FedMax: Enabling a Highly-Efficient Federated Learning Framework

[1]Haohang Xu,[1]Jin Li, [1]Hongkai Xiong, [2]Hui Lu
[1]*Shanghai Jiao Tong University, Shanghai, China*
[2]*Department of Computer Science, Binghamton University*
[1]Email: {xuhaohang,deserve_lj,xionghongkai}@sjtu.edu, [2]Email: huilu@binghamton.edu

***Abstract*—IoT devices produce a wealth of data desired for learning models to empower more intelligent applications. However, such data is often privacy sensitive making data owners reluctant upload their data to a central server for learning purposes. Federated learning provides a promising privacy-preserving learning approach, which decouples the model training from the need of accessing to the sensitive data. However, realizing a deployed, dependable federated learning system faces critical challenges, such as frequent dropouts of learning workers, heterogeneity of workers computation, and limited communication. In this paper, we focus on the systems aspects to advance federated learning and contribute a highly efficient and reliable distributed federated learning framework, FedMax, aiming to tackle these challenges. In designing FedMax, we contribute new techniques in light of the properties of a real federated learning setting, including a relaxed synchronization communication scheme and a similarity-based worker selection approach. We have implemented a prototype of FedMax and evaluated FedMax upon multiple popular machine learning models and datasets, showing that FedMax significantly increases the robustness of a federated learning system, speeds up the convergence rate by 25%, and increases the system efficiency by 50%, in comparison with state-of-the-art approaches.**

***Index Terms*—Federated learning, distributed systems, machine learning, privacy-preserving**

I. Introduction

Mobile phones, ubiquitous sensors, autonomous vehicles, and numerous Internet of Things (IoT) devices have greatly extended traditional cloud architecture by providing additional computing resources for a variety of low-latency IoT and real time applications [5]. Further, these devices have access to an unprecedented amount of data, and machine learning modelers may wish to train a model on this rich data to empower more intelligent applications. However, such data often contains sensitive information (e.g., private messages, photos and videos) making data owners reluctant upload their data to a central server for learning purposes [16].

To achieve privacy-preserving machine learning, *Federated Learning* [1] has been proposed, wherein a shared global model is trained under the coordination of a central server (e.g., in the cloud) that receives updates from participating device workers (e.g., at the edge). Each worker computes an update based on its local training dataset and only communicates this update to the global model maintained by the central server. Federated learning decouples the model training from the need of directly accessing to the training data, thus reducing the privacy and security risks significantly.

However, realizing a deployed, dependable distributed federated learning system faces many practical challenges: (1) As many IoT devices usually have limited power and/or access to network connectivity, it is common to see that workers only sporadically participate in the global model update; (2) Due to the heterogeneous nature of participating workers (e.g., datasets and computation capacity), a federated learning system would expect varying model update speeds from workers; (3) As IoT devices usually have a limited upload bandwidth (e.g., 1 MB/s or less), communication cost in federated learning mostly dominates the model optimization.

Unfortunately, existing distributed machine learning systems [2]–[4], [10]–[12] largely target a *data center* scenario, where training data is available and accessed in a centralized manner; non-trivial limitations exist in directly converting them to a federated learning setting. For example, in Figure 1, if we simply apply a distributed learning system [4] in a federated learning setting, the training model either converge slowly or even can not converge. It is because existing approaches assume the distribution of training data as independent and identically distributed (IID), while data in federated learning is commonly non-IID — the data on a given worker is not representative of the population distribution.

In this paper, we focus on the systems aspects to advance federated learning by contributing a new framework, called **FedMax**, for highly-efficient privacy-preserving machine learning in emerging cloud architecture. In designing FedMax, we especially make the following contributions:

- *Relaxed Worker Synchronization.* The ways how workers update their local models to the central server are important for both system efficiency and model convergence. Typically, a synchronous update scheme [1] leads to fast model convergence rate, while an asynchronous one [2] leads to high system efficiency — more data processed in the same time. To strike a good balance between system efficiency and model convergence meanwhile tolerating dropouts of sporadic workers, FedMax advances a *relaxed synchronization communication scheme* with a dynamic workload balancing mechanism allowing distributed, heterogeneous workers to update their models at close paces.
- *Similarity-based Worker Selection.* A federated learning system (randomly) selects a fraction of workers to participate in each communication round, not only because it

2159-6190/20/$31.00 ©2020 IEEE
DOI 10.1109/CLOUD49709.2020.00064

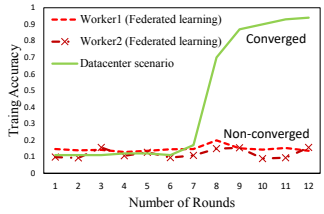

Fig. 1: The training model does not converge with two workers with non-IID data under MXNet in federated learning.

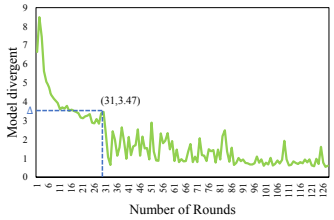

Fig. 2: FedMax Model divergence during training process with dataset CIFAR10 and model VGG.

