Adaptive Federated Learning in Resource Constrained Edge Computing Systems

Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung,
Christian Makaya, Ting He, Kevin Chan

***Abstract*—Emerging technologies and applications including Internet of Things (IoT), social networking, and crowd-sourcing generate large amounts of data at the network edge. Machine learning models are often built from the collected data, to enable the detection, classification, and prediction of future events. Due to bandwidth, storage, and privacy concerns, it is often impractical to send all the data to a centralized location. In this paper, we consider the problem of learning model parameters from data distributed across multiple edge nodes, without sending raw data to a centralized place. Our focus is on a generic class of machine learning models that are trained using gradient-descent based approaches. We analyze the convergence bound of distributed gradient descent from a theoretical point of view, based on which we propose a control algorithm that determines the best trade-off between local update and global parameter aggregation to minimize the loss function under a given resource budget. The performance of the proposed algorithm is evaluated via extensive experiments with real datasets, both on a networked prototype system and in a larger-scale simulated environment. The experimentation results show that our proposed approach performs near to the optimum with various machine learning models and different data distributions.**

***Index Terms*—Distributed machine learning, federated learning, mobile edge computing, wireless networking**

I. Introduction

The rapid advancement of Internet of Things (IoT) and social networking applications results in an exponential growth of the data generated at the network edge. It has been predicted that the data generation rate will exceed the capacity of today's Internet in the near future [2]. Due to network bandwidth and data privacy concerns, it is impractical and often unnecessary to send all the data to a remote cloud. As a result, research organizations estimate that over 90% of the data will be stored and processed locally [3]. Local data storing and processing with global coordination is made possible by the emerging technology of mobile edge computing (MEC) [4], [5], where edge nodes, such as sensors, home gateways, micro servers, and small cells, are equipped with storage and computation capability. Multiple edge nodes work together with the remote cloud to perform large-scale distributed tasks that involve both local processing and remote coordination/execution.

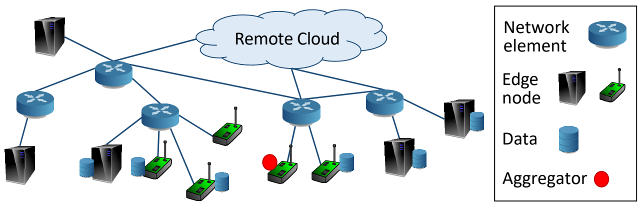


Fig. 1: System architecture.

To analyze large amounts of data and obtain useful information for the detection, classification, and prediction of future events, machine learning techniques are often applied. The definition of machine learning is very broad, ranging from simple data summarization with linear regression to multi-class classification with support vector machines (SVMs) and deep neural networks [6], [7]. The latter have shown very promising performance in recent years, for complex tasks such as image classification. One key enabler of machine learning is the ability to learn (train) models using a very large amount of data. With the increasing amount of data being generated by new applications and with more applications becoming data-driven, one can foresee that machine learning tasks will become a dominant workload in distributed MEC systems in the future. However, it is challenging to perform distributed machine learning on resource-constrained MEC systems.

In this paper, we address the problem of how to efficiently utilize the limited computation and communication resources at the edge for the optimal learning performance. We consider a typical edge computing architecture where edge nodes are interconnected with the remote cloud via network elements, such as gateways and routers, as illustrated in Fig. 1. The raw data is collected and stored at multiple edge nodes, and a machine learning model is trained from the distributed data *without* sending the raw data from the nodes to a central place. This variant of distributed machine learning (model training) from a federation of edge nodes is known as *federated learning* [8]–[10].

We focus on gradient-descent based federated learning algorithms, which have general applicability to a wide range of machine learning models. The learning process includes *local update* steps where each edge node performs gradient descent to adjust the (local) model parameter to minimize the loss function defined on its own dataset. It also includes

S. Wang *(corresponding author)*, T. Salonidis, and C. Makaya are with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. Email: wangshiq@us.ibm.com, tsaloni@us.ibm.com, chrismak@ieee.org

T. Tuor and K. K. Leung are with Imperial College London, UK. Email: tiffany.tuor14@imperial.ac.uk, kin.leung@imperial.ac.uk

T. He is with Pennsylvania State University, University Park, PA, USA. Email: t.he@cse.psu.edu

K. Chan is with Army Research Laboratory, Adelphi, MD, USA. Email: kevin.s.chan.civ@mail.mil

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

A preliminary version of this work entitled "When edge meets learning: adaptive control for resource-constrained distributed machine learning" was presented at IEEE INFOCOM 2018 [1].

global aggregation steps where model parameters obtained at different edge nodes are sent to an aggregator, which is a logical component that can run on the remote cloud, a network element, or an edge node. The aggregator aggregates these parameters (e.g., by taking a weighted average) and sends an updated parameter back to the edge nodes for the next round of iteration. The frequency of global aggregation is configurable; one can aggregate at an interval of one or multiple local updates. Each local update consumes computation resource of the edge node, and each global aggregation consumes communication resource of the network. The amount of consumed resources may vary over time, and there is a complex relationship among the frequency of global aggregation, the model training accuracy, and resource consumption.

We propose an algorithm to determine the frequency of global aggregation so that the available resource is most efficiently used. This is important because the training of machine learning models is usually resource-intensive, and a non-optimal operation of the learning task may waste a significant amount of resources. Our main contributions in this paper are as follows:

1) We analyze the convergence bound of gradient-descent based federated learning from a theoretical perspective, and obtain a novel convergence bound that incorporates non-independent-and-identically-distributed (non-i.i.d.) data distributions among nodes and an arbitrary number of local updates between two global aggregations.
2) Using the above theoretical convergence bound, we propose a control algorithm that learns the data distribution, system dynamics, and model characteristics, based on which it dynamically adapts the frequency of global aggregation in real time to minimize the learning loss under a fixed resource budget.
3) We evaluate the performance of the proposed control algorithm via extensive experiments using real datasets both on a hardware prototype and in a simulated environment, which confirm that our proposed approach provides near-optimal performance for different data distributions, various machine learning models, and system configurations with different numbers of edge nodes.

II. Related Work

Existing work on MEC focuses on generic applications, where solutions have been proposed for application offloading [11], [12], workload scheduling [13], [14], and service migration triggered by user mobility [15], [16]. However, they do not address the relationship among communication, computation, and training accuracy for machine learning applications, which is important for optimizing the performance of machine learning tasks.

The concept of federated learning was first proposed in [9], which showed its effectiveness through experiments on various datasets. Based on the comparison of synchronous and asynchronous methods of distributed gradient descent in [17], it is proposed in [9] that federated learning should use the synchronous approach because it is more efficient than asynchronous approaches. The approach in [9] uses a fixed global aggregation frequency. It does not provide theoretical convergence guarantee and the experiments were not conducted in a network setting. Several extensions have been made to the original federated learning proposal recently. For example, a mechanism for secure global aggregation is proposed in [18]. Methods for compressing the information exchanged within one global aggregation step is proposed in [19], [20]. Adjustments to the standard gradient descent procedure for better performance in the federated setting is studied in [21]. Participant (client) selection for federated learning is studied in [22]. An approach that shares a small amount of data with other nodes for better learning performance with non-i.i.d. data distribution is proposed in [23]. These studies do not consider the adaptation of global aggregation frequency, and thus they are orthogonal to our work in this paper. To the best of our knowledge, the adaptation of global aggregation frequency for federated learning with resource constraints has not been studied in the literature.

An area related to federated learning is distributed machine learning in datacenters through the use of worker machines and parameter servers [24]. The main difference between the datacenter environment and edge computing environment is that in datacenters, shared storage is usually used. The worker machines do not keep persistent data storage on their own, and they fetch the data from the shared storage at the beginning of the learning process. As a result, the data samples obtained by different workers are usually independent and identically distributed (i.i.d.). In federated learning, the data is collected at the edge directly and stored persistently at edge nodes, thus the data distribution at different edge nodes is usually non-i.i.d. Concurrently with our work in this paper, optimization of synchronization frequency with running time considerations is studied in [25] for the datacenter setting. It does not consider characteristics of non-i.i.d. data distributions which is essential in federated learning.

Distributed machine learning across multiple datacenters in different geographical locations is studied in [26], where a threshold-based approach to reduce the communication among different datacenters is proposed. Although the work in [26] is related to the adaptation of synchronization frequency with resource considerations, it focuses on peer-to-peer connected datacenters, which is different from the federated learning architecture that is not peer-to-peer. It also allows asynchronism among datacenter nodes, which is not the case in federated learning. In addition, the approach in [26] is designed empirically and does not consider a concrete theoretical objective, nor does it consider computation resource constraint which is important in MEC systems in addition to constrained communication resource.

From a theoretical perspective, bounds on the convergence of distributed gradient descent are obtained in [27]–[29], which only allow one step of local update before global aggregation. Partial global aggregation is allowed in the decentralized gradient descent approach in [30], [31], where after each local update step, parameter aggregation is performed over a non-empty subset of nodes, which does not apply in our federated learning setting where there is no aggregation at all after some of the local update steps. Multiple local updates before aggregation is possible in the bound derived in [26], but the number of local updates varies based on the thresholding procedure and cannot be specified as a given constant. Concurrently with our work, bounds with a fixed number of local updates between global aggregation steps are derived in [32], [33]. However, the bound in [32] only works with i.i.d. data distribution; the bound in [33] is independent from how different the datasets are, which is inefficient because it does not capture the fact that training on i.i.d. data is likely to converge faster than training on non-i.i.d. data. Related studies on distributed optimization that are applicable for machine learning applications also include [34]–[36], where a separate solver is used to solve a local problem. The main focus of [34]–[36] is the trade-off between communication and optimality, where the complexity of solving the local problem (such as the number of local updates needed) is not studied. In addition, many of the existing studies either explicitly or implicitly assume i.i.d. data distribution at different nodes, which is inappropriate in federated learning. To our knowledge, the convergence bound of distributed gradient descent in the federated learning setting, which captures both the characteristics of different (possibly non-i.i.d. distributed) datasets and a given number of local update steps between two global aggregation steps, has not been studied in the literature.

In contrast to the above research, our work in this paper formally addresses the problem of dynamically determining the global aggregation frequency to *optimize the learning with a given resource budget* for federated learning in MEC systems. This is a non-trivial problem due to the complex dependency between each learning step and its previous learning steps, which is hard to capture analytically. It is also challenging due to non-i.i.d. data distributions at different nodes, where the data distribution is unknown beforehand and the datasets may have different degrees of similarities with each other, and the real-time dynamics of the system. We propose an algorithm that is derived from theoretical analysis and adapts to real-time system dynamics.

We start with summarizing the basics of federated learning in the next section. In Section IV, we describe our problem formulation. The convergence analysis and control algorithm are presented in Sections V and VI, respectively. Experimentation results are shown in Section VII and the conclusion is presented in Section VIII.

III. Preliminaries and Definitions

A. Loss Function

Machine learning models include a set of parameters which are learned based on training data. A training data sample j usually consists of two parts. One is a vector $\mathbf{x}_j$ that is regarded as the input of the machine learning model (such as the pixels of an image); the other is a scalar y_j that is the desired output of the model (such as the label of the image). To facilitate the learning, each model has a loss function defined on its parameter vector $\mathbf{w}$ for each data sample j. The loss function captures the error of the model on the training data, and the model learning process is to minimize the loss function on a collection of training data samples. For each data sample j, we define the loss function as $f(\mathbf{w}, \mathbf{x}_j, y_j)$, which we write as $f_j(\mathbf{w})$ in short[1].

TABLE I: Loss functions for popular machine learning models

Model	Loss function $f(\mathbf{w}, \mathbf{x}_j, y_j)$ $(\triangleq f_j(\mathbf{w}))$
Squared-SVM	$\frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\max\left\{0; 1 - y_j\mathbf{w}^{\mathrm{T}}\mathbf{x}_j\right\}^2$ (λ is const.)
Linear regression	$\frac{1}{2}\|y_j - \mathbf{w}^{\mathrm{T}}\mathbf{x}_j\|^2$
K-means	$\frac{1}{2}\min_l \|\mathbf{x}_j - \mathbf{w}_{(l)}\|^2$ where $\mathbf{w} \triangleq [\mathbf{w}_{(1)}^{\mathrm{T}}, \mathbf{w}_{(2)}^{\mathrm{T}}, ...]^{\mathrm{T}}$
Convolutional neural network	Cross-entropy on cascaded linear and non-linear transforms, see [7]

Examples of loss functions of popular machine learning models are summarized[2] in Table I [6], [7], [37]. For convenience, we assume that all vectors are column vectors in this paper and use $\mathbf{x}^{\mathrm{T}}$ to denote the transpose of $\mathbf{x}$. We use "$\triangleq$" to denote "is defined to be equal to" and use $\|\cdot\|$ to denote the $\mathcal{L}^2$ norm.

