Tests and Implementations of Synchronous and Asynchronous Gradients Aggregation Algorithms for Federated Learning on Google Cloud Platform

Sihui Wang
sihui_wang@sfu.ca
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

ABSTRACT

Motivated by designing a training and prediction network of *violence scene detection* for the community, in this project we investigated the system design requirements for such networks. We realized that networks of these kinds have certain features which make it difficult for them to fit into traditional distributed machine learning paradigms. We found out that *federated learning* might be the potential solution for machine learning on these kinds of networks.

In this project, we will mainly focus on the trade-off between communication efficiency and model accuracy in the design of gradient aggregation algorithms for federated learning. In federated learning, in order to obtain an accurate model from the heterogenous, non-i.i.d. data, we need to perform synchronization among the clients. Synchronization, however, creates the problem of stragglers. The pure asynchronous algorithms, on the other hands, are not desirable, either, because they compromise the accuracy of the models and create a heavy communication burden for the server. So, some researchers proposed to take a hybrid approach. In this project, we will implement some of the synchronous methods and asynchronous methods and test their performance on a small cluster of computers deployed on Google Cloud Platform.

KEYWORDS

Federated Learning; Gradient Aggregation Algorithms; Communication-Efficient; Synchronization; Google Cloud Platform

ACM Reference Format:
Sihui Wang. 2021. Tests and Implementations of Synchronous and Asynchronous Gradients Aggregation Algorithms for Federated Learning on Google Cloud Platform. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Initial Motivation

1.1.1 Violence Scene Detection. Let's consider building a *violence scene detection* network for detecting and predicting violence occurrence to reduce and prevent violence and crime for the community. In this network, different computers/devices have different roles:

A **server** receives pre-trained models for violence scene detection from national information center. It hands out the pre-trained models to the clients so that they can detect if any violence occurs. The server should also be able to receive feedback from the clients in order to fine-tune or improve the model.

Clients consist of numerous devices, such as surveillance cameras for shops and households. These devices are connected to local computers. Once they receive the pre-trained models from the server, they use that information for detection of violence scenes. However, sometimes the pre-trained models are flawed and can't fit well with local datasets. In this case, the local users report the mistakes and the clients will perform local training according to users' complaints. Eventually these local training results will be uploaded to the server so that the server can adjust its models.

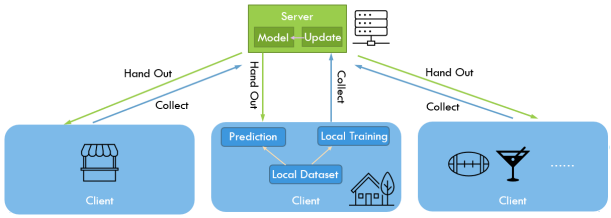

Figure 1: System Design for Violence Scene Detection

1.1.2 System Design Requirements. In our preliminary analysis, we find out that our project poses many challenges to system design. The major challenges are the following:

Scalability The *violence scene detection* network is built for the community. It should be able to provide service to a great number of clients, and it should allow a large number of new users to join in.

Communication Efficiency Since the server needs to provide service to and collect information from numerous clients, it can easily become the communication bottleneck. Hence, it is important that we should design communication-efficient frameworks for this project.

Data Privacy In our project, the information collected by surveillance cameras is privacy sensitive. It is henceforth required that clients should avoid sharing data or uploading data to the server.

Heterogeneity of resources From the perspective of a machine learning engineer, the data collected by each device are heterogeneous, meaning that they are non-i.i.d. data sets, which doesn't conform the general assumption that all data are sampled from the same statistical distribution. This data heterogeneity requires machine learning algorithms that are specifically designed for these scenarios.

Furthermore, numerous devices tend to have a very large variation in their performance. This heterogeneity of devices adds complexity to the design of the collaboration frameworks.

Un-Reliable Connection In our project, most of the clients are cameras and personal computers, and we can't expect that all devices should be available all the time. In our scenario, the server should be functioning well under the assumption that only a proportion of the clients are available.

