ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

REPORT

SCALABILITY OF FEDERATED LEARNING

EPFL SEMESTER PROJECT

Peng Ren

8th January 2022

ABSTRACT

This paper investigates the various influencing factors that affect the performance of the Federated Learning. The research was carried out with experiments based on our extended TFF framework and related work. We extend and modify TFF framework, make it support distributed multi-machine runtime to speed up and optimize simulation. Designing experiments concentrate on exploring the effect of fraction (C), the effect of scalability of devices simulating machine, the comparison between edge pattern and central pattern. Making conclusion for experiments results. Finally, we comprehensive discuss the impact of various factors under real world application scenario and simulation experiment scenario, some future work directions is proposed.

CHAPTER 1

INTRODUCTION

1.1 EXPERIMENT CONTEXT

With the rapid development of the mobile internet, mobile devices, represented by mobile phone, wearable devices and autonomous vehicles, are becoming increasingly popular. These devices generate huge amounts of data every minute of every day through the user's operations and the large number of sensors attached to the device. Sources of data may include, for example, data generated by users typing, browsing the web, taking photos, talking and GPS themselves.This data is the best source of data for machine learning and can greatly improve the ease of usability of mobile devices and increase machine intelligence.

The traditional way is to collect data to a central server and train it, but there are three serious drawbacks to this approach. The first is that this data is often highly privacy-sensitive, and collecting it on a central server would violate the user's personal privacy. The second is that the communication cost in collecting data to the central server and the storage pressure on the server are very high. The transfer of large amounts of data to the server can seriously impair the user experience. The third is storing the whole dataset on a single node becomes infeasible, it is a much more efficient and sensible option to let data stored in a decentralised manner on different devices.

To solve such three problems, Federated Learning was brought up. Concisely described, the server will somehow(such as independent initialization or common initialization) initialise the machine learning model that we want to optimise. The server will broadcast this model to devices participating in training, then after several epochs of training, devices will send an update $\delta \in \mathrm{R}^d$ to server, where d is the dimension of the model being computed/improved. When server receive the update, some algorithms, such as Federated Averaging algorithm, will aggregate updates from devices and use result to update global model on the server. Iterating through this process until the global model reaches the specified accuracy[1][2].

The data and devices involved in Federated Learning has these characteristics.

* Non-IID: Based on factors such as different users, time and geographical location, data on different devices is generated from different distributions. The data on a single device is not representative of the global data.
* Unbalanced: Based on the user's behaviour. The amount of data available on different devices can vary considerably. And the amount of data on different devices is likely to have a long-tail distribution.
* Massively distributed: The number of devices participating in training will be much larger than the average data amount per device.

* Limited communication: The communication environment of the device is unstable and may be offline most of the time (not on a wifi network), but the device may have a large amount of idle computing resources available for computing.

* System capabilities: Due to variability in hardware, network connection and power, the storage, computational and communication capacities of devices may vary significantly.[3]

In this general context, Google introduced a framework TensorFlow Federated (TFF), which is used for machine learning and other computations on decentralized data, enables developers to simulate the Federated Learning algorithms on their models and data.

1.2 EXPERIMENT DESIGN

This project code is based on open-source framework TensorFlow Federated (TFF), but the problem is TFF now does not have a multi-machine distributed runtime, when we perform a simulation, we can only simulate the Federation Learning Server and its corresponding devices on one machine.

There are two main disadvantages of this single machine runtime simulation, the first is simulation on single machine runtime is extremely slow, when running the TFF framework on multiple nodes, simulations should be sped up considerably. The second is that single machine runtime does not simulate well the core behaviour in Federated Learning - communication, simulate on multiple nodes allow us to customise the communication patterns, synchronisation, network topology, etc. between servers and devices (which in the simulation means different nodes).

TFF's interfaces are organized in two main layers: Federated Learning (FL) API and Federated Core (FC) API, FL is a set of high-level interfaces, FC is a set of lower-level interfaces, it can expressing federated algorithms by combining TensorFlow with distributed communication operators within a strongly-typed functional programming environment. We mainly make use of the FC API.