Assume that we have N edge nodes with local datasets $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_i, ..., \mathcal{D}_N$. For each dataset $\mathcal{D}_i$ at node i, the loss function on the collection of data samples at this node is

$$F_i(\mathbf{w}) \triangleq \frac{1}{|\mathcal{D}_i|}\sum_{j\in\mathcal{D}_i} f_j(\mathbf{w}). \tag{1}$$

We define $D_i \triangleq |\mathcal{D}_i|$, where $|\cdot|$ denotes the size of the set, and $D \triangleq \sum_{i=1}^{N} D_i$. Assuming $\mathcal{D}_i \cap \mathcal{D}_{i'} = \emptyset$ for $i \neq i'$, we define the global loss function on all the distributed datasets as

$$F(\mathbf{w}) \triangleq \frac{\sum_{j\in\cup_i\mathcal{D}_i} f_j(\mathbf{w})}{|\cup_i\mathcal{D}_i|} = \frac{\sum_{i=1}^{N} D_i F_i(\mathbf{w})}{D}. \tag{2}$$

Note that $F(\mathbf{w})$ *cannot* be directly computed without sharing information among multiple nodes.

B. The Learning Problem

The learning problem is to minimize $F(\mathbf{w})$, i.e., to find

$$\mathbf{w}^* \triangleq \arg\min F(\mathbf{w}). \tag{3}$$

Due to the inherent complexity of most machine learning models, it is usually impossible to find a closed-form solution to (3). Thus, (3) is often solved using gradient-descent techniques.

C. Distributed Gradient Descent

We present a canonical distributed gradient-descent algorithm to solve (3), which is widely used in state-of-the-art federated learning systems (e.g., [9]). Each node i has its local model parameter $\mathbf{w}_i(t)$, where $t = 0, 1, 2, ...$ denotes the iteration index. At $t = 0$, the local parameters for all nodes i are initialized to the same value. For $t > 0$, new values of $\mathbf{w}_i(t)$ are computed according to a gradient-descent update rule on the local loss function, based on the parameter value in the previous iteration $t-1$. This gradient-descent step on the local loss function (defined on the local dataset) at each node is referred to as the *local update*. After one or multiple

[1]Note that some unsupervised models (such as K-means) only learn on $\mathbf{x}_j$ and do not require the existence of y_j in the training data. In such cases, the loss function value only depends on $\mathbf{x}_j$.

[2]While our focus is on non-probabilistic learning models, similar loss functions can be defined for probabilistic models where the goal is to minimize the negative of the log-likelihood function, for instance.

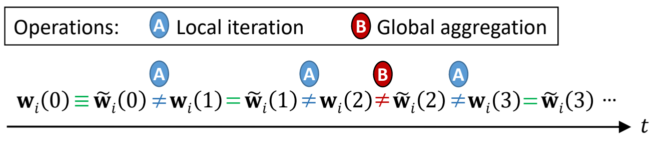

Operations:
A Local iteration
B Global aggregation
A A B A
$\mathbf{w}_i(0) \equiv \widetilde{\mathbf{w}}_i(0) \neq \mathbf{w}_i(1) = \widetilde{\mathbf{w}}_i(1) \neq \mathbf{w}_i(2) \neq \widetilde{\mathbf{w}}_i(2) \neq \mathbf{w}_i(3) = \widetilde{\mathbf{w}}_i(3) \cdots$
t

Fig. 2: Illustration of the values of $\mathbf{w}_i(t)$ and $\widetilde{\mathbf{w}}_i(t)$ at node i.

Algorithm 1: Distributed gradient descent (logical view)

```
Input: $\tau$, $T$
Output: Final model parameter $\mathbf{w}^{\mathrm{f}}$
1 Initialize $\mathbf{w}^{\mathrm{f}}$, $\mathbf{w}_i(0)$ and $\widetilde{\mathbf{w}}_i(0)$ to the same value for all $i$;
2 for $t = 1, 2, ..., T$ do
3     For each node $i$ in parallel, compute local update using (4);
4     if $t$ is an integer multiple of $\tau$ then
5         Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}(t)$ for all $i$, where $\mathbf{w}(t)$ is defined in (5);
          //Global aggregation
6         Update $\mathbf{w}^{\mathrm{f}} \leftarrow \arg\min_{\mathbf{w} \in \{\mathbf{w}^{\mathrm{f}}, \mathbf{w}(t)\}} F(\mathbf{w})$;
7     else
8         Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}_i(t)$ for all $i$; //No global aggregation
```

local updates, a *global aggregation* is performed through the aggregator to update the local parameter at each node to the weighted average of all nodes' parameters. We define that each *iteration* includes a local update step which is possibly followed by a global aggregation step.

After global aggregation, the local parameter $\mathbf{w}_i(t)$ at each node i usually changes. For convenience, we use $\widetilde{\mathbf{w}}_i(t)$ to denote the parameter at node i after possible global aggregation. If no aggregation is performed at iteration t, we have $\widetilde{\mathbf{w}}_i(t) = \mathbf{w}_i(t)$. If aggregation is performed at iteration t, then generally $\widetilde{\mathbf{w}}_i(t) \neq \mathbf{w}_i(t)$ and we set $\widetilde{\mathbf{w}}_i(t) = \mathbf{w}(t)$, where $\mathbf{w}(t)$ is a weighted average of $\mathbf{w}_i(t)$ defined in (5) below. An example of these definitions is shown in Fig. 2.

The local update in each iteration is performed on the parameter after possible global aggregation in the previous iteration. For each node i, the update rule is as follows:

$$\mathbf{w}_i(t) = \widetilde{\mathbf{w}}_i(t-1) - \eta \nabla F_i\left(\widetilde{\mathbf{w}}_i(t-1)\right) \quad (4)$$

where $\eta > 0$ is the step size. For any iteration t (which may or may not include a global aggregation step), we define

$$\mathbf{w}(t) = \frac{\sum_{i=1}^{N} D_i \mathbf{w}_i(t)}{D}. \quad (5)$$

This global model parameter $\mathbf{w}(t)$ is *only observable to nodes in the system if global aggregation is performed at iteration* t, but we define it for all t to facilitate the analysis later.

We define that the system performs τ steps of local updates at each node between every two global aggregations. We define T as the total number of local iterations at each node. For ease of presentation, we assume that T is an integer multiple of τ in the theoretical analysis, which will be relaxed when we discuss practical aspects in Section VI-B. The logic of distributed gradient descent is presented in Algorithm 1, which ignores aspects related to the communication between the aggregator and edge nodes. Such aspects will be discussed later in Section VI-B.

The final model parameter $\mathbf{w}^{\mathrm{f}}$ obtained from Algorithm 1 is the one that has produced the minimum global loss after each global aggregation throughout the entire execution of the algorithm. We use $\mathbf{w}^{\mathrm{f}}$ instead of $\mathbf{w}(T)$, to align with the theoretical convergence bound that will be presented in Section V-B. In practice, we have seen that $\mathbf{w}^{\mathrm{f}}$ and $\mathbf{w}(T)$ are usually the same, but using $\mathbf{w}^{\mathrm{f}}$ provides theoretical rigor in terms of convergence guarantee so we use $\mathbf{w}^{\mathrm{f}}$ in this paper. Note that $F(\mathbf{w})$ in Line 6 of Algorithm 1 is computed in a distributed manner according to (2); the details will be presented later.

The rationale behind Algorithm 1 is that when $\tau = 1$, i.e., when we perform global aggregation after every local update step, the distributed gradient descent (ignoring communication aspects) is equivalent to the centralized gradient descent, where the latter assumes that all data samples are available at a centralized location and the global loss function and its gradient can be observed directly. This is due to the linearity of the gradient operator. Due to space limitation, see our online technical report [38, Appendix A] as well as [39] for detailed discussions about this.

The main notations in this paper are summarized in Table II.

TABLE II: Summary of main notations

$F(\mathbf{w})$	Global loss function
$F_i(\mathbf{w})$	Local loss function for node i
t	Iteration index
$\mathbf{w}_i(t)$	Local model parameter at node i in iteration t
$\mathbf{w}(t)$	Global model parameter in iteration t
$\mathbf{w}^{\mathrm{f}}$	Final model parameter obtained at the end of learning process
$\mathbf{w}^*$	True optimal model parameter that minimizes $F(\mathbf{w})$
η	Gradient descent step size
τ	Number of local update steps between two global aggregations
T	Total number of local update steps at each node
K	Total number of global aggregation steps, equal to T/τ
M (m)	Total number of resource types (the m-th type of resource)
R_m	Total budget of the m-th type of resource
c_m	Consumption of type-m resource in one local update step
b_m	Consumption of type-m resource in one global aggregation step
ρ	Lipschitz parameter of $F_i(\mathbf{w})$ $(\forall i)$ and $F(\mathbf{w})$
β	Smoothness parameter of $F_i(\mathbf{w})$ $(\forall i)$ and $F(\mathbf{w})$
δ	Gradient divergence
$h(\tau)$	Function defined in (11), gap between the model parameters obtained from distributed and centralized gradient descents
φ	Constant defined in Lemma 2, control parameter
$G(\tau)$	Function defined in (18), control objective
τ^*	Optimal τ obtained by minimizing $G(\tau)$

IV. Problem Formulation

When there is a large amount of data (which is usually needed for training an accurate model) distributed at a large number of nodes, the federated learning process can consume a significant amount of resources. The notion of "resources" here is generic and can include time, energy, monetary cost etc. *related to both computation and communication*. One often has to limit the amount of resources used for learning each model, in order not to backlog the system and to keep the operational cost low. This is particularly important in edge computing environments where the computation and communication resources are not as abundant as in datacenters.

Therefore, a natural question is how to make efficient use of a given amount of resources to minimize the loss function of model training. For the distributed gradient-descent based learning approach presented above, the question narrows down to determining the optimal values of T and τ, so that the global loss function is minimized subject to a given resource constraint for this learning task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2019.2904348, IEEE Journal on Selected Areas in Communications

We use K to denote the total number of global aggregations within T iterations. Because we assumed earlier that T is an integer multiple of τ, we have $K = \frac{T}{\tau}$. We define

$$\mathbf{w}^{\mathrm{f}} \triangleq \underset{\mathbf{w}\in\{\mathbf{w}(k\tau):k=0,1,2,...,K\}}{\arg\min} F(\mathbf{w}). \tag{6}$$

It is easy to verify that this definition is equivalent to $\mathbf{w}^{\mathrm{f}}$ found from Algorithm 1.

To compute $F(\mathbf{w})$ in (6), each node i first computes $F_i(\mathbf{w})$ and sends the result to the aggregator, then the aggregator computes $F(\mathbf{w})$ according to (2). Since each node only knows the value of $\mathbf{w}(k\tau)$ after the k-th global aggregation, $F_i(\mathbf{w}(k\tau))$ at node i will be sent back to the aggregator at the $(k+1)$-th global aggregation, and the aggregator computes $F(\mathbf{w}(k\tau))$ afterwards. To compute the last loss value $F(\mathbf{w}(K\tau)) = F(\mathbf{w}(T))$, an additional round of local and global update is performed at the end. We assume that at each node, local update consumes the same amount of resource no matter whether only the local loss is computed (in the last round) or both the local loss and gradient are computed (in all the other rounds), because the loss and gradient computations can usually be based on the same intermediate result. For example, the back propagation approach for computing gradients in neural networks requires a forward propagation procedure that essentially obtains the loss as an intermediate step [7].

We consider M different types of resources. For example, one type of resource can be time, another type can be energy, a third type can be communication bandwidth, etc. For each $m \in \{1, 2, ..., M\}$, we define that each local update step at *all* nodes consumes c_m units of type-m resource, and each global aggregation step consumes b_m units of type-m resource, where $c_m \geq 0$ and $b_m \geq 0$ are both *finite* real numbers. For given T and τ, the total amount of consumed type-m resource is $(T+1)c_m + (K+1)\,b_m$, where the additional "+1" is for computing $F(\mathbf{w}(K\tau))$, as discussed above.

