SmartPC: Hierarchical Pace Control in Real-Time Federated Learning System

Li Li[*,1], Haoyi Xiong[*,2,**], Jun Wang[3], Cheng-Zhong Xu[1,4],Zhishan Guo[5,**],
[1]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen China
[2]Big Data Lab (BDL) and PaddlePaddle, Baidu, Inc., Beijing, China
[3]Department of Electrical and Computer Engineering, McGill University, Montréal, Canada
[4]Department of Computer and Information Science, University of Macau, Macau, China
[5]Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL

***Abstract*—Federated Learning is a technique for learning AI models through the collaboration of a large number of resource-constrained mobile devices, while preserving data privacy. Instead of aggregating the training data from devices, Federated Learning uses multiple rounds of parameter aggregation to train a model, wherein the participating devices are coordinated to incrementally update a shared model with their own parameters locally learned. To efficiently deploy Federated Learning system over mobile devices, several critical issues including real-timeliness and energy efficiency should be well addressed.**

This paper proposes SmartPC, a hierarchical online pace control framework for Federated Learning that balances the training time and model accuracy in an energy-efficient manner. SmartPC consists of two layers of pace control: global and local. Prior to every training round, the global controller first oversees the status (e.g., connectivity, availability, and energy/resource remained) of every participating device, then selects qualified devices and assigns them a well-estimated virtual deadline for task completion. Within such virtual deadline, a statistically significant proportion (e.g., $\geq$60%) of the devices are expected to complete one round of their local training and model updates, while the overall progress of multi-round training procedure is kept up adaptively. On each device, a local pace controller then dynamically adjusts device settings such as CPU frequency so that the learning task is able to meet the deadline with the least amount of energy consumption. We performed extensive experiments to evaluate SmartPC on both Android smartphones and simulation platforms using well-known datasets. The experiment results show that SmartPC reduces up to 32.8% energy consumption on mobile devices and achieves a speedup of 2.27 in training time without model accuracy degradation.

I. Introduction

With rapid development of real-time embedded systems, mobile devices (e.g., smartphone and wearable devices) have been widely used to provide ubiquitous services together with computation-intensive artificial intelligence (AI) tasks. As these devices are carried about everyday and everywhere, they could collect a huge amount of private data about the mobile users using the embedded sensors (e.g., camera, microphone, GPS, accelerometer). With the collected data, learning predictive models to adapt users' behaviors/contexts becomes a promising way to improve user experience [1]–[3]. However, the sensitive nature of the user data means there are serious issues collecting and storing them in a centralized location [4].

To ensure data privacy in the learning procedure, Federated Learning systems have been proposed to enable a large number of mobile devices to train a shared model collaboratively without sharing local private data [5], [6]. To achieve the goal, a Federated Learning system usually organizes all participating mobile devices around a central server for model/parameter sharing (instead of data sharing). Training is started at the same time from a common initialization and will continue in multiple training rounds. During each round of the learning procedure, every mobile device computes its own updates based on the current shared model using its local training data, then forwards the local updates to the central server. On the other hand, once the central server receives the updates from these mobile devices, it improves the shared model and sends the updated one to the participating devices. This process iterates until an accuracy level of the learning model is reached. In this way, the accurate predictive model can be obtained while the user data privacy is well protected, as the local training data are not shared directly. Thus, different kinds of data-sensitive applications can be well supported by Federated Learning systems, such as face detection [7], next-word prediction, on-device item ranking, content suggestions for on-device keyboards, next word prediction [8] and human activity recognition.

Despite all the promising benefits, several obstacles exist for Federated Learning to be viable. When conducting training on mobile devices, federated learning can be costly because the whole learning process requires multiple rounds of communication between the central server and the devices before the model converges. From the perspective of energy consumption, on-device training is highly energy demanding and hurts the battery lifetime of mobile devices. Previous research simply assumes that the training program is only carried out when the smartphone is being charged [7]. Yet, such strong assumption contradicts the original purpose of analyzing data on mobile devices, i.e., understanding the collected data ubiquitously and timely. Conducting machine learning in real-time is becoming more and more important in time-critical applications and fast changing environments, such as autonomous vehicles, military

∗Equal contribution.
∗∗Corresponding authors: xionghaoyi@baidu.com, zsguo@ucf.edu.

applications, health-care informatics, and business analytics [9]. Moreover, a lot of time-sensitive applications can also benefit from real-time mobile-based federated learning. For instance, the renowned ride-share service provider Uber uses machine learning through clients' smartphones to analyze the demand of vehicles in *real time* for their pricing model [10].

From the perspective of training completion time, the *heterogeneity* in a Federated Learning system can also greatly impact the efficiency of the training process. Mobile devices in a Federated Learning system often have different hardware configurations and computing capacities. Moreover, the training data on different mobile devices are usually highly unbalanced. Heterogeneous hardware configurations coupled with unbalanced training data lead to large variations in completion time in each training round. Current Federated Learning system adopts the synchronous model averaging approach [11], which means that the system does not enter the next training round until the central server receives the updated weights from all the participating devices. The progress of the overall training process is thus bottle-necked by the less-powerful devices that require more time to complete the training rounds. **Thus, a framework that can effectively balance the training progress, model accuracy and energy consumption in *real time* is urgently required for Federated Learning systems.**

In this paper, we explore the possibility of enabling Federated Learning on multiple battery-powered devices. In order to make Federated Learning practical, we propose SmartPC, a hierarchical pace control framework that efficiently coordinates the training progress of the whole Federated Learning system and optimizes the energy consumption of the participating mobile devices with heterogeneous hardware configurations. Specifically, SmartPC contains two main layers, 1) a global control layer and 2) a local control layer. The global pace controller intelligently determines the global training deadline for each training round according to the hardware configuration and runtime behavior collected from the participating mobile devices. A proper training deadline is important for training pace control. If the deadline is too tight, most of the participating devices can not catch the deadline and successfully submit their weight updates, and the model accuracy can be severely affected. On the other hand, if the deadline is too loose, it takes a large amount of time to complete the training for each communication round and prolongs the overall training progress. In this work, we design a feedback-based deadline assignment mechanism that dynamically determines the training deadline based on actual training progress information as well as the computing capacity of the devices to guarantee that a specific percentage of participants can successfully submit the weight updates to achieve the predefined model accuracy level.

After receiving the training deadline, the local pace controller dynamically adjusts the system configuration of the participating device so that the participant can meet the training deadline while minimizing the energy consumption. Our experiments show that the default governor on Android system is not energy optimal for Federated Learning system; it always selects unnecessarily high system configurations. The designed energy minimization approach overrides the default governor and dynamically selects the optimal system configuration according to the local training progress in order to complete the local training in an energy efficient way.

We compare SmartPC with a state-of-the-art Federated Learning control system and show that SmartPC achieves up to 32.8% less energy consumption and accelerates the overall training progress up to 2.27 times. We also test the performance of SmartPC with different apps concurrently running in the foreground. Our results show that SmartPC can intelligently control the local training progress with negligible impact on the performance of the foreground app. To our best knowledge, SmartPC is the *first* work that studies the trade-off between training progress and energy efficiency in real-time Federated Learning systems. Specifically, our major contributions are as follows:

- We propose SmartPC, a hierarchical pace control framework that intelligently coordinates the overall training progress in a Federated Learning system.
- We make the observation that only a proportion of updates from the devices is required to achieve high model accuracy. Consequently, we design a deadline assignment mechanism to determine the training deadline in each training round in order to intelligently trade off model accuracy and training completion time.
- We design an energy optimizer to minimize the energy consumption of the participating devices during the training process while catching each training deadline.
- We prototype SmartPC on a Federated Learning system consisting of commercial mobile devices with heterogeneous hardware configurations.
- We demonstrate that Federated Learning can be successfully performed even when the participating devices have foreground apps running.

