CO-OP: Cooperative Machine Learning from Mobile Devices

by

Yushi Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Massive amounts of user behavior logs and sensor data are generated on mobile devices, which can help to improve the usability of social media apps and other intelligent apps. However, collecting such personal data may spark privacy and legal concerns. Recently, many efforts in both academia and industry have been devoted to developing distributed machine learning methods and architectures to scale up to a large amount of distributed data. However, most such methods focus on a server cluster or data residing on several datacenters. This work takes one more step toward the more ambitious objective of collectively training machine learning models based on data from mobile devices, yet without collecting such private data centrally. We propose *CO-OP*, an asynchronous protocol which leverages the computing power of each mobile client to train local models based on small amounts of newly generated training samples, and merges such local models into a global model on the server judiciously, balancing the model accuracy and communication overhead. We implemented a *CO-OP* Android app and tested it on 60 real clients distributed in different continents. Results suggest that *CO-OP* can achieve an accuracy of more than 80% on image classification using neural networks based on the MNIST datasets, even when the clients are intermittently available and training data are generated dynamically on the go. Additional simulation results also demonstrate the effectiveness of *CO-OP* in training Support Vector Machine (SVM) and Logistic Regression models.

Preface

The result of Chapter 3, Chapter 4 and part of Chapter 5 has been submitted to INFOCOM 2018 as T. Zhang, Y. Wang, C. Li and D. Niu, "CO-OP: Cooperative Machine Learning from Mobile Devices"[1]. I was responsible for the design and implementation of CO-OP system, organizing the experiment and analyzing the experimental results. T. Zhang and I collaboratively worked on the design for the model merging algorithm and age filter behaviours. D. Niu was the supervisory author and assisted in the design and manuscript editing.

Acknowledgments

First of all, I want to thank my parents, my wife and my 14-month-old son. Without their support, I would give up for a hundred of times.

Then I am deeply indebted to my academic advisors, Dr. Di Niu. His guidance and supervision helped me to learn how to be a good researcher and pursue an academic career. His willingness to discussion helped me through two important years of my life.

I also would like to thank all the people who contributed in some way to the work described in this thesis.

Last but not least, I want to thank myself.

Table of Contents

List of Figures

Chapter 1

Introduction

With the increasing popularity of social media, online shopping and location-based services, mobile phones and tablets have gradually become the major personal computing devices [2]. An enormous amount of data are constantly being generated from these mobile devices , including data from sensors and cameras, behavior and click logs in apps and location information. Such massive data, together with machine learning, have powered the success of modern intelligent apps and recommender systems [3]–[5].

However, collecting such private and sensitive personal data from mobile devices for machine learning purposes has sparked ever-increasing privacy and legal concerns [6]–[8]. On one hand, app operators must take a greater responsibility protecting user data from leakage both during transmission and in storage. On the other hand, many users do not want to share their data at all out of privacy reasons.

There is a growing need to design decentralized algorithms and systems that can effectively perform widely adopted machine learning tasks without collecting the user data to a central site. A related field, distributed machine learning, has recently attracted significant attention both in industry and in academia, which however mainly focuses either on training models on a cluster of co-located servers based on the Parameter Server architecture [9]–[13] or on learning a shared model when datasets reside on several different datacenters [8], [14]. None of these schemes can directly be used to reap insights from data that come from a potentially large number of independent and unorganized mobile devices.

Google Research has initiated a project called "Federated Learning", aiming at testing the feasibility of training supervised learning models, e.g., image classification, on mobile devices without collecting data centrally, with research results published recently [15]–[17]. Unlike traditional distributed machine learning [9]–[11] which aggregates the error gradient computed by each node, the key idea of Federated Learning is to leverage the computing power which modern mobile devices are equipped with to generate high-quality local models and aggregate these local models directly in each iteration, thus reducing the required rounds of server-client communication. However, existing work has only scratched the surface of what is possible in the ambitious goal of decentralized learning from mobile devices—a number of practical issues are yet to be addressed: *First*, the dataset on a mobile device is not static in reality: the training samples are generated and added to each client on the go. *Second*, Federated Learning adopts a synchronous protocol, where in each communication round, the server selects a number of random clients to pull their local models for aggregation. However, it is hard to synchronize the clients in reality; local models are computed asynchronously. *Third*, a pull-based scheme is used, where the server contacts the clients to pull local models, whereas in reality, a client may only be available intermittently or never respond due loss of connection. *Finally*, Mobile devices have significantly heterogeneous bandwidth, latency, and computing powers. Communication and model updating protocols need to be designed to respect the usefulness and timeliness of each local computed model, eliminate unnecessary transmissions, and avoid biasing toward faster users.

To solve these challenges, this thesis presents the design of *CO-OP*, cooperative machine learning from mobile devices without collecting the raw data. Our major contributions are highlighted as follows:

- **A push-based protocol**: in *CO-OP*, it is the clients, instead of the server, that initiate all the communication. A client may choose to contact the server when it has trained its local model based on a new batch of data and has free Wifi connection or is plugged in. Unlike prior work [15]–[17], the server does not schedule any update. This design suits the real scenario well, where mobile clients are only intermittently available, while the server is always available.

- **Asynchronous updates**: each client uploads its local model asynchronously. We propose an asynchronous model merging algorithm: whenever a qualified local model is received, the server immediately takes action to merge it into the global model without waiting for updates from other clients.

- **Heterogeneity and Age-Dependency**: *CO-OP* adopts several key strategies to respect the heterogeneity of devices in computing power and connection speed. We keep track of the model "age" both at the server and at each client, and throttle a client from updating if it becomes too aggressive while discarding obsolete updates and let slow clients to catch up with the global model. The "age" of a local model also decides the weighting at which it is merged into the global model.

- **Real experiments**: we implemented a *CO-OP* Android app and conducted experiments on 60 real Android clients, including mobile phones and tablets of diverse setups, to collectively train an image classification neural network model based on $60,000$ MNIST data samples which are assumed to be generated at the clients in mini-batches over a period of 6 hours. *CO-OP* successfully trained a model with a test accuracy of 80%, even when participating clients are highly dynamic, heterogeneous and unorganized.

To the best of our knowledge, this thesis presents the first experimental results from a medium-scale real-world deployment of distributed machine learning on mobile devices. In addition, we also conducted extensive simulation studies to verify the effectiveness of *CO-OP* in training several popular types of supervised learning models including logistic regression, neural networks, and support vector machines (SVM), as compared to the centralized mini-batch learning as well as Federated Learning. Simulation results demonstrate the generalizability of *CO-OP* in performing diverse types of machine learning tasks.

Chapter 2

Related work

The topic of distributed machine learning based on data partitioned in multiple machines is first addressed in the context of tightly coupled server clusters [12], [18]–[21], where all the worker nodes, each holding a part of the data, still reside in a same datacenter and communicate with each other through local-area networks (LANs). There are three basic distributed machine learning architectures: iterative MapReduce [22], Dataflow [23] and the Parameter Server (PS) [9]–[11], [13], [24]. The MapReduce architecture, only using synchronous update, has been adopted in MLlib[25] and Mahout [26]. The Dataflow architecture has been successfully applied to TensorFlow [18]. However, Dataflow-based systems have high-level abstraction and low-level flexibility. Comparing to the other two architectures, the Parameter Server supports different model aggregation mechanisms, especially asynchronous model updates. Due to its performance advantage, Parameter Server is widely adopted in systems such as Petuum [27] and Google's DistBelief [9]. However, in distributed machine learning, even though data are distributed in different workers, they are still in one datacenter and each worker has a large amount of static data, which is not the case in our settings. Also, in the above-mentioned architectures, workers need to communicate with each other intensively through LANs to synchronize their models, which is impossible when all the workers are mobile devices and communicate through wireless networks.

Geo-distributed machine learning has been studied in Gaia [14] and [8], which propose decentralized machine learning systems where training data are distributed among more than one datacenter and the communication between datacenters is through wide-area networks (WANs). These systems take the variable network bandwidth into consideration. In Gaia [14], each datacenter will individually apply the same training algorithm onto its local data to optimize its own model, while synchronizing the model parameters to all other datacenters using the so-called Approximate Synchronous Parallel (ASP), which is a loosely synchronous protocol. ASP does not only take workers' computational speed into account, but also considers the data transmission speed, which is similar to the needs in our system. However, in geo-distributed machine learning, all the workers are highly reliable servers in datacenters. Each worker is always available and has a large static dataset to work on. We do not have such privileges in our system, where workers are unreliable mobile devices.

Federated Learning [15], [16] is the first study that identifies distributed machine learning from mobile devices as an important research direction. Two approaches are proposed in [16]: 1) Structured Updates, where an update is learned from a restricted lower-dimensional space, and 2) Sketched Updates, where the model is compressed before being sent to the server. In [15], synchronized simulation has been conducted to show the correctness of their approaches. However, both [16] and [15] use the same pull-based algorithm where in each synchronized communication round, the server initiates the contact and chooses a random set of clients to pull their trained local models for aggregation, whereas in real world, a mobile device may never respond and it is hard to synchronize mobile devices. Furthermore, in Federated Learning [15], [16], the dataset in each worker is assumed to be static, whereas in reality the training samples are generated on mobile devices dynamically. This thesis takes one more step to propose an asynchronous push-based protocol, which is optimized to perform well in a highly heterogeneous environment. We are also the first work that puts cooperative machine learning from mobile devices to reality check on a medium-scale real-world experiment among 60 mobile clients.

Finally, many efforts have been devoted to the theoretical studies of distributed machine learning algorithms [28]–[32]. These algorithms all try to strike balance between the model accuracy and communication overhead. Bulk Synchronous Parallel (BSP) [33] synchronizes updates after all the workers finish their local training. Before the next iteration starts, all the workers will obtain the most up-to-date model. Stale Synchronous Parallel (SSP) [11] only allows the fastest worker to be ahead of the slowest worker by up to a bounded number of rounds. Total Asynchronous Parallel (TAP)[34] removes all synchronization between workers. BSP and SSP can guarantee algorithm convergence in theory but have heavier communication cost. While there is no convergence guarantee for TAP, it has the least communication cost. However, in these algorithms, each worker performs a simple gradient update before sending it to the server, where the gradients (instead of the models) are aggregated for model updates. Thus, a large number of communication rounds between each worker and the server is required for these algorithms to converge.

