When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning

***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 rate 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.**

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 [1]. 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 [2]. Local data storing and processing with global coordination is made possible by the emerging technology of mobile edge computing (MEC) [3], [4], 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.

To analyze large amounts of data and obtain useful information for the detection, classification, and prediction of future events, machine learning techniques are usually 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 [5], [6]. 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

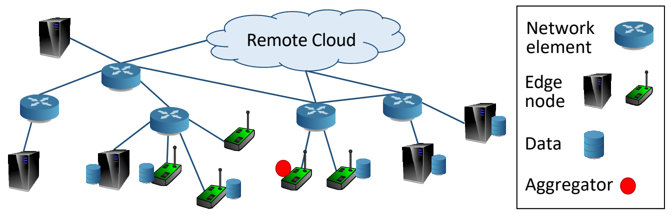


Fig. 1: System architecture.

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 best 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.

We focus on gradient-descent based 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 in order to minimize the loss function defined on its own dataset. It also includes *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 one of the edge nodes. The aggregator aggregates these parameters (often 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, and one can perform the global aggregation 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.

A. Related Work

Existing work on MEC focuses on generic applications, where solutions have been proposed for application offloading [7], [8], workload scheduling [9], [10], and service migration triggered by user mobility [11], [12]. 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 learning tasks.

Distributed machine learning based on gradient descent has been studied from a theoretical perspective in [13], [14], [15], where asymptotic bounds on the training convergence and communication cost are obtained. The results were then extended to general classes of distributed learning approaches in [16]. To some extent, these results characterize the communication and computation trade-off. However, none of them consider the adaptation of global aggregation frequency. Some of the analysis also have unrealistic assumptions such as independent and identically distributed (i.i.d.) data at different nodes [13], [14], whereas the more general case involving non-i.i.d. data distributions is much harder to analyze. From the practical perspective, [17] proposed a variant of distributed gradient descent where the global aggregation is performed in a synchronous manner. Experiments using various datasets confirmed the effectiveness of this approach. The global aggregation frequency is fixed in [17]. It does not provide any theoretical guarantees and the experiments were not conducted in a network setting.

In contrast to the above research, our work in this paper addresses the problem of dynamically determining the global aggregation frequency based on real-time observations during the distributed learning process. 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 and the real-time dynamics of the system.

B. Our Contributions

Our main contributions in this paper are as follows:

1) We analyze the convergence rate of gradient-descent based distributed learning from a theoretical perspective, and obtain a novel convergence bound that incorporates non-i.i.d. data distributions and an arbitrary number of local updates between two global aggregations.

2) Using the above theoretical result, 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.

TABLE I: Loss functions for popular machine learning models

Model	Loss function $f(\mathbf{w}, \mathbf{x}_j, y_j)$ ($\triangleq f_j(\mathbf{w})$)
Smooth 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 [6]

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

II. 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. 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].

Examples of loss functions of popular machine learning models are summarized[2] in Table I. 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

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

For convenience, 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

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

which is equal to the weighted average of the local losses at each edge node i given in (1).

[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.

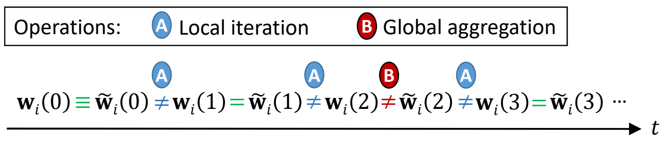

Operations:
A Local iteration
B Global aggregation
A
A
B
A
$\mathbf{w}_i(0) \equiv \widetilde{\mathbf{w}}_i(0) \neq \mathbf{w}_i(1) = \widetilde{\mathbf{w}}_i(1) \neq \mathbf{w}_i(2) \neq \widetilde{\mathbf{w}}_i(2) \neq \mathbf{w}_i(3) = \widetilde{\mathbf{w}}_i(3) \cdots$
t

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

B. The Learning Problem

The learning problem is to find

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

Due to the inherent complexity of most machine learning models, it is often 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 the state-of-the-art [17]. 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, based on the parameter value in the previous iteration $t - 1$. This gradient-descent step on the local loss function at each node is referred to as the *local update*. After one or multiple 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 followed by a possible global aggregation.