The rest of the paper is organized as follows. Section II introduces the background about Federated Learning and the key observations that motivate the design of SmartPC. Section III discusses the design of the global layer and the local layer, respectively. Section IV presents the system implementation and evaluation of SmartPC. Then Section V discusses prior research that is closely related to SmartPC. Finally, Section VI concludes the paper.

II. Background and Observation

In this section, we first briefly introduce the related background about Federated Learning. After that, we discuss the three key observations that motivate our design of SmartPC.

A. Background about Federated Learning system

A Federated Learning system usually consists of two main components: 1) a central server and 2) multiple mobile devices that participate in the training process. Figure 1 represents the workflow of a Federated Learning system which contains the following main steps:

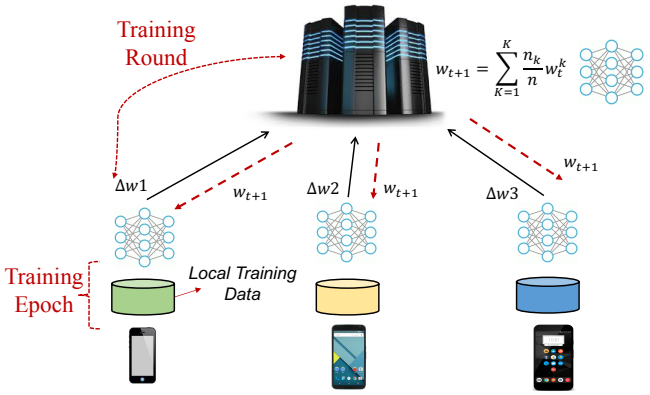
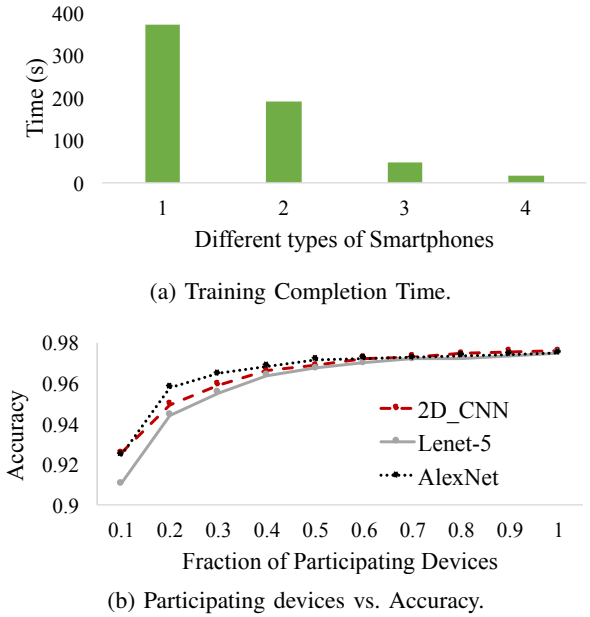


Fig. 1: A typical architecture of federated learning over cloud-edge infrastructures.



(a) Training Completion Time.

(b) Participating devices vs. Accuracy.

Fig. 2: Runtime information and model accuracy. ((a) training completion time on different mobile devices: 1-RedMi2, 2-Samsung Galaxy S4, 3-Huawei Honor 8, 4-Honor V9. (b) Impact of percentage of participants on model accuracy.)

1) At the beginning of each training round, the central server selects a set of online devices to participate in the training process.
2) The selected devices download the current global model state (e.g., current model parameters(w_t)).
3) Each mobile device performs local training based on the global model state and its local training dataset for a specific number of training *epochs*.
4) After completing the local training process, each mobile device sends the model updates (e.g., Δw) back to the central server.
5) After receiving the model updates from all the mobile devices, the central server aggregates these gradient updates and generates the updated global model. Then, the system enters a new training *round*.
6) The whole process iterates until the global model converges.

We can note that, at no point in time in the entire training process does the central server directly or indirectly access the local training data (e.g., raw data generated during user interaction with mobile devices). Data privacy is therefore well preserved-a key advantage of Federated Learning.

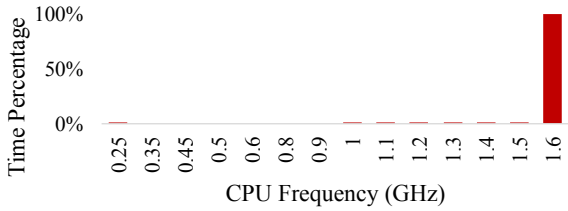
A number of important technical issues must be addressed for a Federated Learning system to be viable. Previous research in this area focus on such problems as: 1) reducing the communication cost [11], 2) guaranteeing the system security [12] and 3) analyzing the convergence bound [13]. However, in this paper, we aim at addressing a critical problem for efficient deployment of federated learning on mobile devices (e.g., pace control in a Federated Learning system to effectively trade off the energy efficiency, model accuracy and training progress).

B. Motivation

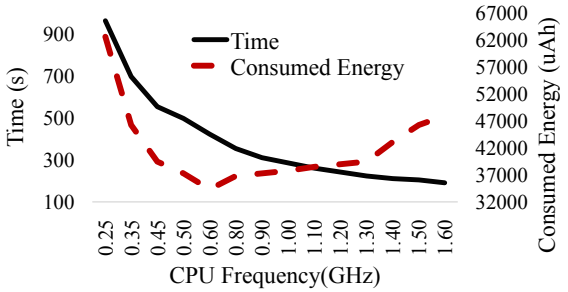
With the above background on Federated Learning, we now discuss a few key observations that motivate this work.

Observation1: A certain percentage of successful weight updates is enough to guarantee the predictive accuracy. A Federated Learning system usually includes mobile devices of diverse hardware configurations. Therefore, in each training round, the completion time of the devices can have great variations. We have conducted a Federated Learning task to train the image classification model Lenet5 [14] on four different smartphones, i.e., RedMi2, Samsung Galaxy S4, Huawei Honor 8, and Honor V9, respectively, and collected the local training completion time during the training process. We can see from the results in Figure 2a that the completion time of the same training process with the same size of data set, differs greatly – RedMi 2 spends 12× more time compared to Honor V9. This is largely due to difference in hardware configurations. For instance, the RedMi2 is powered by a 4-core 1.2GHz ARM Cortex-A53 CPU while Huawei Honor V9 has an 8-core CPU with 4 2.36GHz ARM Cortex-A73 and 4 1.84GHz Cortex-A53. Therefore, if the central server in a Federated Learning system has to wait for all the participating devices to send their updates before entering the next round, it is likely to be bottle-necked by the slowest devices. To reduce the overall training time, it is necessary *not* to wait for all devices to complete their round.

Figure 2b shows the impact of the proportion of weight updates on the model accuracy in a Federated Learning system. In this experiment, we train different models, including 2_Layer CNN, Lenet-5, AlexNet [15], with the Mnist [16] data set. The training data are evenly distributed among 200 participants. The x-axis shows the percentage of participants whose weight updates are successfully received by the central server. The y-axis shows the corresponding model accuracy. We can find that the model accuracy does not increase linearly with the fraction of participants that successfully send their weight updates. Above a certain threshold (e.g., 0.8 in this



(a) Frequency Selection.

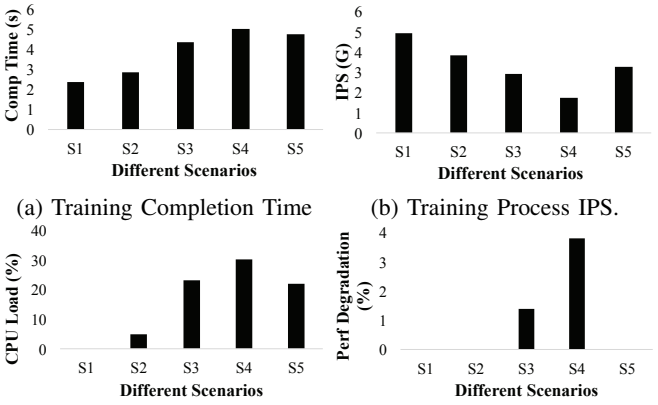


(b) Completion Time Vs. Energy.