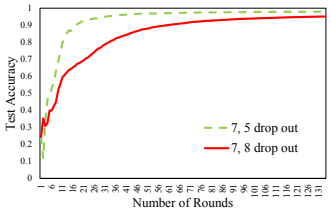

Fig. 3: Randomly drop out workers to reduce communication cost for a 8-worker scenario.

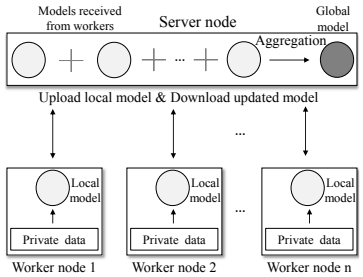

Fig. 4: An overview of federated learning.

can greatly reduce communication cost, but also because adding more workers beyond a certain point may expect diminishing returns [1]. To more explicitly identify participating workers for enhanced model convergence, FedMax introduces a *similarity-based worker selection approach*, which chooses the most effective workers with least dataset similarity (e.g., more non-IID). As data is not accessible by the central server in federated learning, FedMax smartly implies dataset similarity from workers' updates.

We have implemented a prototype of FedMax upon MXNet [4], a popular machine learning library. Our extensive evaluation based on popular neural network learning models with multiple datasets shows that FedMax addresses the above challenges successfully: For example, FedMax significantly accelerates the convergence rate by 25% and increases the system efficiency by 50%, in comparison with state-of-the-art federated learning solutions [1].

II. Background and Motivation

A. Federated Learning Background

As illustrated in Figure 4, to achieve privacy preserving machine learning for real-world data from IoT devices, federated learning [1] allows the worker of a device to compute a minimal update, such as stochastic gradient descent (SGD), with its local training dataset, and communicate this update to a central server. The central server keeps updating the global training model with all (or part of) these local updates. As workers only communicate the minimal updates, instead of the raw data, to the central server, federated learning reduces the attack surface significantly. Combined with other security techniques, such as differential privacy [17] and secure multi-party computation [16], federated learning can provide much stronger security guarantees.

The ways how workers update their local models to a central server are important for model convergence and system efficiency. We summarize three common synchronization controls used in state-of-the-art distributed machine learning systems [15]. (1) *Bounded Synchronous Parallel (BSP):* Workers perform a computation round followed by a synchronization phase where they exchange updates (e.g., with the central server in federated learning [1]). (2) *Asynchronous Parallel (ASP):* Opposite to BSP, ASP allows computations to execute as fast as possible by running workers completely asynchronously (i.e., the central server will not block individual workers). (3) *Stale Synchronous Parallel (SSP).* Instead of asking all workers to be on the same iteration, the system decides if a worker may proceed based on how far it falls behind the fastest worker (e.g., with a time τ delay).

B. Challenges

A distributed federated learning system faces several practical challenges such as dropouts, heterogeneity of computation, and limited communication. To investigate these challenges, we converted MXNet [4], a machine learning library with the parameter server support [2], to a federated learning setting via following the workflow in Figure 4. As data distribution in federated learning follows a non-IID pattern, we distribute an image dataset (MNIST [7]) to workers, each with different categories (i.e., emulating a non-IID scenario).

First, as any worker can *drop out* at any time in a federated learning environment (e.g., a mobile phone may be disconnected or run out of battery), BSP cannot be straightforwardly applied; otherwise the learning system will be blocked if any worker drops out. However, if we choose ASP, we may encounter a situation that the global training model does not converge at all. As shown in Figure 1, after several rounds of training, the model accuracy of federated learning (with two workers) keeps stable at around 10%. As the MNIST dataset we used for illustration has only ten categories, 10% accuracy is equivalent to random prediction. In contrast, in a traditional data center scenario, as training data is available and accessed in a centralized manner, each worker is usually assigned with a portion of data from all categories (i.e., IID) — a significant rise of model accuracy from 10% to 90% is observed in Figure 1. Intuitively, SSP seems a good fit in federated learning. Yet, we face another problem — workers on various IoT devices have different computation capacities and could generate data at varying speeds. As a result, the

update speed of different workers may vary a lot. For example, we conducted a simple test in a 8-worker cluster: We set the fastest worker with 5 CPUs, while the slowest worker with only 1 CPU; the waiting time ratio of the fastest worker is 74.1%, indicating that 74.1% time in a computation round is wasted for the fastest worker.

Moreover, in a federated learning system, it is not possible (or even necessary) for all workers to participate in each update round. However, the existing approach [1] which naively picks up random participating workers before each update round is far from enough, as we observe that different worker selection may lead to quite different model convergence. For example, in Figure 3, in the similar 8-worker cluster as above but with each worker having the same computation capability, the case that we drop out worker 7 & 5 yields a much better performance (in terms of model accuracy and convergence speed) than that we drop out worker 7 & 8.