Let R_m denote the total budget of type-m resource. We seek the solution to the following problem:

$$\begin{aligned} \min_{\tau,K\in\{1,2,3,...\}} \quad & F(\mathbf{w}^{\mathrm{f}}) \\ \text{s.t.} \quad & (T+1)c_m+(K+1)\,b_m \leq R_m\,,\ \forall m\in\{1,...,M\} \\ & T = K\tau. \end{aligned} \tag{7}$$

To solve (7), we need to find out how τ and K (and thus T) affect the loss function $F(\mathbf{w}^{\mathrm{f}})$ computed on the final model parameter $\mathbf{w}^{\mathrm{f}}$. It is generally impossible to find an exact analytical expression to relate τ and K with $F(\mathbf{w}^{\mathrm{f}})$, because it depends on the convergence property of gradient descent (for which only upper/lower bounds are known [40]) and the impact of the global aggregation frequency on the convergence. Further, the resource consumptions c_m and b_m can be time-varying in practice which makes the problem even more challenging than (7) alone.

We analyze the convergence bound of distributed gradient descent (Algorithm 1) in Section V, then use this bound to approximately solve (7) and propose a control algorithm for adaptively choosing the best values of τ and T to achieve near-optimal resource utilization in Section VI.

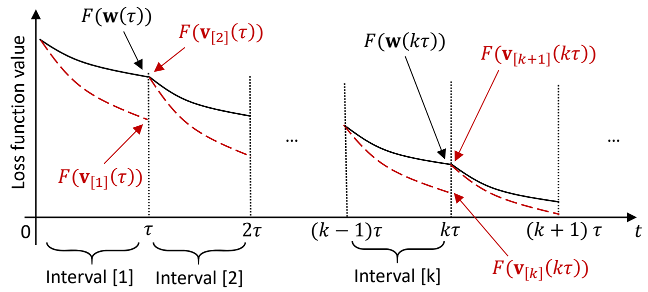


Fig. 3: Illustration of definitions in different intervals.

V. Convergence Analysis

We analyze the convergence of Algorithm 1 in this section and find an upper bound of $F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*)$. To facilitate the analysis, we first introduce some notations.

A. Definitions

We can divide the T iterations into K different intervals, as shown in Fig. 3, with only the first and last iterations in each interval containing global aggregation. We use the shorthand notations $[k]$ to denote the iteration interval[3] $[(k-1)\tau, k\tau]$, for $k = 1, 2, ..., K$.

For each interval $[k]$, we use $\mathbf{v}_{[k]}(t)$ to denote an auxiliary parameter vector that follows a *centralized* gradient descent according to

$$\mathbf{v}_{[k]}(t) = \mathbf{v}_{[k]}(t-1) - \eta\nabla F(\mathbf{v}_{[k]}(t-1)) \tag{8}$$

where $\mathbf{v}_{[k]}(t)$ is only defined for $t \in [(k-1)\tau, k\tau]$ for a given k. This update rule is based on the global loss function $F(\mathbf{w})$ which is only observable when all data samples are available at a central place (thus we call it centralized gradient descent), whereas the iteration in (4) is on the local loss function $F_i(\mathbf{w})$.

We define that $\mathbf{v}_{[k]}(t)$ is "synchronized" with $\mathbf{w}(t)$ at the beginning of each interval $[k]$, i.e., $\mathbf{v}_{[k]}((k-1)\tau) \triangleq \mathbf{w}((k-1)\tau)$, where $\mathbf{w}(t)$ is the average of local parameters defined in (5). Note that we also have $\widetilde{\mathbf{w}}_i((k-1)\tau) = \mathbf{w}((k-1)\tau)$ for all i because the global aggregation (or initialization when $k = 1$) is performed in iteration $(k-1)\tau$.

The above definitions enable us to find the convergence bound of Algorithm 1 by taking a two-step approach. The first step is to find the gap between $\mathbf{w}(k\tau)$ and $\mathbf{v}_{[k]}(k\tau)$ for each k, which is the difference between the distributed and centralized gradient descents after τ steps of local updates without global aggregation. The second step is to combine this gap with the convergence bound of $\mathbf{v}_{[k]}(t)$ within each interval $[k]$ to obtain the convergence bound of $\mathbf{w}(t)$.

For the purpose of the analysis, we make the following assumption to the loss function.

Assumption 1. *We assume the following for all i:*

1) *$F_i(\mathbf{w})$ is convex*
2) *$F_i(\mathbf{w})$ is ρ-Lipschitz, i.e., $\|F_i(\mathbf{w}) - F_i(\mathbf{w}')\| \leq \rho\|\mathbf{w} - \mathbf{w}'\|$ for any $\mathbf{w}, \mathbf{w}'$*

[3]With slight abuse of notation, we use $[(k-1)\tau, k\tau]$ to denote the integers contained in the interval for simplicity. We use the same convention in other parts of the paper as long as there is no ambiguity.

3) $F_i(\mathbf{w})$ is β-smooth, i.e., $\|\nabla F_i(\mathbf{w}) - \nabla F_i(\mathbf{w}')\| \leq \beta \|\mathbf{w} - \mathbf{w}'\|$ for any $\mathbf{w}, \mathbf{w}'$

Assumption 1 is satisfied for squared-SVM and linear regression (see Table I). The experimentation results that will be presented in Section VII show that our control algorithm also works well for models (such as neural network) whose loss functions do not satisfy Assumption 1.

Lemma 1. *$F(\mathbf{w})$ is convex, ρ-Lipschitz, and β-smooth.*

Proof. Straightforwardly from Assumption 1, the definition of $F(\mathbf{w})$, and triangle inequality. □

We also define the following metric to capture the *divergence* between the gradient of a local loss function and the gradient of the global loss function. This divergence is *related to how the data is distributed at different nodes*.

Definition 1. *(Gradient Divergence) For any i and $\mathbf{w}$, we define δ_i as an upper bound of $\|\nabla F_i(\mathbf{w}) - \nabla F(\mathbf{w})\|$, i.e.,*

$$\|\nabla F_i(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta_i. \tag{9}$$

We also define $\delta \triangleq \frac{\sum_i D_i \delta_i}{D}$.

B. Main Results

The below theorem gives an upper bound on the difference between $\mathbf{w}(t)$ and $\mathbf{v}_{[k]}(t)$ when t is within the interval $[k]$.

Theorem 1. *For any interval $[k]$ and $t \in [k]$, we have*

$$\left\|\mathbf{w}(t) - \mathbf{v}_{[k]}(t)\right\| \leq h(t-(k-1)\tau) \tag{10}$$

where

$$h(x) \triangleq \frac{\delta}{\beta}\left((\eta\beta+1)^x - 1\right) - \eta\delta x \tag{11}$$

for any $x = 0, 1, 2, \ldots$.

Furthermore, as $F(\cdot)$ is ρ-Lipschitz, we have $F(\mathbf{w}(t)) - F(\mathbf{v}_{[k]}(t)) \leq \rho h(t-(k-1)\tau)$.

Proof. We first obtain an upper bound of $\left\|\widetilde{\mathbf{w}}_i(t) - \mathbf{v}_{[k]}(t)\right\|$ for each node i, based on which the final result is obtained. See our online technical report [38, Appendix B] for details. □

Note that we always have $\eta > 0$ and $\beta > 0$ because otherwise the gradient descent procedure or the loss function becomes trivial. Therefore, we have $(\eta\beta+1)^x \geq \eta\beta x + 1$ for $x = 0, 1, 2, \ldots$ due to Bernoulli's inequality. Substituting this into (11) confirms that we always have $h(x) \geq 0$.

It is easy to see that $h(0) = h(1) = 0$. Therefore, when $t = (k-1)\tau$, i.e., at the beginning of the interval $[k]$, the upper bound in (10) is zero. This is consistent with the definition of $\mathbf{v}_{[k]}((k-1)\tau) = \mathbf{w}((k-1)\tau)$ for any k. When $t = (k-1)\tau + 1$ (i.e., the second iteration in interval $[k]$), the upper bound in (10) is also zero. This agrees with the discussion at the end of Section III-C, showing that there is no gap between distributed and centralized gradient descents when only one local update is performed after the global aggregation. If $\tau = 1$, then $t - (k-1)\tau$ is either 0 or 1 for any interval $[k]$ and $t \in [k]$. Hence, the upper bound in (10) becomes exact for $\tau = 1$.

For $\tau > 1$, the value of $x = t - (k-1)$ can be larger. When x is large, the exponential term with $(\eta\beta+1)^x$ in (11) becomes dominant, and the gap between $\mathbf{w}(t)$ and $\mathbf{v}_{[k]}(t)$ can increase exponentially with t for $t \in [k]$. We also note that $h(x)$ is proportional to the gradient divergence δ (see (11)), which is intuitive because the more the local gradient is different from the global gradient (for the same parameter $\mathbf{w}$), the larger the gap will be. The gap is caused by the difference in the local gradients at different nodes starting at the second local update after each global aggregation. In an extreme case when all nodes have exactly the same data samples (and thus the same local loss functions), the gradients will be always the same and $\delta = 0$, in which case $\mathbf{w}(t)$ and $\mathbf{v}_{[k]}(t)$ are always equal.

Theorem 1 gives an upper bound of the difference between distributed and centralized gradient descents for each iteration interval $[k]$, assuming that $\mathbf{v}_{[k]}(t)$ in the centralized gradient descent is synchronized with $\mathbf{w}(t)$ at the beginning of each $[k]$. Based on this result, we first obtain the following lemma.

Lemma 2. *When all the following conditions are satisfied:*

1) $\eta \leq \frac{1}{\beta}$
2) $\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon^2} > 0$
3) $F\left(\mathbf{v}_{[k]}(k\tau)\right) - F(\mathbf{w}^*) \geq \varepsilon$ for all k
4) $F\left(\mathbf{w}(T)\right) - F(\mathbf{w}^*) \geq \varepsilon$

for some $\varepsilon > 0$, where we define $\varphi \triangleq \omega\left(1 - \frac{\beta\eta}{2}\right)$ and $\omega \triangleq \min_k \frac{1}{\left\|\mathbf{v}_{[k]}((k-1)\tau) - \mathbf{w}^*\right\|^2}$, then the convergence upper bound of Algorithm 1 after T iterations is given by

$$F(\mathbf{w}(T)) - F(\mathbf{w}^*) \leq \frac{1}{T\left(\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon^2}\right)}. \tag{12}$$

Proof. We first analyze the convergence of $F\left(\mathbf{v}_{[k]}(t)\right)$ within each interval $[k]$. Then, we combine this result with the gap between $F(\mathbf{w}(t))$ and $F(\mathbf{v}_{[k]}(t))$ from Theorem 1 to obtain the final result. See our online technical report [38, Appendix C] for details. □

We then have the following theorem.

Theorem 2. *When $\eta \leq \frac{1}{\beta}$, we have*

$$F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*) \leq \frac{1}{2\eta\varphi T} + \sqrt{\frac{1}{4\eta^2\varphi^2T^2} + \frac{\rho h(\tau)}{\eta\varphi\tau}} + \rho h(\tau). \tag{13}$$

Proof. Condition 1 in Lemma 2 is always satisfied due to the condition $\eta \leq \frac{1}{\beta}$ in this theorem.

When $\rho h(\tau) = 0$, we can choose ε to be arbitrarily small (but greater than zero) so that conditions 2–4 in Lemma 2 are satisfied. We see that the right-hand sides of (12) and (13) are equal in this case (when $\rho h(\tau) = 0$), and the result in (13) follows directly from Lemma 2 because $F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*) \leq F(\mathbf{w}(T)) - F(\mathbf{w}^*)$ according to the definition of $\mathbf{w}^{\mathrm{f}}$ in (6).

We consider $\rho h(\tau) > 0$ in the following. Consider the right-hand side of (12) and let

$$\varepsilon_0 = \frac{1}{T\left(\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon_0^2}\right)}. \tag{14}$$

Solving for ε_0, we obtain

$$\varepsilon_0 = \frac{1}{2\eta\varphi T} + \sqrt{\frac{1}{4\eta^2\varphi^2 T^2} + \frac{\rho h(\tau)}{\eta\varphi\tau}} \tag{15}$$

where the negative solution is ignored because $\varepsilon > 0$ in Lemma 2. Because $\varepsilon_0 > 0$ according to (15), the denominator of (14) is greater than zero, thus condition 2 in Lemma 2 is satisfied for any $\varepsilon \geq \varepsilon_0$, where we note that $\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon^2}$ increases with ε when $\rho h(\tau) > 0$.