Fig. 3: Runtime information during local training process on a Galaxy S4 smartphone ((a) CPU frequency selected by the default governor, (b) energy consumption and training completion time under different CPU frequency level).

case), the accuracy improvement is negligible. Thus, weight updates from all the participated device are not required. A certain percentage of successful weight updates is enough in order to guarantee the model accuracy.

Observation2: Default DVFS governor does not well balance the training progress and energy in a Federated Learning system. In order to further understand the system behavior during the local training process, we collect the CPU frequency selected by the default governor (e.g., the `interactive` governor on Android system). Figure 3a shows the results on Samsung Galaxy S4 as an example. We can find that the system spends more than 98% of the time on the highest CPU frequency (1.6GHz in this case) during the local training. This is because, on a typical Android device, the default governor determines the CPU frequency solely based on CPU load. If the CPU load is higher than a certain threshold, the governor would select the high frequencies. Therefore, for CPU-intensive tasks such as on-device training, the highest frequency will often be selected. Although this allows the task to be completed in the shortest possible time, it is generally not energy optimal on a mobile device. Figure 3b shows the completion time and energy consumption (the whole smartphone, based energy + training energy) at different fixed CPU frequencies on the Galaxy S4. As expected, the training time decreases monotonically as frequency increases. However, energy consumption does not share that trend. In fact, beyond a certain point (0.6GHz) energy increases monotonically as frequency increases. Moreover, in a Federated Learning system, though the highest frequency can complete the training in a short time, the whole system still needs to wait for weight updates from other devices in order to enter the next round.



(a) Training Completion Time (b) Training Process IPS.

(c) Foreground Scenario Load. (d) Foreground Perf Degradation.

Fig. 4: Impact of different concurrent running foreground tasks on the background local training process (S1-Reading, S2-Typing, S3-Gaming1 (2D-AngryBirds), S4-Gaming2 (3D-Basketball), S5-Video Playing).

Observation 3: The concurrent running foreground apps can highly impact the background local training process. Figure 4 shows the runtime information when the local training process runs concurrently with foreground apps in different common scenarios (e.g., S1-Reading, S2-Typing, S3-Gaming1 (2D-AngryBirds), S4-Gaming2(3D BasketBall), S5-Video Playing). Specifically, Figure 4a, 4b, 4c, 4d show the local training completion time, Instructions Per Second (IPS) of the local training process, CPU load of the foreground app and performance degradation of the foreground app based on a Nexus 6 smartphone, respectively. We can find that user interaction with the foreground app can highly impact the background training process. For instance, the training completion time of a minibatch is 2.368s (IPS 4.91GHz) with the reading scenario (CPU Load = 0%), while it increases up to 5.02s (IPS 1.75GHz) with the 3D gaming scenario (CPU Load = 31%). The concurrently running foreground apps can compete for the computing resources in different ways and thus impact the local training process. Moreover, mobile system usually assign higher priority to foreground apps and during this process the performance degradation of the foreground app is negligible as shown in Figure 4d.

III. System Design

In this section, we discuss the system design of *SmartPC*, a pace control framework that effectively balances model accuracy and training time and at the same time optimizes energy efficiency for mobile-based Federated Learning systems. Specifically, we first briefly introduce the system overview and then present the two-layer (i.e., global layer and local layer) design in detail.

A. System Overview

Figure 5 shows the system architecture of *SmartPC* which mainly contains two layers, the global layer and the local layer. The central server hosts the global pace controller while each

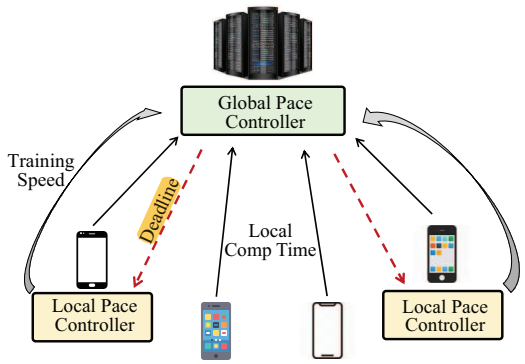


Fig. 5: System Architecture of SmartPC.

participating device has a local pace controller. The job of the global pace controller is to intelligently balance the progress of each training round and the model accuracy, whereas the local pace controller is responsible for trading off the pace and energy consumption of the local training process for its hosting device. Together, the two layers strive to dynamically achieve optimal balance among training time, model accuracy, and energy efficiency for the Federated Learning system.

The system workflow of *SmartPC* can be represented as the following main steps:

1) At the initialization step, the global controller first oversees the status (e.g., connectivity, availability, and energy/resource remained) of every participating device, then selects qualified devices. After that, each selected mobile device sends the following local information to the central server: 1) hardware information (e.g., available CPU frequency range) and 2) size of local training data.
2) After receiving the local information from the selected participants, the global pace controller estimates a virtual deadline of the upcoming training round to balance the overall training progress and model accuracy. The controller then broadcasts the virtual deadline to all the participants.
3) The local pace controller performs the local optimization to determine the optimal scheduling of hardware resources (e.g., CPU frequency) in order to minimize the energy consumption of local training while meeting the virtual deadline.
4) When the virtual deadline arrives, the global controller checks the progress of the current training round (i.e., percentage of model updates the server has received from all the participants). If the central server has already received enough weight updates to guarantee the model accuracy, the system directly enters the next training round. Otherwise, the controller notifies the participating mobile devices the synchronization deadline (timely secure the synchronization of local parameter updates for gradients aggregation in the central server per round) which is configured based on the requirement of the specific application.
5) The participants that have not completed the local training adjust their hardware configuration based on the received synchronization deadline (performed by the local pace controller) and try to complete their remaining training job.
6) When the synchronization deadline arrives, the global pace controller determines whether this round of training is successful or not based the percentage of model updates received by the central server. If the current training round is successful, the central server performs model averaging and the system enters the next training round. Otherwise, the current training round restarts to avoid waiting indefinitely.

It is important to note that SmartPC is mainly limited to soft real-time applications. Moreover, the training process on mobile devices can be highly dynamic. Different uncertainties (e.g., mobile devices may experience delay or even failure in sending back updates due to connectivity or battery issues) can take place during this process. The following mechanisms are designed to handle the uncertainties during the training process in each training round. First, SmartPC does not require *all* the devices to successfully upload their local updates to the central server, but only requires the central server to receive updates from a certain percentage of participating devices in order to guarantee the model accuracy. This effectively reduces the impact of random uncertainties that could occur on the participating mobile devices. Moreover, the synchronization deadline is designed to prevent the system from waiting indefinitely in a certain round in order to effectively handle the uncertainties. Figure 6 shows the deadline assignment (global pace control) and local training process (local pace control) in each training round. It is a challenging problem to efficiently trade off the training progress, model accuracy and energy consumption in such a heterogeneous system. In the following sections, we discuss the design of the global and local pace control in detail.

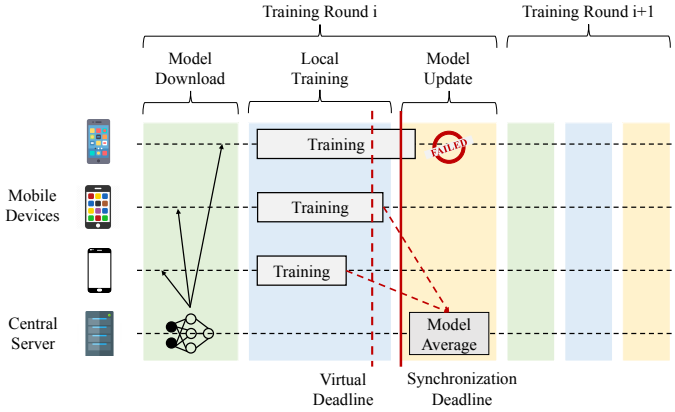


Fig. 6: Deadline assignment and local training.