1.2 Federated Learning

Based on previous discussions, we realized that our violence scene detection networks have certain requirements which make it difficult for them to fit into traditional distributed machine learning paradigms where training is performed centrally in the data center. In our scenario the storage and computation are likely to be decentralized, which conforms the definition of *federated learning* that data processing and training are performed at the "edge" nodes. Hence, we consider federated learning to be the most relevant paradigm for our purpose.

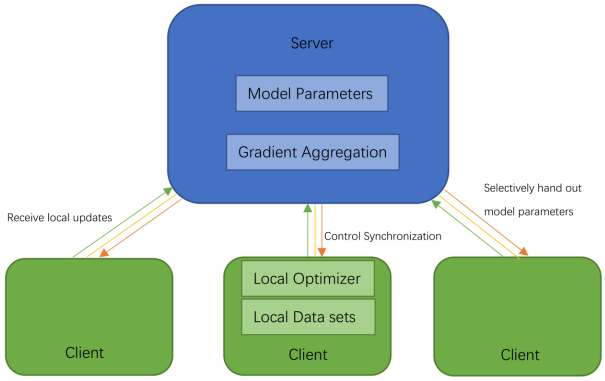

Figure 2: System Model for Federated Learning

1.2.1 Related Works. Since its proposal in 2016 [1] , Federated Learning has attracted wide attention from both academic and industrial communities. There are many challenges in the domain of Federated Learning [2]. In the aspect of optimization, there are challenges such as reducing the bias and variance of learned models, and increasing the convergence rate of optimization algorithms [3]. In the aspect of system design, there are challenges such as improving the efficiency of communication and resource allocation, enhancing robustness against adversarial devices, and ensuring fairness and data privacy. In this project, we will mainly focus on the trade-off between communication efficiency and model accuracy.

In Federated Learning, FedAvg [4] is a gradient aggregation algorithm which enables multiple clients who receive non-i.i.d. data to collaboratively train a model. In [5], the idea of FedAvg was further elaborated as "periodic averaging" and "partial node participation". [5] also introduced the mechanism of "quantized message-passing" to transfer the parameters in a compressed manner and proposed a variant of FedAvg, FedPAQ. These two algorithms are synchronous gradient aggregation algorithms, and the purpose of synchronization is to reduce the bias and improve the accuracy of the trained models with non-i.i.d. datasets [5].

However, due to unbalanced loads, and heterogeneity of the devices and resources in federated learning systems, synchronous gradient aggregation algorithms are prone to the problem of stragglers. To address the problem of stragglers, asynchronous gradient aggregation algorithms are proposed.

With asynchrony, a new problem arises: if all clients are allowed to communicate with the server asynchronously, then the server can easily become a communication bottleneck since it might have to deal with updates from tens of thousands of clients simultaneously.

To mitigate this problem, [6] [7] combine synchronous training and asynchronous training together. They implemented tier schedulers which divide the clients into tiers according to their latency metrics. Within the same tier, synchronous training is performed because stragglers are not likely to appear if all clients have similar performance. For cross-tier training, the parameters are updated asynchronously to avoid the problem of stragglers. Since asynchrony is happened on the tier level rather than the client level, it is manageable, and the server won't easily become the bottleneck. To reduce the bias of the trained models, they also proposed to assign higher weights for the slower tiers and lower weights for the faster tiers so that the trained model won't "skew towards" the data sets of the clients in the faster tiers.

1.2.2 Preliminary Solution. According to the system design requirements posed by violence scene detection networks, we proposed that our solution should have the following features:

Local Training The videos collected by cameras are large, and it is not realistic to upload them to the server because of the constraints in the aspect of communication. In addition, these videos can be privacy sensitive, and sharing the data among devices should be avoided. So, data should be stored locally and training should be performed by the clients themselves.

Reduced Synchronization Videos collected by different clients are considered to be non-i.i.d. data sets. To avoid the scenarios that clients work separately ending up with training a number of different models, it is required that certain kind of synchronization is performed to guarantee that a single unbiased model is produced.

On the other hand, in order to reduce communication overhead, we also want to avoid the scenario where clients perform synchronization and update to the server frequently. We have to find an optimal trade-off between communication efficiency and model accuracy.