Suppose that there exists $\varepsilon > \varepsilon_0$ satisfying conditions 3 and 4 in Lemma 2, so that all the conditions in Lemma 2 are satisfied. Applying Lemma 2 and considering (14), we have

$$F(\mathbf{w}(T)) - F(\mathbf{w}^*) \leq \frac{1}{T\left(\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon^2}\right)} < \frac{1}{T\left(\eta\varphi - \frac{\rho h(\tau)}{\tau\varepsilon_0^2}\right)} = \varepsilon_0$$

which contradicts with condition 4 in Lemma 2. Therefore, there *does not* exist $\varepsilon > \varepsilon_0$ that satisfy both conditions 3 and 4 in Lemma 2. This means that either 1) $\exists k$ such that $F\left(\mathbf{v}_{[k]}(k\tau)\right) - F(\mathbf{w}^*) \leq \varepsilon_0$ or 2) $F\left(\mathbf{w}(T)\right) - F(\mathbf{w}^*) \leq \varepsilon_0$. It follows that

$$\min\left\{\min_{k=1,2,\ldots,K} F\left(\mathbf{v}_{[k]}(k\tau)\right); F\left(\mathbf{w}(T)\right)\right\} - F(\mathbf{w}^*) \leq \varepsilon_0. \tag{16}$$

From Theorem 1, $F(\mathbf{w}(k\tau)) \leq F(\mathbf{v}_{[k]}(k\tau)) + \rho h(\tau)$ for any k. Combining with (16), we get

$$\min_{k=1,2,\ldots,K} F\left(\mathbf{w}(k\tau)\right) - F(\mathbf{w}^*) \leq \varepsilon_0 + \rho h(\tau)$$

where we recall that $T = K\tau$. Using (6) and (15), we obtain the result in (13). □

We note that the bound in (13) has no restriction on how the data is distributed at different nodes. The impact of different data distribution is captured by the gradient divergence δ, which is included in $h(\tau)$. It is easy to see from (11) that $h(\tau)$ is non-negative, non-decreasing in τ, and proportional to δ. Thus, as one would intuitively expect, for a given total number of local update steps T, the optimality gap (i.e., $F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*)$) becomes larger when τ and δ are larger. For given τ and δ, the optimality gap becomes smaller when T is larger. When $\tau = 1$, we have $h(\tau) = 0$, and the optimality gap converges to zero as $T \to \infty$. When $\tau > 1$, we have $h(\tau) > 0$, and we can see from (13) that in this case, convergence is only guaranteed to a non-zero optimality gap as $T \to \infty$. This means that when we have unlimited budget for all types of resources (i.e., $R_m \to \infty, \forall m$), it is always optimal to set $\tau = 1$ and perform global aggregation after every step of local update. However, when the resource budget R_m is limited for some m, the training will be terminated after a finite number of iterations, thus the value of T is finite. In this case, it may be better to perform global aggregation less frequently so that more resources can be used for local update, as we will see later in this paper.

VI. Control Algorithm

We propose an algorithm that approximately solves (7) in this section. We first assume that the resource consumptions c_m and b_m ($\forall m$) are known, and we solve for the values of τ and T. Then, we consider practical scenarios where c_m, b_m, and some other parameters are unknown and may vary over time, and we propose a control algorithm that estimates the parameters and dynamically adjusts the value of τ in real time.

A. Approximate Solution to (7)

We assume that η is chosen small enough such that $\eta \leq \frac{1}{\beta}$, and use the upper bound in (13) as an approximation of $F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*)$. Because for a given global loss function $F(\mathbf{w})$, its minimum value $F(\mathbf{w}^*)$ is a constant, the minimization of $F(\mathbf{w}^{\mathrm{f}})$ in (7) is equivalent to minimizing $F(\mathbf{w}^{\mathrm{f}}) - F(\mathbf{w}^*)$. With this approximation and rearranging the inequality constraints in (7), we can rewrite (7) as

$$\min_{\tau, K \in \{1,2,3,\ldots\}} \frac{1}{2\eta\varphi T} + \sqrt{\frac{1}{4\eta^2\varphi^2 T^2} + \frac{\rho h(\tau)}{\eta\varphi\tau}} + \rho h(\tau) \tag{17}$$
$$\text{s.t. } K \leq \frac{R'_m}{c_m\tau + b_m}, \quad \forall m \in \{1, \ldots, M\}$$
$$T = K\tau$$

where $R'_m \triangleq R_m - b_m - c_m$.

It is easy to see that the objective function in (17) decreases with T, thus it also decreases with K because $T = K\tau$. Therefore, for any τ, the optimal value of K is $\left\lfloor \min_m \frac{R'_m}{c_m\tau + b_m} \right\rfloor$, i.e., the largest value of K that does not violate any inequality constraint in (17), where $\lfloor\cdot\rfloor$ denotes the floor function for rounding down to integer. To simplify the analysis, we approximate by ignoring the rounding operation and substituting $T = K\tau \approx \min_m \frac{R'_m\tau}{c_m\tau + b_m} = 1 \big/ \max_m \frac{c_m\tau + b_m}{R'_m\tau}$ into the objective function in (17), yielding

$$G(\tau) \triangleq \frac{\max_m \frac{c_m\tau + b_m}{R'_m\tau}}{2\eta\varphi} + \sqrt{\frac{\left(\max_m \frac{c_m\tau + b_m}{R'_m\tau}\right)^2}{4\eta^2\varphi^2} + \frac{\rho h(\tau)}{\eta\varphi\tau}} + \rho h(\tau) \tag{18}$$

and we can define the (approximately) optimal τ as

$$\tau^* = \arg\min_{\tau \in \{1,2,3,\ldots\}} G(\tau) \tag{19}$$

from which we can directly obtain the (approximately) optimal K as $K^* = \left\lfloor \min_m \frac{R'_m}{c_m\tau^* + b_m} \right\rfloor$, and the (approximately) optimal T as $T^* = K^*\tau^* = \left\lfloor \min_m \frac{R'_m}{c_m\tau^* + b_m} \right\rfloor \tau^*$.

Proposition 1. *When $\eta \leq \frac{1}{\beta}$, $\rho > 0$, $\beta > 0$, $\delta > 0$, we have $\lim_{R_{\min}\to\infty} \tau^* = 1$, where $R_{\min} \triangleq \min_m R_m$.*

Proof. Because $R_{\min} \to \infty \iff R_m \to \infty, \forall m \iff R'_m \to \infty, \forall m$, we have $\lim_{R_{\min}\to\infty} \max_m \frac{c_m\tau + b_m}{R'_m\tau} = \max_m \lim_{R'_m\to\infty} \frac{c_m\tau + b_m}{R'_m\tau} = 0$. Thus, $\lim_{R_{\min}\to\infty} G(\tau) = \sqrt{\frac{\rho h(\tau)}{\eta\varphi\tau}} + \rho h(\tau)$. Let $B \triangleq \eta\beta + 1$. With a slight abuse of notation, we consider continuous values of $\tau \geq 1$. We have

$$d\left(\frac{h(\tau)}{\tau}\right) \Big/ d\tau = \frac{\delta}{\beta\tau^2}\left(B^\tau \log B^\tau - (B^\tau - 1)\right)$$
$$\geq \frac{\delta}{\beta\tau^2}\left(B^\tau\left(1 - \frac{1}{B^\tau}\right) - B^\tau - 1\right) \geq 0$$

where the first inequality is from a lower bound of logarithmic function [41]. We also have

$$\begin{aligned}\frac{dh(\tau)}{d\tau} &= \frac{\delta}{\beta}(B^\tau \log B - \eta\beta) \geq \frac{\delta}{\beta}\left(\frac{2\eta\beta B^\tau}{2+\eta\beta} - \eta\beta\right) \\ &= \frac{\delta(2\eta\beta B^\tau - 2\eta\beta - \eta^2\beta^2)}{\beta(2+\eta\beta)} \\ &\geq \frac{\delta(2\eta\beta B - 2\eta\beta - \eta^2\beta^2)}{\beta(2+\eta\beta)} = \frac{\delta\eta^2\beta^2}{\beta(2+\eta\beta)} > 0\end{aligned}$$

where the first inequality is from a lower bound of $\log B$ [41], the second inequality is because $B > 1$ and $\tau \geq 1$.

Thus, for any $\tau \geq 1$, $h(\tau)$ increases with τ, and $\frac{h(\tau)}{\tau}$ is non-decreasing with τ. We also note that $\sqrt{x}$ increases with x for any $x \geq 0$, and $h(1) = 0$. It follows that $\lim_{R_{\min}\to\infty} G(\tau)$ increases with τ for any $\tau \geq 1$. Hence, $\lim_{R_{\min}\to\infty} \tau^* = 1$. □

Combining Proposition 1 with Theorem 2, we know that using τ^* found from (19) guarantees convergence with zero optimality gap as $R_{\min} \to \infty$ (and thus $R'_m \to \infty, \forall m$ and $T^* \to \infty$), because $\lim_{R_{\min}\to\infty} \tau^* = 1$ and $h(1) = 0$. For general values of R_m (and R'_m), we have the following result.

Proposition 2. *When $\eta \leq \frac{1}{\beta}$, $\rho > 0$, $\beta > 0$, $\delta > 0$, there exists a* finite *value τ_0, which only depends on η, β, ρ, δ, φ, c_m, b_m, R'_m ($\forall m$), such that $\tau^* \leq \tau_0$. The quantity τ_0 is defined as*

$$\tau_0 \triangleq \max\left\{\max_m \frac{b_m R'_\nu - b_\nu R'_m}{c_\nu R'_m - c_m R'_\nu}\,;\; \frac{\varphi(2+\eta\beta)}{2\rho\delta}\left(\frac{2c_\nu b_\nu}{C_2} + \frac{2b_\nu^2}{C_2}\right);\right. \\ \left.\frac{1}{\rho\delta\eta\log B}\left(\frac{b_\nu}{C_1} + \rho\eta\delta\right) - \frac{1}{\eta\beta};\;\; \frac{1}{\eta\beta} + \frac{1}{2}\right\}$$

where index $\nu \triangleq \arg\max_{m\in V} \frac{b_m}{R'_m}$ (set $V \triangleq \arg\max_m \frac{c_m}{R'_m}$), $B \triangleq \eta\beta + 1$, $C_1 \triangleq 2\eta\varphi R'_\nu$, $C_2 \triangleq 4\eta^2\varphi^2 R'^2_\nu$. Here, for convenience, we allow $\arg\max$ to interchangeably return a set and an arbitrary value in that set, we also define $\frac{0}{0} \triangleq 0$.

We also note that $0 < \eta\beta \leq 1$, thus $\tau_0 \geq \frac{1}{\eta\beta} + \frac{1}{2} > 1$.

Proof. We can show that $\max_m \frac{b_m R'_\nu - b_\nu R'_m}{c_\nu R'_m - c_m R'_\nu}$ is finite according to the definition of ν and $\frac{0}{0}$, then it is easy to see that τ_0 is finite. We then show $\arg\max_m \frac{c_m\tau + b_m}{R'_m \tau} = \nu$ for any $\tau > \tau_0$, in which case the maximization over m in (18) becomes fixing $m = \nu$. Then, the proof separately considers the terms inside and outside the square root in (18). It shows that the first order derivatives of both parts are always larger than zero when $\tau > \tau_0$. Because the square root is an increasing function, $G(\tau)$ increases with τ for $\tau > \tau_0$, and thus $\tau^* \leq \tau_0$. See our online technical report [38, Appendix D] for details. □

There is no closed-form solution for τ^* because $G(\tau)$ includes both polynomial and exponential terms of τ, where the exponential term is embedded in $h(\tau)$. Because τ^* can only be a positive integer, according to Proposition 2, we can compute $G(\tau)$ within a finite range of τ to find τ^* that minimizes $G(\tau)$.

B. Adaptive Federated Learning

In this subsection, we present the complete control algorithm for adaptive federated learning, which recomputes τ^* in every global aggregation step based on the most recent system state. We use the theoretical results above to guide the design of the algorithm.

As mentioned earlier, the local updates run on edge nodes and the global aggregation is performed through the assistance of an aggregator, where the aggregator is a logical component that may also run on one of the edge nodes. The complete procedures at the aggregator and each edge node are presented in Algorithms 2 and 3, respectively, where Lines 8–12 of Algorithm 3 are for local updates and the rest is considered as part of global aggregation, initialization, or final operation. We assume that the aggregator initiates the learning process, and the initial model parameter $\mathbf{w}(0)$ is sent by the aggregator to all edge nodes. We note that instead of transmitting the entire model parameter vector in every global aggregation step, one can also transmit compressed or quantized model parameters to further save the communication bandwidth, where the compression or quantization can be performed using techniques described in [19], [20], for instance.

1) Estimation of Parameters in $G(\tau)$: The expression of $G(\tau)$, which includes $h(\tau)$, has parameters which need to be estimated in practice. Among these parameters, c_m and b_m ($\forall m$) are related to resource consumption, ρ, β, and δ are related to the loss function characteristics. These parameters are estimated in real time during the learning process.