I designed two patterns for multi-machine runtime, edge pattern and central pattern. For example, we use four machines to simulate hundreds of mobile devices and one machine to simulate a server, if we use central pattern code, each simulated device will send the updated model parameters directly back to the server, if we simulate 50 mobile devices per machine in charge of simulating the devices, the server will receive 200 different copies of the updated model parameters, the server then do Federated Averaging for these two hundred different updated model parameters and the resulting model is used as the global model.

For "edge pattern", each machine in charge of simulating the mobile devices can be seen as a combination of dozens of devices simulated and an virtual edge server. Firstly, the server simulating machine generates the initial model , and then sends the parameters of the initial model to the machine in charge of simulating the mobile devices. When each device return the updated model parameters, the edge server will first do an averaging of the updated model parameters returned by the devices simulated by this machine, then send the result of averaging to the server. The server only need do Federated Averaging for 4 updated model parameters returned by 4 edge server. This virtual edge server will also be in charge of tasks such as broadcasting the latest global model sent by the server to the devices selected for training in the current round. This pattern will significantly reduce the communication load and the computational workload of the server when doing Federated Averaging of the updated parameter models for all involved mobile devices in current round.

Each experiment will be assessed using three metrics

* The number of communication rounds required for the model to converge to the specified accuracy.

* The total time required for the model to converge to the specified accuracy.
* The total data transfer load for the model to converge to the specified accuracy.

1.3 EXPERIMENT GOALS

This project has four main goals.

The first one is build a distributed multi-machine runtime in python which utilizes TFF, aim to speeding up and optimize the simulation and customising the communication patterns, synchronisation, network topology, etc.

The Second one is to gain insights into the training process of Federated Learning, precisely to understand the impact of client fraction (C) on the convergence speed and overall performance, finally validate the hypothesis that the marginal benefit of increasing client fraction decreases and eventually becomes zero after sufficient number of clients participate. In the meantime, investigate other factors that may affect performance of Federated Learning .

The third one is to explore scalability of simulation machines, during the experiment we will simulate the same total number of clients on different number of device simulation machines and study the difference in their performance, the purpose of this part is to investigate the impact of the scalability of the simulation machine on the performance of Federated Learning simulation, prove multi-machine distributed runtime has a significant speed advantage compared to single machine runtime, and based on the laboratory environment, a multi-machine distributed runtime is a better implementation than a single machine runtime.

The fourth one is evaluating the differences between the 'edge' and 'central' simulation patterns, assessing the performance, advantages and disadvantages of each pattern.

CHAPTER 2

RELATED WORK

In paper *Communication-Efficient Learning of Deep Networks from Decentralized Data* by H. Brendan McMahan et.al[4], the concept of Federated Learning is presented. In addition to exploring the fundamental problems of Federated Learning and the definition of data, they also present the FederatedAveraging algorithm, which illustrates the training and aggregation process of Federated Learning and their formula definition. They also explored the various factors that affect the performance of Federated Learning.

Firstly, they investigate the effects of increasing parallelism, in other words, increasing fraction. they conclude from figure 2.1, increasing the client fraction has a small advantage, C = 0.1 is a good trade-off between computational efficiency and convergence rate. Secondly, they investigate the effects of increasing computation per client by decreasing batchsize or increasing epoch.The results is in figure 2.2, conclusion is increasing computation can achieve significant decrement on the number of rounds to reach the target accuracy. For the difference between IID dataset and Non-IID dataset, authors think that may because some local dataset can be very valuable and representative.

The other two things they proposed is different initialize methods affects the performance of Federated Learning and model averaging produces a regularization benefit similar to that achieved by dropout.

To facilitate experiments and simulations for Federated Learning, A number of benchmark and experimental datasets have also been proposed, such as LEAF[5]. LEAF contains 6 datasets that have been processed into a federated format, categories include image classification and NLP tasks, and the number of data points is skewed across devices. LEAF also established a rigorous evaluation metrics to assess how a learning solution behaves in federated scenarios. There are two interesting conclusion from LEAF paper, the first one is about the minimum number of samples per user, as we can see from the left side of figure 2.1, the larger the amount of data on each client, the more we can ensure that the vast majority of users (>75%) get better predictions or classification results when using this model. In other words, model is representative of data on most devices.