Partial Participation of Clients For one thing, not all devices are available all the time in our scenario. For another thing, we also want to reduce the communication overhead by allowing the server

to selectively connect to only a small proportion of the clients each time.

Quantized Parameters To reduce the communication overhead, we allow the clients to transfer only the difference between the old and the new model parameters, and we only transfer the quantized difference in order to further compress the data to be transferred.

Mechanisms to Mitigate the Problem of Stragglers To deal with stragglers without large communication overhead, the server should routinely divide the clients into different tiers according to their latency metrics. While intra-tier training is performed in a synchronous manner, inter-tier training should be performed asynchronously. To guarantee that the final model is unbiased, we need to assign more weights to the slower tiers while assign fewer weights to the faster tiers.

1.3 Refined Scope of this Project

On one hand, federated learning has many open problems and it takes a lot of time and resources to conduct experiments and build a system that can live up to the real world requirements. On the other hand, we only have limited time and resource quotas on Google Cloud Platform. This is the reason why we want to narrow down the scope of our project.

In this preliminary stage, we will use data sets such as MNIST for experiments; We will focus on implementations and tests of synchronous and asynchronous gradient aggregation algorithms and compare their performance in terms of training time (which is an indicator of communication efficiency) and model accuracy (which measures the trade-off). We will consider two scenarios: 1) all clients have equal capacities and it's not likely to have stragglers in the system; 2) clients have unequal capacities and it's likely that stragglers will appear in the system.

In this project, the objectives of this project are defined as follows:

1) To implement synchronous gradient aggregation algorithms, such as FedAvg and FedPAQ;

2) To implement gradient aggregation algorithms with asynchronous components;

3) To test and compare their performance in simplified real-world environments on Google Cloud Platform.

4) To test and compare their performance when there are stragglers.

2 SOLUTION DESIGN

2.1 Federated Learning with Synchronous Gradient Aggregation Algorithms

Meta Data The server stores all registration information of the clients in this part. This helps the server make connections with the clients. Here the server also stores global parameters, such as the interval of two consecutive synchronizations. This configures how the federated learning process is organized.

Model Parameters The server stores the parameters of the models here. The server routinely sends these parameters to the clients for training and receives updated parameters from the gradient aggregator.

Client Selector In each turn, client selector randomly selects some clients according to the registration information. Once the selection is made, the client selector informs that the server should send model parameters and relevant meta data to corresponding clients to start the round of training.

Counter The clients receive meta data and learn the interval of two synchronizations. This counter judges whether the client has finished this round of training.

Local Data The training data sets are stored here. Once the clients are ordered to perform training, they load data sets from here.

Local Optimizer The clients can implement optimization algorithms such as SGD here to train the model.

Cache Once the client finishes the work for this round of training, it sends the local updates of the parameters to the cache of the server and waits for further instructions from the process scheduler.

Synchronizer Once the server receives all local updates from all designated clients, the synchronizer informs the clients that this round of training is ended.

Process Scheduler The process scheduler receives instructions from the synchronizer and informs the clients whether or not to block the training process.

Gradient Aggregator Once the round of training is finished, the gradient aggregator loads all local updates of model parameters and performs gradient aggregation algorithms. Once gradient aggregation is done, gradient aggregator updates the model parameters.

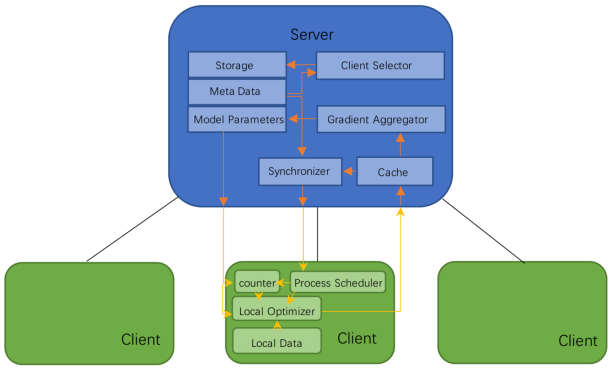

Figure 3: System Architecture of Federated Learning with Synchronous Gradient Aggregation Algorithms