After the 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 $\widetilde{\mathbf{w}}_i(t)$ and $\mathbf{w}_i(t)$ are not equal and we set $\widetilde{\mathbf{w}}_i(t) = \mathbf{w}(t)$, where $\mathbf{w}(t)$ is 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:

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

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

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

The value of $\mathbf{w}(t)$ is *only observable to 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 iterations. For simplicity, we assume that T is an integer multiple of τ, which will be relaxed when we discuss practical aspects in Section V-B. The distributed gradient descent algorithm is presented in Algorithm 1, which ignores details such as the communication between the aggregator and edge nodes. Such implementation details will be discussed later in Section V-B.

Algorithm 1: Distributed gradient descent

```
Input: $\tau$, $T$
Output: $\mathbf{w}(T)$
1 Initialize $\mathbf{w}_i(0)$ 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         (Global aggregation) Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}(t)$ for all $i$, where $\mathbf{w}(t)$
           is defined in (5);
6     else
7         (No global aggregation) Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}_i(t)$ for all $i$;
```

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 becomes the same as 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. See Appendix A in [18] for details.

III. 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 distributed 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 $F(\mathbf{w})$ is minimized subject to a given resource constraint for this learning task.

Formally, we define that each step of lo$\underline{\text{c}}$al update at *all* participating nodes consumes c units of resource, and each step of glo$\underline{\text{b}}$al aggregation consumes b units of resource, where $c \geq 0$ and $b \geq 0$ are both real numbers. For given T and τ, the total amount of consumed resource is then $T\left(c + \frac{b}{\tau}\right)$. Let R denote the total resource budget. We seek the solution to the following problem:

$$\begin{aligned} \min_{\tau, T} \quad & F(\mathbf{w}(T)) \\ \text{s.t.} \quad & T\left(c + \frac{b}{\tau}\right) \leq R \end{aligned} \tag{6}$$

Note that we do not distinguish among different types of resources in the formulation. As mentioned earlier, the resource can be time, energy, monetary cost or a combination of them.

To solve (6), we need to find out how the values of τ and T affect the loss function (after T iterations) $F(\mathbf{w}(T))$. It

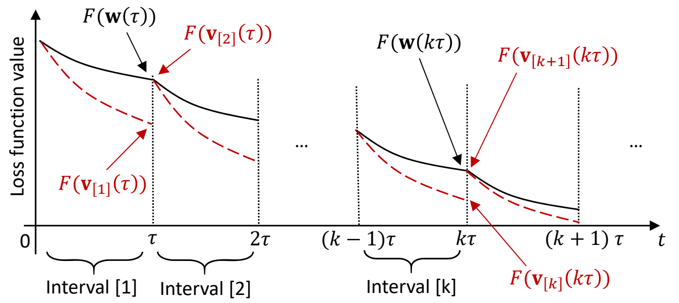


Fig. 3: Illustration of definitions in different intervals.

is generally impossible to find an exact analytical expression to relate τ and T with $F(\mathbf{w}(T))$, because it depends on the convergence property of gradient descent (for which only upper/lower bounds are known [19]) and the impact of the global aggregation frequency. Further, the resource consumptions c and b can be time-varying in practice which makes the problem even more challenging than (6) alone.

We analyze the convergence rate upper bound of distributed gradient descent (Algorithm 1) in Section IV. We then use this bound as an approximation to solve (6) and propose a control algorithm for adaptively choosing the best values of τ and T to achieve near-optimal resource utilization in Section V.

IV. Convergence Analysis

We analyze the convergence of Algorithm 1 in this section and find an upper bound of $F(\mathbf{w}(T)) - F(\mathbf{w}^*)$, where we recall that T is the total number of iterations. To facilitate the analysis, we first introduce some notations.