The second is system analysis, which is similar to my performance analysis, the two metrics is total number of FLOPS across all devices and total number of bytes uploaded to network. we can see from the right side of figure 2.3. When keep all other conditions same, only increase the number of epoch, although the computation resource consumed increase a lot, but the amount of communication data has also been reduced. This is a valuable enhancement for practical application scenarios. The experiment pattern is a central-like pattern, so we can see if increase fraction(C), computation resource consumed and the amount of communication data both increase. So combine this result and our experiment result, there is a very important conclusion, it is very important to choose a suitable fraction to minimise the number of communication rounds required to reach convergence while minimising the computational and communication load on the devices and servers.

2NN	IID		NON-IID	
C	$B = \infty$	$B = 10$	$B = \infty$	$B = 10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E = 5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

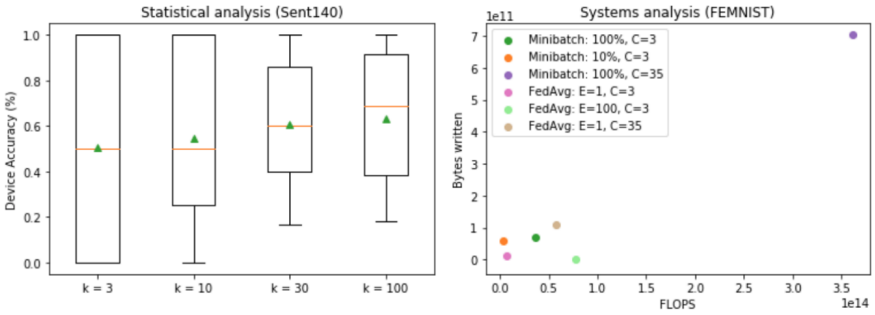
H

FIGURE 2.1
effects of parallelism

MNIST CNN, 99% ACCURACY					
CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)
SHAKESPEARE LSTM, 54% ACCURACY					
LSTM	E	B	u	IID	NON-IID
FEDSGD	1	∞	1.0	2488	3906
FEDAVG	1	50	1.5	1635 (1.5×)	549 (7.1×)
FEDAVG	5	∞	5.0	613 (4.1×)	597 (6.5×)
FEDAVG	1	10	7.4	460 (5.4×)	164 (23.8×)
FEDAVG	5	50	7.4	401 (6.2×)	152 (25.7×)
FEDAVG	5	10	37.1	192 (13.0×)	41 (95.3×)

H

FIGURE 2.2
effects of increasing computation per client



H

FIGURE 2.3
metrics plot from LEAF paper

The third one is different modeling approaches may be more or less appropriate for different federated datasets.

The paper *Federated Optimization: Distributed Machine Learning for On-Device Intelligence*[1] presents and describes a number of federated optimisation algorithms and compares the performance of these federated optimisation algorithms. This paper also proposes both synchronous and asynchronous optimisation strategies and suggests that in the future multiple models can be considered to be trained simultaneously, a global model based on global data and a personalised model based on local data.

Paper *A Performance Evaluation of Federated Learning Algorithms*[6] also do much performance test for Federated Learning, they use three Federated optimization algorithms, Federated Averaging (FedAvg), Federated Stochastic Variance Reduced Gradient and CO-OP. The dataset they use is iid-MNIST and non-iid-MNIST. They compare and prove two things, the first being that the FedAvg algorithm has top performance against most datasets, and the second proving that Federated Learning still has weaker performance against non-iid data compared to traditional data-centre style training methods.