2.2 Federated Learning with Asynchronous Gradient Aggregation Algorithms

Tier-based asynchronous federated learning systems are actually hybrid systems with both synchronous and asynchronous components. The synchronous systems are Layer-2 systems in which the clients and the server are communicating with each other synchronously. The tier-based asynchronous systems, on the other hand, are Layer-3 systems in which an additional layer of tier managers are regulating the communication between the clients and the server. In tier-based asynchronous systems, the clients are divided into clusters according to their latency metrics. Each cluster of clients who share similar performance with each other, are communicating with the tier manager synchronously. Since there is little performance variance within each cluster, the problem of stragglers is mitigated. The tier managers communicate with the server

asynchronously. Compared with the system design in which the clients are allowed direct, asynchronous communication with the server, here in the tier-based systems only a small number of tier managers are allowed direct communication with the server, which efficiently reduces the communication overhead for the server. So, the design of intra-tier synchronous communication and the inter-tier asynchronous communication solves the problem of stragglers for synchronous systems without incurring much communication overhead because of asynchronous communications.

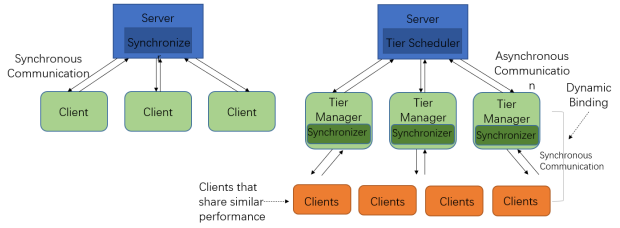

Figure 4: Comparison between System Designs of Synchronous and Asynchronous Federated Learning

Additional components and mechanisms are needed to support the functionality of tier-bases asynchronous systems.

Tier Evaluator The tier evaluator manages the dynamic binding between clients and tier managers. It routinely tries to make connections to each registered client. It divide the clients into tiers according to the latency metrics and allocates tier managers to clusters of clients. Once the tiers are made, the information will be updated for the synchronizer, so that the synchronizer can take control of the intra-tier synchronization. Tier Evaluator also sends the latency metrics to the weight adaptor so that the weight adaptor can assign different weights to different tiers.

Weight Adaptor The weight adaptor receives latency metrics from the tier evaluator. Then it generates weights for each tier, so that the gradient aggregator will obtain a weighted average of local updates from each tier. The motivation here is to offset the effect that faster tiers will have larger impact on training if vanilla averaging of gradients is performed.

Model Status Manager In asynchronous system designs, time stamps are required to be transferred together with the model parameters, so that the server can take control of the staleness of the model. Usually the server has a staleness tolerance parameter. In asynchronous systems, the server can to some degree tolerate out-dated model parameters. However, the server will throw away the old parameters and wait for new ones if the staleness of the old parameters exceed certain threshold. If the server has zero tolerance toward staleness, then the system becomes a synchronous one.

3 IMPLEMENTATION

3.1 Platform

In this project, we implemented both synchronous and asynchronous algorithms and conducted tests on Google Cloud Platform.

For synchronous algorithms, we have deployed them on 4 virtual N1 machines with v2 CPU and 7.5GB memory on Google Cloud Platform. One virtual machine takes the role of the server, and the other three take the role of clients.

For the asynchronous algorithm, we have deployed it on another 4 virtual N1 machines with v2 CPU and 7.5GB memory on Google Cloud Platform. Because of the resource quotas, we can't use more than 4 virtual machines. So, this deployment only simulates the asynchronous communications between one server and three clients.

Previously we planned to set up 4 virtual machines with different hardware configurations in order to simulate the heterogeneous environment. Our previous plan is:

Server: N1 machine, v1 CPU, 3.75GB memory;
Client1: N1 machine, v4 CPU, 15GB memory;
Client2: N1 machine, v2 CPU, 7.5GB memory;
Client3: N1 machine, v1 CPU, 3.75GB memory.

We carried out experiments and found out that this heterogeneity of hardware configurations didn't create the straggler problems as we once expected. So, we turned to other methods to generate the problem of stragglers in synchronous training, and on Google Cloud Platform we no longer preserve these 4 virtual machine instances because of resource quotas.