B. Global Pace Control

The Global Pace Control assigns the training deadline for each training round in order to balance model accuracy and

training progress. However, determining the *right* deadline is not trivial. If the deadline is too tight, *many* of the mobile devices may fail to complete the local training in time and to submit their model updates. This negatively impacts the model accuracy. If the deadline is too loose, the overall training progress is slowed down. Thus, determining the amount of weight updates required to guarantee the model accuracy and accurately estimating the local completion time is critical for deadline determination.

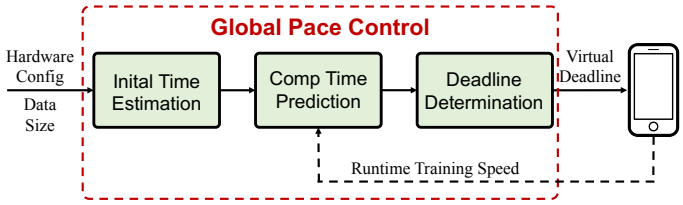


Fig. 7: Workflow of Global Pace Control.

Global Pace Control Overview. Figure 7 shows the workflow of global pace control. In the initialization step, after receiving the hardware configuration information and the local training data size, the global pace control estimates the local completion time of each participating device. Then, with the completion information and the required weight update percentage value, the *Deadline Determination* component determines the global training deadline and broadcasts it as input to the local pace controllers. At the end of each training round, the global pace control receives the feedback of the local training speed from the local pace controllers for completion time prediction in order to effectively estimate the new deadline for the next round.

Completion Time Model. With the received hardware configuration information and the size of local training data, the time required by device i to complete the local training process can be modeled [6] as:

$$t_i = \frac{c_i D_i}{f_i} \tag{1}$$

where, c_i represents the number of CPU cycles required to process one data object on mobile device i, which can be obtained through offline profiling, D_i represents the number of data objects in the local training data set, and f_i is a particular CPU frequency available on device i. Because the data objects (e.g., pictures) in a training set usually have the same size, the number of CPU cycles required for device i to run a local iteration can be modeled as $c_i D_i$.

Given this model, if f_{i_max} is the maximum CPU frequency on device i, the shortest training completion time is:

$$T_{i\_min} = \frac{c_i D_i}{f_{i\_max}} \tag{2}$$

Completion Time Prediction. When a mobile device has no foreground app running, using Eqn. 1 to predict local training completion time can be sufficiently accurate. However, as discussed above, we intend to have mobile devices participate in Federated Learning even when they are in use. As Figure 4 shows, foreground apps can affect the performance of the background training process. This has to be accommodated in order to accurately predict training time. In *SmartPC*, we adopt an exponential moving average (EMA) formula for the completion time prediction as follows. For each training round j, the local pace control located on each mobile device monitors the starting time $t^j_{i\_start}$ and the ending time $t^j_{i\_end}$ of the training process. Moreover, it monitors the number of data objects S^j_i that have been processed in each round. We define the training speed of mobile device i in round j as follows:

$$r^j_i = \frac{S^j_i}{t^j_{i\_end} - t^j_{i\_start}} \tag{3}$$

With this definition, the training speed of the upcoming training round k, R^k_i can be predicted as follows:

$$R^k_i = \begin{cases} r^1_i, & k = 1 \\ \alpha * r^{k-1}_i + (1-\alpha) * R^{k-1}_i, & k > 1 \end{cases} \tag{4}$$

where R^k_i $(k > 1)$ is the new predicted value, R^{k-1}_i is the last predicted value, r^{k-1}_i is the latest measured training speed on device i, and α is a constant attenuation factor in the range between 0 to 1. The parameter α controls the relative weight of recent and past history in the prediction– when $\alpha > 0.5$, more weight is given to the most recent sample, and vice versa. In our implementation, we use the the real user interaction trace from LiveLab [17], which records the user interaction from different users, to select the best value of α in order to achieve the best prediction accuracy. With the predicted training speed R^k_i, the completion time of the upcoming round k can be estimated as

$$t^k_i = \frac{D_i}{R^k_i}, \quad k > 1 \tag{5}$$

For the very first round, we use T_{i_min} in Eqn. 2 as the estimation.

Deadline Determination. As shown in Figure 2b, a certain percentage (not 100%) of successful weight updates is sufficient for achieving a high level of model accuracy. Thus, the training deadline is determined as the shortest time within which a specific percentage ($U_{required}$) of participating devices can send back their weight updates to the central server. Using $U_{required}$ ($< 100\%$) has two main advantages: 1) it makes the system more robust considering mobile devices can be offline due to various reasons (e.g., out of battery, user shutdown, system failure), and 2) it accelerates the overall training process – the Federated Learning system will not be bottle-necked by the low-end devices and/or those experiencing performance issues at the moment. We formulate the deadline determination problem for training round k as follows:

$$Min\{d_k\} \tag{6}$$

$$\text{subject to} \quad \frac{\sum_{i=1}^{i=N} I(t^k_i)}{N} \geq U_{required} \tag{7}$$

where t_i^k is the estimated completion time (Eqn. 5), and the indicator function $I(t_i^k)$ is defined as follows:

$$I(t_i^k) = \begin{cases} 0 & t_i^k > d_k \\ 1 & t_i^k \leq d_k \end{cases} \tag{8}$$

After that, the determined deadline d_k is sent out as input to the local pace controllers.

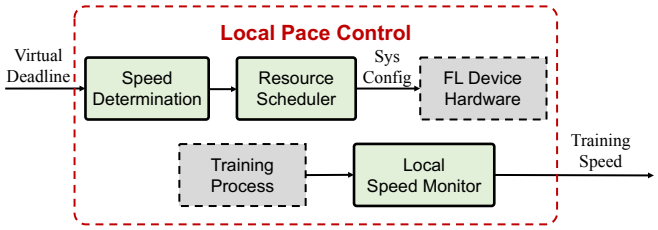


Fig. 8: Local pace control in SmartPC.

C. Local Pace Control

After receiving the global deadline d_k from the inter-device layer, the local pace control aims to meet the deadline while minimizing the energy consumption of the local training process. Figure 8 shows the workflow of the local pace control. First, the *Speed Determination* component calculates the optimal speed (in terms of IPS) for the local training process that allows the deadline to be met with minimal energy. After that, the *Resource Scheduler*, guided by the optimal speed, computes a resource schedule for the device to balance the power and performance for the local training process. Additionally, the *Local Speed Monitor* monitors the average training speed and sends back to the global pace control for deadline determination of the next training round.

1) Speed Determination: As discussed in Section II-B, due to its compute-intensive nature, the local training process on an Android system is executed in a race-to-idle manner, i.e., the highest CPU frequencies are used for the entire duration of the task, which may not be energy optimal. In *SmartPC*, we find a suitable speed for the local training process such that the deadline is met while energy consumption is minimized.

We model the energy consumption of a device in one training round as follows:

$$E = p^{train} * t^{train} + p^{idle} * t^{idle} \tag{9}$$

where p^{idle} represents the base power when the smartphone is powered on but not actively used, p^{train} represents the power consumed while the training process is running, and t^{idle} and t^{train} are the time spent in the idle and training state, respectively.

Given a deadline and assuming that the training round can finish before the deadline, running the training process at a higher CPU frequency means it can be completed sooner (i.e., smaller t^{train}), but with a higher p^{train}. We therefore formulate the training speed determination as a constrained optimization problem, as explained below.