A. Definitions

We use K to denote the total number of global aggregations within T iterations. Because we assumed that T is an integer multiple of τ, we have $K = \frac{T}{\tau}$. Further, 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 notation $[k]$ to denote the iteration interval $[(k-1)\tau, k\tau]$, 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

$$\mathbf{v}_{[k]}(t) = \mathbf{v}_{[k]}(t-1) - \eta \nabla F(\mathbf{v}_{[k]}(t-1)) \tag{7}$$

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 the 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 rate of $\mathbf{v}_{[k]}(t)$ within each interval $[k]$ to obtain the convergence rate 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) *$F_i(\mathbf{w})$ is β-smooth, i.e., $\|\nabla F_i(\mathbf{w}) - \nabla F_i(\mathbf{w}')\| \leq \beta\|\mathbf{w} - \mathbf{w}'\|$ for any $\mathbf{w}, \mathbf{w}'$*

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

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

Proof. This is straightforward 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.,*

$$\|\nabla F_i(\mathbf{w}) - \nabla F(\mathbf{w})\| \leq \delta_i \tag{8}$$

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*

$$\left\|\mathbf{w}(t) - \mathbf{v}_{[k]}(t)\right\| \leq h(t - (k-1)\tau) \tag{9}$$

where

$$h(x) \triangleq \frac{\delta}{\beta}\left((\eta\beta + 1)^x - 1\right) - \eta\delta x \tag{10}$$

for any $x > 0$. 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. See Appendix B in [18]. □

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 $x \geq 0$ due to Bernoulli's inequality. Substituting this into (10) confirms that we always have $h(x) \geq 0$.

It is easy to see that we have $h(0) = h(1) = 0$. Therefore, when $t = (k-1)\tau$, i.e., at the beginning of the interval $[k]$, the upper bound in (9) is zero. This is consistent with the definition of $\mathbf{v}_{[k]}((k-1)\tau) = \mathbf{w}((k-1)\tau)$ for any k.

When $t = 1 + (k-1)\tau$, the upper bound in (9) is also zero. This agrees with the discussion in Section II-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 $\tau = 1$, we have that $t - (k-1)\tau$ is either 0 or 1 for any interval $[k]$ and $t \in [k]$. Hence, the upper bound in (9) 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 (10) 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 (10)), 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 have the following theorem.

Theorem 2. *The convergence upper bound of Algorithm 1 after T iterations is given by*

$$F(\mathbf{w}(T)) - F(\mathbf{w}^*) \leq \frac{1}{T\left(\omega\eta\left(1 - \frac{\beta\eta}{2}\right) - \frac{\rho h(\tau)}{\tau\varepsilon^2}\right)} \tag{11}$$

for some $\varepsilon > 0$, when the following conditions are satisfied:

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

where we define $\omega \triangleq \min_k \frac{1}{\left\|(\mathbf{v}_{[k]}((k-1)\tau) - \mathbf{w}^*)\right\|^2}$

Proof. See Appendix C in [18]. □

In the above theorem, condition 1 says that the step size should be small enough in order to guarantee convergence. A larger step size is allowed for a smoother function (i.e., with smaller β). When condition 1 holds, condition 2 always holds for $\tau = 1$, because $h(1) = 0$. When $\tau > 1$, it holds for small enough $h(\tau)$, implying that in order to guarantee convergence, the gap in (9) needs to be small. This means that the global aggregation needs to be performed frequently enough so that the local gradients do not deviate too much from the global gradient, otherwise the algorithm may not converge. The value of ε together with conditions 3 and 4 specify lower bounds on the optimality gap. When $\tau > 1$ and ε is small, i.e., when the loss function is close to its optimal value, the term with $\frac{\rho h(\tau)}{\tau\varepsilon^2}$ is large and the bound becomes loose. When ε is below a certain threshold, condition 2 does not hold anymore. Because $F\left(\mathbf{w}(T)\right) - F(\mathbf{w}^*)$ generally becomes smaller when T gets larger, yielding a smaller ε, intuitively, this means that when $\tau > 1$, convergence is only guaranteed to a non-zero optimality gap as $T \to \infty$. Convergence to zero optimality gap is guaranteed with $\tau = 1$ (because $h(1) = 0$ in which case the bound does not depend on ε). Instead of focusing on the $T \to \infty$ case, we consider limited resource budget and thus finite T in this paper. Therefore, in most of our considered problem scenarios, the values of $\mathbf{w}(T)$ and $\mathbf{v}_{[k]}(t)$ are not "too close" to the optimum and there exists a value of ε such that all the conditions in Theorem 2 hold.

Because a global aggregation step is performed at iteration T, the value of $\mathbf{w}(T)$ is observable to the system and it is the final result of distributed gradient descent. We also note that the bound in (11) has no restriction on how the data is distributed at different nodes.