3.2 Software Dependencies

The synchronous algorithms are built upon the federated learning package Flower [8]. For our implementation, Ubuntu 20.04 LTS, Pytorch 1.10.0 (CPU version), and pip3 are required.

The asynchronous algorithm is the implementation of [9], and the code is adapted from[10]. This implementation is built upon another federated learning package, PySyft. In our implementation, Ubuntu 20.04 LTS, Pytorch 1.4.0, pip3, and PySyft 0.2.4 are required.

3.3 System Implementation Design

3.3.1 Flower Framework for Synchronous Federated Learning. In Flower, the clients connect to the server via the client proxies. Once the connection is established, the client proxy makes a registration to the client manager, signaling that the client is available. The server will block itself until minimal number of clients become available. Once that minimal requirement is met, the server unblocks itself and proceeds to sample from the clients. By sampling, the server starts one communication round with a subset of the clients.

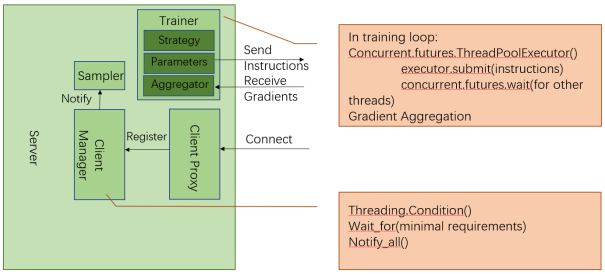

Figure 5: Flower's System Design for Synchronous Federated Learning

In one communication round, the server starts multiple threads, each takes control of one client. Each thread sends the model parameters and training instructions to the client, and the server

will block itself and wait for all updates from the clients so that it can proceed to perform gradient aggregation. One communication round is completed once the server finishes gradient aggregation for this round.

3.3.2 Asynchronous Federated Learning Based on PySyft. In PySyft-based implementation [10], the server and the clients connect with each other by the PySyft objects, *WebSocketServer* and *WebSocketClient*. The server starts multiple threads, each of which takes control of sending training instructions to one client. The server and the clients pass the model parameters by python MPI's message *Queue* mechanism. In the server, the evaluator is a process who regularly sends model parameters to the clients, checks for updates of model parameters from the clients, and monitors the staleness of the model parameters.

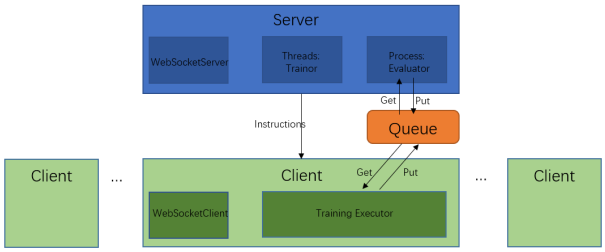

Figure 6: System Design for Asynchronous Federated Learning Based on PySyft

3.4 Implementation of Gradient Aggregation Algorithms

3.4.1 FedAvg. FedAvg is provided by the library, Flower. In this project, we use FedAvg as a benchmark for tests and experiments.

3.4.2 FedPAQ. In this project, my work is to implement FedPAQ and add FedPAQ as a *strategy* to the Flower library.

According to [5], FedPAQ should have three extra components compared with FedAvg: partial participation of clients, reduced level of synchronization, and quantized transfer of parameters. Since partial participation has been implemented for FedAvg in Flower, I mainly focus on 1) reduced level of synchronization and 2) quantized transfer of parameters in my implementation for FedPAQ.

In my implementation, FedPAQ is a *strategy* class who takes FedAvg as the parent class. I added two keywords for FedPAQ's configurations: "sync level" and "precision", who take charge of reduced level of synchronization and quantized transfer of parameters. To enable FedPAQ's implementation, I also adapted the code both in the library of Flower and on the user side accordingly.

3.4.3 Asynchronous Algorithm. In this project, the asynchronous algorithm is borrowed from [10]. The original code is performed locally in a PC for simulation purpose. I made minimal change to the code so that it can run on multiple machines on Google Cloud Platform. I use this asynchronous algorithm to test if asynchronous algorithms can solve the problem of stragglers in Section 4.