To reduce communication rounds, distributed learning algorithms by iteratively averaging locally trained models (instead of gradients) have also been studied, for example, Perceptron [35], speech recognition DNNs [36] and Elastic Averaging SGD [37]. *CO-OP* is another variant of asynchronous model averaging (instead of gradient aggregation) put into the specific setting of using mobile devices as workers.

Chapter 3

The Asynchronous *CO-OP* Protocol

Unlike prior work [14], [16], [15] which assumes each client has a fixed dataset that can be used for training, in this thesis, we deal with dynamic data, which are generated as users use mobile devices and are added to their individual training datasets on the go. Suppose there are K mobile devices, or *clients*, participating in the cooperative learning. Each client is independent and accumulates its own training data according to its user's habit. Once B training samples have arrived at a a client (B is referred to as the *local batch size*), the client starts to train the local model using gradient descent based on the new batch of samples. Fig. 3.1 shows the framework of *CO-OP*.

In contrast to Federated Learning [16],[15] where a server contacts some random clients to perform model updates in synchronized communication rounds, we let each client initiate model update attempts asynchronously. Client can conduct the local training process based on gradient descent anytime at any place whenever it wants to update its local model based on the new data, e.g., when it is plugged in. When client has better network connection, e.g., it can attempt to upload its local model to the server for aggregation (subject to the *CO-OP* protocol) and subsequently download the updated global model from the server to replace its local model.

Let $w \in \mathbb{R}^d$ denote the global model parameters at the server and $w_k \in \mathbb{R}^d$ represent the model parameters at each client $k \in 1, \ldots, K$. Denote by n the age of the global model, i.e., the number of times that the global model has been merged with any local models. Each client keeps a local model age n_k, which is updated according to the *CO-OP* protocol. The *CO-OP* protocol is described in Algorithm 3.1.

Algorithm 3.1 The asynchronous CO-OP protocol.

```
1:  **Initialization**: set $w = w_1 = \ldots = w_K := w_0$; set $n := a$, $n_1 = \ldots = n_K := 0$.
2:  Each client $k$ performs the following independently:
3:  while true do
4:      Accumulate a new batch of $B$ samples $D_k$.
5:      $w_k$ =ClientUpdate($k, D_k$)
6:      When the client is ready to contact the server, pull the model age $n$ from the server.
7:      if $n - n_k > b$ then
8:          {Update is too old}
9:          Pull $w, n$ from the server and set $n_k := n$, $w_k := w$.
10:         $w_k$ =ClientUpdate($k, D_k$)
11:     else if $n - n_k < a$ then
12:         {Updated too often}
13:         Continue
14:     else
15:         {Normal upload}
16:         Upload $w_k$ to the server. Upon receiving $w_k$, the server immediately performs the update:
            $$w := \left(1 - \frac{1}{\sqrt{(n - n_k) + 1}}\right) w + \frac{1}{\sqrt{(n - n_k) + 1}} \cdot w_k,$$
            and sets $n := n + 1$.
17:         Download the global model $w$ and set $w_k := w$, $n_k := n$.
```

Initially, each client, when joining the system, downloads the global model w_0 and sets the the local model age to 0. Once a client k has accumulated a new batch of B training samples, it will train the local model w_k using gradient descent (for R rounds) based on these B samples. When client k is ready to upload its newly trained local model, w_k will be accepted by server for merging into the global model subject to an age filter, i.e., the local model age n_k will be compared with the global model age n. The intuition is that the global model will only be updated if w_k is neither obsolete nor too frequently updated. In the following, we first explain the model merging scheme, followed by the interpretation of the age filter.

3.1 Model Merging

Model aggregation is a critical problem, which will directly affect the model convergence and accuracy. In Federated Learning [16], [15], the FedAvg algorithm updates w as $w := (w_{k_1} + \cdots + w_{k_C})/C$, where $w_{k_1}, \ldots, w_{k_C}$ are the C local models collected in a particular synchronized round. Although FedAvg works well in the synchronous scenario, it is not suitable for the asynchronous case in reality. As local model updates arrives asynchronously, we must merge them into the global model one by one. In our algorithm, the global model w is updated with the combination of the current w and the newly received local mode w_k in a simple weighting equation $w := (1 - \alpha)w + \alpha w_k$.

Setting a proper α is important to the algorithm performance. When the server aggregates asynchronously updated local models, there might be some local models that were trained with outdated global models as the initial starting point. Therefore, it is unfair to give local models equal weights. In *CO-OP*, we assign α depending on the age gap $n - n_k$ between the local and global models, i.e., setting $\alpha = 1/\sqrt{n - n_k + 1}$.

Intuitively speaking, when $n - n_k$ is smaller, the local model w_k is more important. In the extreme case when $n - n_k = 0$, the global model will be updated to $w := w_k$, which is exactly the same update scheme in mini-batch learning. On the other hand, when $n - n_k$ is large, which means that the local model was trained with a much earlier global model as the starting point, w_k is not reliable. Thus, the merging weight α should be small in this case to ensure convergence of the global model.

3.2 The Age Filters

Note that in *CO-OP*, a normal upload and update will occur only when $a < n - n_k < b$. This leads to the discussion of another key design in *CO-OP*, i.e., we have adopted age filters to reduce the upload traffic as well as to avoid unbalanced data which hurts the convergence performance.

Uploading and downloading models could still incur high traffic volume. Especially, for a mobile device, the upload bandwidth could be up to 10x smaller than the download bandwidth.

In *CO-OP*, a client checks the age gap $n - n_k$ before sending out a local model w_k. If $n - n_k > b$, w_k was trained based on an old global model and will not be reliable for model merging. In this case, the server will not let client k upload w_k to save traffic. Instead, it will send the current global model to client k, which subsequently sets $n_k := n$ and $w_k := w$, and trains the model again to catch up.

On the other hand, $n - n_k < a$ implies that client k is very active; it generates data and uploads their local models at a much higher frequency than other clients. In this case, accepting w_k will make the whole dataset unbalanced and bias the global model toward the personal model of such clients, which actually decreases the generalizability of the global model on test data. Therefore, for such an active client k, *CO-OP* will let it continue training on its next local batches without uploading or downloading until $n - n_k$ falls into the normal range.

This idea is essentially similar to the Staleness Synchronous Parallel (SSP) [11] algorithm in parameter servers; when the server detects that the fastest client is ahead of the lowest client by $b - a$ rounds, it will block the fastest client from uploading or downloading until other clients catch up.

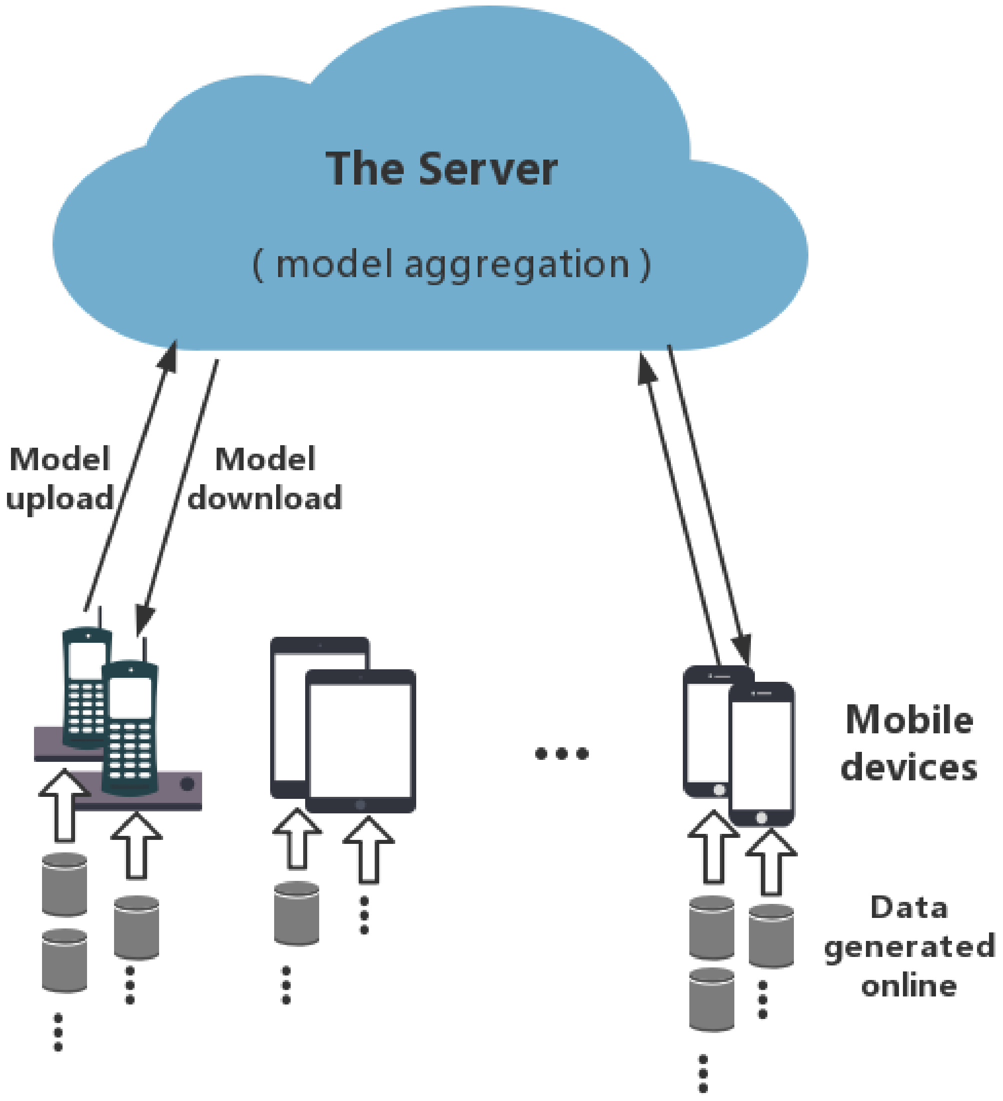

Fig. 3.1. The framework of the *CO-OP* system for asynchronous distributed machine learning from mobile devices. The training data are dynamically generated