Training Power Model. The idle power p^{idle} in Eqn. 9 can be easily obtained through measurement. The training power, on the other hand, can be modeled as follows [13]:

$$p^{train} = \beta * f^3 \tag{10}$$

where β is the effective capacitance coefficient of the computing chipset of the mobile device which can be obtained through the profiled power data at different frequency levels, and f represents the CPU frequency adopted during the training process.

Problem Formulation. With the power model in Eqn. 10 and the completion time model in Eqn. 1, we formulate the local speed determination problem as a constrained optimization problem. For a specific global training round with training deadline d_k, we aim to minimize the energy consumption of the mobile device under the condition that the training round be completed before d_k is reached. Denote by $E_i(f_i)$ the energy consumption of device i in a particular training round when the training process is run with CPU frequency f_i. The problem therefore is to find:

$$\underset{f_i}{\text{argmin}}\ E_i(f_i), \quad f_i^{min} \leq f_i \leq f_i^{max} \tag{11}$$

$$\text{s.t. } t_i^{idle} + t_i^{train}(f_i) = d_k, \ \ 0 \leq t^{idle}, t^{train}(f_i) \leq d_k \tag{12}$$

where $E_i(f_i)$ is the energy consumption in Eqn. 9 when the CPU frequency is f_i.

In the above, Eqn. 11 indicates that the optimization problem is to find the CPU frequency in the available range on the device that results in the least amount of energy consumption. Equation 12 requires that the training completion time meet the deadline d_k set for the training round.

The solution to Eqn. 11 and 12 gives the energy-optimal completion time t_i^{train}. Based on this, we can compute the IPS target as $i_i * D_i / t_i^{train}$, where i_i is the number of instructions for processing one data object and D_i is the size of the data set. This target is then passed as input to the Resource Scheduler.

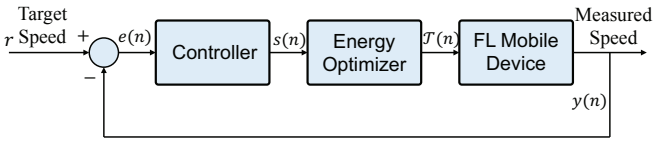


Fig. 9: Resource Scheduler feedback loop.

2) Resource Scheduler: As discussed, the *Speed Determination* component computes a training time target that allows the training deadline to be met with the least amount of energy, which can be converted to IPS. The *Resource Scheduler* then takes the IPS target as input and dynamically computes schedules for the device hardware resources in order to achieve the target in an energy efficient way. In this paper, we limit the resources to CPU frequencies.

We adopt a two-component design for the Scheduler that consists of an integral controller and an energy optimizer in a feedback loop, as shown in Figure 9.

The Resource Scheduler operates in a cyclic manner. The workflow of one cycle of the scheduling process is as follows: (1) The performance of the training process, $y(n)$, is measured and the control error $e(n) = r - y(n)$ is calculated, where r is the static target IPS for the round. (2) Using $e(n)$ as input, the controller, taking into account historical errors, calculates a *dynamic* performance as speedup $s(n)$ that minimizes the accumulated error. (3) The energy optimizer takes $s(n)$ as input and based on the performance model in Eqn. 1 and power model in Eqn. 10 computes a CPU frequency schedule that can achieve $s(n)$ with the least amount of energy. (4) The schedule is applied, and the process is repeated.

Integral Controller. To minimize the accumulated error $\sum e(n)$ between the static target performance r and the measured performance $y(n)$, we utilize an integral controller [18]. The input to the controller is the control error $e(n)$, i.e., the difference between the target performance r and the actual performance $y(n)$, and the output is the dynamic target performance in the form of a speedup $s(n)$ as described below.

First, the dynamic target performance $u(n)$ considering accumulated error is formulated as follows:

$$u(n) = u(n-1) + g \cdot e(n-1), \tag{13}$$

where $u(0) = r$, and the constant g, $0 < g < 1$, is the controller gain that is introduced for stability reasons. In practice, a value around 0.5 is a reasonable choice.

The performance model in Eqn. 1 is a simple model. To address the inevitable modeling errors, we decompose the performance into a base speed $b(n)$ and a dimensionless speedup $s(n)$, as follows:

$$u(n) = s(n) \cdot b(n) \tag{14}$$

the base speed $b(n)$ is the speed of the local training process when it runs with the minimal CPU frequency. Note that $b(n)$ is a variable and is estimated every cycle using a Kalman filter as detailed in [19]. With this formulation of the performance, the actual output of the controller is the desired speedup as follows:

$$s(n) = s(n-1) + g \cdot \frac{e(n-1)}{b(n-1)} \tag{15}$$

Energy Optimizer. Once the required speedup $s(n)$ is calculated by the integral controller, the energy optimizer determines the energy-optimal CPU frequency schedule that achieves $s(n)$. This schedule is then applied for the next control cycle of T time units. As in [19], this optimization problem can be formulated as a linear programming problem.

Each mobile device has a set of CPU frequencies. Assuming that for a device we have a selected list of N CPU frequency values $\{v_1, v_2, ..., v_N\}$, Corresponding to each v_i, there is a training process performance s_i in the form of speedup as described above, as well as a power consumption p_i. The task of the energy optimizer is to determine a schedule $\tau = [\tau_1, \tau_2, ...\tau_N]$, where τ_i means applying frequency v_i for τ_i time units. Thus, the optimization problem can be formulated as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} \tau_i \cdot p_i \tag{16}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \tau_i \cdot s_i = s(n) \tag{17}$$

$$\text{and} \quad \sum_{i=1}^{N} \tau_i = T, \quad 0 \le \tau_i \le T \tag{18}$$

where T is the control period of the Resource Scheduler. Eqn. 16 is the optimization goal that minimizes the energy consumption in the control cycle. Eqn. 17 represents the performance constraint, while Eqn. 18 shows the constraints on the time horizon.

The solution to the optimization problem determines a set of CPU frequencies and their corresponding applied duration, such that it minimizes the energy consumed by the system for a time period of T units while maintaining a performance (speedup) of $s(n)$. Based on the theory of linear programming, there exists an optimal solution to Equation 16 with at most two non-zero values for τ_i in the solution. This means for the frequency schedule τ we simply need to find a pair of values τ_i and τ_j that satisfies Equations 16 – 18.

3) Speed Monitor: The *Speed Monitor* monitors the processed data objects between the starting and ending points through instrumenting the local training process. As discussed, this information is sent back to the global controller at the end of each round so that it can update the prediction model for completion time.

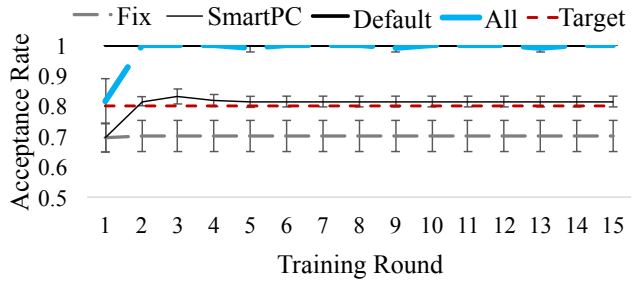
IV. Evaluation

SmartPC is designed to efficiently balance the training performance and energy consumption for on-device Federated Learning. We evaluate the performance of SmartPC using both simulation and physical testbed. We use this hybrid testbed to cross validate the effectiveness and correctness of corresponding models and system design. In this section, we first introduce the experimental methodology and baselines, and then discuss the corresponding experiments.