4 EVALUATION

For each algorithm to be tested, we collect two metrics: the total training time and the model accuracy. The total training time reflects the communication overhead of the algorithms, whereas the model accuracy indicates the trade-off between model performance and communication efficiency.

Total training time can be tracked by the logging component of the server, whereas model accuracy can be evaluated by clients' local optimizers on each local data set.

For the asynchronous algorithm, logging is taken care of by a package, yappi.

4.1 Performance evaluation of synchronous gradient aggregation algorithms under homogeneous environments

The hardware configurations are discussed in Section 3.1 (4 virtual N1 machines with v2 CPU and 7.5GB memory). Here we tested FedAvg's validation loss curve for the first 30 epochs (Figure 7), and FedAvg's total training time for 30 epochs' federated learning.

To test FedPAQ's communication efficiency, I tested how different level of synchronization and different level of clients' participation would impact the total training time (communication efficiency) and the validation loss (model accuracy) for 30 epochs. The results are shown below.

From the figures we can see that reduced level of synchronization significantly reduces the total training time and improves the communication efficiency (Figure 9). Contrary to general expectations, in our experiments, the gain in communication efficiency is at little cost of model accuracy (Figure 8). This is partly due to the fact that in the experiments we used MNIST dataset which might be unable to simulate non-i.i.d. dataset in federated learning. We will consider the use of real-world dataset in future to improve the experiment.

For clients' partial participation, I tested the training time and model accuracy when 1,2,3 client(s) is (are) involved in gradient aggregation in each communication round. The results show that full participation group tends to have longer training time and higher model accuracy. Less participation of clients, however, will