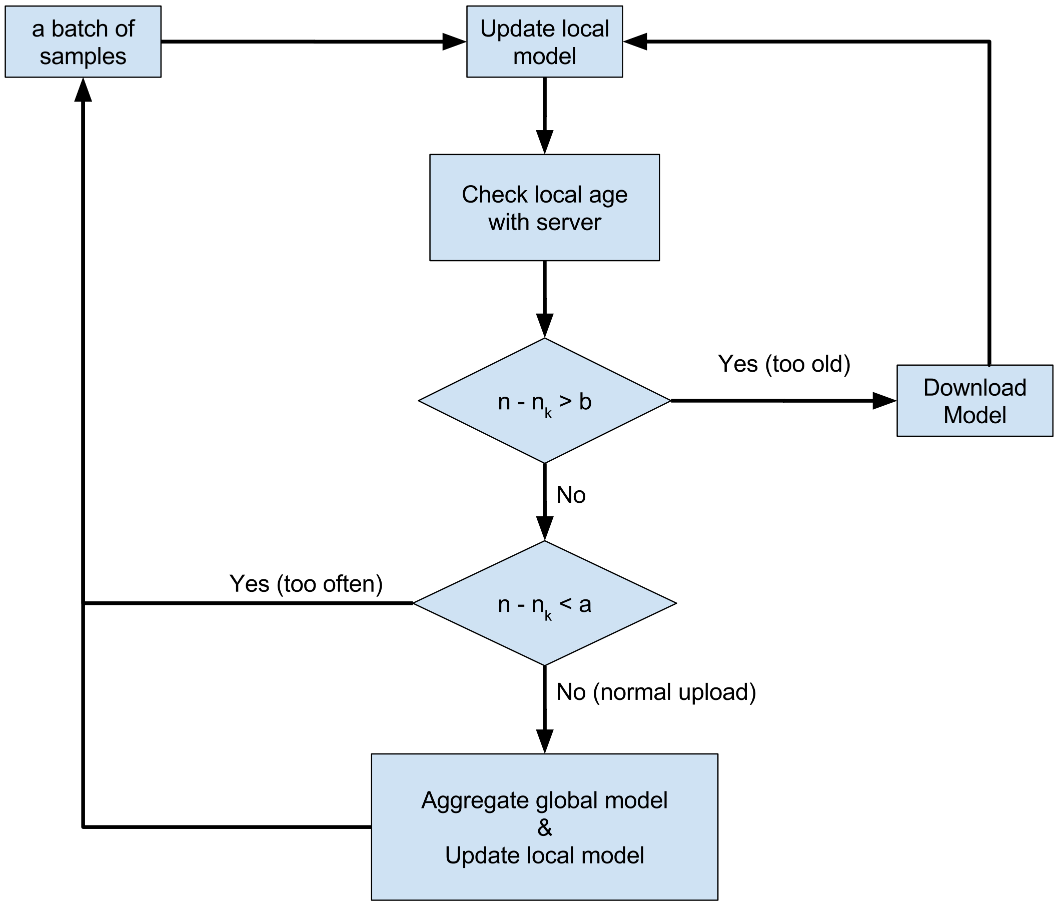

Fig. 3.2.   The flowhchart of asynchronous CO-OP protocol.

Chapter 4

Simulation results

In this chapter, we show the applicability of *CO-OP* to general machine learning problems, through simulations performed on three widely-used machine learning models, including Neural Networks, Logistic Regression (LR) and Support Vector Machine (SVM) on three different datasets, respectively.

We simulate the asynchronous update scenario with round-based simulation, where each round represents a small unit of time. The real asynchronous case will be evaluated in the experiments. Suppose there are 100 clients. Each client receives a new batch of data at a random frequency and performs gradient descent local training on the new batch for a number of iterations. After a new local model is ready, the client will contact the server according to the *CO-OP* protocol. We model the interval (in rounds) between the points of time when a local model is ready as a certain truncated random variable bounded below by 0 and rounded to integer simulation rounds. Such an interval is approximately equal to the batch arrival interval.

We evaluate the effectiveness of *CO-OP*, in comparison to two other state-of-the-art algorithms:

1) Centralized (batched): centralized learning based on mini-batch gradient descent (GD) is one of the most commonly used learning algorithms, which tries to find a balance between the robustness of stochastic GD and conver-

gence of full-batch GD. The dataset is split into multiple mini-batches, which are fed to the model separately.

2) Federated Averaging (FedAvg) [15]: the best algorithm in Federated Learning. The server randomly selects C clients to pull their local models $w_{k_1}, \ldots, w_{k_C}$ in each communication round. And the global model is updated as $w := \sum_{i=1}^{C} w_{k_i}/C$.

To adapt FedAvg, which is essentially a synchronized protocol, to our simulated asynchronous case, in each simulation round, the server does not select C clients to pull models. Instead, those clients with a new local model ready will upload their models to the server for direct averaging. For Centralized (batched), all local batches of data, once generated, are sent to the server where the mini-batch GD algorithm is performed to training the global model.

4.1 Neural Networks

Neural network is one of the most powerful models in machine learning, which has been successfully implemented in many applications. In this work, we test *CO-OP* on a feedforward neural network with a single hidden layer. There are 512 neurons in the hidden layer using $tanh$ as the active function. The dataset is MNIST, which is a handwritten digit database of $28*28$ images, containing 70,000 samples in total. We split it into the training and testing datasets, containing 60,000 and 10,000 data points, respectively. The batch arrival intervals at each client are generated from a Gaussian Distribution with a mean of 20, and a variance of 10. The local batch size B is 100.

Fig. 4.1 shows the accuracy of the global model under different algorithms on the test set over time. The performance of *CO-OP* is close to the centralized learning in spite of a few unstable points, and is much better than Federated Averaging. The unstable situation happens because of some clients with a low age. Fig. 4.2 is the comparison among different versions of *CO-OP*. It is obviously that the curve of

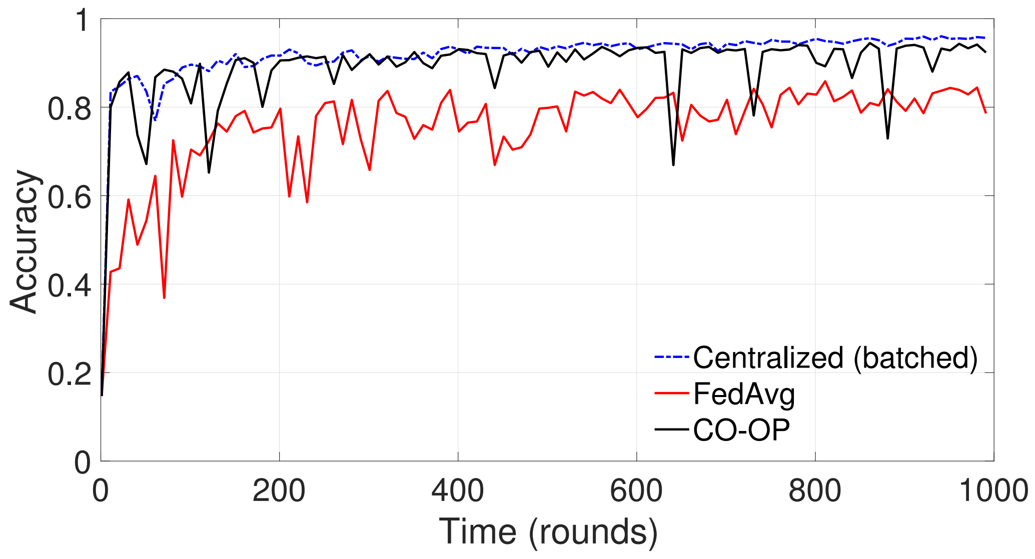

Fig. 4.1. A comparison of different algorithms for the neural network.

CO-OP (without the age filter and with random weights) is oscillating more wildly, which demonstrates the effectiveness of age-dependent weighting. Comparing *CO-OP* with and without the age filter , we can see that *CO-OP* with the age filter is better, because the age filter helps to block some unreliable local model updates. The age filter also helps to greatly reduce the total number of server contacts (especially the uploads) performed by the clients. With the age filter the number of uploads is 3392, while the number of downloads is 5597. Without the age filter the numbers of uploads and downloads are both 5130.

4.2 Logistic Regression

Logistic Regression (LR) is a classical and simple machine learning algorithm for classification problems. Using a logit function, LR is able to learn from some dependent variables. To evaluate the performance of *CO-OP* on training LR models, we choose a dataset called Shuttle in LIBSVM, which contains over 40,000 samples. Each sample has 9 input features and the output falls in 7 categories. We split the dataset evenly into two parts, one for training and the other for testing. The local batch size B is 20, and the batch arrival interval at each client follows a $(60, 0.5)$

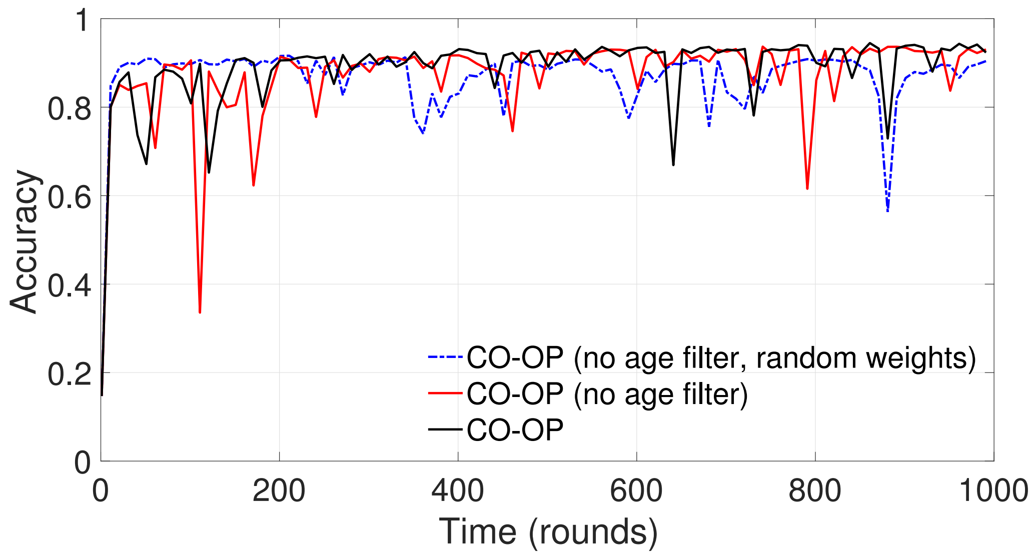

Fig. 4.2. The effects of the age filter and age-dependent weighting in *CO-OP* in the neural network.