III. Design of FedMax

A. FedMax Architecture Overview

To build a deployed, efficient federated learning framework, we introduce FedMax. As illustrated in Figure 5, FedMax first realizes the workflow of federated learning (Figure 4): A FedMax worker runs on each device participating in a federated learning system, mainly responsible for local model training and communicating the update to a central server. The central server [1] collects model updates from participating workers, applies such updates to the global model, and then communicates the updated model back to the participating workers for the next round update. Data is communicated between the workers and the central server via `push` and `pull` operations: Each worker pushes its update (e.g., model parameters or gradient) to the server after one training round, and then pulls the updated global model back. Workers also receive *control metadata* from the server for adjusting training behaviors (e.g., speed).

Further, to address the challenges highlighted in Section II-B, FedMax incorporates other two main components: (1) It realizes a relaxed synchronization communication scheme with a workload balancing mechanism by taking heterogeneous computation capacities of workers into account (Section III-B). (2) It involves a similarity-based worker selection approach, which picks up the most effective participating workers with least dataset similarity for model update (Section III-C).

B. FedMax's Model Synchronization

In Section II, we have demonstrated that none of the three typical synchronization controls can be directly applied to a federated learning setting.

1) Relaxed Synchronization: FedMax adopts a *relaxed synchronization* communication scheme, where FedMax's synchronizer, as shown in Figure 5, delays fast workers for a threshold time τ (i.e., waiting for slow workers). More specifically, in each round, once FedMax receives an update from the fastest worker, it waits (at most) time τ before starting the next round, during which s updates (out of total n workers) can be collected. After either receiving updates from all participating workers ($s = n$) or waiting time τ expires ($s < n$), FedMax applies these updates to the global model and returns the s workers for the next round. FedMax keeps receiving updates from other workers that miss the deadline (unless they lag too much). With this, FedMax ensures that s workers receive the most recently updated global model in a relaxed synchronization manner, while still reckoning the lagging workers in an asynchronous manner.

FedMax prefers to put more computation locally to reduce communication cost. To this, FedMax allows workers to process a certain amount of training data before communicating the update to the central server in each update round. More specifically, a large (local) training dataset is usually divided into multiple batches for processing — one batch is processed at a time. In one epoch, all batches are processed. FedMax can decide after how many batches/epochs, workers need to communicate with the central server for model update (i.e., one round), by sending control metadata to individual workers.

FedMax's global model update in round t for s updates (out of n workers) with time τ shows as follows:

$$\Delta W_t = \frac{1}{s}\sum_{i=1}^{s} \lambda(\Delta W_t^i)\Delta W_t^i \quad (1)$$

$$W_t = W_{t-1} + \Delta W_t \quad (2)$$

where W_t is the global model of round t; ΔW_t is the update to the global model; $\Delta W_t^i = W_t^i - W_{t-1}^i$ refers to the local model update of worker i in round t. Noticeably, FedMax introduces parameter λ — the learning rate — to address the model staleness issue (detailed below).

2) Model Staleness: Similar to an asynchronous update scheme, having workers run in a relaxed synchronization manner gives rise to the expense of workers (e.g., s workers within time τ) not seeing the global model updates by other lagging workers (e.g., the $n - s$ workers that fall behind) — namely model staleness.

If training data across workers is IID, the level of model staleness is low, as there are a lot "similarities" among updates from different workers. That is, even a worker misses several such updates, its local model is not far away from the current global model. Thus, in the IID case, an asynchronous or relaxed synchronization update scheme may still have as good model convergence as the synchronous case. However, if training data across workers is non-IID — common in federated learning — model staleness could be very high with an asynchronous or relaxed synchronization scheme, resulting in slow model convergence or even non-convergence. This explains what we have seen in Figure 1 in Section II.

To mitigate the negative impact of model staleness, a scale factor λ is involved in Formula (1). We have investigated how λ affects model convergence with a wide range of settings. In Figure 6 and Figure 7, we demonstrate such impact using

[1] A federated server can consist of a group of nodes, each maintaining a partition of the parameters of the global model.

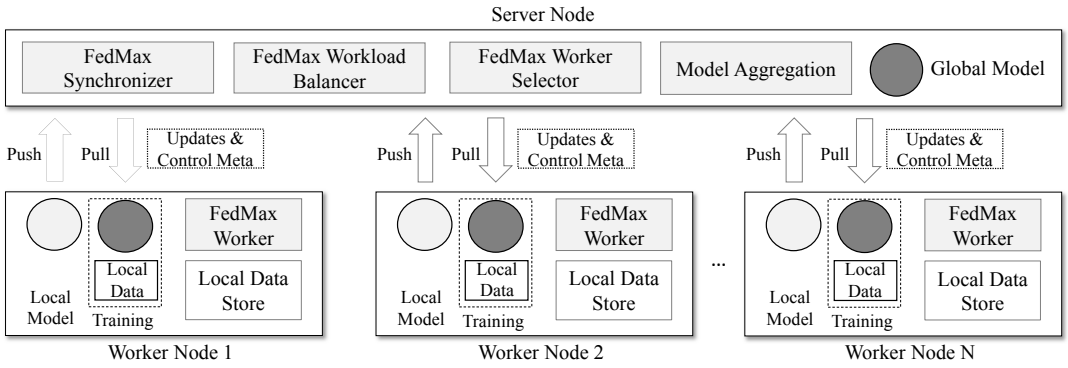

Fig. 5: Overview of FedMax architecture.