V. Control Algorithm

We propose an algorithm that approximately solves (6) in this section. We first assume that the resource consumptions c and b are known and solve for the values of τ and T. Then, we consider practical scenarios where c, b, 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 (6)

We use the upper bound in (11) as an approximation of $F(\mathbf{w}(T)) - F(\mathbf{w}^*)$. Because for a given global loss function $F(\mathbf{w})$, its minimum value $F(\mathbf{w}^*)$ is a constant, the minimization of $F(\mathbf{w}(T))$ in (6) is equivalent to minimizing $F(\mathbf{w}(T)) - F(\mathbf{w}^*)$. With this approximation and rearranging the inequality constraint in (6), we can rewrite (6) as

$$\min_{\tau,T} \quad \frac{1}{T\left(\omega\eta\left(1 - \frac{\beta\eta}{2}\right) - \frac{\rho h(\tau)}{\tau\varepsilon^2}\right)} \tag{12}$$
$$\text{s.t.} \quad T \leq \frac{R\tau}{c\tau + b} \text{ and conditions in Theorem 2}$$

In the following, we assume that η and ε are chosen small enough so that conditions 1, 3, and 4 in Theorem 2 are satisfied. When condition 2 holds, the denominator in (12) is positive, and thus, the optimal value of T is $T = \frac{R\tau}{c\tau+b}$. Substituting this into (12) and taking the reciprocal of the objective function, we have

$$\max_{\tau=1,2,3,\ldots} \quad \frac{R\tau}{c\tau + b}\left(\omega\eta\left(1 - \frac{\beta\eta}{2}\right) - \frac{\rho h(\tau)}{\tau\varepsilon^2}\right) \tag{13}$$
$$\text{s.t.} \quad \text{Conditions 1, 3, 4 in Theorem 2}$$

The optimal solution to (13) always satisfies condition 2 in Theorem 2, because $\tau = 1$ (in which case $h(\tau) = 0$) yields the objective function to be positive and thus satisfying the condition. Dividing the objective function in (13) by $\frac{R\omega}{c}$ yields

$$G(\tau) \triangleq \frac{\tau}{\tau + a}\left(\eta\left(1 - \frac{\beta\eta}{2}\right) - \frac{\rho h(\tau)}{\tau\varepsilon^2\omega}\right) \tag{14}$$

where $a \triangleq \frac{b}{c}$ is the relative resource consumption of global aggregation normalized by that of local update.

Based on the above reasoning, we know that the optimal τ that minimizes (12) and maximizes (13) and (14) are the same. Therefore, we can solve

$$\tau^* = \arg\max_{\tau} G(\tau) \tag{15}$$

from which we can also obtain $T^* = \frac{R\tau^*}{c\tau^*+b}$. The following proposition shows that $G(\tau)$ has a unique maximum.

Proposition 1. *When $\eta \leq \frac{1}{\beta}$, $G(\tau)$ is a strictly concave function for $\tau \geq 1$.*

Proof. See Appendix D in [18]. □

The equation $\frac{dG(\tau)}{d\tau} = 0$ has no closed-form solution and hence there is no closed-form expression for τ^*. Because τ takes integer values, we can solve for τ^* using a binary search procedure similar to that in [12] which has a complexity of $O(\log \tau_{\max})$, where $\tau_{\max}$ is the maximum value of τ in the search which will be discussed further in the next subsection.

B. Dynamic Control Algorithm

The expression of $G(\tau)$ (which includes $h(\tau)$) has parameters which may be unknown in practice. Among these parameters, c and b (and thus a) are related to the resource consumption, β and δ are related to the loss function characteristics. These four parameters are estimated in real time during the learning process, which will be further explained later. The parameter η is the gradient-descent step size which is pre-specified and known. The remaining parameters ρ, ε, and ω all appear in a single term in the definition of $G(\tau)$ (see (14)), and we define $\varphi \triangleq \frac{\rho}{\varepsilon^2\omega}$ as a control parameter that is manually chosen and remains fixed for the same machine learning model. Experimentation results presented in the next section show that a fixed value of φ works well across different data distributions and various numbers of nodes. We note that a larger value of φ amplifies the gradient divergence δ (which is proportional to $h(\tau)$) and yields a smaller value of τ^*, and vice versa. Thus, 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 (also see the results on the sensitivity of φ in Section VI-B4). We are also not concerned about the conditions in Theorem 2 in the practical implementation.