Binomial distribution.

Fig. 4.3 shows the performance of *CO-OP* as compared to the Centralized algorithm and Federated Averaging. As we see, all the algorithms, including Centralized (batched) fluctuate a little bit, because LR is sensitive itself due to the S shape of the logistic function. From the figure we can see all three algorithms have a mean accuracy around 95% in the final stage.

Fig. 4.4 shows the communication overhead reduction due to the use of age filters. Especially, we can observe that with the age filter, the number of uploads from clients is significantly reduced, because the local models that are either too "old" or too "aggressive" are blocked from being uploaded. This is an advantage, since the upload bandwidth of a mobile device could be $10\times$ slower than its download bandwidth.

4.3 Support Vector Machine

Support vector machine is another popular model in machine learning. The standard linear SVM classification problem attempts to create a hyperplane which separates the given data points in a feature space based on their associated binary labels. To

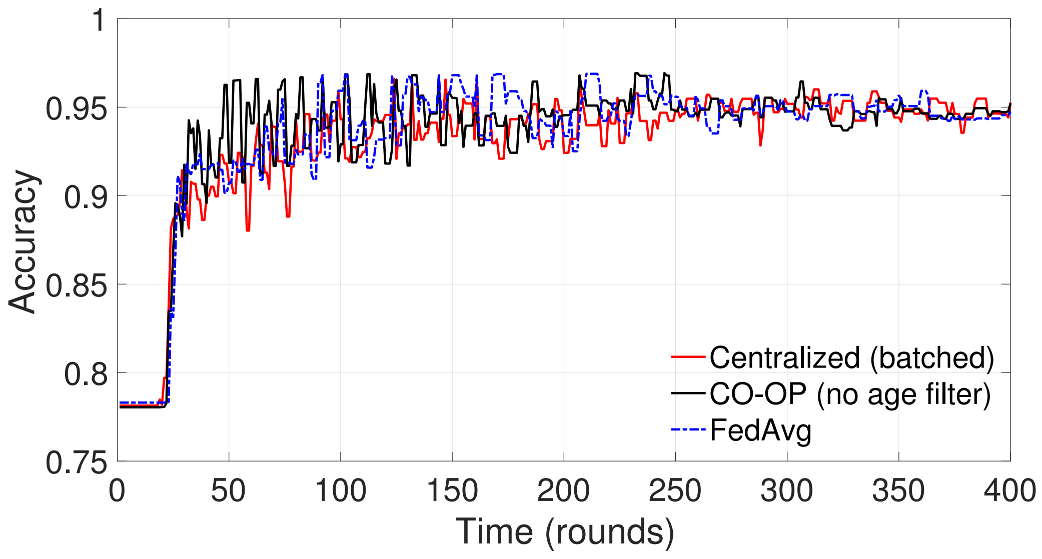

Fig. 4.3. A comparison of different algorithms for Logistic Regression.

test the performance of *CO-OP* on SVM, the USPS dataset of handwritten digits, consisting of 7,291 training samples and 2007 testing samples, was used. The batch arrival interval at each client follows a truncated Gaussian distribution with a mean of 15 rounds and a variance of 10. The local batch size is $B = 30$. We repeated feed the dataset to clients in batches, with each training sample utilized about 5 times throughout the simulation. Additionally, the learning rate was established as $batchsize^{-2}$, which was empirically optimized.

Fig. 4.5 shows the algorithm comparison result. We can conclude that *CO-OP* and Centralized (batched) are better than Federated Averaging in terms of both prediction accuracy and model convergence. Again, Fig. 4.6 shows the reduction in the number of uploads and downloads by clients with the age filter technique applied. We can see that both uploads and downloads have been reduced by the age filter, especially the uploads.

To summarize, applying *CO-OP* on three classic machine learning models, neural networks, Logistic Regression and SVM, we find that *CO-OP*, as an asynchronous protocol works well on all three models. The age-awareness introduced into the protocol not only helps to stabilize the algorithm performance but also

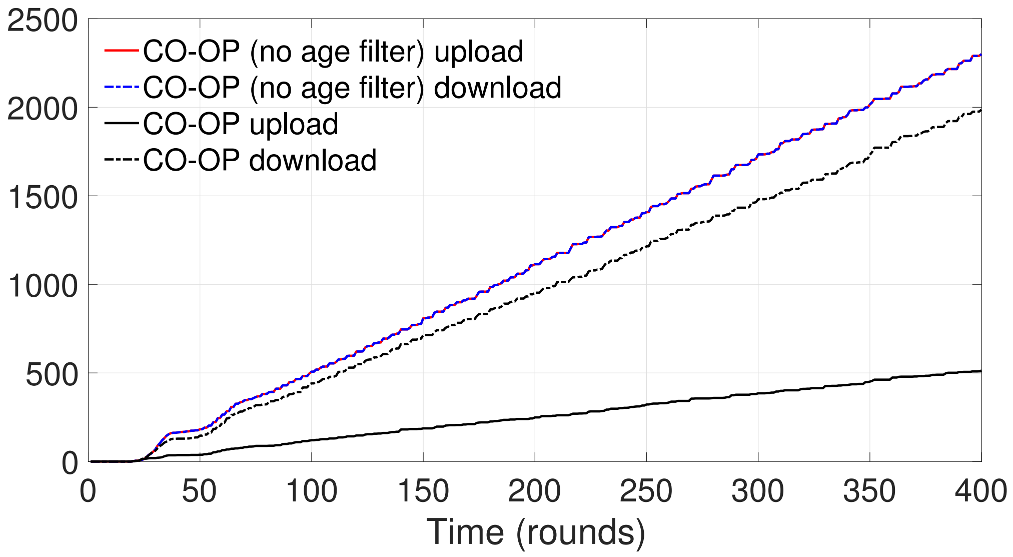

Fig. 4.4. The number of uploads and downloads in *CO-OP* over time for LR.

greatly reduces unnecessary communication without hurting the accuracy.

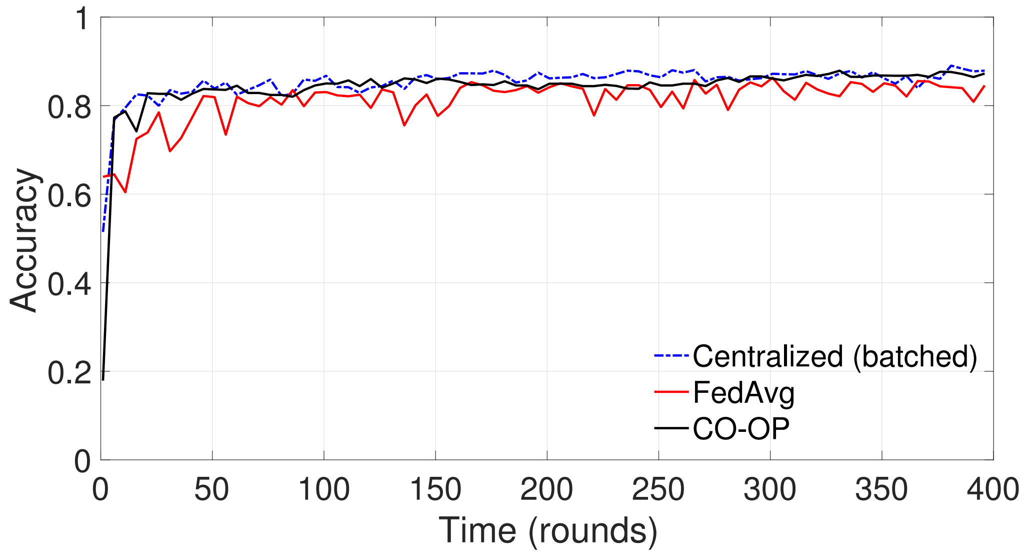

Fig. 4.5. A comparison of different algorithms for SVM.

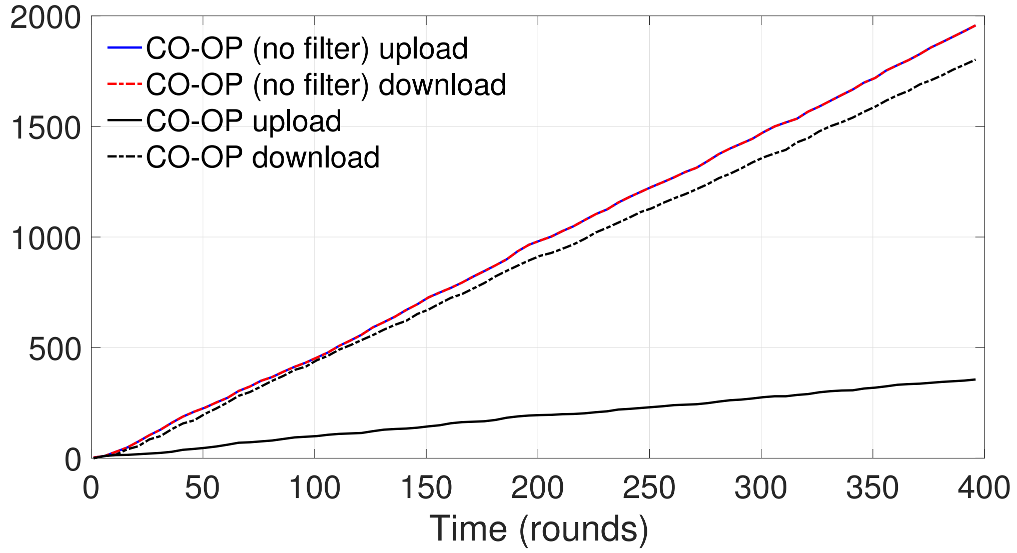

Fig. 4.6. The number of uploads and downloads in *CO-OP* over time for SVM.