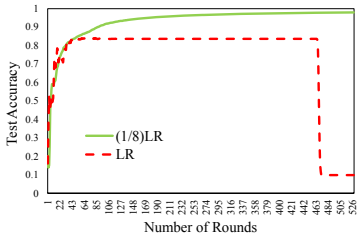

Fig. 6: Model convergence impact from λ with 8 workers (same capacity).

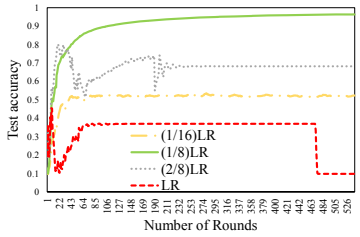

Fig. 7: Model convergence impact from λ with 8 workers (varying capacities).

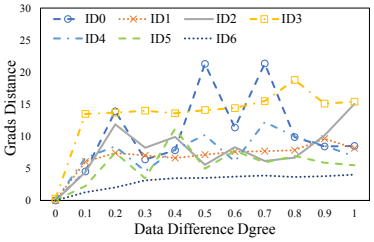

Fig. 8: Gradient similarity vs. data similarity.

LeNet [9] as the machine learning model and MNIST [18] as the dataset. First, in a fully asynchronous case (i.e., $s == 1$), we run 8 workers each with the same computation capacity. In Figure 6, LR means that λ is set to 1, while $\frac{1}{8}$LR means that λ is set to $\frac{1}{8}$ (i.e., 8 is the number of workers). We observe that when $\lambda = 1$, the model does not converge. "$\lambda = 1$" indicates that each worker's local model contributes to the global model at the full speed. Again, for IID data, this is a good estimation, while for non-IID data, it results in high model staleness, and hence poor model convergence as shown in Figure 6 — after certain rounds, the asynchronous case does not converge.

Instead, we choose to only apply a portion of a worker's update to the global model with a fraction being λ, as shown in Formula (1). This way, after all workers upload local models in a round, the sum of all such local updates approximate the global model's *one* update. We have measured the impact on model convergence with different λ in a more real, heterogeneous environment — 8 workers with varying computation capacities. As shown in Figure 7, when $\lambda = 1$, we observe the similar results as that in Figure 6: a sudden drop of model accuracy occurs after several rounds. When $\lambda = \frac{1}{8}$ (i.e., 8 is the number of participating workers), the model quickly converges with the highest accuracy. When $\lambda = \frac{2}{8}$, there also exists some oscillations. The reason is that a bigger learning rate (than $\frac{1}{8}$) introduces higher model staleness. When $\lambda = \frac{1}{16}$, the model converges but very slowly. It is because, with a smaller λ model staleness becomes smaller, hence no oscillations. However, with a reduced learning speed, it takes more time to converge. These examples clearly show that λ can greatly affect the model convergence and it may vary from one training model to another.

3) Workload Balancing: Similar to a synchronous update scheme, in FedMax, workers with less computation capacity may still slow down the whole learning system speed. It is because fast workers need to wait for slow workers before the next update round within τ, during which the computation resource of fast workers is wasted. To overcome this problem and improve system overall efficiency, FedMax involves a *dynamic workload balancing mechanism*. FedMax dynamically adjusts the workload of workers — the number of data batches/epochs for training in one update round — based on the observed update speeds of these workers. For instance, if a fast worker is 10 times faster than the *slowest* worker, FedMax assigns 10 times of workload to the fast worker for its next round. Alternatively, FedMax can also reduce the workload of slow workers. In federated learning, as FedMax prefers to put more computation locally, it mostly adopts the former strategy. More concretely, FedMax initially assigns same amount of workload in local training for each participating worker. After several rounds of observation, FedMax detects the update speed of each worker, and then increases workload of faster workers. Gradually, completion time of different workers with varying computation capacities converges.

To summarize, FedMax's relaxed synchronization communication scheme strikes a good balance between synchronous and asynchronous updates. It tackles the model staleness and

load imbalance problems to achieve high model accuracy and system efficiency. Notice that, if we set a large τ, s becomes N and FedMax behaves the same as the synchronous scheme. In contrast, if we set a small τ, s becomes 1 and FedMax behaves just like the asynchronous update. In FedMax, λ and τ are two key tuning factors that vary dependent on training models and dataset — a subject of ongoing investigations.

C. FedMax's Worker Selection

To reduce communication cost in federated learning, a straightforward approach is to randomly pick up a fraction of workers to contribute to the global model in each update round. FedMax greatly advances this approach with a new *worker selection approach*, which more wisely identifies the effective workers yielding enhanced model convergence. FedMax's selection approach targets a deep neural network model widely used in federated learning [1], where SGD is the basic update algorithm.

1) Neural Network Model: In general, a deep neural network represents a map: $F : X \rightarrow \hat{Y}$ with parameter W. For example, in image classification tasks, $\hat{Y}$ refers to the predicted label of an input image X. We can also write it as:

$$\hat{Y} = F(X; W) \tag{3}$$

In SGD, model parameters are initialized as W_0. Then the bias between model output and ground truth of input can be computed as $E = \|F(X; W_0) - Y\|$, where Y is the truth label of input images. Because our final goal is to get an optimized W that can map input images to their corresponding labels, we see W as an unknown variable while input X as a known variable. The update ΔW can be calculated as:

$$\Delta W = -\eta \frac{\partial F(X, W)}{\partial W}\bigg|_{W=W_0} \tag{4}$$

Given by initialized parameter W_0, ΔW can be seen as the function of input data X.