As mentioned earlier, the local updates run on edge nodes and the global aggregation is performed through the assistance of an aggregator (which is a logical component that may also run on one of the edge nodes). The procedures at the aggregator and each edge node are presented in Algorithms 2 and 3, respectively, where Lines 7–11 of Algorithm 3 are for local updates and the rest is considered as part of global aggregation or initialization. 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.

The value of τ is recomputed during each global aggregation step, based on the most updated parameter estimations. As shown in Line 17 of Algorithm 2, we search for new values of τ up to γ times the current value of τ, and find the value that maximizes $G(\tau)$, where $\gamma > 0$ is a fixed value. The presence of γ limits the search space and also avoids τ from growing too quickly as initial parameter estimates may be inaccurate. The new value of τ is sent to each node together with $\mathbf{w}(t)$ (Line 4 of Algorithm 2).

Algorithm 2: Procedure at the aggregator

```
Input: Resource budget $R$, control parameter $\varphi$, search range param. $\gamma$
Output: $\mathbf{w}(T)$
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  repeat
4      Send $\mathbf{w}(t)$ and $\tau$ to all edge nodes, also send STOP if it is set;
5      $t_0 \leftarrow t$;   //Save iteration index of last transmission of $\mathbf{w}(t)$
6      $t \leftarrow t + \tau$;   //Next global aggregation is after $\tau$ iterations
7      Receive $\mathbf{w}_i(t)$, $\hat{c}_i$ from each node $i$;
8      Compute $\mathbf{w}(t)$ according to (5);
9      if STOP then
10         return $\mathbf{w}(t)$ as the final result;
11     Estimate resource consumptions $\hat{c}$ and $\hat{b}$ according to received $\hat{c}_i$
         from all nodes $i$ and local measurements at the aggregator;
12     $s \leftarrow s + \hat{c}\tau + \hat{b}$;
13     if $t_0 > 0$ then
14         Receive $\hat{\beta}_i$, $\nabla F_i(\mathbf{w}(t_0))$ from each node $i$;
15         Estimate $\hat{\beta} \leftarrow \frac{\sum_{i=1}^N D_i \hat{\beta}_i}{D}$;
16         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 $\hat{\delta} \leftarrow \frac{\sum_{i=1}^N D_i \hat{\delta}_i}{D}$;
17         Compute new value of $\tau$ according to (15) via binary search
             on integer values within $[1, \tau_{\max}]$, where we set $\tau_{\max} \leftarrow \gamma\tau$;
18     if $s + \hat{c}\tau + \hat{b} \geq R$ then
19         Decrease $\tau$ so that the estimated resource consumption for
             remaining iterations is within budget $R$, set STOP;
```