CHAPTER 3

3-EXPERIMENTS RESULTS

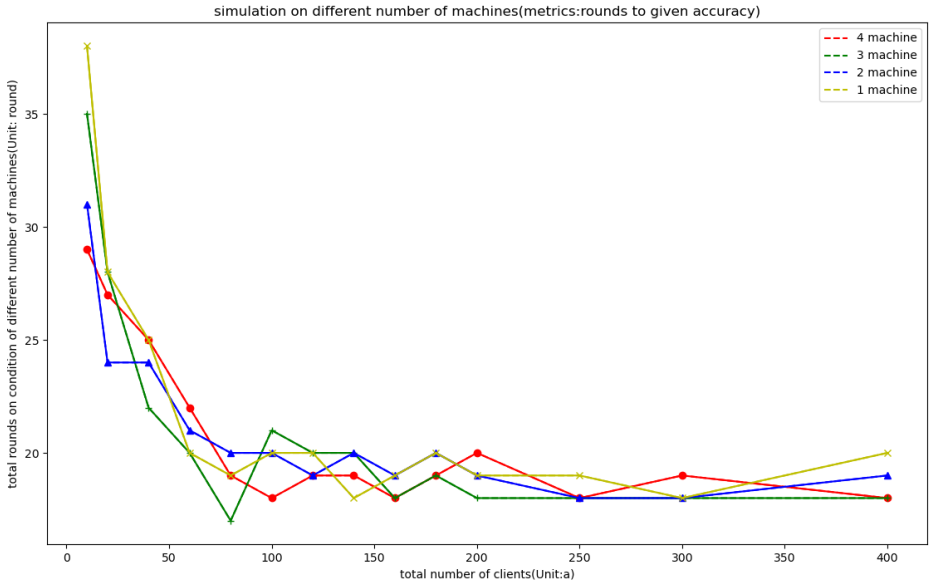
3.1 EXPERIMENT SETTINGS

* Data: Federated EMNIST data, the data is distributed to 3383 clients in a Non-IID manner.
* System: Ubuntu 20.04.2, Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz, 128 GB RAM.
* Framework: tensorflow 2.3.4, tensorflow-federated 0.17.0.
* Hyper-parameter: epoch = 1, batchsize = 5.
* Parallel mechanism VS tandem mechanism: I use a threading mechanism, which allows for a fully parallel relationship between the each device simulation machines. Since the code is based on the TFF framework, I presume that the individual devices within the device simulation machine are in a tandem relationship. The map function in the TFF framework causes the device to perform the training process one by one. If we use central pattern to train, the communication between devices and server also is tandem relationship. If using edge pattern to train, this is not a problem, as the communication process between the device and the server can be seen as parallel.
* Clients selection: In each communication round, the server randomly selects a designated number of clients from the 3383 clients for training, the horizontal axis indicators is the Total number of clients participating to each training round.

3.2 IMPACT OF CLIENT FRACTION (C) ON THE CONVERGENCE SPEED AND OVERALL PERFORMANCE

The number of communication rounds is our most important performance evaluation metric. The federated version of emnist data used for this experiment simulated 3383 client devices. From any curve of figure 3.1 and 3.4 we can see, for arbitrary number of device simulation machine, when there are few client fractions involved in each round, many communication rounds (30+) are required to converge to an accuracy of 0.6. When we increase the total number of clients involved in training to roughly 100 per round, the number of rounds required for convergence is roughly between 18-20, and the number of rounds required for convergence does not decrease as we continue to increase the number of clients involved in training per round.

So in terms of the number of communication rounds required for convergence as a metric, we can conclude the marginal benefit of increasing client fraction decreases and eventually becomes zero after sufficient number of clients participate. In other words, the knowledge and information contained in 100 clients is



H

FIGURE 3.1
rounds of converge to given accuracy between different number of client simulation machine

