shows the corresponding results for data completion ratio. We see that MCFL effectively controls the completion ratio to the predefined reference point through the feedback loop between the macro and micro coordinators. However, *FixTime* can hardly achieve the predefined ratio in most cases. *GL* also fails to complete sufficient local training in certain scenarios (e.g., when each device has 250 images).



(a) CPU frequency selected by different schemes.



(b) Memory bandwidth selected by different schemes.

Fig. 11: CPU frequency and memory bandwidth selected by different schemes on a Nexus 6 smartphone.

D. Evaluation of the Micro-coordinator

The micro-coordinator intelligently coordinates different system components in order to complete the on-device training process with minimum energy while meeting the time limit received from the macro coordinator in each training round. Figure 10 shows the power consumption under different schemes on a Neuxs 6 smartphone. Specifically, *Coordinate* represents the micro coordinator in MCFL in which CPU frequency and memory bandwidth are jointly scheduled. *CPU_Only* represents the energy optimizer adopted by *L_only*, *GL* and *FixTime* in which only CPU frequency is controlled. *Default* represents the scheme in which both CPU frequency and memory bandwidth are managed by the default governor on Android. The default scheme completes local training in 29s with an average power of 2947mW. *CPU_only* completes the training process in 159s with 387mW and *Coordinate* takes 161s with 312mW within the time limit. Compared to *Default*, *CPU_only* achieves 30.8% energy savings, while *Coordinate* achieves 43.3%.

Figure 11 shows histograms of system configuration. Figure 11a shows the result of CPU frequency. We see that *Default* selects the highest frequencies in most of the time (e.g., 62% in level 18 and 10% in level 17). However, *Coordinate* and *CPU_Only* conduct similar selection for CPU frequencies but spend nearly 60% of time in the lowest frequency and almost 40% of time in the second lowest one. Lower frequency selection leads to lower energy consumption at the same time. Figure 11b shows the selection of memory bandwidth during the local training process. *CPU_Only* spends most of the time in level 3, 4, 5, and 6 (19% in level 3, 38% in level 4, 16% in level 5 and 16% in level 6). However, *Coordinate* spends most of the time in the lowest bandwidth. This can be explained with Figure 12, which shows the training completion time speedup and the normalized power at different memory bandwidth levels when the CPU frequency is fixed (level 0 in this experiment). It shows that the acceleration in training completion time is negligible when the memory bandwidth

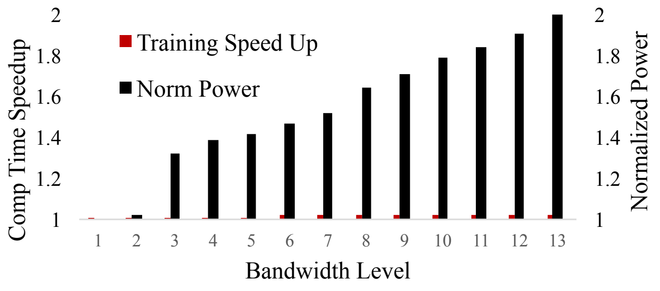


Fig. 12: Normalized training completion time and power with different memory bandwidth levels.

increases, while higher bandwidth level leads to much higher energy consumption. Thus, compared with *CPU_Only*, *Coordinate* completes the local training process using similar amount of time and achieves further energy saving through jointly scheduling CPU frequency and memory bandwidth.

We also evaluate the impact of the foreground app during the on-device training (in the background) procedure. We measure the FPS (Frame Per Second) of different apps which represents the user-perceived performance. The performance degradation is less than 7.14% which is negligible. The macro coordinator mainly runs on the central server, whose overhead is not the main concern. The overhead of the micro coordinator is less than 50mw which is negligible.

VIII. Conclusion

In this paper, we propose MCFL, a multi-layer online coordination framework for mobile-based Federated Learning in real time. MCFL consists of two layers: a macro layer on the central server and a micro layer on each participating mobile device. In each training round, the macro coordinator selects the right devices to participate, and estimates a time limit, such that the overall training time is significantly reduced while still guaranteeing the model accuracy. Unlike the state-of-the-art systems, MCFL removes the restriction that participating devices must be connected to power sources, thus allowing more timely and ubiquitous training. The micro coordinator determines optimal schedules for hardware resources in order to meet the time limit set by the macro coordinator with the least amount of energy consumption. We evaluate the effectiveness of MCFL on both hardware and simulation testbed. The evaluation results show that MCFL effectively balances the convergence rate, model accuracy and energy efficiency. Compared with existing systems, MCFL achieves a speedup up to $8.66\times$ and reduces energy consumption by up to 75.6%.

IX. Acknowledgement

This work is supported by the National Key R&D Program of China (No. 2019YFB2102100), Science and Technology Development Fund of Macao S.A.R (FDCT) under number 0015/2019/AKP, Guangdong Key R&D Project (No. 2020B010164003), Shenzhen Discipline Construction Project for Urban Computing and Data Intelligence.

References

[1] S. Halpern, "The champaign for mobile phone voting is getting a midterm test." https://www.newyorker.com/tech/annals-of-technology/, 2018.
[2] H. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.
[3] K. Bonawitz, "Towards federated learning at scale: System design," *SYSML*, 2019.
[4] Terry Erisman, "Achieving real-time machine learning and deep learning with in-memory computing," https://jaxenter.com/in-memory-computing-machine-learning-145623.html, 2018.
[5] L. Wang *et al.*, "Cmfl: Mitigating communication overhead for federated learning," in *ICDCS*, 2019.
[6] J. Konečnỳ *et al.*, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
[7] Y. Lin *et al.*, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.
[8] D. C. Verma *et al.*, "Approaches to address the data skew problem in federated learning," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, 2019.
[9] Yann Lecun, "Lenet5," http://yann.lecun.com/exdb/lenet/.
[10] L. Xiao *et al.*, "Scene classification with improved alexnet model," in *ISKE*, 2017.
[11] Yann Lecun, "The MNIST Database," http://yann.lecun.com/exdb/mnist/.
[12] C. Beleites *et al.*, "Sample size planning for classification models." *Analytica chimica acta*.
[13] R. L. Figueroa *et al.*, "Predicting sample size required for classification performance." in *BMC Med Inform Decis Mak*, 2012, pp. 21–28.
[14] "Cpu frequency and voltage scaling code in the linux kernel." [Online]. Available: https://goo.gl/Mlr9U1
[15] T. Burton *et al.*, "Nonlinear system identification using a subband adaptive volterra filter," in *IEEE Instrumentation and Measurement Technology Conference*, 2008.
[16] F. Gers *et al.*, "Lstm recurrent networks learn simple context free and context sensitive languages," *IEEE Transactions on Neural Networks*, 2019.
[17] Rice University, "LiveLab: Measuring wireless networks adn smartphone users in the field," http://livelab.recg.rice.edu/traces.html.
[18] L. Zhang *et al.*, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *CODES+ISSS*, 2010.
[19] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing*. Springer, 1994, pp. 449–471.
[20] J. Wang *et al.*, "An application-specific approach to energy optimization on android mobile devices," *IEEE Transactions on Mobile Computing*, 2019.
[21] K. Ogata and Y. Yang, *Modern control engineering*. Pearson Upper Saddle River, NJ, 2010.
[22] B. Kolman and R. E. Beck, *Elementary linear programming with applications*. Academic Press, 1995.
[23] "Pytorch," https://pytorch.org/.
[24] M. Hamblen, "Deep Learning For Java," https://deeplearning4j.org/.
[25] Monsoon Solutions Inc, "High Voltage Power Monitor," https://www.msoon.com/online-store.