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
             $\hat{\beta}_i \leftarrow \|\nabla F_i(\mathbf{w}_i(t)) - \nabla F_i(\mathbf{w}(t))\| / \|\mathbf{w}_i(t) - \mathbf{w}(t)\|$;
7      for $m = 1, 2, ..., \tau$ do
8          $t \leftarrow t + 1$;   //Start of next iteration
9          Perform local update and obtain $\mathbf{w}_i(t)$ according to (4);
10         if $m < \tau$ then
11             Set $\widetilde{\mathbf{w}}_i(t) \leftarrow \mathbf{w}_i(t)$;
12     Estimate resource consumption $\hat{c}_i$ for one local update at node $i$;
13     Send $\mathbf{w}_i(t)$, $\hat{c}_i$ to aggregator;
14     if $t_0 > 0$ then
15         Send $\hat{\beta}_i$, $\nabla F_i(\mathbf{w}(t_0))$ to aggregator;
16 until STOP is received;
```

The values of c and b are estimated based on measurements of resource consumptions at the edge nodes and the aggregator (Line 11 of Algorithm 2). The estimation method depends on the type of resource under consideration. For example, when the resource is energy, the sum energy consumption (per local update) at all nodes is considered as c; whereas when the resource is time, the maximum computation time (per local update) at all nodes is considered as c. The aggregator also monitors the total resource consumption based on the estimates, and compares the total resource consumption against the resource budget R (Line 18 of Algorithm 2). If the consumed resource is at the budget limit, it stops the learning and returns the final result.

The values of β and δ are estimated based on the local and global gradients computed at $\mathbf{w}(t)$ and $\mathbf{w}_i(t)$ (see Lines 14–16 of Algorithm 2 and Lines 6 and 15 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 Line 6 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 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[3].

VI. Experimentation Results

A. Setup

To evaluate the performance of our proposed control algorithm, we conducted a large number of 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 (described below) 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.

We consider time as the resource in our experiments. For the prototype system, we train each model for a fixed amount of time budget. The values of c and b 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 set as the average values of the measurements from the prototype.

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 [5], [6], which uses the gradient computed on the loss function defined on a randomly sampled subset of data to approximate the real gradient. Although the theoretical analysis in this paper is based on deterministic gradient descent (DGD), we consider both DGD and SGD in the experiments to evaluate the general applicability of the proposed algorithm.

1) Models and Datasets: We evaluate the training of four different models on three different datasets. The models include smooth SVM, linear regression, K-means, and deep convolutional neural networks (CNN)[4]. See Table I for a summary of the loss functions of these models, and see [5], [6] for more details. Among them, the loss functions for smooth-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 and CNN are trained on the MNIST dataset [20], which contains $70,000$ handwritten digits ($60,000$ for training and $10,000$ for testing). For SVM, we use even and odd digits as a binary label, and we use multi-class labels of 10 different digits for CNN. The SVM training via DGD only uses $1,000$ training and $1,000$ testing data samples out of the entire dataset, because DGD cannot process a large amount of data. The SGD variant of SVM and CNN uses the entire MNIST dataset. Linear regression is performed on Facebook metrics dataset [21], which has 500 samples with multiple attributes related to posts published on a cosmetics brand. The model finds a linear relationship between the total interaction number and all other attributes. K-means is performed on the user knowledge modeling dataset [22], 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.