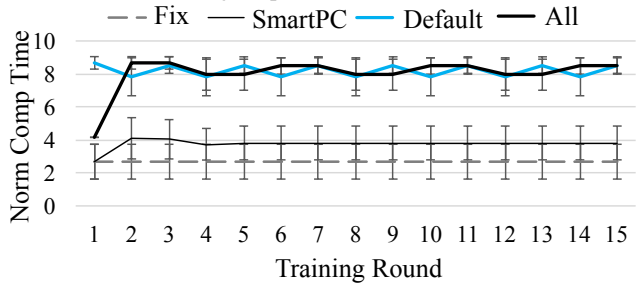
A. Experimental Setup

We build a prototype on-device Federated Learning system using Android smartphones with heterogeneous hardware configurations, as listed in Table I. The devices have different Android versions (i.e., 5.0.5-8.0), different number of CPU cores (i.e., 4, 6, 8) and different sets of CPU frequencies. The local training process is implemented based on the DL4J [20] with a PaddlePaddle based the parameter server. It runs as an *asynctask* in the background and has no user interface. Communications between the devices and the central server are based on the client-server model as shown in Figure 1.

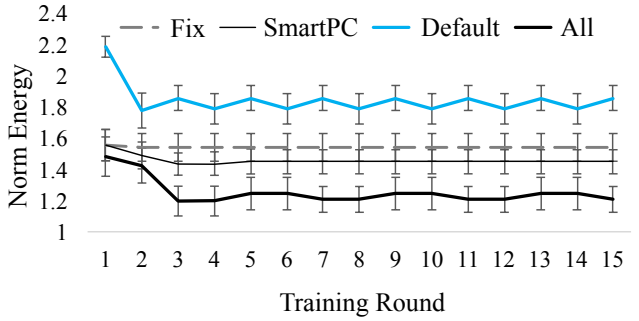
In the local pace controller, we dynamically adjust the CPU frequency of a mobile device to provide the performance-energy trade-off. Setting CPU frequency to a specific level is achieved by writing to pertinent files in the *sysfs* to first set the



(a) Weight updates submission rate.



(b) Completion time of each round.



(c) Energy consumption of each round.

Fig. 10: Comparison of schemes in global pace control.

TABLE I: Device Profiles

Device	Android Version	#Core	Frequency
Honor	8.0	8	1.40 - 2.11GHz
Lenovo	5.0.2	4	0.29 - 1.04GHz
ZTE	5.1.1	4	0.20 - 1.09GHz
Mi	5.1.1	6	0.46 - 1.44GHz
Nexus	6.0	4	0.30 - 2.65GHz

governor to *userspace*, and then to set the CPU frequency. We use the *SimplePerf* tool to collect runtime performance data such as instruction count, and for all power measurements a Monsoon power monitor is physically connected to the device.

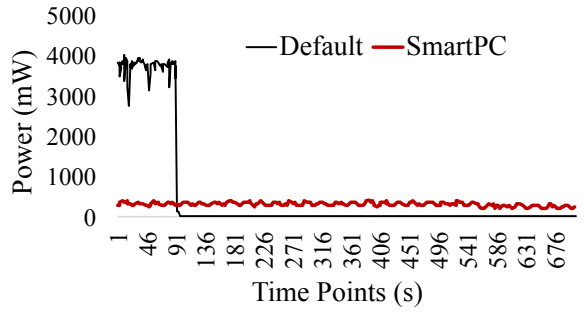
Baselines. We evaluate the effectiveness of SmartPC against the following baselines.

- *Default* represents the state-of-the-practice Federated Learning system in which each mobile device completes the local training process with the default CPU governor and the central server performs model averaging after receiving *all* the local updates from the mobile devices.
- *Train-with-all* represents the Federated Learning approach in which the central server must receive local updates from *all* the mobile devices. On the device side, each device performs the same energy optimization process as SmartPC does.
- *Fixed-deadline* represents the scheme in which the training deadlines are fixed for all the training rounds.

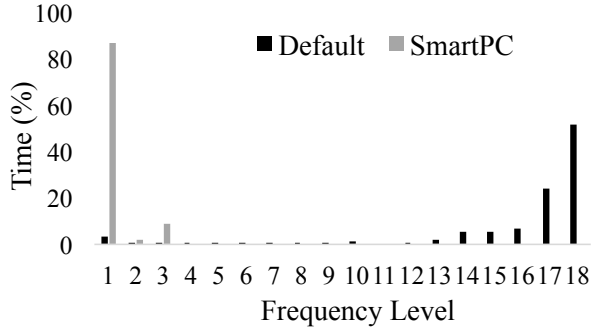
B. Evaluation of Global Pace Control

In this section, we evaluate the effectiveness of the global pace control from different perspectives. We construct a simulation testbed consisting of 100 smartphones with hardware configurations randomly selected from Table I. We evaluate SmartPC in the more general situation where learning is performed with concurrently running foreground apps. To that end smartphone usage traces from LiveLab [17] are adopted that emulate the user interactions with different foreground apps. We train the Lenet5 [14] model with the Mnist [16] data set. The training data are evenly distributed among the mobile devices. In this experiment, we use 80% as the target of percentage of weight updates the server has to receive at the end of each training round, which in practice can be configured by the Federated Learning service provider. It is important to note that SmartPC focuses on the system perspective of Federated Learning (e.g., balance the training progress, model accuracy and energy consumption). We use the well-known image classification task (e.g., Lenet5) as an example to demonstrate the effectiveness of SmartPC. However, SmartPC can be generally applied to different federated learning tasks (e.g., face detection, next-word prediction, on-device item ranking, content suggestions for on-device keyboards, next word prediction and human activity recognition).

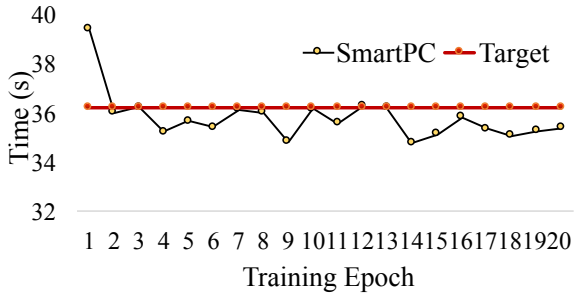
Figure 10a shows the percentage of local weight updates successfully received by the central server with different schemes. We can see that *Fixed-Deadline* can not achieve the predefined completion percentage (i.e., 80%). This is because *Fixed-Deadline* only considers the hardware configurations (e.g., available CPU frequencies) during the deadline determination process and does not consider the impact of user interactions on the completion time of the local training process. For instance, the low-end devices, e.g., Lenovo and ZTE in this case, need to run with the highest CPU frequency in order to meet the training deadline. However, due to resource contention with the foreground app as shown in Figure 4, the local training completion time can be extended significantly, causing them to miss the pre-assigned deadlines. With the pre-configured acceptance rate of 80%, the average acceptance rate with *Fixed-Deadline* is 70% with the worst case being 62%, which means the Federated Learning system has to restart the training every time. With SmartPC, in the very first round the acceptance rate is the same as that with *Fixed-Deadline*, since SmartPC also determines the deadline of the first round based on the hardware configurations. However, in the next few training rounds, the acceptance rate gradually converges to the pre-defined target. This is because the training deadline is adaptively determined based on the predicted training speed of each participating device. With the *Default* and *Train-with-*



(a) Power consumption of one training round.



(b) Frequency level selection with different schemes.



(c) Local pace control performance on a Nexus 6 smartphone.

Fig. 11: Local pace control performance on a Nexus 6.

All schemes, the acceptance rates are close to 100%, since they do not enter the next training round until the central server has received the weight updates from all the participants.

As shown in Figure 10a, *SmartPC*, *Default* and *Train-with-All* can all successfully achieve the pre-set acceptance rate of the weight updates in each training round. In Figure 10b we compare training completion time (first measured in seconds and then normalized to the smallest value) of different schemes. We can find that *Default* and *Train-with-All* have similar local training completion time, which is 2.27 times of that with *SmartPC* on average. Because these two schemes always wait for the weight updates from all the participants, the overall training completion time is bounded by the low-end devices, particularly those with the highest interference from the foreground app. *SmartPC*, on the other hand, proves to be more robust because it only requires a certain percentage of accepted weight updates that is statistically significant. This essentially removes the outlier participants from consideration without significantly affecting the model accuracy. For large scale Federated Learning this is especially important because the probability of having outliers is high. For instance, a mobile device can be offline due to a number of reasons (e.g., networking issue, out of battery). When this happens, the overall training process of either *Default* or *Train-with-All* can be slowed down considerably.