2) Gradient Similarity: It can be observed from Formula (4) that similar input X would lead to similar gradient ΔW given the same initialized parameter W_0. Conversely, data X similarity could be induced from gradient ΔW similarity with the same initialized parameter W_0. In federated learning scenario, all workers need to be synchronized with the server after global model aggregation (i.e., all workers will be assigned with same initialized parameter W_0 after global model aggregation). (Recall that, in federated learning, the central model can only obtain gradient ΔW instead of data X.) Thus to induce data similarity (between two workers), we can measure their gradient similarity after a synchronous update. In particular, the parameters of a neural network can be expressed as a "metric", so are gradients. We can then use the Euclidean distance between the metrics of two worker's gradients to measure their gradient "similarity".

To illustrate how gradient similarity reflects data similarity, we show a concrete example using the MNIST dataset [18]. First, we create a "baseline" case, where we run 8 workers (with ID from 0 to 7) each having images of a unique label. With this, the 8 workers have disparate data from each other. Then, we create a "contrast" case, where we still run 8 workers (again with ID from 0 to 7). Initially, the workers of the same ID in the contrast case have the same data as that in the baseline case. Then we add "differences" in each worker's data in the contrast case — each worker (e.g., ID) replaces a portion of its original data with the data from its neighbor worker (e.g., $ID + 1$). We vary such a portion from 0% to 100%: 0% means that the worker keeps the same data, while 100% means that the worker replaces all of its data with its neighbor's. For each pair of workers (i.e., workers from the baseline and contrast cases with the same ID), we calculate the Euclidean distance between their gradients. In Figure 8, as the data "difference" increases, there is an obvious trend for most worker pairs that, the Euclidean distances between their gradients also increase.

3) Worker Selection: As stated above, data similarity between two workers could be induced from their gradient similarity. Hence, $D(\Delta W_k, \Delta W_j)$ can be used as an indication for inferring data similarity between two workers, k and j. Based on this, FedMax introduces a worker selection approach to pick up the effective workers for the next update round: FedMax drops one of the two workers with the smallest distance $D(\Delta W_K, \Delta W_J)$ at a time until the number of the remaining workers reaches K — the expected number of workers to participate in the next round:

The concept of the similarity-based worker selection can also be applied in the global model update to further reduce communication cost. Figure 2 shows that the model divergence (defined in Section III-B2) becomes smaller as the training process proceeds. It indicates that the local model updates of all workers are getting close to a converged global model. It makes intuitive sense that, given all workers with their local model updates close to the global model, we can slow down (some of) their update speeds to reduce communication cost. More specifically, FedMax monitors the Euclidean distances between the local model update of each worker and the global model during the training process, once the largest distance among them (i.e., model divergence) is below a threshold — indicating that all workers converge — a subset of workers can be informed to slow down their update speed (e.g., by skipping several rounds of updates).

IV. Evaluation

We have implemented a prototype of FedMax upon MXNet [4] and evaluated it using multiple popular deep neural models including LeNet [9] and VGG [20], and datasets including MNIST [7], [18] and CIFAR10 [8].

We first show the properties and benefits of FedMax in detail using a small-scale test-bed (with 8 nodes). We further deploy and evaluate FedMax in a large-scale setup with 70+ nodes. For a fair comparison between FedMax and existing synchronization and asynchronization approaches, we have the following common configurations: (1) The batch size of local dataset is configured to be the same in all test cases (i.e., 64).

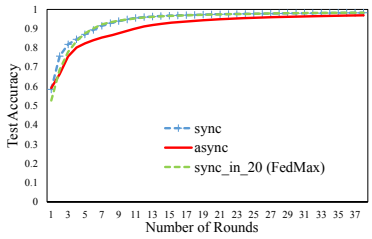

Fig. 9: MNIST: comparisons on a homogeneous scenario: 8 workers have the same computation capacity.

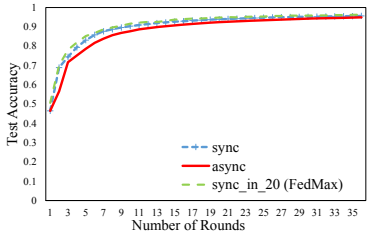

Fig. 10: MNIST: comparisons on a heterogeneous scenario: 8 workers have varying computation capacities.

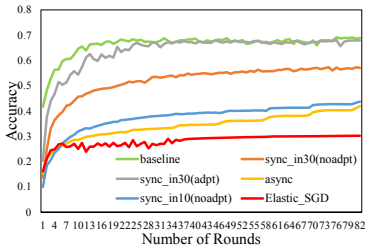

Fig. 11: CIFAR10: comparisons on a heterogeneous scenario: 8 workers have varying computation capacities.