2) Data Distribution at Different Nodes (Cases 1–4): 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 unbiased (but not full) information. In *Case 2*, all the data samples in each node have the same label[5]. This represents the case where each node has biased 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 as in Case 1; the other samples are distributed as in Case 2. This represents a combined biased and unbiased case.

3) Training and Control Parameters: In all our experiments, we set the search range parameter $\gamma = 10$. Unless otherwise specified, we set the control parameter $\varphi = 0.2$ for SVM, linear regression, and K-means, and we set $\varphi = 10^{-4}$ for CNN. The gradient descent step size is $\eta = 0.01$ for SVM, linear regression, and CNN, and $\eta = 0.1$ for K-means.

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, where the resource (time) budget for each model training instance is fixed to 15 seconds. Due to the resource limitation of Raspberry Pi devices, the CNN model was trained in a simulated environment of 5 nodes, with a total budget equivalent to 200 local updates, and a global aggregation consuming $a = 5.0$ times the resource of one local update. The value $a = 5.0$ was obtained from the time measurements of SVM (SGD) on the prototype.

[3]See the condition in Line 13 of Algorithm 2 and Lines 5 and 14 of Algorithm 3. Also note that the parameters $\hat{\beta}_i$ and $\nabla F_i(\mathbf{w}(t_0))$ sent in Line 15 of Algorithm 3 are obtained at the previous global aggregation step (t_0 and $\hat{\beta}_i$ are obtained in Lines 4–6 of Algorithm 3).

[4]The CNN has 7 layers with the following structure: $5 \times 5 \times 32$ Convolutional $\rightarrow 2 \times 2$ MaxPool $\rightarrow 5 \times 5 \times 32$ Convolutional $\rightarrow 2 \times 2$ MaxPool $\rightarrow 1568 \times 256$ Fully connected $\rightarrow 256 \times 10$ Fully connected $\rightarrow$ Softmax.

[5]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.

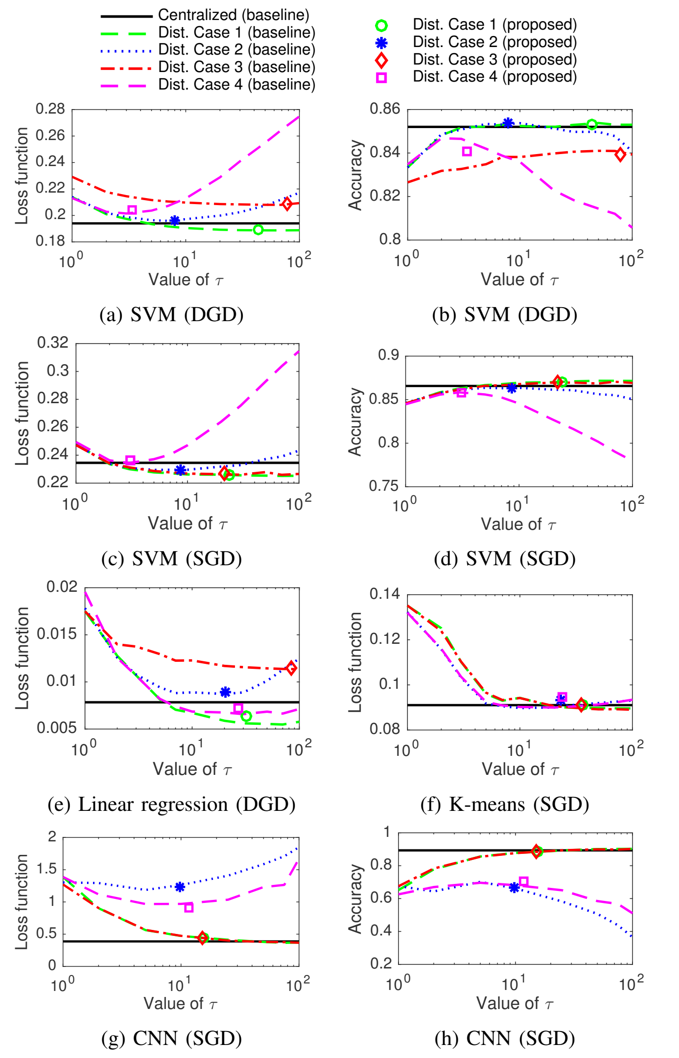


Fig. 4: Loss function values and classification accuracy with different τ. Only SVM and CNN classifiers have accuracy values. The curves show the results from the baseline with different fixed values of τ. Our proposed solution (represented by a single marker for each case) yields an average τ and loss/accuracy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

We compare the loss function values of our proposed algorithm (with adaptive τ) to baseline approaches that include centralized training and distributed training with fixed τ. We also compare the classification accuracies for the SVM and CNN classifiers. The centralized baseline is obtained in a simulated environment by running only local updates on a single node subject to the total resource budget, where the local update resource consumption is estimated based on measurements on the prototype system averaged over all cases of each model. The distributed baselines run on the prototype system (except for CNN as discussed earlier) under the same setup but with different fixed values of τ (i.e., no adaptation). Note that this setup with non-adaptive τ *is the same as the state-of-the-art approach in [17]*. When fixing $\tau = 1$, it is also equivalent to the synchronous SGD approach in [23].

The average results of 30 different experiments (15 for CNN) 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

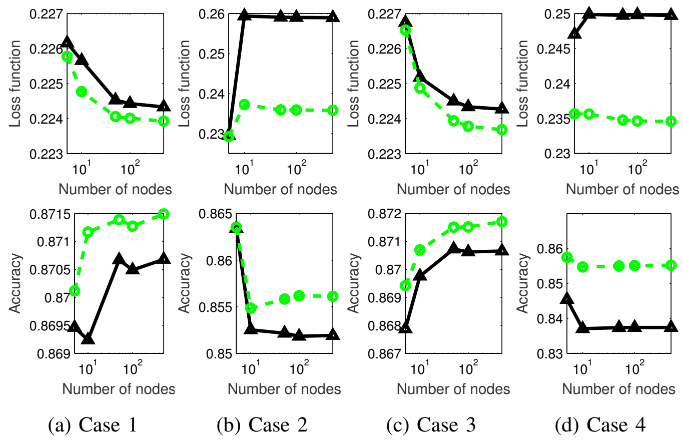


Fig. 5: Loss function values and classification accuracy with different numbers of nodes for SVM (SGD), where the solid lines with "△" markers correspond to fixed $\tau = 10$, and the dashed lines with "○" markers correspond to the proposed approach with adaptive τ.