We also evaluate the model accuracy achieved with SmartPC. Table II compares the model accuracy obtained with SmartPC and with *Default* when training three different models. We can see that SmartPC produces the same level of model accuracy despite the fact that it only uses updates from a portion of the participants. Because the training algorithm itself (e.g., model averaging, stochastic gradient descent) is not modified in anyway, the only difference from the training algorithm's standpoint is the amount of parameter updates. These results show that, in a Federated Learning system, using a high enough percentage of inputs can give similar model accuracy to using all inputs. For the reason that, the Dalvik Virtual Machine in Android system has a limit for the heap size (e.g., 512MB on Nexus 6), we select the models and dataset with less memory requirement for evaluation. However, SmartPC can be generally applied to other models and data set.

TABLE II: Model Accuracy

	2D_CNN	Lenet-5	AlexNet
Default	97.61%	97.49%	97.5%
SmartPC	97.46%	97.24%	97.35%

Finally, Figure 10c shows the energy consumption (first measured in mAh and then normalized to the smallest value) of each training round. The *Default* scheme consumes the highest energy during the training process. Compared with the *Default* scheme, *SmartPC* achieves 28.4% energy savings on average, due to the intelligent pace control in the local layer, which is discussed in the following section. We can find that the scheme *All* has the lowest energy consumption. This is because the *All* scheme expects all the participated device to successfully send their weight updates, which gives the local layer more time range to conduct energy optimization. However, the training completion time of the *All* scheme is highly extended.

C. Evaluation of Local Pace Control

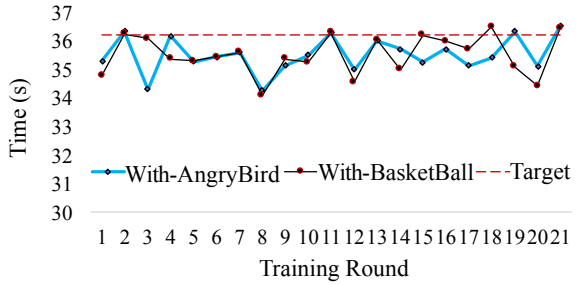
In this section, we dive into the mobile device to evaluate the effectiveness of local pace control. In the experiment, we construct a Federated Learning system consisting of 5 devices, one each of the models listed in Table I. We first present results obtained with no foreground apps, and then discuss the impact of foreground apps in the next section. Figure 11 shows the results in one training round on a Nexus 6 smartphone. The global pace controller sets the deadline to 724s, and within this deadline, 20 training epochs (configured by service provider) should be completed.

Figure 11a shows the power consumption of the local training process. We can see that with the default governor the local training process completes as soon as possible and then the CPU goes into idle. The average power consumption of the local training process, p^{train}, is 3517.57mW, for a duration t^{train} of 90.2s. The idle power p^{idle} is measured at 27mW and the idle time t^{idle} is 633.8s. Thus, the total energy consumption of the training round on this device is 334.4J. On the other hand, with *SmartPC*, with the deadline of 724s, the Speed Determination component determines that it is energy optimal to run the training process for the entire duration of 724s. Under the Resource Scheduler, the local training process runs for the entire 689s (during the real local training process) with an average power consumption of 324.74mW. The energy consumption is 224.6J, representing a 32.8% reduction compared to the *Default* scheme. To better understand the energy savings, in Figure 11b we compare the CPU frequencies selected by *SmartPC* vs. *Default* during the local training process. The x-axis represents the 18 CPU frequency levels available on the Nexus 6 smartphone. The y-axis represents the percentage of time the CPU is at a certain frequency level. We can find that the default governor always uses the relatively high frequency levels to complete the local training process in a short time but with high power consumption at the same time. Specifically, the default governor spends 51.8% of the time at level 18 (the highest frequency level) and 24.3% of the time at level 17 (the second highest frequency level). By contrast, with *SmartPC*, the local pace controller spends 87% of the time at level 1 (the lowest frequency level) in order to complete the local training process in an energy efficient way while catching the training deadline.

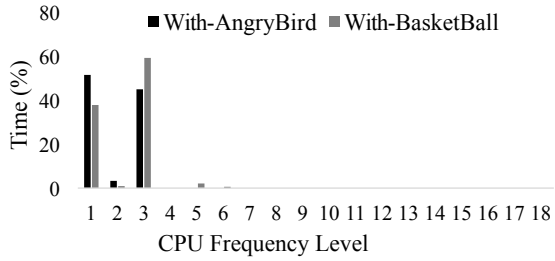
Figure 11c shows the training completion time during the training process. The x-axis represents different training epochs and the y-axis represents the completion time of each training epoch. We can see that the local pace controller can effectively achieve the training completion time target.

D. Impact of Different Foreground Apps

One important feature of SmartPC is its ability to run training while the device is being used for other purposes, as opposed to limiting the training task to running only when the device is being charged. In this section, we evaluate the effectiveness of the local pace control with different apps running in the foreground. We use two apps (i.e., AngryBirds and BasketBall as discussed in Figure 4) as examples. Figure 12a shows the completion time of each training epoch with different apps running in the foreground. The x-axis represents different training epochs and the y-axis represents the corresponding training completion time. Most of the training epochs do not miss the deadline. This is because the controller is able to obtain the IPS (instruction per second) value for the training process alone and can adjust the CPU frequency to meet the target IPS. Therefore, the local pace control is able to effectively perform its task when there is a foreground app running concurrently.

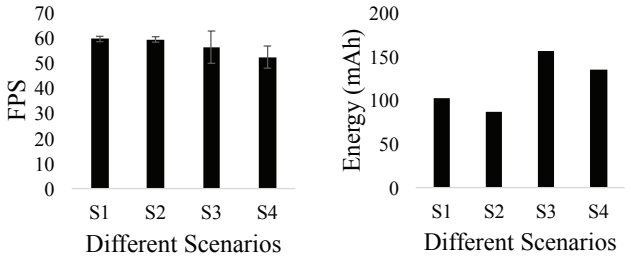


(a) Training Completion Time.



(b) Frequency Level Selection.

Fig. 12: Intra-Device Control with Different Foreground Apps.



(a) App Performance. (b) Energy Consumption.

Fig. 13: App performance and energy consumption of intra-device control with different foreground apps. (S1-AngryBird only, S2-AngryBird+Training, S3-BasketBall only, S4-BasketBall+Training).

Figure 12b shows the CPU frequency selected by *SmartPC* with different apps running in the foreground. When AngryBirds is running in the foreground, the CPU spends 51% of the time on frequency level 1 and 45% of the time on frequency level 3. Whereas, when BasketBall is running in the foreground, 37% of the time is spent on level 1 and 59% of the time is spent on level 3. We can find that the controller selects higher frequency level during the local training process in these two cases than when there is no app running in the foreground. This is because the foreground app can compete for CPU time with the background training process. Thus, in order to achieve the training performance target, the local pace controller spends more time on the higher frequency level. The reason that the local pace controller spends more time on higher frequency level when the Basketball app is running in the foreground is that the BasketBall app has higher CPU load (30%) than that of AngryBirds (22%).