Chapter 5

Real Experiments

In this chapter, we present the experimental results of testing *CO-OP* on 60 geographically distributed Android clients for training an image classifier using neural networks based on the MNIST dataset dynamically fed into the clients.

5.1 System Implementation

Our system includes two parts: the client devices each running the *CO-OP* Android mobile app and a central server that stores the global model.

5.1.1 Server Side Setup

In our experiment, the server simply runs on a MacBook Pro (mid 2015) with 4 cores 2.5 GHz Intel Core i7 and 16GB 1600 MHz DDR3 memory, connecting to the university LAN which has relatively ample bandwidth connection.

On the central server, we run a Node.js application. Node.js is an asynchronous event-driven JavaScript runtime, and designed to build scalable network applications [38]. It is a different solution from popular network solutions called concurrency model.

For concurrency models, when a request reaching a web server, the web server launches a new OS thread to process the request, handle the service and send response back. Meanwhile, if another request reaches the server before the first one finished, the web server would launch another OS thread, which is identical to the first one, to do the same jobs for the second request. It would raise problems when multiple threads trying to access the same resource. For example, in our case, multiple clients connect to the central server, and are served by different threads simultaneously. When they all upload their local model to the central server at the same time, race conditions for network controllers occur.

On the other hand, Node.js is asynchronous and single-threaded. Network I/O controllers are scheduled to divide their time to network port fairly equally, such that no connection would be dropped because of timeout. All I/O accesses (including networking) are performed asynchronously, so the process never blocks. This ensures all the clients' local models get uploaded to the central server.

Fig. 5.1 shows the API setup for the central server. When clients call checking age API, one of three results will be sent back. If the client model is too old, the client needs to call the download API to get the updated global model and discard its local outdated model. If the client model is updated too often, the client would neither call uploading API nor downloading API. If the client model's age confirms update requirement, the client would call uploading API to upload the local model, and the server would send back the merged global model.

5.1.2 Android Client Side App

Each *CO-OP* app simply runs the SGD algorithm to train the local model based on training samples that appear in batches. There are two types of client devices we have used in our experiment. The first type is 30 real Android mobile devices, including mobile phones and tablets, from past 3–5 years. They run different versions of the Android kernel and different distributions of the Android system, such as the Samsung distribution version and Huawei version. As most of the devices we could find are older than the latest Android devices on the market, we introduce the second type of 30 clients to emulate faster latest Android devices. These clients are laptops running the *CO-OP* app via Android emulators, including BlueStacks and

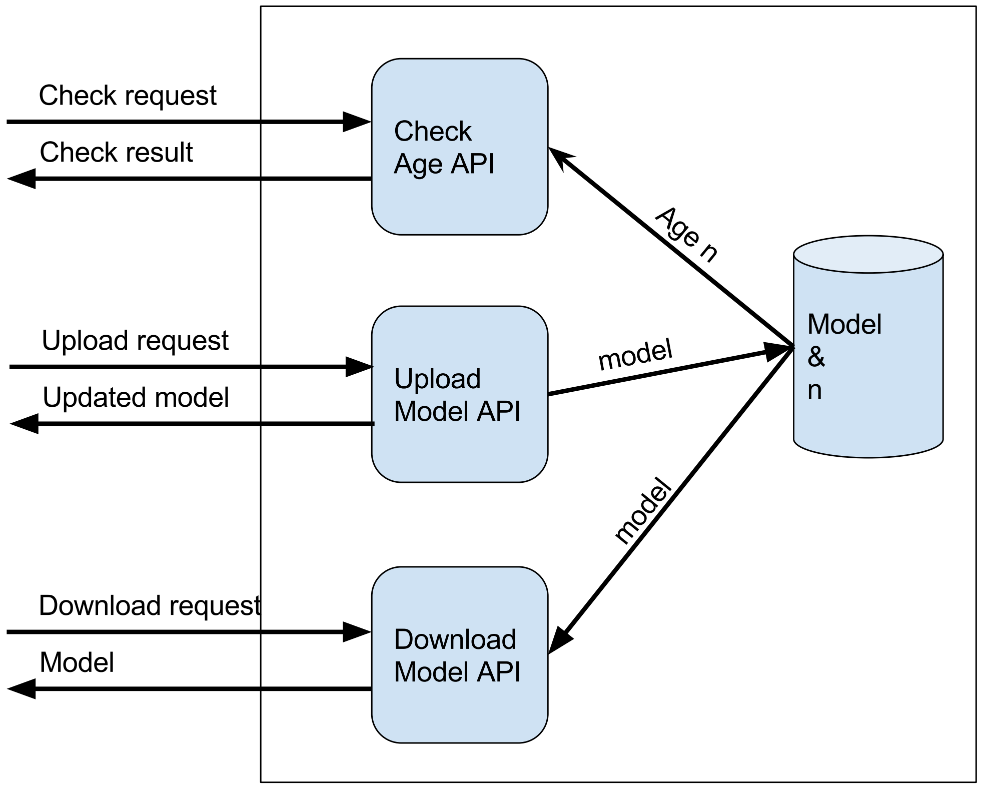

Fig. 5.1. The architecture of the server side application of *CO-OP* system and the Web API expose to client side application

Android Studio emulators running one of the latest Android systems with the latest kernel. It is worth noting that we chose the Android platform over iOS, because it is much easier to distribute a demo app on Android by just installing an `.apk` (Android application package) file.

In order to provide better user experience, Android requires applications to put computation heavy tasks (like batched gradient descent calculation) and time consuming tasks (like networking uploading and downloading) off the main thread (UI thread), to prevent the UI thread getting blocked and causing user interface frozen or lagging [39]. The common solutions are asynchronous tasks [40] and services [41]. Either of them runs tasks on a separated thread, taking the advantages of multiple cores of CPU inside of modern cell phones. We choose asynchronous tasks as our solution because it is lighter and easier to implement. I would not cover the differences between asynchronous tasks and services in detail, as it is out of scope for this thesis.

Fig. 5.2 shows the workflow of the client applications. When a client joins the *CO-OP* system, we assume it has already generated a batch of data. Then the client starts its first cycle of processing by loading the first batch of data. After each cycle is done, the client goes into next cycle of processing, loading the next batch of data. The client applications would keep generating and processing data batch by batch, until all the data in its dataset has been processed. The following shows the details of data processing in each cycle.

1. Load next batch of data.

2. Train the local model with stochastic gradient descent on the batch of data.

3. Call checking age API on server.

4. Based on different server responses, the client application would perform different tasks.

   - If $n - n_k < a$, the client uploads too often.

     4.1. Wait for next batch of data to be generated.

     4.2. Go back to step 1.

   - If $a \leq n - n_k \leq b$, the client could is allowed to update global model.

     4.1. Call API to upload local model to the central server to aggregate with global model, and then download the updated global model, replacing the local one.

     4.2. Wait for next batch of data to be generated.

     4.3. Go back to step 1.

   - If $n - n_k > b$, the age of the client is too old.

4.1. Call download API to fetch the newest global model from server, replacing the local one.

4.2. Go back to step 2.

I also want to clarify that the uploading step and downloading step do not have to be separated into two asynchronous tasks. With a single request, the client app would send its local model to central server, and in the corresponding response, the central server would send back the updated global model to the client. So there is only one round of communication between the client and the central server, without having to establish redundant connections.

5.2 Application

We evaluate our system for training a neural network image classification based on the MNIST dataset, which contains 60,000 training samples and 10,000 test samples. Again, the goal here is to train a neural network to learn the relationship between the pixels (in grayscale) of a 28×28 image and the digit (0–9) it represents. Note that image classification on the MNIST dataset is a complicated enough model to *stress test* the capability of our cooperative learning framework, while in reality many popular machine learning tasks such as linear/logistic regression or next-action prediction requires far less computing power than the application considered in our experiment.

We adopt a single-hidden-layer neural network with 300 hidden nodes as the model to be trained. We choose this model to strike a balance between the power of the model and the model size in order to avoid overly high computation burden on mobile devices, since for a highly sophisticated model such as deep learning, the best resort is still to collect data centrally and train the model on dedicated hardware.

5.3 Experiment Setup

The MNIST training data (containing 60,000 samples) are equally divided into 60 smaller datasets, each containing 1000 samples. We assume each client holds one of such smaller datasets, and does not want to share it to the server for privacy concerns. Furthermore, the 1000 samples on each client are generated in batches of 50 samples to mimic the scenario where samples are dynamically added to the training set on each client while he/she uses the device. We assume the inter-arrival time between consecutive batches on each client is a Gaussian random variable with a mean of 30 seconds, and a standard deviation of 5 seconds. Note that such a dynamic batch-based scenario is closer to reality than the static datasets considered in prior work [15], [16].

We performed the experiment for 6 hours in total and the 60 clients join the system at random times during the 6 hours, depending on the availability of the human user holding each device. There are 8–15 devices simultaneously online at each particular point of time. Upon joining the system, each client first fetches all the hyper-parameters from the server, including the hidden layer dimension, local learning rate, the number of iterations of local SGD training to be performed on each batch of data, etc. Then it downloads the initial global model and carries out the *CO-OP* algorithm. At each client, the local SGD training for each local batch of 50 samples runs for 50 iterations, with the local learning rate set to 0.02, which is 1/batch size.

Finally, in order to reduce server load and save bandwidth, we only transfer each model parameter up to two decimal places, which can save network traffic by 3/4. Again, the justification is that we want to stress test the capability of *CO-OP* in a network of everyday mobile devices even with bandwidth cut. And it is not an objective of *CO-OP* to achieve extreme accuracy requirement for sophisticated models.

5.4 Results

Fig. 5.3 plots the CDF of client lifetime defined as the time interval between the first time a client contacts the central server, trying to upload its local model and the last time it does so. The lifetime does not depend on how fast the client does the computation or how many uploads it tries to make. As long as a client participated, the time difference described above will be used as its lifetime, even though some slower clients may only have a few tries during the entire experiment. As the figure shows, most of the clients stay online for a relative long time period of over 100 minutes, and the mean of the lifetime of all clients is around 41 minutes.