The values of c_m and b_m ($\forall m$) are estimated based on measurements of resource consumptions at the edge nodes and the aggregator (Line 22 of Algorithm 2). The estimation depends on the type of resource under consideration. For example, when the type-m resource is energy, the sum energy consumption (per local update) at all nodes is considered as c_m; when the type-m resource is time, the maximum computation time (per local update) at all nodes is considered as c_m. The aggregator also monitors the total resource consumption of each resource type m based on the estimates, and compares the total resource consumption against the resource budget R_m (Line 24 of Algorithm 2). If the consumed resource is at the budget limit for some m, it stops the learning and returns the final result.

The values of ρ, β, and δ are estimated based on the local and global losses and gradients computed at $\mathbf{w}(t)$ and $\mathbf{w}_i(t)$, see Line 11 and Lines 17–19 of Algorithm 2 and Lines 6, 7, and 17 of Algorithm 3. To perform the estimation, each edge node needs to have access to both its local model parameter $\mathbf{w}_i(t)$ and the global model parameter $\mathbf{w}(t)$ for the same iteration t (see Lines 6 and 7 of Algorithm 3), which is only possible when global aggregation is performed in iteration t. Because $\mathbf{w}(t)$ is only observable by each node after global aggregation, estimated values of ρ, β, and δ are only available for recomputing τ^* starting from the second global aggregation step after initialization, which uses estimates obtained in the previous global aggregation step[4].

Remark: In the extreme case where $\mathbf{w}_i(t) = \mathbf{w}(t)$ in Lines 6 and 7 of Algorithm 3, we estimate $\hat{\rho}_i$ and $\hat{\beta}_i$ as zero. When

[4]See the condition in Line 10 of Algorithm 2 and Lines 5 and 16 of Algorithm 3. Also note that the parameters $\hat{\rho}_i$, $\hat{\beta}_i$, $F_i(\mathbf{w}(t_0))$, $\nabla F_i(\mathbf{w}(t_0))$ sent in Line 17 of Algorithm 3 are obtained at the previous global aggregation step (t_0, $\hat{\rho}_i$, and $\hat{\beta}_i$ are obtained in Lines 4–7 of Algorithm 3).

Algorithm 2: Procedure at the aggregator

```
Input: Resource budget $R$, control parameter $\varphi$, search range
    parameter $\gamma$, maximum $\tau$ value $\tau_{\max}$
Output: $\mathbf{w}^{\mathrm{f}}$
1  Initialize $\tau^* \leftarrow 1$, $t \leftarrow 0$, $s \leftarrow 0$;   //$s$ is a resource counter
2  Initialize $\mathbf{w}(0)$ as a constant or a random vector;
3  Initialize $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(0)$;
4  repeat
5      Send $\mathbf{w}(t)$ and $\tau^*$ to all edge nodes, also send STOP if it is set;
6      $t_0 \leftarrow t$;   //Save iteration index of last transmission of $\mathbf{w}(t)$
7      $t \leftarrow t + \tau^*$;   //Next global aggregation is after $\tau$ iterations
8      Receive $\mathbf{w}_i(t)$, $\hat{c}_i$ from each node $i$;
9      Compute $\mathbf{w}(t)$ according to (5);
10     if $t_0 > 0$ then
11         Receive $\hat{\rho}_i$, $\hat{\beta}_i$, $F_i(\mathbf{w}(t_0))$, $\nabla F_i(\mathbf{w}(t_0))$ from each node $i$;
12         Compute $F(\mathbf{w}(t_0))$ according to (2)
13         if $F(\mathbf{w}(t_0)) < F(\mathbf{w}^{\mathrm{f}})$ then
14             $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(t_0)$;
15         if STOP flag is set then
16             break;   //Break out of the loop here if STOP is set
17         Estimate $\hat{\rho} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\rho}_i}{D}$;
18         Estimate $\hat{\beta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\beta}_i}{D}$;
19         Compute $\nabla F(\mathbf{w}(t_0)) \leftarrow \frac{\sum_{i=1}^{N} D_i \nabla F_i(\mathbf{w}(t_0))}{D}$, estimate
             $\hat{\delta}_i \leftarrow \|\nabla F_i(\mathbf{w}(t_0)) - \nabla F(\mathbf{w}(t_0))\|$ for each $i$, from which
             we estimate $\hat{\delta} \leftarrow \frac{\sum_{i=1}^{N} D_i \hat{\delta}_i}{D}$;
20         Compute new value of $\tau^*$ according to (19) via linear search
             on integer values of $\tau$ within $[1, \tau_{\mathrm{m}}]$, where we set
             $\tau_{\mathrm{m}} \leftarrow \min\{\gamma\tau^*; \tau_{\max}\}$;
21     for $m = 1, 2, ..., M$ do
22         Estimate resource consumptions $\hat{c}_m$, $\hat{b}_m$, using $\hat{c}_{m,i}$ received
             from all nodes $i$ and local measurements at the aggregator;
23         $s_m \leftarrow s_m + \hat{c}_m\tau + \hat{b}_m$;
24     if $\exists m$ such that $s_m + \hat{c}_m(\tau + 1) + 2\hat{b}_m \geq R_m$ then
25         Decrease $\tau^*$ to the maximum possible value such that the
             estimated resource consumption for remaining iterations is
             within budget $R_m$ for all $m$, set STOP flag;
26 Send $\mathbf{w}(t)$ to all edge nodes;
27 Receive $F_i(\mathbf{w}(t))$ from each node $i$;
28 Compute $F(\mathbf{w}(t))$ according to (2)
29 if $F(\mathbf{w}(t)) < F(\mathbf{w}^{\mathrm{f}})$ then
30     $\mathbf{w}^{\mathrm{f}} \leftarrow \mathbf{w}(t)$;
```

$\delta = \beta = 0$ and $\frac{\delta}{\beta}$ in $h(\tau)$ is undefined, we define that $h(\tau) = 0$ for all $\tau \geq 1$. This is because for $t > 0$, $\mathbf{w}_i(t) = \mathbf{w}(t)$ only occurs when different nodes have extremely similar (often equal) datasets, in which case a large value of τ does not make the convergence worse than a small value of τ, thus it makes sense to define $h(\tau) = 0$ in this case.

The parameter η is the gradient-descent step size which is pre-specified and known. The remaining parameter φ includes ω which is non-straightforward to estimate because the algorithm does not know $\mathbf{w}^*$, thus we regard φ as a control parameter that is manually chosen and remains fixed for the same machine learning model[5]. Experimentation results presented in the next section show that a fixed value of φ works well across different data distributions, various numbers of nodes, and various resource consumptions/budgets. If we multiply both sides of (18) by φ, we can see that a larger value of φ gives a higher weight to the terms with $h(\tau)$, yielding a smaller value of τ^* (because $h(\tau)$ increases with τ), and vice versa. Therefore, in practice, it is not hard to tune the value of φ on a small and simple setup, which can then be applied to general cases. See also the results on the sensitivity of φ in Section VII-B6.

[5]Although φ is related to β and we estimate β separately, we found that it is good to keep φ a constant value that does not vary with the estimated value of β in practice, because there can be occasions where the estimated β is large causing $\varphi < 0$, which causes abnormal behavior when computing τ^* from $G(\tau)$.

Algorithm 3: Procedure at each edge node i

```
1  Initialize $t \leftarrow 0$;
2  repeat
3      Receive $\mathbf{w}(t)$ and new $\tau^*$ from aggregator, set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}(t)$;
4      $t_0 \leftarrow t$;   //Save iteration index of last transmission of $\mathbf{w}(t)$
5      if $t > 0$ then
6          Estimate $\hat{\rho}_i \leftarrow \|F_i(\mathbf{w}_i(t)) - F_i(\mathbf{w}(t))\| \,/\, \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$;
7          Estimate
             $\hat{\beta}_i \leftarrow \|\nabla F_i(\mathbf{w}_i(t)) - \nabla F_i(\mathbf{w}(t))\| \,/\, \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$;
8      for $\mu = 1, 2, ..., \tau^*$ do
9          $t \leftarrow t + 1$;   //Start of next iteration
10         Perform local update and obtain $\mathbf{w}_i(t)$ according to (4);
11         if $\mu < \tau^*$ then
12             Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}_i(t)$;
13     for $m = 1, 2, ..., M$ do
14         Estimate type-$m$ resource consumption $\hat{c}_{m,i}$ for one local
             update at node $i$;
15     Send $\mathbf{w}_i(t)$, $\hat{c}_{m,i}$ ($\forall m$) to aggregator;
16     if $t_0 > 0$ then
17         Send $\hat{\rho}_i$, $\hat{\beta}_i$, $F_i(\mathbf{w}(t_0))$, $\nabla F_i(\mathbf{w}(t_0))$ to aggregator;
18 until STOP flag is received;
19 Receive $\mathbf{w}(t)$ from aggregator;
20 Send $F_i(\mathbf{w}(t))$ to aggregator;
```

2) Recomputing τ^*: The value of τ^* is recomputed by the aggregator during each global aggregation step, based on the most updated parameter estimations. When searching for τ^*, we use the following search range instead of the range in Proposition 2 due to practical considerations of estimation error. As shown in Line 20 of Algorithm 2, we search for new values of τ^* up to γ times the current value of τ^*, and find τ^* that minimizes $G(\tau)$, where $\gamma > 0$ is a fixed parameter. The presence of γ limits the search space and also avoids τ^* from growing too quickly as initial parameter estimates may be inaccurate. We also impose a maximum value of τ, denoted by $\tau_{\max}$, because if τ^* is too large, it is more likely for the system to operate beyond the resource budget due to inaccuracies in the estimation of local resource consumption, see Line 24 of Algorithm 2. The new value of τ^* is sent to each node together with $\mathbf{w}(t)$ (Line 5 of Algorithm 2).

3) Distributed Gradient Descent: The local update steps of distributed gradient descent at the edge node include Lines 8–12 of Algorithm 3, where Line 10 of Algorithm 3 corresponds to Line 3 of Algorithm 1 and Line 12 of Algorithm 3 corresponds to Line 8 of Algorithm 1. When global aggregation is performed, Line 9 of Algorithm 2 computes the global model parameter $\mathbf{w}(t)$ at the aggregator, which is sent to the edge nodes in Line 5 of Algorithm 2, and each edge node receives $\mathbf{w}(t)$ in Line 3 of Algorithm 3 and sets $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}(t)$ to use $\mathbf{w}(t)$ as the initial model parameter for the next round of local update; this corresponds to Line 5 of Algorithm 1.

The final model parameter $\mathbf{w}^{\mathrm{f}}$ that minimizes $F(\mathbf{w})$ is obtained at the aggregator in Lines 13–14 of Algorithm 2, corresponding to Line 6 of Algorithm 1. As discussed in Section IV, the computation of $\mathbf{w}^{\mathrm{f}}$ lags for one round of global aggregation, because for any iteration t_0 that includes a global aggregation step, $F(\mathbf{w}(t_0))$ can only be computed after each edge node has received $\mathbf{w}(t_0)$ and sent the local loss $F_i(\mathbf{w}(t_0))$ to the aggregator in the next round of global aggregation. To take into account the final value of $\mathbf{w}(t)$ in the computation of $\mathbf{w}^{\mathrm{f}}$, Lines 26–30 of Algorithm 2 and Lines 19–20 of Algorithm 3 perform an additional round of computation of the loss and $\mathbf{w}^{\mathrm{f}}$, as also discussed in Section IV.

Overall, when global aggregation is executed for K times in total, the computational complexity of Algorithm 2 is $O(K(NM + \tau_{\max}))$, because each global aggregation step includes the computation of global parameters from the local parameters collected from N different nodes for M resource types and the linear search step in Line 20 of Algorithm 2 which has at most $\tau_{\max}$ steps. When T steps of local updates are performed in total, Algorithm 3 has a computational complexity of $O(T + KM)$, where the additional term KM corresponds to the additional local processing (at each node) in global aggregation steps.

C. Extension to Stochastic Gradient Descent

When the amount of training data is large, it is usually computationally prohibitive to compute the gradient of the loss function defined on the entire (local) dataset. In such cases, stochastic gradient descent (SGD) is often used [6], [7], [37], which uses the gradient computed on the loss function defined on a randomly sampled subset (referred to as a mini-batch) of data to approximate the real gradient. Although the theoretical analysis in this paper is based on deterministic gradient descent (DGD), the proposed approach can be directly extended to SGD. As discussed in [39], SGD can be seen as an approximation to DGD.