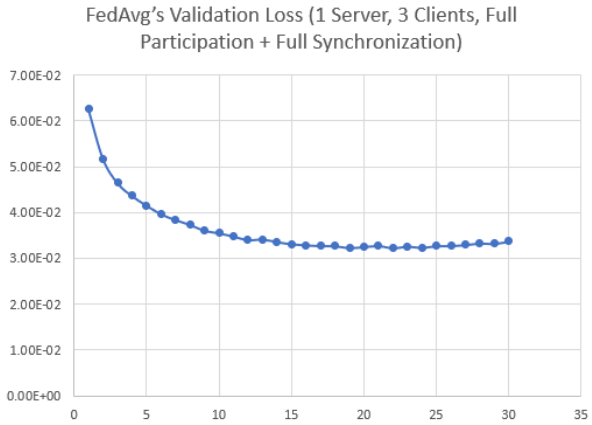

Figure 7: FedAvg's validation loss over 30 epochs

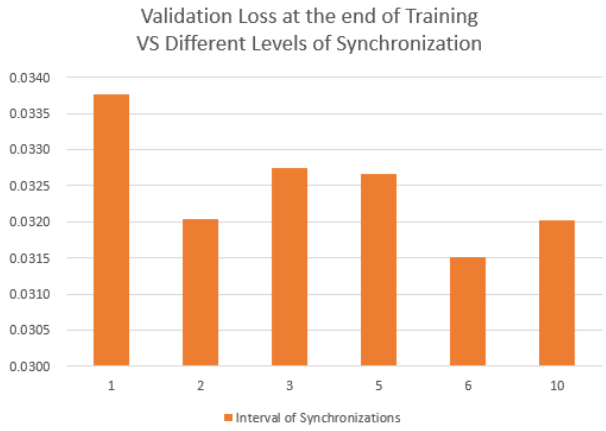

Figure 8: Reduced Synchronization's impact on Model Accuracy

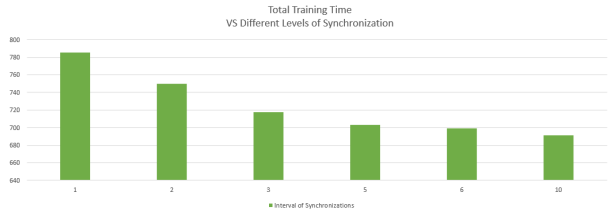

Figure 9: Reduced Synchronization's impact on Training Time

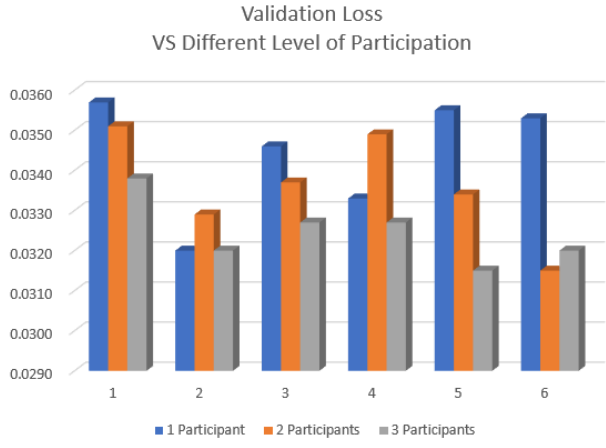

Figure 10: Reduced Participation's impact on Model Accuracy

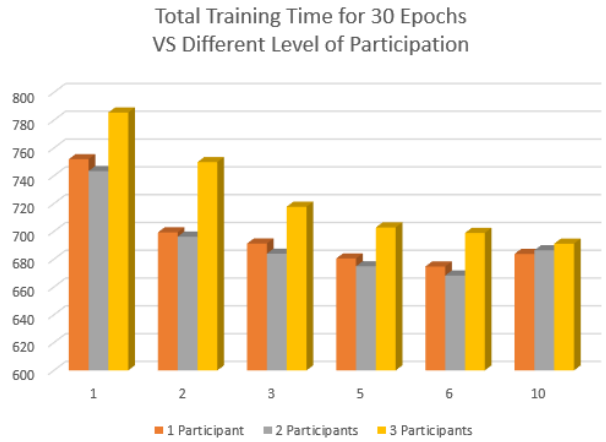

Figure 11: Reduced Participation's impact on Training Time

save training time at the cost of model accuracy.

So, in our experiments, reduced synchronization can enhance communication efficiency without significant loss of model accuracy. Reduced participation, on the other hand, will improve communication efficiency at the cost of model accuracy.

4.2 Performance evaluation of synchronous gradient aggregation algorithms under heterogeneous environments

At first, I carried out experiments on 4 virtual machines with different hardware configurations on Google Cloud Platform, as was discussed in Section 3.1. However, it turns out that this heterogeneity is not enough to induce the problem of stragglers.

So, to simulate the slow devices, I manually add the code to the stragglers' processes so that they will sleep for a random period of time during training. Using this method to simulate the stragglers' behaviour in the system, I obtained the results in figure 12.

From the figure below we can see that even one straggler can significantly reduce the overall performance of the system. This is the biggest problem for synchronous federate learning, because

Figure 12: Straggler's impact on Training Time

this limits synchronous federated learning's ability to scale up and deal with heterogeneity of devices.

4.3 Performance evaluation of asynchronous gradient aggregation algorithms under heterogeneous environments

In this section, I manually add codes to adapted version of [10] to simulate the straggler's effect. In my simulations, there are one server and three clients, and one of the clients is the straggler who sleeps for a random period of time during training. In this code, I can set the staleness tolerance parameter to zero to simulate the synchronous training, and I can set the staleness tolerance parameter very high to simulate the totally asynchronous training. So, I can use the code [10] to conduct a relatively fair comparison between synchronous and asynchronous federated training performance when stragglers are presented in the system.