For the next, let us talk about the accuracy of the model. Fig. 5.4 shows the change of model accuracy over time for the *CO-OP* system and the centralized learning in batches, which collects all data from clients, then performs batched learning with stochastic gradient descent. We can see that the accuracy for centralized learning climbs fast, and achieves its highest value of 93%. However, since all the data is collected centrally, it violates the privacy-preserving requirement and only serves as a baseline for the accuracy comparison here.

The *CO-OP* system starts relatively slowly and achieves its highest accuracy value of around 80%. The lower accuracy for the *CO-OP* system can be expected, as the *CO-OP* system is asynchronous and the learning is performed in a decentralized fashion only based on unreliable real mobile devices. However, such asynchronous and decentralized model learning brings more features and benefits that centralized learning system does not, such as better privacy protection and a higher level of personal customization. As user's data stays on the devices locally, it prevents all kinds of hacks and attacks during data transmission. And as the newly trained model can be directly used on the user's device, it provides better user experience for the applications. We can conclude that under such a stress test of a medium-sized neural network image classification model, *CO-OP* can yield some positive training results by only using mobile devices as computing workers without collecting the raw data.

The vertical dashed line divides the figure into two parts. On the right part, all datasets have been re-fed into each client again, and be used to train the systems one more time to ensure the system achieve their highest accuracy value, although both centralized learning system and the *CO-OP* system have already achieved their highest model accuracy value before the data points have been re-fed into the systems (the left part).

We also notice that, for the *CO-OP* system, the model accuracy moves up and down drastically at the beginning of the experiment. We believe there are two reasons for it. On one hand, the initial model is randomly generated. It might have a good accuracy by some chance. On the other hand, at the beginning of the experiment, there are just a small number of clients joined into the *CO-OP* system, and one or a few of them are much faster than the others. They trained their local models fast, contacted the server to try to update global model frequently. However, as the difference of global age and local age is smaller than the lower bounds of the filter, the updates sent by these fast clients had been evaluated as "too often". When the updates got rejected, these clients kept training their local models with next batch of data generated. After a few times of trying and rejecting, the local model on these clients accumulatively trained with much more data than models on other clients. Thus, these models are much more accurate than models on other clients. Once global model updated enough times (global age increases, and becomes large enough), the updating submitted by these faster clients finally got approved. When the global model aggregated with a more accurate model, the spike appeared.

Now let us take a look at how the mobile clients behave in the system. In Fig. 5.5, the inter-upload times for each client are shown. Most of the clients would successfully update the global model every 2 to 5 minutes on average. Among them, the fastest client updates the global model with an average period of less than 1 minute (53 seconds), while the slowest client updates every 45 minutes. The fastest client should be one of the emulators running on our desktop computers, and the slowest client should be a Samsung Galaxy S4 phone which was released in 2013.

Fig. 5.6 shows the boxplot of the number of attempts clients make to update the global model (the first box), and the age-checking results responded by server (the following three boxes). Any try/check would be evaluated as one of three possible results: “too often", “normal uploads", and “too old". Therefore, the number of tries/checks is equal to the total numbers of checking results. As shown in the figure, about 70% of tries/checks finally lead into local model uploading followed by global model updating. And 30% of them are rejected, and most of them are from those faster clients. For the last part of the figure, it shows the number of checks evaluated as “too old", which means the model the clients used to train their data was too old, and was not suitable to be merged into the global model. It also shows that attempts/checks from 9 clients have been evaluated as “too old", and rejected. Half of these 8 clients have only been rejected for once, while the other half is been rejected for 8 to 10 times. (That is why only 4 points are shown on the figure).

Last but not the least, let us talk about data transmission. Fig. 5.7 and Fig. 5.8 show the amount of data transferred (in bytes) through network and the number of uploads and downloads in *CO-OP* system (with filter), comparing with in centralized system. Even though, in Fig. 5.7, the total number of uploads and downloads in *CO-OP* is larger the number of uploads (there is no download) in centralized training system, it could not be a disadvantage of *CO-OP* system. In real world scenario, the download bandwidth is much larger than the upload bandwidth (10 times larger), so saving the number of uploads is more effective and critical than saving the number of downloads. In Fig. 5.8, there are much more data transmitted in *CO-OP* system than in centralized training system. This cannot be the disadvantage of *CO-OP* system as well. First, there are more data transmitted, but there is less user information transmitted. The *CO-OP* system keeps user’s privacy more safe. Second, the amount of data transmitted not only depends on the number of uploads/downloads, but also depends on the size of data sample and the size of model. In our experiment, the size of each MNIST data sample is 784 bytes (an 28 * 28 image with only 1 color channel). So in centralized training system, the size of data in each upload is 38 KB (784 * 50). And for the neural network we used in the experiment, the model size is over 238 KB, which is also the size of data for each upload in *CO-OP* system. However, the size of data sample in real world applications could be much larger. A good example could be the photos user taking with their cell phones, which equipped with a multi-mega-pixel camera. These photos could be as large as 6-10 MB each. With a dataset like this, the size of data transferred in centralized training system would be much larger the size of model transferred in *CO-OP* system.

Fig. 5.9 shows the amount of data transmitted in *CO-OP* system comparing with itself but without age filter. About 2/7 amount of data transmission is saved with the filter. The age filter would significantly save the bandwidth for data transmission.

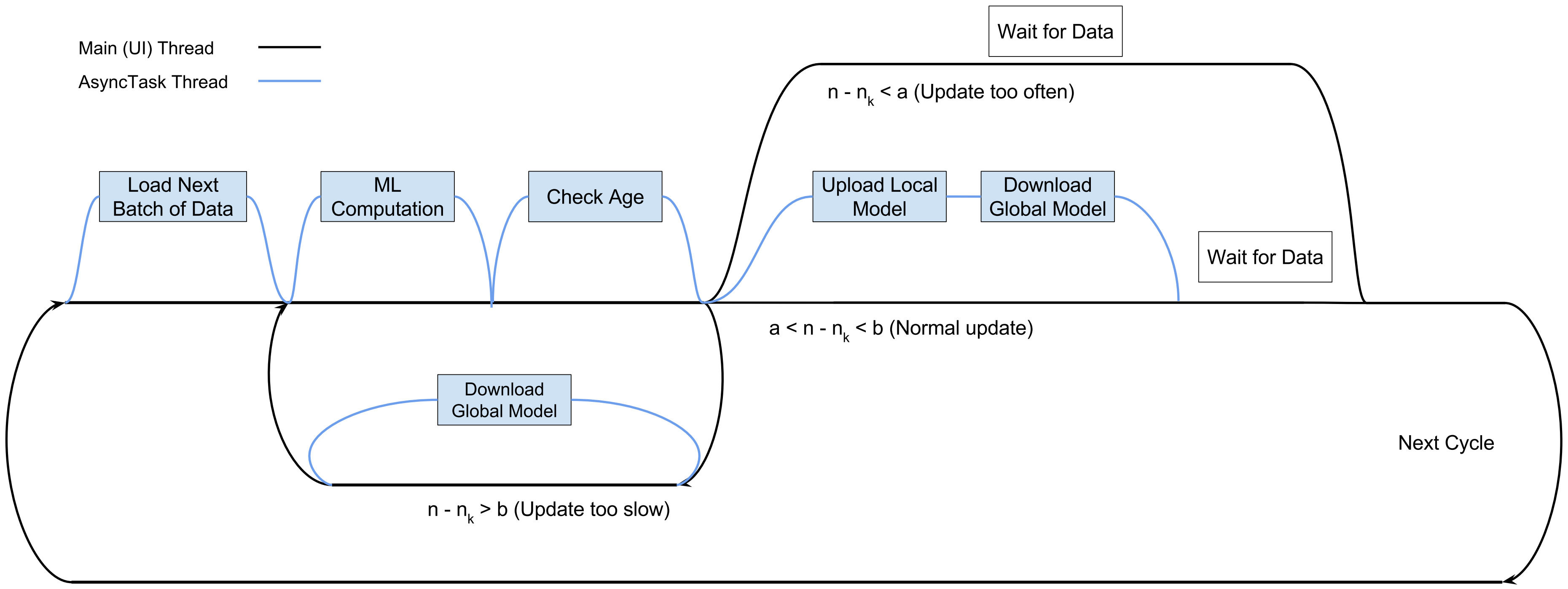

Fig. 5.2. The workflow of Android client application based on different check response from server

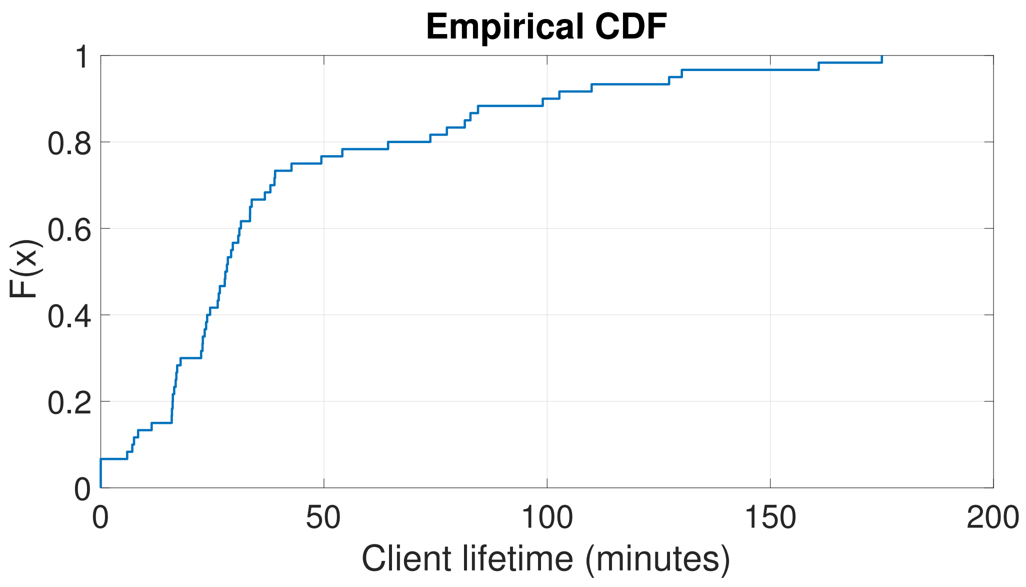

Fig. 5.3. The CDF of the lifetime of clients, with a mean lifetime of 41 minutes, maximum lifetime of 175 minutes and minimum lifetime of 0 minute