When using SGD with our proposed algorithm, all losses and their gradients are computed on mini-batches. Each local iteration step corresponds to a step of gradient descent where the gradient is computed on a mini-batch of local training data. The mini-batch changes for every step of local iteration, i.e., for each new local iteration, a new mini-batch of a given size is randomly selected from the local training data. However, to reduce errors introduced by random data sampling when estimating the parameters ρ, β, and δ, the first iteration after global aggregation uses the same mini-batch as the last iteration before global aggregation. When $\tau = 1$, the mini-batch changes if the same mini-batch has already been used in two iterations, to ensure that different mini-batches are used for training over time.

To avoid approximation errors caused by mini-batch sampling when determining $\mathbf{w}^{\mathrm{f}}$, when using SGD, the aggregator informs the edge nodes whether the current $\mathbf{w}(t_0)$ is selected as $\mathbf{w}^{\mathrm{f}}$ using an additional flag sent together with the message in Line 5 of Algorithm 2. The edge nodes save their own copies of $\mathbf{w}^{\mathrm{f}}$. When an edge node computes $F_i(\mathbf{w}(t_0))$ that is sent in Line 17 of Algorithm 3, it also recomputes $F_i(\mathbf{w}^{\mathrm{f}})$ using the same mini-batch as for computing $F_i(\mathbf{w}(t_0))$. It then sends both $F_i(\mathbf{w}^{\mathrm{f}})$ and $F_i(\mathbf{w}(t_0))$ to the aggregator in Line 17 of Algorithm 3. The aggregator recomputes $F(\mathbf{w}^{\mathrm{f}})$ based on the most recently received $F_i(\mathbf{w}^{\mathrm{f}})$. In this way, the values of $F(\mathbf{w}^{\mathrm{f}})$ and $F(\mathbf{w}(t_0))$ used for the comparison in Lines 13 and 29 of Algorithm 2 are computed on the same mini-batch at each edge node.

VII. Experimentation Results

A. Setup

To evaluate the performance of our proposed adaptive federated learning algorithm, we conducted experiments both on networked prototype system with 5 nodes and in a simulated environment with the number of nodes varying from 5 to 500. The prototype system consists of three Raspberry Pi (version 3) devices and two laptop computers, which are all interconnected via Wi-Fi in an office building. This represents an edge computing environment where the computational capabilities of edge nodes are heterogeneous. All these 5 nodes have local datasets on which model training is conducted. The aggregator is located on one of the laptop computers, and hence co-located with one of the local datasets.

1) Resource Definition: For ease of presentation and interpretation of results, we let $M = 1$ and consider time as the single resource type in our experiments. For the prototype system, we train each model for a fixed amount of time budget. The values of c and b (we omit the subscript $m = 1$ for simplicity) correspond to the actual time used for each local update and global aggregation, respectively. The simulation environment performs model training with simulated resource consumptions, which are randomly generated according to Gaussian distribution with mean and standard deviation values (see [38, Appendix E] for these values) obtained from measurements of the squared-SVM model on the prototype. See Section VII-A4 below for definitions of models and datasets.

2) Baselines: We compare with the following baseline approaches:

(a) Centralized gradient descent [6], [7], where the entire training dataset is stored on a single edge node and the model is trained directly on that node using a standard (centralized) gradient descent procedure;
(b) Canonical federated learning approach presented in [9], which is equivalent to using a fixed (non-adaptive) value of τ in our setting;
(c) Synchronous distributed gradient descent [17], which is equivalent to fixing $\tau = 1$ in our setting.

For a fair comparison, we implement the estimation of resource consumptions for all baselines and the training stops when we have reached the resource (time) budget. When conducting experiments on the prototype system, the centralized gradient descent is performed on a Raspberry Pi device. To avoid resource consumption related to loss computation, centralized gradient descent uses the last model parameter $\mathbf{w}(T)$ (instead of $\mathbf{w}^{\mathrm{f}}$) as the result, because convergence of $\mathbf{w}(T)$ can be proven in the centralized case [40]. We do not explicitly distinguish the baselines (b) and (c) above because they both correspond to an approach with non-adaptive τ of a certain value. When τ is non-adaptive, we use the same protocol as in Algorithms 2 and 3, but remove any parts related to parameter estimation and recomputation of τ.

3) DGD and SGD: We consider both DGD and SGD in the experiments to evaluate the general applicability of the proposed algorithm. For SGD, the mini-batch sampling uses the same initial random seed at all nodes, which means that when the datasets at all nodes are identical, the mini-batches at all nodes are also identical in the same iteration (while they are generally different across different iterations). This setup is for a better consideration of the differences between equal and non-equal data distributions (see Section VII-A5 below).

4) Models and Datasets: We evaluate the training of four different models on five different datasets, which represent a large variety of both small and large models and datasets, as one can expect all these variants to exist in edge computing scenarios. The models include squared-SVM, linear regression, K-means, and deep convolutional neural networks (CNN)[6]. See Table I for a summary of the loss functions of these models, and see [6], [7], [37] for more details. Among them, the loss functions for squared-SVM (which we refer to as SVM in short in the following) and linear regression satisfy Assumption 1, whereas the loss functions for K-means and CNN are non-convex and thus do not satisfy Assumption 1.

SVM is trained on the original MNIST dataset (referred to as *MNIST-O*) [43], which contains gray-scale images of $70,000$ handwritten digits ($60,000$ for training and $10,000$ for testing). The SVM outputs a binary label that corresponds to whether the digit is even or odd. We consider both DGD and SGD variants of SVM. The DGD variant only uses $1,000$ training and $1,000$ testing data samples out of the entire dataset in each simulation round, because DGD cannot process a large amount of data. The SGD variant uses the entire MNIST dataset.

Linear regression is performed with SGD on the energy dataset [44], which contains $19,735$ records of measurements from multiple sensors and the energy consumptions of appliances and lights. The model learns to predict the appliance energy consumption from sensor measurements.

K-means is performed with DGD on the user knowledge modeling dataset [45], which has 403 samples each with 5 attributes summarizing the user interaction with a web environment. The samples can be grouped into 4 clusters representing different knowledge levels, but we assume that we do not have prior knowledge of this grouping.

CNN is trained using SGD on three different datasets, including MNIST-O as described above, the fashion MNIST dataset (referred to as *MNIST-F*) which has the same format as MNIST-O but includes images of fashion items instead of digits [46], and the CIFAR-10 dataset which includes $60,000$ color images ($50,000$ for training and $10,000$ for testing) of 10 different types of objects [47]. A separate CNN model is trained on each dataset, to perform multi-class classification among the 10 different labels in the dataset.

5) Data Distribution at Different Nodes (Cases 1–4): For the distributed settings, we consider four different ways of distributing the data into different nodes. In *Case 1*, each data sample is randomly assigned to a node, thus each node has uniform (but not full) information. In *Case 2*, all the data samples in each node have the same label[7]. This represents the case where each node has non-uniform information, because the entire dataset has samples with multiple different labels. In *Case 3*, each node has the entire dataset (thus full information). In *Case 4*, data samples with the first half of the labels are distributed to the first half of the nodes as in Case 1; the other samples are distributed to the second half of the nodes as in Case 2. This represents a combined uniform and non-uniform case. For datasets that do not have ground truth labels, such the energy dataset used with linear regression, the data to node assignment is based on labels generated from an unsupervised clustering approach.

6) Training and Control Parameters: In all our experiments, we set the search range parameter $\gamma = 10$, the maximum τ value $\tau_{\max} = 100$. Unless otherwise specified, we set the control parameter $\varphi = 0.025$ for SVM, linear regression, and K-means, and $\varphi = 5 \times 10^{-5}$ for CNN. The gradient descent step size is $\eta = 0.01$. The resource (time) budget is set as $R = 15$ seconds unless otherwise specified. Except for the instantaneous results in Section VII-B5, the average results of 15 independent experiment/simulation runs are shown.

B. Results

1) Loss and Accuracy Values: In our first set of experiments, the SVM, linear regression, and K-means models were trained on the prototype system. Due to the resource limitation of Raspberry Pi devices, the CNN model was trained in a simulated environment of 5 nodes, with resource consumptions generated in the way described in Section VII-A1.

We compare the loss function values of our proposed algorithm (with adaptive τ) to baseline approaches, and also compare the classification accuracies for the SVM and CNN classifiers. The results are shown in Fig. 4. We note that *the proposed approach only has one data point (represented by a single marker in the figure) in each case*, because the value of τ is adaptive in this case and the marker location shows the average τ^* with the corresponding loss or accuracy. The centralized case also only has one data point but we show a flat line across different values of τ for the ease of comparison. We see that the proposed approach performs close to the optimal point for all cases and all models[8]. We also see that the (empirically) optimal value of τ is different for different cases and models, so a fixed value of τ does not work well for all cases. In some cases, the distributed approach can perform better than the centralized approach, because for a given amount of time budget, federated learning is able to make use of the computation resource at multiple nodes. For DGD approaches, Case 3 does not perform as well as Case 1,

[6]The CNN has 9 layers with the following structure: $5 \times 5 \times 32$ Convolutional $\rightarrow$ 2×2 MaxPool $\rightarrow$ Local Response Normalization $\rightarrow$ $5 \times 5 \times 32$ Convolutional $\rightarrow$ Local Response Normalization $\rightarrow$ 2×2 MaxPool $\rightarrow$ $z \times 256$ Fully connected $\rightarrow$ 256×10 Fully connected $\rightarrow$ Softmax, where z depends on the input image size and $z = 1568$ for MNIST-O and MNIST-F and $z = 2048$ for CIFAR-10. This configuration is similar to what is suggested in the TensorFlow tutorial [42].

[7]When there are more labels than nodes, each node may have data with more than one label, but the number of labels at each node is no more than the total number of labels divided by the total number of nodes rounded to the next integer.

[8]Note that the loss and accuracy values shown in Fig. 4 can be improved if we allow a longer training time. For example, the accuracy of CNN on MNIST data can become close to 1.0 if we allow a long enough time for training. The goal of our experiments here is to show that our proposed approach can operate close to the optimal point with a *fixed and limited* amount of training time (resource budget) as defined in Section VII-A6.

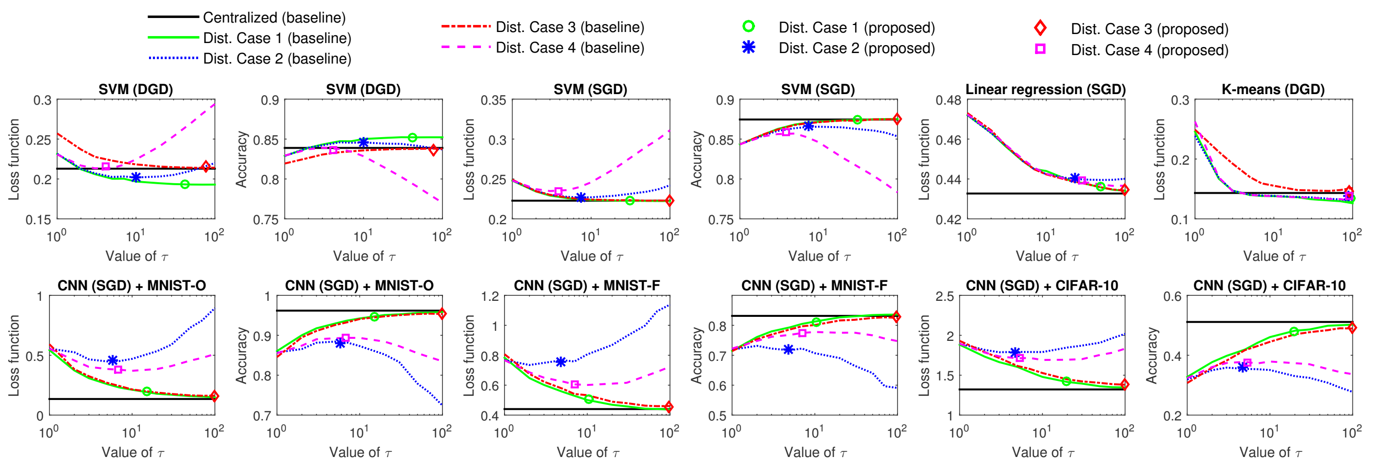
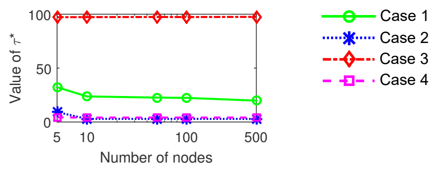
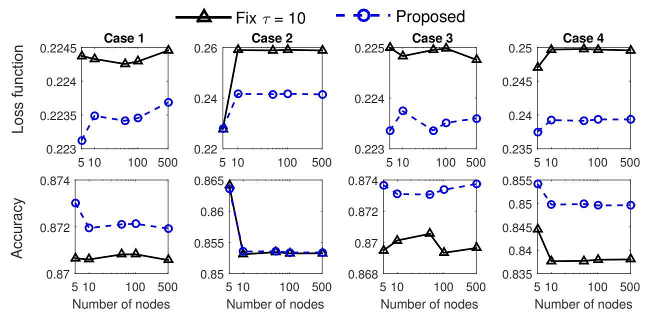


Fig. 4: Loss function values and classification accuracy with different τ. Only SVM and CNN classifiers have accuracy values. The curves show the results from the baseline with different fixed values of τ. Our proposed solution (represented by a single marker for each case) gives an average τ and loss/accuracy that is close to the optimum in all cases.