Figure 13a shows the impact of the background training process on the performance of the foreground app. S1 represents the scenario where AngryBirds runs in the foreground without the training process running in the background. S2 represents the scenario where AngryBirds runs with the background training process concurrently. S3 and S4 represent the corresponding scenarios for the Basketball app, respectively. In the experiment, we use FPS (Frames Per Second) as the performance metric. For S1, the average FPS is 59.41. For S2, the average FPS is 59.23. For S3, the average FPS is 56.14 and for S4, it is 52.21. We can see that the impact of user-perceived performance of the foreground app is negligible. Figure 13b shows the energy consumption of the whole smartphone under the four scenarios. Compared with the default scheme, SmartPC achieves 17.1% energy saving when AngryBirds is concurrently running in the foreground and 14.3% when Basketball is on the foreground, respectively. It is important to note that the consistent results obtained from our hardware deployment and simulated testbed verify the efficiency and correctness of the corresponding mechanism and models.

E. System Overhead

In SmartPC, the global pace controller runs on the central server and the local pace controller runs on each mobile device. Thus, the system overhead of the local pace controller is the main concern for SmartPC. As described in Sec. III-C, what the local pace controller does is to periodically compute a resource schedule so that the training process can meet the deadline with the lowest amount of energy. Since the computation involved is very limited, most of the time the controller is in sleep mode. On a Nexus 6 smartphone, with a control cycle of 2s, each cycle the controller takes less than 10ms to compute the frequency schedule. Consequently, the power consumption of the local pace controller itself is measured at 44mW, which is negligible.

V. Related Work

Our work is closely related to two major research topics, *distributed learning* and *federated learning*.

Distributed Learning. In order to leverage large amount of data located at different places to train various kinds of deep learning models, distributed learning has attracted a lot of attention [6], [21]–[29]. Zhang et al. [21] design a cluster scheduling system to approximate machine learning training jobs in order to maximize the overall job quality. So et al. [22] propose an approach to make efficient parallelization of the distributed training process and keep the training information (e.g., training data and model) private in order to guarantee a secure training process. Li et al. [6] propose a parameter server framework for distributed learning in order to manage asynchronous data communication between nodes and support flexible consistency models, elastic scalability and continuous fault tolerance. Bao et al. [25] propose a deep learning-driven ML cluster scheduler to place different jobs in corresponding machines to minimize the interference and maximize performance. Though these approaches can efficiently improve the performance of distributed learning system, they cannot be directly applied on Federated Learning system which has its own characteristics. The training data are placed and the training process is usually performed on central data centers (e.g., large amount of servers) in a distributed system, however in a Federated Learning system, the process is mainly completed on mobile devices which has much higher limitation on battery lifetime (e.g., energy consumption) and system heterogeneity.

Federated Learning. Federated Learning is then proposed to efficiently leverage the data generated from mobile devices to support intelligent applications [4], [5], [8], [11]–[13], [30]–[32]. Lalitha et al. [30] propose a distributed learning algorithm to train a machine learning model over a network of users in a fully decentralized framework. Konecny et al. [5] focus on the communication efficiency in a federated learning system and propose two schemes (e.g., sketched update and structured update) to reduce the uplink communication costs. Smith et al. [4] propose a system-aware optimization approach to consider issues of high communication cost, stragglers, and fault tolerance for distributed multi-task learning. McMahan et al. [11] design a practical approach for the federated learning of deep networks based on iterative model averaging. Existing research about federated learning mainly focuses on the following perspectives: 1) reducing the communication cost, 2) improving the security during the training process and 3) analyzing the convergence. However, the trade-off among the energy efficiency, training progress and model accuracy is ignored by pervious study. In this paper, we try to solve the problems in a federated learning system from a new perspective.

VI. Conclusion

This paper has proposed SmartPC, a hierarchical pace control framework for Federated Learning that intelligently balances the training time and model accuracy in an energy-efficient manner, through incorporating with two major components: a *Global pace controller* and a *Local pace controller*. At the start of every training round, the global controller first collects the status of every participating devices, then estimates a virtual deadline for all qualified devices per selection. More specific, such virtual deadline is selected to allow a statistically significant proportion (e.g., $\geq$60%) of the devices to complete their work and upload model updates, which guarantees the model accuracy for every round of model update and ensure the timeliness of the overall progress for the multi-round training procedure. On each device, a local pace controller then dynamically adjusts device settings such as CPU frequency so that the learning task is able to meet the deadline with the least amount of energy consumption. We performed extensive experiments to evaluate SmartPC on both Android smartphones and simulation platforms using well-known datasets. The experiment results show that SmartPC can reduce 32.8% energy consumption without model accuracy degradation. At the same time, SmartPC can achieve a speedup of 2.27x in training time.

ACKNOWLEDGEMENT

We appreciate efforts made by shepherd, reviewers, and PC members. The research is partially supported by National Key R&D Program of China (2018YFB1004804), Chinese Academy of Sciences Shenzhen Basic Research Program (No. JCYJ20170818153016513), and National Science Foundation CNS-1850851. Parts of SmartPC have been transformed into Baidu PaddlePaddle Federated Learning Systems: `https://github.com/PaddlePaddle/models`.

REFERENCES

[1] A. Ram, "How smartphone apps track users and share data," https://ig.ft.com/mobile-app-data-trackers/, 2018.

[2] Marketing-Schools, "Marketing Mobile Phones," http://www.marketing-schools.org/consumer-psychology, 2018.

[3] S. Halpern, "The champaign for mobile phone voting is getting a midterm test." https://www.newyorker.com/tech/annals-of-technology/, 2018.

[4] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[5] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[6] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014, pp. 583–598.

[7] L. Hautala, "Google tool lets any AI app learn without taking all your data," https://www.cnet.com/news/google-ai-tool-lets-outside-apps-get-smart-without-taking-all-your-data/, 2018.

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

[9] Terry Erisman, "Achieving real-time machine learning and deep learning with in-memory computing," https://jaxenter.com/in-memory-computing-machine-learning-145623.html.

[10] RedAlkemi, "5 Real Time Applications of Machine Learning," https://www.redalkemi.com/blog/post/5-real-time-applications-of-machine-learning.

[11] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[12] F. Mo and H. H, "Efficient and private federated learning using tee," in *EuroSys*, 2019.

[13] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, 2019.

[14] Yann Lecun, "Lenet5," http://yann.lecun.com/exdb/lenet/.

[15] L. Xiao, Q. Yan, and S. Deng, "Scene classification with improved alexnet model," in *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, Nov 2017, pp. 1–6.

[16] Yann Lecun, "Mnist," http://yann.lecun.com/exdb/mnist/.

[17] Rice University, "LiveLab: Measuring wireless networks adn smartphone users in the field," http://livelab.recg.rice.edu/traces.html.

[18] K. Ogata, *Modern control engineering*, 5th ed. Prentice Hall Upper Saddle River, NJ, 2009.

[19] C. Imes, D. H. Kim, M. Maggio, and H. Hoffmann, "Poet: a portable approach to minimizing energy under soft real-time constraints," in *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 75–86.

[20] M. Hamblen, "Deep Learning For Java," https://deeplearning4j.org/.

[21] H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 390–404.

[22] J. So, B. Guler, A. S. Avestimehr, and P. Mohassel, "Codedprivateml: A fast and privacy-preserving framework for distributed machine learning," *arXiv preprint arXiv:1902.00641*, 2019.

[23] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "Mlbase: A distributed machine-learning system." in *Cidr*, vol. 1, 2013.

[24] L. Mai, C. Hong, and P. Costa, "Optimizing network performance in distributed machine learning," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[25] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *IEEE INFOCOM*, 2019.

[26] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1269–1284.

[27] S. Sun, W. Chen, J. Bian, X. Liu, and T.-Y. Liu, "Slim-dp: A multi-agent system for communication-efficient distributed deep learning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 721–729.

[28] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 328–339.

[29] P. Watcharapichat, V. L. Morales, R. C. Fernandez, and P. Pietzuch, "Ako: Decentralised deep learning with partial gradient exchange," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, 2016, pp. 84–97.

[30] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[31] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[32] M. R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, "Asynchronous federated learning for geospatial applications," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018, pp. 21–28.