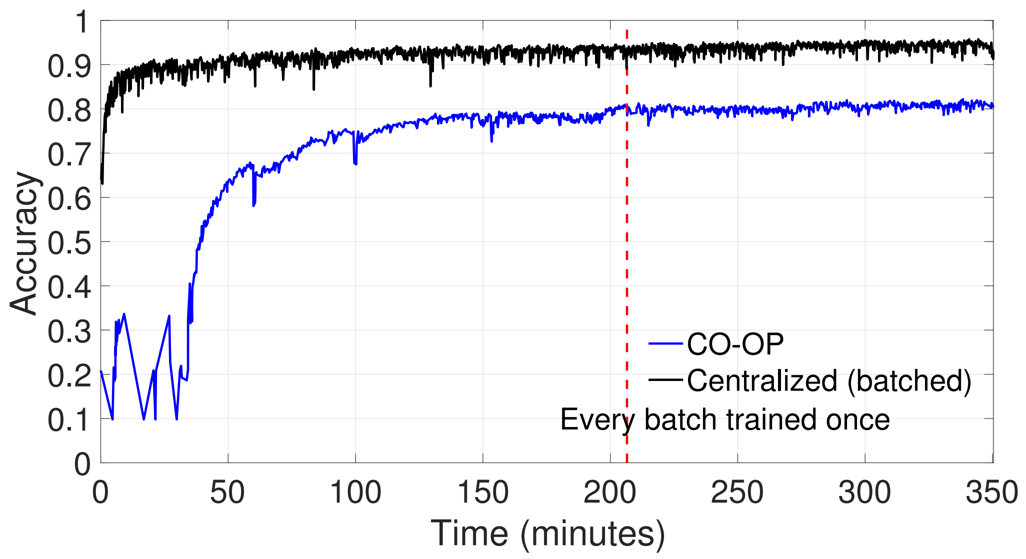

Fig. 5.4. The accuracy for *CO-OP* models over time vs. the accuracy of batched centralized model over time

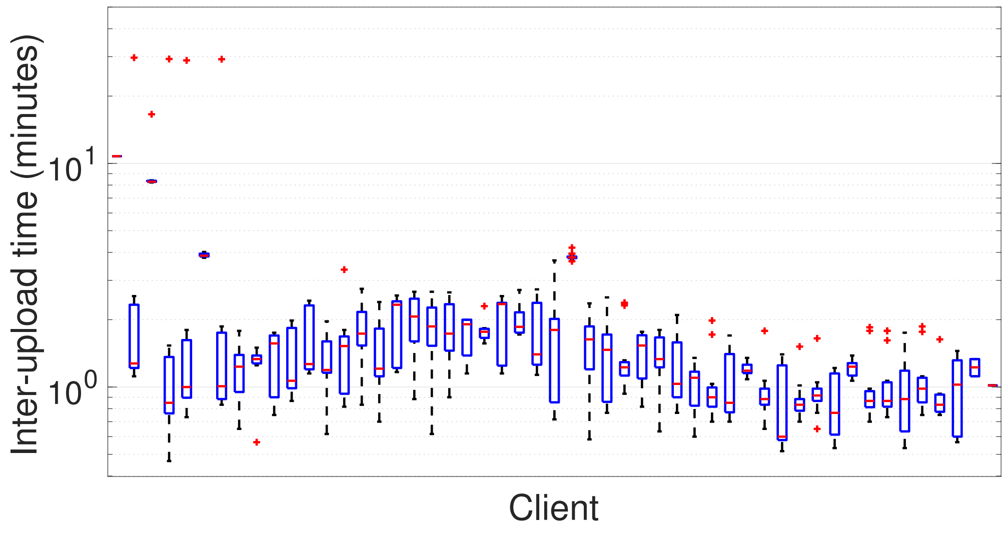

Fig. 5.5. The boxplot of of time intervals between each adjacent effective uploads for each client

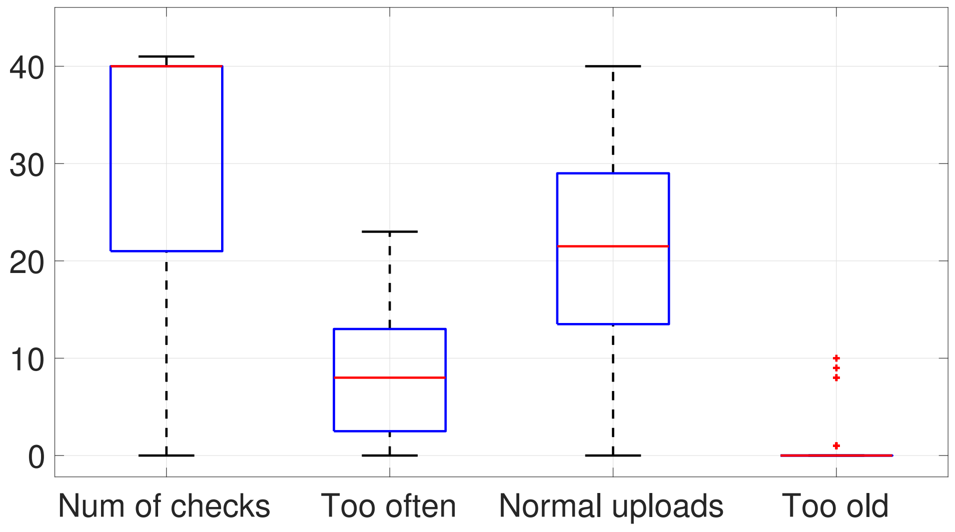

Fig. 5.6. The boxplot of total number of checks, number of too often, number of normal uploads and number of too old for all 60 clients during the whole experiment

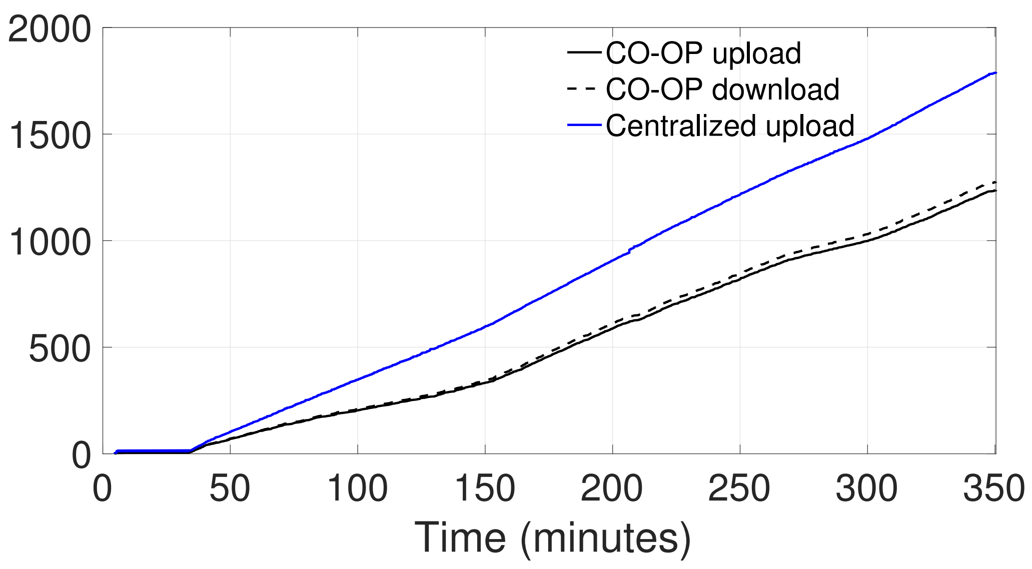

Fig. 5.7. The number of uploads and downloads in *CO-OP* system over time, comparing with the number of uploads in centralized training system over time

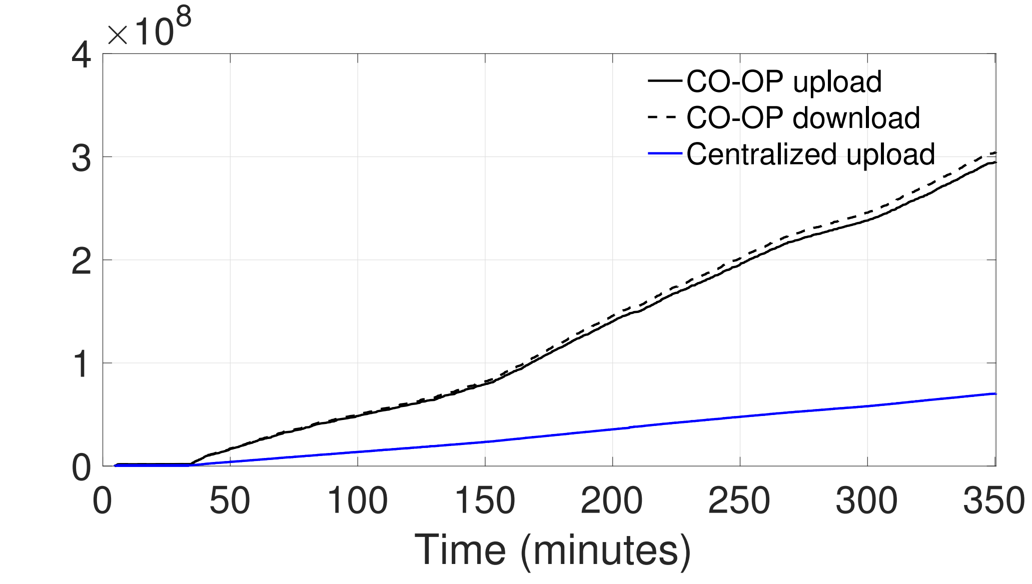

Fig. 5.8. The cumulative amount of data in bytes downloaded/uploaded in *CO-OP* system in experiment, comparing with cumulative amount of uploaded in centralized training system