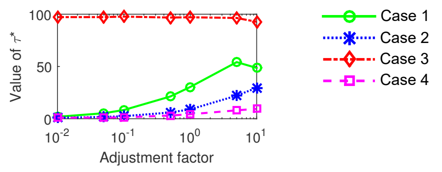


(a) τ^* in proposed algorithm

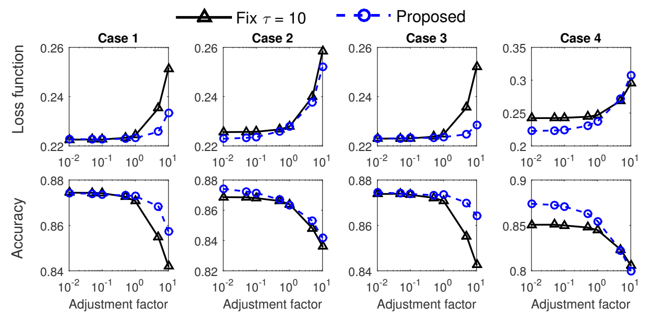


(b) Loss function values and classification accuracy

Fig. 5: SVM (SGD) with different numbers of nodes.



(a) τ^* in proposed algorithm



(b) Loss function values and classification accuracy

Fig. 6: SVM (SGD) with different global aggregation times.

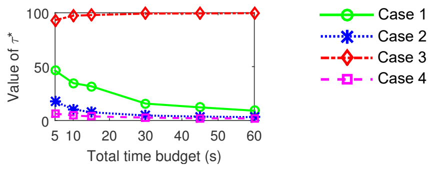
because the amount of data at each node in Case 3 is larger than that in Case 1, and DGD processes the entire amount of data thus Case 3 requires more resource for each local update.

Due to the high complexity of evaluating CNN models and the fact that linear regression and K-means models do not provide accuracy values, we focus on the SVM model in the following and provide further insights on the system.

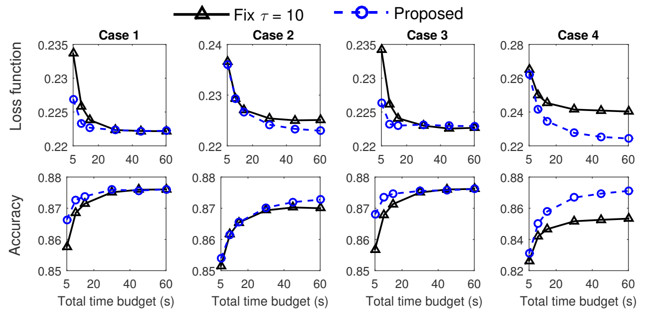
2) Varying Number of Nodes: Results of SVM (SGD) for the number of nodes varying from 5 to 500 are shown in Fig. 5, which are obtained in the simulated environment. Our proposed approach performs better than or similar to the fixed $\tau = 10$ baseline in all cases, where we choose fixed $\tau = 10$ as the baseline in this and the following evaluations because it is empirically a good value for non-adaptive τ in different cases according to the results in Fig. 4.

3) Varying Global Aggregation Time: To study the impact of different resource consumption (time) for global aggregation, we modify the simulation environment so that the global aggregation time is scaled by an *adjustment factor*. The actual time of global aggregation is equal to the original global aggregation time multiplied by the adjustment factor, thus a small adjustment factor corresponds to a small global aggregation time. The results for SVM (SGD) are shown in Fig. 6. Additional results for SVM (DGD) are included in [38, Appendix F]. We can see that as one would intuitively expect, a larger global aggregation time generally results in a larger τ^* for the proposed algorithm, because when it takes more time to perform global aggregation, the system should perform global aggregation less frequently, to make the best use of available time (resource). The fact that τ^* slightly decreases when the adjustment factor is large is because in this case, the global aggregation time is so large that only a few rounds of global aggregation can be performed before reaching the resource budget, and the value of τ^* will be decreased in the last round to remain within the resource budget (see Line 25 of Algorithm 2). Comparing to the fixed $\tau = 10$ baseline, the proposed algorithm performs better in (almost) all cases.

4) Varying Total Time Budget: We evaluate the impact of the total time (resource) budget on the prototype system. Results for SVM (SGD) are shown in Fig. 7. Further results for SVM (DGD) are included in [38, Appendix G]. We see that



(a) τ^* in proposed algorithm



(b) Loss function values and classification accuracy

Fig. 7: SVM (SGD) with different total time budgets.

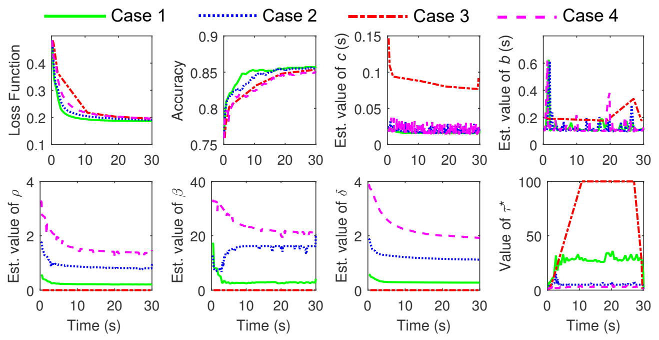


Fig. 8: Instantaneous results of SVM (DGD) with the proposed algorithm.

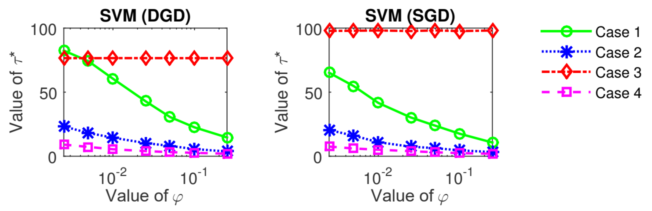


Fig. 9: Impact of φ on the average value of τ^* in the proposed algorithm.

except for Case 3 where all nodes have the same dataset, the value of τ^* of the proposed algorithm decreases with the total time budget. This aligns with the discussion in Section VI-A that τ^* becomes close to one when the resource budget is large enough. We also see that the proposed algorithm performs better than or similar to the fixed $\tau = 10$ baseline in all cases.

5) *Instantaneous Behavior:* We further study the instantaneous behavior of our system for a single run of 30 seconds (for each case) on the prototype system. Results for SVM (DGD) is shown in Fig. 8. Further results for SVM (SGD) are available in [38, Appendix H]. We see that the value of τ^* remains stable after an initial adaptation period, showing that the control algorithm is stable. The value of τ^* decreases at the end due to adjustment caused by the system reaching the resource budget (see Line 25 of Algorithm 2). As expected, the gradient deviation δ is larger for Cases 2 and 4 where the data samples at different nodes are non-uniform. The same is observed for ρ and β, indicating that the model parameter $\mathbf{w}$ is in a less smooth region for Cases 2 and 4. In Case 3, the data at different nodes are equal so we always have $\mathbf{w}_i(t) = \mathbf{w}(t)$ regardless of whether global aggregation is performed in iteration t. Thus, the estimated ρ and β values are zero by definition, as explained in the remark in Section VI-B1. Case 3 of SVM (DGD) has a much larger value of c because it processes more data than in other cases and thus takes more time, as explained before. The value of b exhibits fluctuations because of the randomness of the wireless channel.

6) *Sensitivity of φ:* The sensitivity of the control parameter φ evaluated on the prototype system is shown in Fig. 9. We see that the relationship among τ^* in different cases is mostly maintained with different values of φ. The value of τ^* decreases approximately linearly with $\log \varphi$, which is consistent with the fact that there is an exponential term w.r.t. τ in $h(\tau)$ (and thus $G(\tau)$). For Case 3, τ^* remains the same with different φ, because $h(\tau) = 0$ in this case by definition (see the remark in Section VI-B1) and the value of φ does not affect τ^*, as $G(\tau) \propto \frac{1}{\varphi}$ independently of τ in this case according to (18). We also see that small changes of φ does not change τ^* much, indicating that one can take big steps when tuning φ in practice and the tuning is not difficult.

7) *Comparison to Asynchronous Distributed Gradient Descent:* Asynchronous gradient descent [17] is an alternative to the typically used synchronous gradient descent in federated learning. With asynchronous gradient descent, the edge nodes operate in an asynchronous manner. Each edge node pulls the most up-to-date model parameter from the aggregator, computes the gradient on its local dataset, then sends the gradient back to the aggregator. The aggregator performs gradient descent according to the step size η weighted by the dataset sizes of each node, similar to the combination of (4) and (5). The process repeats until the training finishes. Asynchronous gradient descent is able to fully utilize the available computational resource at each node by running more gradient descent steps at more powerful (faster) nodes. However, the asynchronism may hurt the overall performance.

It was shown in [17] that synchronous gradient descent has benefits over asynchronous gradient descent in a datacenter setting. Here, we study their differences in the edge computing setting with heterogeneous resources (laptops and Raspberry Pis in our experiment) and different data distributions (Cases 1–4). The results for DGD and SGD with SVM are shown in Figs. 10 and 11, respectively. We see that the performance of asynchronous gradient descent is much worse than synchronous gradient descent for non-uniform data distribution in Cases 2 and 4, with slower convergence, sudden changes (indicating instability of the training process), and convergence to higher loss and lower accuracy values. This is because the model tends overfit the datasets on the faster nodes, as many more steps of gradient descent are performed on these nodes compared to the slower nodes. With uniform data distribution (Cases 1 and 3), asynchronous gradient descent performs similar as or slightly better than synchronous gradient descent, because when the datasets at different nodes are similar (Case 1) or equal (Case 3), there is not much harm caused by overfitting the data on the faster nodes.

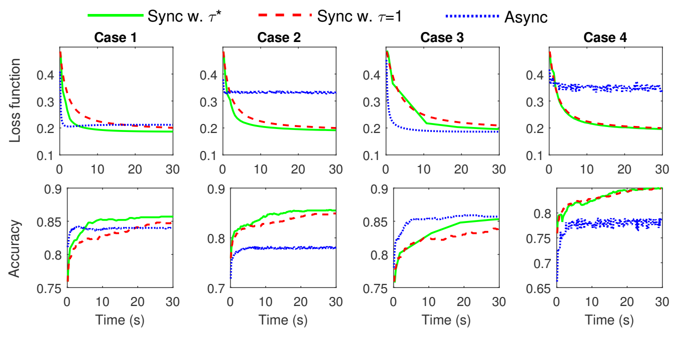


Fig. 10: Synchronous vs. asynchronous distributed DGD with SVM.

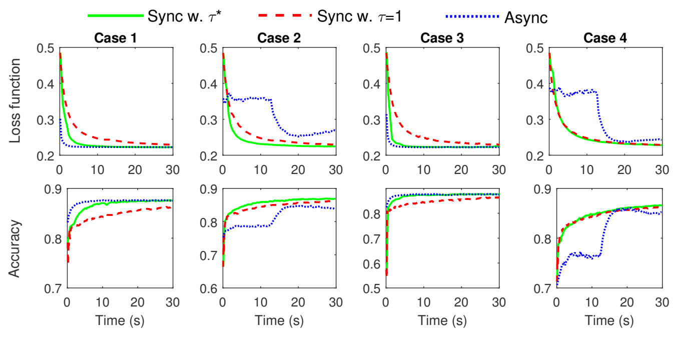


Fig. 11: Synchronous vs. asynchronous distributed SGD with SVM.

Considering the overall performance in all Cases 1–4, we can conclude that it is still better to perform federated learning with synchronous gradient descent as we do throughout this paper. However, how to make more efficient use of heterogeneous resources is something worth investigating in the future.

VIII. Conclusion

In this paper, we have focused on gradient-descent based federated learning that include local update and global aggregation steps. Each step of local update and global aggregation consumes resources. We have analyzed the convergence bound for federated learning with non-i.i.d. data distributions. Using this theoretical bound, a control algorithm has been proposed to achieve the desirable trade-off between local update and global aggregation in order to minimize the loss function under a resource budget constraint. Extensive experimentation results confirm the effectiveness of our proposed algorithm. Future work can investigate how to make the most efficient use of heterogeneous resources for distributed learning, as well as the theoretical convergence analysis of some form of non-convex loss functions representing deep neural networks.