(2) In all test cases, one update round is considered to be that when all N participating workers complete in the same update round. (3) The time of a update round is determined by the slowest worker. As the time spent by the slowest worker is nearly the same under all the tested synchronization approaches, we simply use the number of update rounds to reflect the real training time.

A. FedMax's Relaxed Synchronization Scheme

1) MNIST on a Homogeneous Scenario: First, we evaluate FedMax under a simple homogeneous scenario, where we deploy 8 workers each running on a machine equipped with 2 CPUs and 4 GB memory, and 1 central server running on a machine with sufficient resources – 4 CPUs and 8 GB memory.

The dataset we use is MNIST [18], which is for handwriting digital recognition: There are ten classes of data with handwriting number from 0 to 9; each image is labeled with the corresponding label of the number in the image. The deep neural model we use is LeNet [9]: There are two 5x5 convolution layers (the first with 32 channels and the second with 64 with each followed with 2x2 max pooling); a fully connected layer with 512 units and ReLu activation; and a final softmax output layer.

To emulate a federated learning setting, we specify how the dataset is distributed over workers. Each worker is assigned with five *random* classes of data (out of total 10 classes). By dividing dataset in this way, the non-IID degree is higher than that of traditional distributed learning systems, which usually shuffle the data and distribute them among workers (hence more IID). Note that, this partitioning does not results in a total *non-IID* case, as the workers share certain data classes. We will use a more non-IID scenario, shortly.

We further divide the local dataset into multiple batches with each batch consisting of 64 images. Notice that the actual size of dataset used for one round processing is guided by FedMax's workload balancer. Within one epoch, the machine learning algorithm processes all batches. In our experiments, by default, workers will synchronize with the server after each epoch. Yet, notice that, the size of epoch can be adjusted (e.g., the bigger the epoch number, the lower the communication frequency between workers and the server).

In Figure 9, we compare three cases with different synchronization approaches: (1) synchronous, (2) asynchronous and (3) FedMax's relaxed synchronization (with τ being 20 seconds). In all the three cases, we consider one round (x-axis) as that the server receives updates from all workers. We observe that, though in the *first* round, both the synchronous case and the asynchronous case achieve the close model accuracy, the synchronous case converges faster than the asynchronous case afterwards. That said, the asynchronous approach takes much longer time to converge: When the round number is 37 in Figure 9, the asynchronous approach remains 1.4% behind the synchronous approach.

With τ being 20 seconds, FedMax's relaxed synchronization scheme achieves almost the same convergence rate as the synchronous scheme. It is because, in this homogeneous scenario, the update speeds among these workers are close to each other; all workers can synchronize within 20 seconds in FedMax— same as the synchronous case.

2) MNIST on a Heterogeneous Scenario: To emulate a more realistic heterogeneous scenario in federated learning, we change the computation capability of the above 8 workers. Specifically, 4 workers are assigned with 1 CPU, 3 workers are assigned with 2 CPUs, and 1 worker is assigned with 5 CPUs. In addition, we change the data partitioning method to realize a more non-IID scenario: We first sort the data by its digit labels, divide it into 8 shards in order, and assign each of 8 workers 1 shard. This is a more non-IID partition of the data, as most workers will only have the training data belonging to at most two labels/classes.

Again, we compare the three cases as the above homogeneous scenario. Differently, in this case, the workers have different computation capacities that in each round they finish at different times. For the synchronous case, the server will wait for the slowest worker to complete and then start the next round. In contrast, for FedMax's relaxed synchronization case, the server adjusts the workload ratio for fast workers based on the observed update speeds of 8 workers. Guided by such information, the fast workers are re-assigned the size of dataset to be processed (i.e., we simply run more epochs for fast workers) until all workers reach to a stable state where their update speeds are close.

As illustrated in Figure 10, we observe that the synchronous

case still achieves better accuracy than the asynchronous case. Unlike the above homogeneous scenario, FedMax's relaxed synchronization case outperforms the synchronous case, though slightly. For example, The model accuracy under FedMax is 0.6% higher than that under the synchronous case, when the round number is 37. Further, as listed in Table I, to achieve higher model accuracy, FedMax requires significantly smaller number of rounds. For example, to achieve 96.6% model accuracy, FedMax reduces the number of rounds by 25%. As each round takes roughly the same amount of time to complete for all three synchronization cases (determined by the slowest worker), FedMax significantly improves the convergence rate. Moreover, as dataset become mores complex, the advantage of FedMax widens. We will see shortly that, this advantage becomes more significant when we use another larger, more complex dataset, CIFAR10 [7], in Section IV-A3.

Model Accuracy	90%	92%	94%	96.60%
async	106	146	225	606
sync	72	104	168	424
FedMax	64	88	128	320

TABLE I: FedMax needs fewer rounds to achieve the same model accuracy compared to the other two cases.

One of the main reasons that makes FedMax achieve higher accuracy within the same time is due to its workload balancing mechanism. In this heterogeneous scenario, FedMax adjusts the workload size of fast workers (e.g., with 5 CPUs or 2 CPUs) to make them process more data (epochs) to reach the similar update speed as the slow workers (e.g., with 1 CPU). In addition, the waiting time of the fastest worker (i.e., with 5 CPUs) reduces significantly, from 74.1% to 40.3%, indicating improved system efficiency. Notice that, 40% waiting time seems high. However, it is because, the time of each round with the MNIST dataset is quite small — little disturbance while execution will result in a big percentage difference.