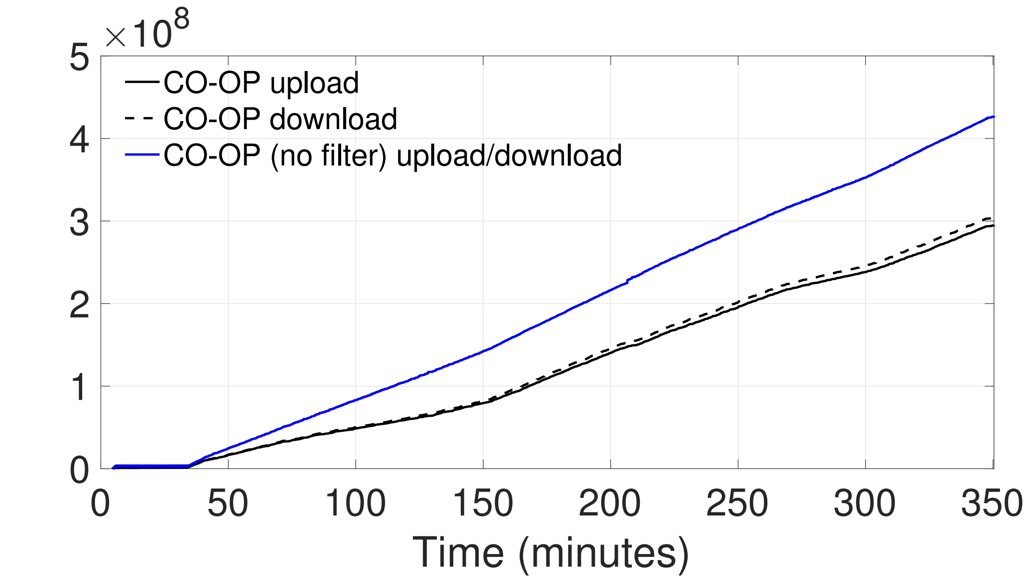

Fig. 5.9. The cumulative amount of data in bytes downloaded/uploaded in *CO-OP* system in experiment, comparing with the cumulative amount of data in *CO-OP* system without any filter

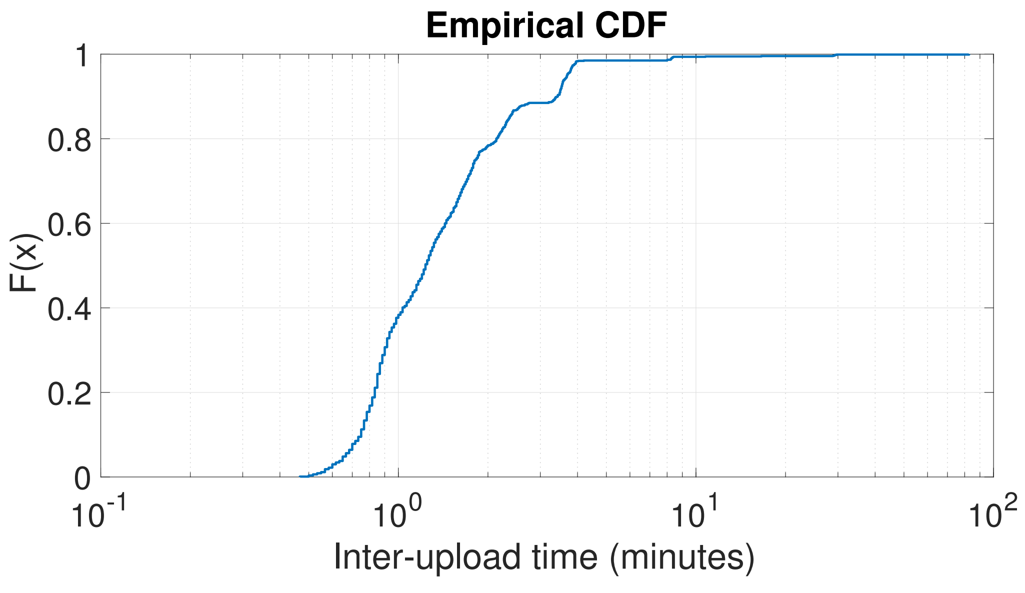

Fig. 5.10. The CDF of update time intervals of clients, with a mean time interval of 1.75 minutes, maximum time interval of 83 minutes, and minimum time interval of 0.5 minute

Chapter 6

Concluding Remarks

In this thesis, we propose *CO-OP*, a privacy-preserving asynchronous system that can train machine learning models based on data batches generated on mobile devices, without collecting these data to central server. Although a number of distributed machine learning algorithms and systems have been developed recently both in academia and in industry, most of them focus on training models on a cluster of co-located servers or on multiple datacenters each holding a part of the data. Federated learning advanced the concept of distributed machine learning to mobile devices with a synchronized Federated Averaging algorithm proposed.

In contrast, *CO-OP* is a completely asynchronous protocol, where the client initiates contacts to the server whenever it connects to a free Wifi network. Also, we consider the more realistic scenario where the data on each device are generated in batches on the go. *CO-OP* adopts an age-based filter to avoid unnecessary uploads and downloads in the case that the download is obsolete or if the updates from a certain client are too often. *CO-OP* also uses a carefully designed age-dependent weighting mechanism to merge qualified local models into the global model. Extensive simulation results have shown that *CO-OP* is capable of training neural networks, logistic regression and SVM models on several different datasets in a distributed setting.

We implemented *CO-OP* and tested the system on 60 Android clients, including mobile phones, tables, and Android emulators, distributed worldwide on the application of training hand-written digits recognition neural networks based on 60,000 MNIST data samples. Experimental results have demonstrated that *CO-OP* can yield a model accuracy of 80% on a separate test set of 10,000 samples even when the mobile clients are highly dynamic, some of which reside in bandwidth-limited environment. To the best of our knowledge, this is the first work that has reported experimental results from a fairly large number of real Android devices to investigate the feasibility of cooperative machine learning on mobile devices.

References

[1] T. Zhang*, Y. Wang*, C. Li, and D. Niu, "Co-op: Cooperative machine learning from mobile devices," submitted to INFOCOM 2018. *These two authors contributed equally to this paper.

[2] J. Poushter, "Smartphone ownership and internet usage continues to climb in emerging economies," *Pew Research Center*, vol. 22, 2016.

[3] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism," *Journal of network and computer applications*, vol. 39, pp. 319–333, 2014.

[4] L. O. Colombo-Mendoza, R. Valencia-García, A. Rodríguez-González, G. Alor-Hernández, and J. J. Samper-Zapater, "Recommetz: A context-aware knowledge-based mobile recommender system for movie showtimes," *Expert Systems with Applications*, vol. 42, no. 3, pp. 1202–1222, 2015.

[5] W.-S. Yang and S.-Y. Hwang, "itravel: A recommender system in mobile peer-to-peer environment," *Journal of Systems and Software*, vol. 86, no. 1, pp. 12–20, 2013.

[6] A. Vulimiri, C. Curino, B. Godfrey, K. Karanasos, and G. Varghese, "Wanalytics: Analytics for a geo-distributed data-intensive world." in *CIDR*, 2015.

[7] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints." in *NSDI*, 2015, pp. 323–336.

[8] I. Cano, M. Weimer, D. Mahajan, C. Curino, and G. M. Fumarola, “Towards geo-distributed machine learning,” *arXiv preprint arXiv:1603.09035*, 2016.

[9] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, “Large scale distributed deep networks,” in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

[10] H. Cui, A. Tumanov, J. Wei, L. Xu, W. Dai, J. Haber-Kucharsky, Q. Ho, G. R. Ganger, P. B. Gibbons, G. A. Gibson *et al.*, “Exploiting iterative-ness for parallel ml computations,” in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–14.

[11] Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing, “More effective distributed ml via a stale synchronous parallel parameter server,” in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[12] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, “Project adam: Building an efficient and scalable deep learning training system.” in *OSDI*, vol. 14, 2014, pp. 571–582.

[13] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, “Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server,” in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 4.

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

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

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

[17] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, "Federated multi-task learning," *arXiv preprint arXiv:1705.10467*, 2017.

[18] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[19] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy, "The unified logging infrastructure for data analytics at twitter," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1771–1780, 2012.

[20] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[21] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[22] C. tao Chu, S. K. Kim, Y. an Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng, "Map-reduce for machine learning on multicore," in *Advances in Neural Information Processing Systems 19*, P. B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 281–288. [Online]. Available: http://papers.nips.cc/paper/3150-map-reduce-for-machine-learning-on-multicore.pdf

[23] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.

[24] J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing, "Managed communication and consistency for fast data-parallel iterative analytics," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 381–394.

[25] Apache Spark MLlib., http://spark.apache.org/mllib/.

[26] Apache Mahout., http://mahout.apache.org.

[27] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu, "Petuum: A new platform for distributed machine learning on big data," *IEEE Transactions on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[28] M. F. Balcan, A. Blum, S. Fine, and Y. Mansour, "Distributed learning, communication complexity and privacy," in *Conference on Learning Theory*, 2012, pp. 26–1.

[29] Y. Zhang, J. Duchi, M. I. Jordan, and M. J. Wainwright, "Information-theoretic lower bounds for distributed statistical estimation with communication constraints," in *Advances in Neural Information Processing Systems*, 2013, pp. 2328–2336.

[30] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *International conference on machine learning*, 2014, pp. 1000–1008.

[31] T. Yang, "Trading computation for communication: Distributed stochastic dual coordinate ascent," in *Advances in Neural Information Processing Systems*, 2013, pp. 629–637.

[32] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč, "Adding vs. averaging in distributed primal-dual optimization," *arXiv preprint arXiv:1502.03508*, 2015.

[33] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[34] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[35] R. McDonald, K. Hall, and G. Mann, "Distributed training strategies for the structured perceptron," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 456–464.

[36] D. Povey, X. Zhang, and S. Khudanpur, "Parallel training of deep neural networks with natural gradient and parameter averaging," *arXiv preprint*, 2014.

[37] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems*, 2015, pp. 685–693.

[38] "About node.js," https://nodejs.org/en/about/, accessed: 2017-08-15.

[39] "Android developer guide: Processes and threads," https://developer.android.com/guide/components/processes-and-threads.html, accessed: 2017-08-16.

[40] "Android api reference: Asynctask," https://developer.android.com/reference/android/os/AsyncTask.html, accessed: 2017-08-16.

[41] "Android developer guide: Services," https://developer.android.com/guide/components/services.html, accessed: 2017-08-16.