References

[1] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM*, Apr. 2018.

[2] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[3] R. Kelly, "Internet of Things data to top 1.6 zettabytes by 2020," Apr. 2015. [Online]. Available: https://campustechnology.com/articles/2015/04/15/internet-of-things-data-to-top-1-6-zettabytes-by-2020.aspx

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[8] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Apr. 2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[9] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2016.

[10] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless network intelligence at the edge," *arXiv*, Dec. 2018. [Online]. Available: http://arxiv.org/abs/1812.02858

[11] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM*, May 2017, pp. 1–9.

[12] L. Tong and W. Gao, "Application-aware traffic scheduling for workload offloading in mobile clouds," in *IEEE INFOCOM*, Apr. 2016.

[13] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM*, Apr. 2016.

[14] H. Tan, Z. Han, X.-Y. Li, and F. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM*, May 2017.

[15] L. Wang, L. Jiao, J. Li, and M. Muhlhauser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *IEEE ICDCS*, June 2017, pp. 1281–1290.

[16] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

[17] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *ICLR Workshop Track*, 2016.

[18] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[19] J. Konen, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop on Private Multi-Party Machine Learning*, 2016. [Online]. Available: https://arxiv.org/abs/1610.05492

[20] C. Hardy, E. Le Merrer, and B. Sericola, "Distributed deep learning on edge-devices: feasibility via adaptive compression," in *Network Computing and Applications (NCA), 2017 IEEE 16th International Symposium on*. IEEE, 2017, pp. 1–8.

[21] J. Konen, H. B. McMahan, D. Ramage, and P. Richtarik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016. [Online]. Available: https://arxiv.org/abs/1610.02527

[22] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," *arXiv preprint arXiv:1804.08333*, 2018.

[23] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[24] 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 *OSDI*, vol. 14, 2014, pp. 583–598.

[25] J. Wang and G. Joshi, "Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD," in *SysML*, Mar.–Apr. 2019. [Online]. Available: http://arxiv.org/abs/1810.08313

[26] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 629–647.

[27] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[28] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization," in *Advances in Neural Information Processing Systems*, 2015, pp. 2737–2745.

[29] S. Zheng, Q. Meng, T. Wang, W. Chen, N. Yu, Z.-M. Ma, and T.-Y. Liu, "Asynchronous stochastic gradient descent with delay compensation," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W.

Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 4120–4129.

[30] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[31] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," *arXiv preprint arXiv:1710.06952*, 2017.

[32] J. Wang and G. Joshi, "Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms," *arXiv*, Jan. 2019. [Online]. Available: http://arxiv.org/abs/1808.07576

[33] H. Yu, S. Yang, and S. Zhu, "Parallel restarted SGD with faster convergence and less communication: Demystifying why model averaging works for deep learning," in *AAAI Conference on Artificial Intelligence*, Jan.–Feb. 2019.

[34] Y. Zhang, M. J. Wainwright, and J. C. Duchi, "Communication-efficient algorithms for statistical optimization," in *Advances in Neural Information Processing Systems*, 2012, pp. 1502–1510.

[35] Y. Arjevani and O. Shamir, "Communication complexity of distributed convex learning and optimization," in *Advances in neural information processing systems*, 2015, pp. 1756–1764.

[36] C. Ma, J. Konečnỳ, M. Jaggi, V. Smith, M. I. Jordan, P. Richtárik, and M. Takáč, "Distributed optimization with arbitrary local solvers," *Optimization Methods and Software*, vol. 32, no. 4, pp. 813–848, 2017.

[37] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[38] 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," Tech. Rep., 2019. [Online]. Available: https://arxiv.org/abs/1804.05271

[39] T. Tuor, S. Wang, K. K. Leung, and K. Chan, "Distributed machine learning in coalition environments: overview of techniques," in *21st International Conference on Information Fusion*, Jul. 2018.

[40] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and trends in Machine Learning*, vol. 8, no. 3-4, 2015.

[41] F. Topsok, "Some bounds for the logarithmic function," *Inequality theory and applications*, vol. 4, 2006.

[42] "Advanced convolutional neural networks." [Online]. Available: https://www.tensorflow.org/tutorials/images/deep%5Fcnn

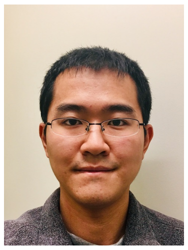
[43] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] L. M. Candanedo, V. Feldheim, and D. Deramaix, "Data driven prediction models of energy use of appliances in a low-energy house," *Energy and Buildings*, vol. 140, pp. 81 – 97, 2017.

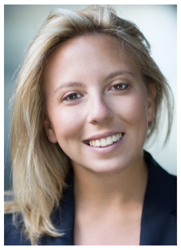
[45] H. Kahraman, S.Sagiroglu, and I.Colak, "Developing intuitive knowledge classifier and modeling of users' domain dependent data in web," *Knowledge Based Systems*, vol. 37, pp. 283–295, 2013.

[46] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[47] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.



Shiqiang Wang (S'13-M'15) received his Ph.D. from the Department of Electrical and Electronic Engineering, Imperial College London, United Kingdom, in 2015. Before that, he received his master's and bachelor's degrees at Northeastern University, China, in 2011 and 2009, respectively. He joined IBM T. J. Watson Research Center in 2016 as a Research Staff Member, where he was also a Graduate-level Co-op in the summers of 2014 and 2013. In the fall of 2012, he was at NEC Laboratories Europe, Heidelberg, Germany. His current research focuses on theoretical and practical aspects of mobile edge computing, cloud computing, and machine learning. He has over 40 scholarly publications. Dr. Wang currently serves as an associate editor of IEEE Access and a technical program committee (TPC) member of IEEE ICDCS 2019 and IFIP Networking 2019. In the past, he served as a TPC member of several international conferences including IEEE GLOBECOM, IEEE ICC, IEEE VTC, and as a reviewer for a number of international journals and conferences. He received multiple Invention Achievement Awards from IBM since 2016, and the Best Student Paper Award of the Network and Information Sciences International Technology Alliance (NIS-ITA) in 2015. He was recognized as an exemplary reviewer of the IEEE Transactions on Communications in 2017.



Tiffany Tuor received her B.S. degree in Communication Systems from the Swiss Federal Institute of Technology of Lausanne (EPFL) in 2014. She obtained an M.S degree in Financial Engineering in 2015 and an M.S. degree in Communications and Signal Processing in 2016 from Imperial College London, UK. Currently, she is working towards a Ph.D. degree in the department of Electrical and Electronic Engineering of Imperial College London. In summer 2017, she interned at IBM T. J. Watson Research Center in New York. Her research interest includes machine learning, cloud computing, and various aspects of future communication networks.

Theodoros Salonidis (S'98-M'04-SM'17) received the Diploma in electronic and computer engineering from the Technical University of Crete, Chania, Greece, in 1997, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, MD, USA, in 1999 and 2004, respectively. He is a Research Staff Member with the IBM T.J. Watson Research Center, Yorktown Heights, NY, USA. He was a Post-doctoral Researcher with Rice University, Houston, TX, USA from 2004 to 2006, and a Researcher with Intel Research, Cambridge, U.K., in 2006 and Thomson/ Technicolor, Paris, France, from 2007 to 2012. His current research interests are in the areas of automated AI, distributed analytics, machine learning, and performance analysis, design, and implementation of mobile and cloud computing systems. Dr. Salonidis is a senior member of IEEE and the Technical Chamber of Greece.

Kin K. Leung (F'01) received his B.S. degree from the Chinese University of Hong Kong in 1980, and his M.S. and Ph.D. degrees from University of California, Los Angeles, in 1982 and 1985, respectively. He joined AT&T Bell Labs in New Jersey in 1986 and worked at its successors, AT&T Labs and Lucent Technologies Bell Labs, until 2004. Since then, he has been the Tanaka Chair Professor in the Electrical and Electronic Engineering (EEE), and Computing Departments at Imperial College in London. He is the Head of Communications and Signal Processing Group in the EEE Department. His current research focuses on protocols, optimization and modeling of wireless networks and computer systems. He also works on multi-antenna and cross-layer designs for wireless networks. He received the Distinguished Member of Technical Staff Award from AT&T Bell Labs (1994), and was a co-recipient of the Lanchester Prize Honorable Mention Award (1997). He was elected an IEEE Fellow (2001), received the Royal Society Wolfson Research Merits Award (2004-09) and became a member of Academia Europaea (2012). Along with his co-authors, he received several best paper awards, including the IEEE PIMRC 2012 and ICDCS 2013. He served as a member (2009-11) and the chairman (2012-15) of the IEEE Fellow Evaluation Committee for ComSoc. He has served as a guest editor for the IEEE JSAC, IEEE Wireless Communications and the MONET journal, and as an editor for the JSAC: Wireless Series, IEEE Trans. on Wireless Communications and IEEE Trans. on Communications. Currently, he is an editor for the ACM Computing Survey and International Journal on Sensor Networks.

Christian Makaya is currently a senior research scientist at HP Labs. Previously, he was a research staff member at IBM T. J. Watson Research Center and a senior research scientist at Telcordia Technologies. His work has been a catalyst behind several new initiatives and technologies resulting in delivery of high-value capabilities to products and services. For his technical contributions, he has been recognized by several high-prestige awards by IBM and Telcordia. Dr. Makaya leads several technical research activities in the areas of distributed computing systems and big data analytics with the mission of delivering deep technical breakthroughs and technology transfer. The focus of his current research interests is on edge computing, machine learning, artificial intelligence, network intelligence, Internet of Things (IoT), network functions virtualization, policy-based management systems, and cyber-security. He has authored numerous technical papers in peer-reviewed journals and conferences. He has several filed and issued patents. Dr. Makaya received his Ph.D. in computer engineering from Polytechnique Montreal, University of Montreal. He serves on the Industry Outreach Board of IEEE Communication Society (ComSoc) and as a member of technical program committees of various conferences. He served as the co-chair of IEEE Young Professionals for the IEEE Princeton/Central Jersey Section.

Ting He (S'04-M'07-SM'13) received the B.S. degree in computer science from Peking University, China, in 2003 and the Ph.D. degree in electrical and computer engineering from Cornell University, Ithaca, NY, in 2007. She is an Associate Professor in the School of Electrical Engineering and Computer Science at Pennsylvania State University, University Park, PA. Between 2007 and 2016, she was a Research Staff Member in the Network Analytics Research Group at the IBM T. J. Watson Research Center, Yorktown Heights, NY. Her work is in the broad areas of network modeling and optimization, statistical inference, and information theory. Dr. He is a senior member of IEEE. She is an Associate Editor for IEEE Transactions on Communications (2017-2020) and IEEE/ACM Transactions on Networking (2017-2019). She was the Membership co-chair of ACM N2Women in 2013-2014 and was listed in N2Women: Rising Stars in Networking and Communications in 2017. She has served on the TPC of a range of communications and networking conferences, including IEEE INFOCOM (Distinguished TPC Member), IEEE ICNP, IEEE SECON, IEEE WiOpt, IEEE/ACM IWQoS, IEEE MILCOM, IEEE ICNC, IFIP Networking, etc. She received the Research Division Award and the Outstanding Contributor Awards from IBM in 2016, 2013, and 2009. She received the Most Collaboratively Complete Publications Award by ITA in 2015, the Best Paper Award at the 2013 International Conference on Distributed Computing Systems (ICDCS), a Best Paper Nomination at the 2013 Internet Measurement Conference (IMC), and the Best Student Paper Award at the 2005 International Conference on Acoustic, Speech and Signal Processing (ICASSP). Her students received the Outstanding Student Paper Award at the 2015 ACM SIGMETRICS and the Best Student Paper Award at the 2013 ITA Annual Fall Meeting.

Kevin Chan (S'02-M'09-SM'18) received his B.S. in Electrical and Computer Engineering and Engineering & Public Policy from Carnegie Mellon University, Pittsburgh, PA in 2001 and his M.S.E.C.E and PhD in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, GA in 2003 and 2008, respectively. Dr. Chan is currently a research scientist with the Computational and Information Sciences Directorate at the U.S. Army Combat Capabilities Development Command Army Research Laboratory (Adelphi, MD). Dr. Chan is actively involved in research on network science, distributed analytics and cybersecurity. He has received multiple best paper awards and the NATO Scientific Achievement Award. He currently is the co-editor for the IEEE Communications Magazine Military Communications and Networks Series.