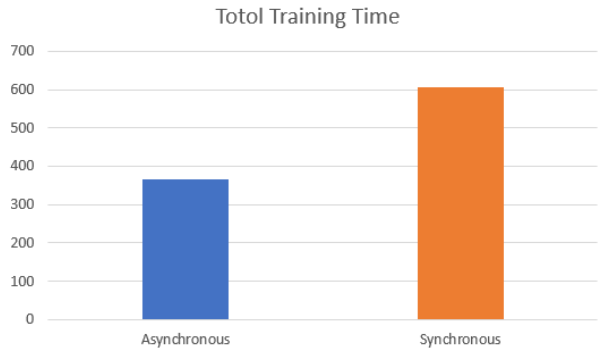

Figure 13: Straggler's impact on Training Time

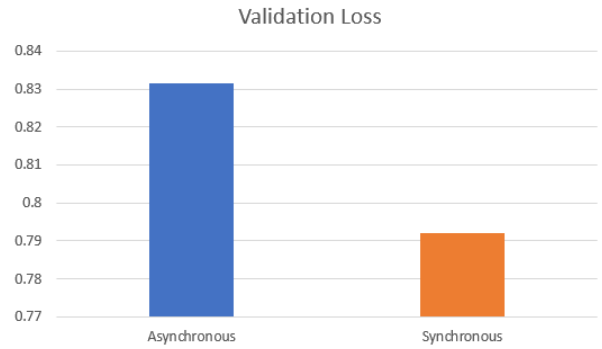

Figure 14: Straggler's impact on Validation Loss

From the figures we can see that asynchronous algorithms can significantly improve the communication efficiency, and the loss of model accuracy due to asynchronous training is small. With more techniques implemented to offset asynchronous training's side effect on model accuracy, we can expect that asynchronous training will be a promising method for improving communication efficiency in federated learning.

5 CONCLUSION

In the project, we implemented part of synchronous and asynchronous gradient aggregation algorithms for federated learning. We implemented clients' partial participation, reduced level of synchronization, and tested their impact on communication efficiency and model accuracy. We also tested both synchronous and asynchronous algorithms' performance with and without the presence of stragglers. The general conclusion is that the techniques such as partial participation, reduced synchronization, and asynchronous training, can all improve communication efficiency with the price of loss of model accuracy.

Asynchronous federated learning is a novel approach which leads to many open problems in federated learning. In this project, we only did minimal work on asynchronous federated learning, due to the difficulties in the implementations. However, this is an intriguing problem and worth doing research in the future.

REFERENCES

[1] H. Brendan McMahan Felix X. Yu Peter Richtárik Ananda Theertha Suresh Konečný, Jakub and Dave Bacon. Federated learning: Strategies for improving communication efficiency. NIPS Workshop on Private Multi-Party Machine Learning, 2016.

[2] H. Brendan McMahan Brendan Avent Aurélien Bellet Mehdi Bennis Arjun Nitin Bhagoji Kallista Bonawitz et al. Kairouz, Peter. Advances and open problems in federated learning. arXiv preprint arXiv:1912.04977, 2019.

[3] Zachary Charles Manzil Zaheer Zachary Garrett Keith Rush Jakub Konečný Sanjiv Kumar Reddi, Sashank and H. Brendan McMahan. Adaptive federated optimization. arXiv preprint arXiv:2003.00295, 2020.

[4] Eider Moore Daniel Ramage Seth Hampson McMahan, Brendan and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. pages 1273–1282. Artificial intelligence and statistics, 2017.

[5] Aryan Mokhtari Hamed Hassani Ali Jadbabaie Reisizadeh, Amirhossein and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. pages 2021–2031. International Conference on Artificial Intelligence and Statistics, 2020.

[6] Ahsan Ali Syed Zawad Stacey Truex Ali Anwar Nathalie Baracaldo Yi Zhou Heiko Ludwig Feng Yan Chai, Zheng and Yue Cheng. Tifl: A tier-based federated learning system. pages 125–136. Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020.

[7] Yujing Chen Ali Anwar Liang Zhao Yue Cheng Chai, Zheng and Huzefa Rangwala. Fedat: a high-performance and communication-efficient federated learning system with asynchronous tiers. The International Conference for High Performance Computing, Networking, Storage, and Analysis., 2021.

[8] Taner Topal Akhil Mathur Xinchi Qiu Titouan Parcollet Pedro PB de Gusmão Beutel, Daniel J. and Nicholas D. Lane. Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390, 2020.

[9] Senapati Sang Diwangkara and Achmad Imam Kistijantoro. Study of data imbalance and asynchronous aggregation algorithm on federated learning system. International Conference on Information Technology Systems and Innovation (ICITSI), IEEE, 2020., 2020.

[10] https://github.com/diwangs/asynchronous-federated-learning.