3) CIFAR10 on a Heterogeneous Scenario: CIFAR10 [8], a more comprehensive dataset, consists of 10 classes of 32x32 images with three RGB channels. These images are closer to real-world scenarios, and harder to be classified for deep neural networks than MNIST. We choose the same model, LeNet, to train CIFAR10. We evaluate FedMax using 8 workers: 1 worker with 5 CPUs, 3 with 2 CPUs, and 4 with 1 CPUs. Each worker is randomly assigned with 5 classes of data (out of 10), allowing them to have certain levels of overlap.

In addition to the above three comparison cases, we add a *best-case* baseline, which is the standard SGD training using LeNet on the full training set (no partitioning). As shown in Figure 11, the baseline achieves an 69% test accuracy [2]. The model accuracy achieved by the asynchronous case is far below the synchronous case, which is in turn lower than the baseline. In contrast, we observe a clear advantage of FedMax, both on model accuracy and convergence speed. FedMax achieves almost the same model accuracy as the baseline as the round number increases. In addition, we observe that the larger threshold time τ we set (e.g., 30 vs. 10 seconds), the higher accuracy the global model can achieve. But of course, a larger τ may degrade system efficiency.

We also compared our relaxed synchronization with a stat-of-the-art distributed SGD algorithm: Elastic Averaging SGD [19], designed for accelerating convergence in IID data. The result in Figure 11 shows that in the non-IID situation, Elastic SGD performs significantly worse than FedMax. Last, we observe that the waiting time of the fastest worker (with 5 CPUs) reduces from 50% to 20%.

B. FedMax's Worker Selection Approach

We have evaluated FedMax's worker selection approach using the same configuration as that in Section IV-A1. We calculate the gradient distance $D(\Delta W_k, \Delta W_j)$ of any pair of workers $\{k, j\}$. To select n workers, FedMax drops $8 - n$ workers based on their gradient similarities. Specifically, each time FedMax drops one of the two workers with the smallest $D(\Delta W_K, \Delta W_J)$. In Figure 12, as FedMax drops the most "similar" workers one by one, we do not observe significant performance drop in the beginning until half of the 8 workers are dropped. In contrast, if we drop workers randomly — for example, as shown in Figure 3, after dropping worker7, instead of dropping worker5 (the most similar one), we drop worker8 — we observe significant drop of model accuracy and convergence speed.

We have further evaluated FedMax's worker selection method by monitoring model divergence with a more complex model, VGG [20]. Based on the model divergence curve in Figure 2, we present a divergence threshold Δ and monitor the real-time model divergence. Here we set the Δ at round 31, as we find that after round 31, model divergence does not perform significant decrease. Once model divergence is under threshold Δ, in our evaluation, 2 workers are randomly selected from 8 workers to update the global model in each round. With this, the average update speed of each worker is slowed down. In Figure 13, we compare the cases with (1) full update, (2) selected update by threshold Δ, and (3) selected update by threshold 0. The results show that the case with selected update with a suitable threshold can achieve almost the same converge rate compared to the case with full update. However, the case with selected update unconstrained ($\Delta = 0$) does not reach a good converge rate, because when the model selection is carried out when the model divergence is still high, the updates from the selected workers are not a good approximation of all workers' updates.

C. Scalability

We have deployed and evaluated FedMax in a large-scale setup with 70 workers of varying computation capacities on Google Cloud Platform. Specifically, we run 40 small instances (i.e., workers) with 1 CPU, 20 medium instances with 2 CPUs, and 10 large instances with 4 CPUs. In Figure 14, we use the same CIFAR10 dataset. Each worker is randomly

[2]Note that the state-of-the-art approaches have achieved a test accuracy of 96.5% for CIFAR10. However, our goal is to evaluate the effectiveness of FedMax, and not focus on tuning training models for optimal accuracy. High-accuracy learning models can be applied to FedMax.

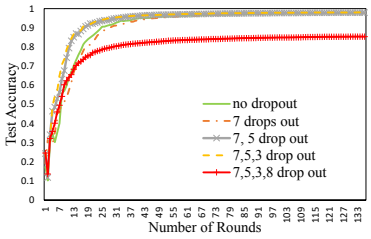

Fig. 12: FedMax drops workers with the most similarity.

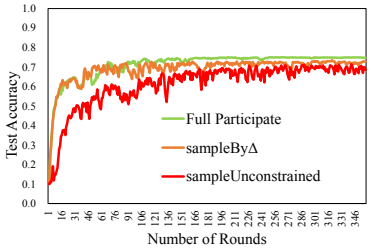

Fig. 13: FedMax select worker update during training.