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. We also see that the 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, distributed learning is able to make use of the computation resource at multiple nodes. It does not always perform better than centralized because the resource consumption of the centralized approach is estimated as the average over all distributed cases for each model, so some will perform worse. For DGD approaches, Case 3 does not perform as well as Case 1, 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 (time) for each local update.

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 with $a = 5.0$ and total budget equal to 868 local updates (both parameters are obtained from measurements on the prototype system). Our proposed approach outperforms the fixed $\tau = 10$ baseline in all cases.

3) Instantaneous Behavior: We further study the instantaneous behavior of our system. Results for SVM (DGD) is shown in Fig. 6 for a single run of 30 seconds (for each case) on the prototype system. Further results of SVM (SGD) are available in Appendix E in [18]. We see that except for Case 3 of SVM (DGD), the value of τ^* converges after a certain amount of time, showing that the control algorithm is stable. The value of τ^* keeps increasing in Case 3 of SVM (DGD) because all nodes have exactly the same data in this case and DGD uses all the data samples (i.e., no random sampling). There is no gradient deviation in this case and the optimal value of τ is infinity. As expected, the gradient deviation δ is

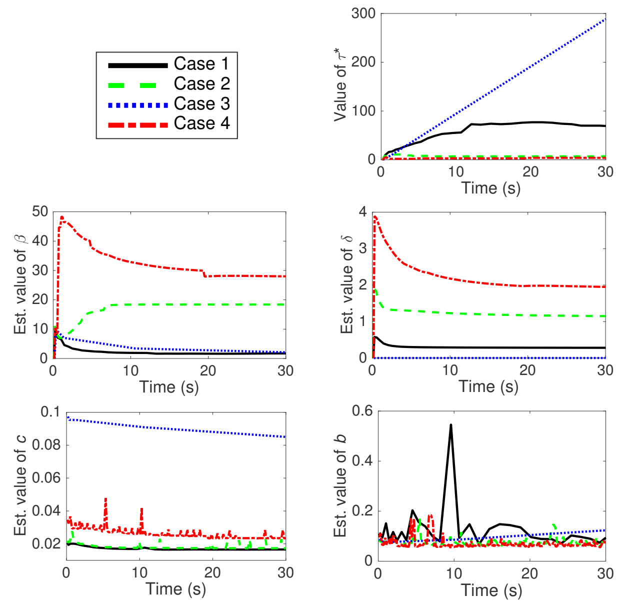
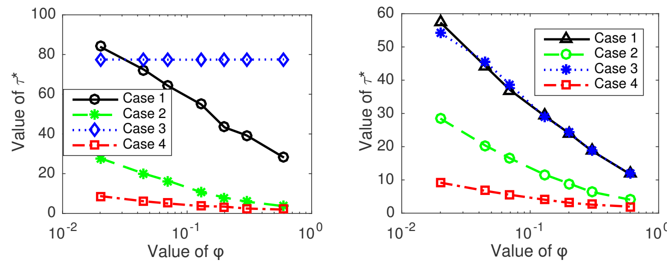


Fig. 6: Instantaneous results of SVM (DGD) with the proposed algorithm.



(a) SVM (DGD) (b) SVM (SGD)

Fig. 7: Impact of φ on the average value of τ^*.

larger for Cases 2 and 4 where the data samples at different nodes are biased. The same is observed for β, indicating that the model parameter $\mathbf{w}$ is in a less smooth region for Cases 2 and 4. 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.

4) *Sensitivity of φ:* The sensitivity of the control parameter φ is shown in Fig. 7, where the experimentation settings are the same as for Fig. 4. 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 of SVM (DGD), τ^* remains the same with different φ, because $\delta = 0$ (thus $h(\tau) = 0$) in this case and the value of φ does not affect $G(\tau)$ (see (14)). 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.

VII. Conclusion

In this paper, we have focused on gradient-descent based distributed 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 distributed 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.

References

[1] 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.

[2] 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

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *CoRR*, vol. abs/1701.01090, 2017. [Online]. Available: http://arxiv.org/abs/1701.01090

[4] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, 2017.

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

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

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

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

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

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

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

[12] 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.

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

[14] 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.

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

[16] C. Ma, J. Konečný, 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.

[17] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of AISTATS*, 2016.

[18] Tech. Rep. [Online]. Available: https://1drv.ms/b/s!AvVdJI6Go0c-alB5sxwLNM6Zw2c

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

[20] 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.

[21] S. Moro, P. Rita, and B. Vala, "Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach." *Journal of Business Research*, vol. 69, no. 9, pp. 3341–3351, 2016.

[22] 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.

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