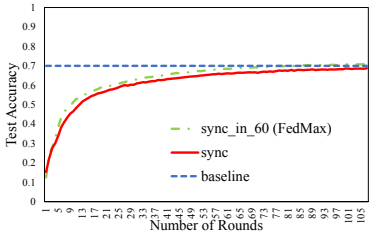

Fig. 14: FedMax runs with 70 workers on Google Cloud Platform.

assigned with 3 classes of data (out of 10). τ is set to 60 seconds. We observe that FedMax achieves 2.5% higher model accuracy than the synchronous scheme, when the round number reaches 100, and is very close to the best-case baseline (measured on the full training set).

Note that, in all these experiments, CPU resources are fully utilized (by the slowest worker(s)) during running, while memory is sufficient to cache the dataset for each worker. In addition, the communication cost in both in-house test-bed and in-cloud setup is low, as the network bandwidth between workers and the central server is relatively high (1 Gbps).

V. Related work

Recent years have witnessed both commercial and academic machine learning datasets growing at an unprecedented pace. The trend is going to continue as IoT goes from concepts to reality, where an increasing number of IoT devices continuously create sheer amount of data. Empowered by such massive datasets, the scale of machine learning models and their training and inference algorithms have been scaled up dramatically and proved to be more effective than small-scale models and algorithms [21], [22]. For easy implementation and deployment of large-scale machine learning models, a number of distributed machine learning frameworks have been developed [4], [10], [23], [24], which leverage large-scale clusters of machines to distribute training and inference tasks.

There are two common approaches for distributed machine learning algorithms, data parallelism [26] and model parallelism [25], [27]. [1] adapts them in a federated learning setting, by assuming strict *synchronous* communication between workers and the central server. FedMax advances this approach with a relaxed synchronization scheme by considering a more realistic heterogeneous distributed environment. A lot of asynchronous algorithms have also been extensively studied [6], [13], [19], mostly considering balanced and IID datasets. Distributed consensus algorithms [14] relaxes such constraints, but the communication cost is high when the cluster size becomes large. However, our evaluation with such a distributed algorithm, Elastic SGD [19], shows that it does not perform well in a non-IID scenario.

VI. Conclusions

We have presented FedMax, a privacy-preserving distributed machine learning framework for federated learning. To build this framework, we addressed several practical challenges such as dropout of workers, heterogeneity of worker computation, and limited communication, with two main techniques: a relaxed synchronization communication scheme and a similarity-based worker selection approach. We have implemented a prototype of FedMax and extensively evaluated it with popular learning models and datasets. Our results on both a in-house test-bed and a large-scale cloud environment demonstrate that FedMax greatly increases the robustness, model accuracy, and system efficiency of a federated learning system.

References

[1] McMahan, Brendan, et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data." Artificial Intelligence and Statistics. 2017.

[2] Li M, Andersen D G, Park J W, et al. Scaling Distributed Machine Learning with the Parameter Server. In OSDI. 2014.

[3] Zaharia M, Chowdhury M, Das T, et al. Fast and interactive analytics over Hadoop data with Spark[J]. Usenix Login, 2012.

[4] Chen, Tianqi, et al. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems."

[5] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016.

[6] Zhang, Wei, et al. "Staleness-aware async-SGD for distributed deep learning." Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. AAAI Press, 2016.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 1998.

[8] Krizhevsky, Alex, and Geoffrey Hinton. "Learning multiple layers of features from tiny images." (2009): 7.

[9] LeCun Y. LeNet-5, convolutional neural networks[J]. URL: http://yann.lecun. com/exdb/lenet, 2015. Based Vision. 2004

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K.Yang, and A. Ng. Large scale distributed deep networks. NIPS, 2012.

[11] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. NSDI, 2011.

[12] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein. Distributed Graphlab: A framework for machine learning and data mining in the cloud. In PVLDB, 2012.

[13] Reddi S J, Hefny A, Sra S, et al. On variance reduction in stochastic gradient descent and its asynchronous variants. NIPS. 2015.

[14] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. ICML.JMLR Workshop, 2014.

[15] Wang, Liang, Ben Catterall, and Richard Mortier. "Probabilistic Synchronous Parallel." arXiv preprint arXiv:1709.07772, 2017.

[16] Bonawitz, Keith, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. "Practical secure aggregation for privacy-preserving machine learning."ACM CCS, 2017.

[17] Abadi, Martin, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep learning with differential privacy." ACM CCS, 2016.
[18] "MNIST DATABASE."http://yann.lecun.com/exdb/mnist/
[19] Zhang S, Choromanska A E, LeCun Y. Deep learning with elastic averaging SGD[C]. In Neural Information Processing Systems 2015. from few training examples: an incremental Bayesian approach tested on 101 object categories. IEEE. CVPR 2004.
[20] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
[21] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
[22] Le, Quoc V. "Building high-level features using large scale unsupervised learning." 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, 2013.
[23] Abadi, Martín, et al. "Tensorflow: A system for large-scale machine learning." 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016.
[24] Li, Mu, et al. "Scaling distributed machine learning with the parameter server." 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). 2014.
[25] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.
[26] Bauer, Michael, et al. "Legion: Expressing locality and independence with logical regions." SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012.
[27] Gao, Mingyu, et al. "Tetris: Scalable and efficient neural network acceleration with 3d memory." Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. 2017.