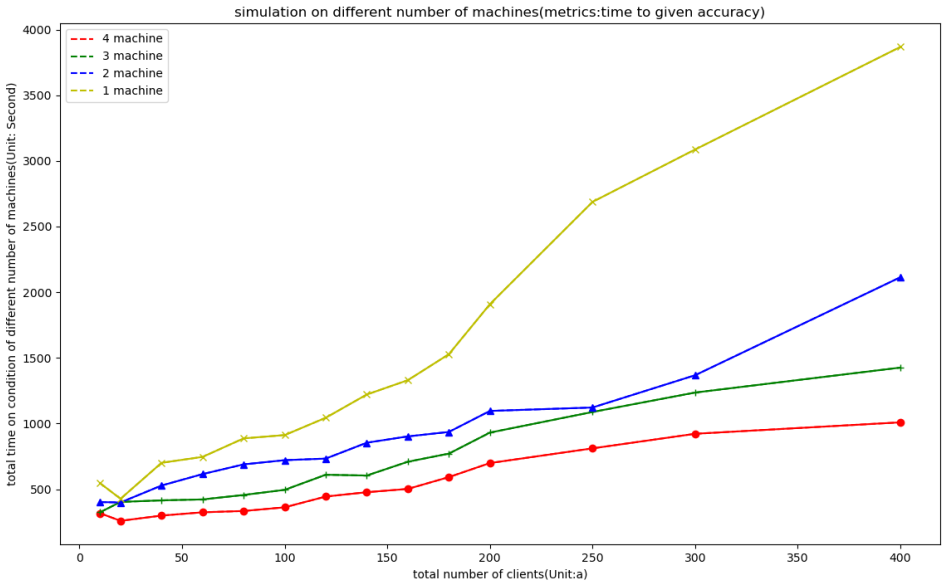
sufficient to represent all the data, and this information is sufficient for the model to learn. When we raise the Client Fraction, the model does not receive more knowledge and information to learn.

This point is also supported by the findings of fraction shown in other papers, for example, we can conclude from the observation in Figure 2.1, that increasing fraction leads to a benefit regarding the number of communication rounds, but with a decreasing trend as fraction increases. The second example is in figure 2.3, by using FedAvg algorithms, keep epoch same as 1, if we increase the fraction about 10 times, the increment of communication data load is lower than 10 times, this also prove the benefit brought from increasing fraction.

In terms of the metric total time consuming to given accuracy, from any curve of figure 3.2 and 3.5 we can see, regardless of the number of machines on which simulations were performed, the total time consumed increased when increasing the number of clients involved in each round of training. But the slope of the time increase was lower when the number of clients was smaller, and the slope of the time increase higher when the number of clients grew above 100. The explanation of this phenomenon is following.

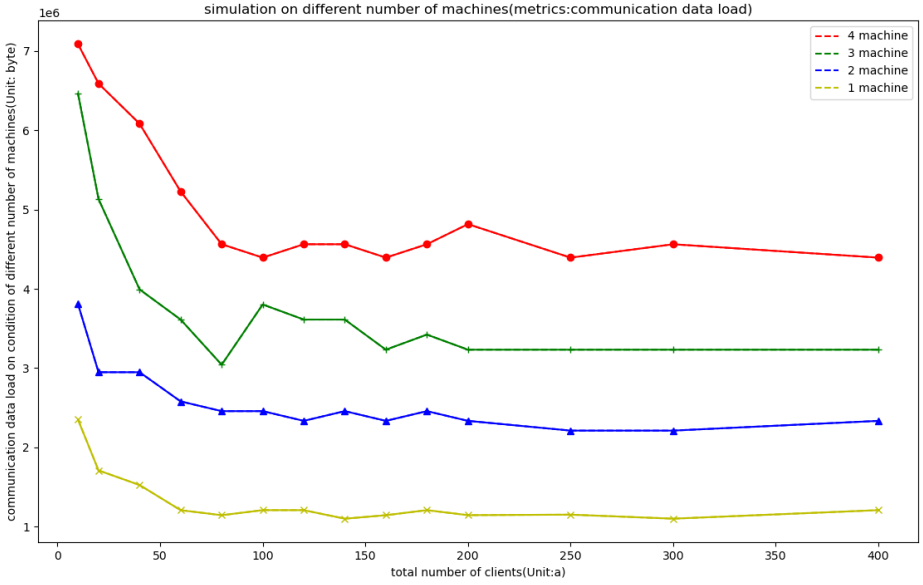
Since it was mentioned earlier that the client performs training in tandem mode, the total time increases when the number of clients involved in training(fraction C) increases. When the number of clients is small, increasing the number of clients (fraction C) reduces the number of training rounds, although the training time for each round is increasing, in this time, the increment in total time is not yet significant. When the number of clients is higher than 100, continuing to increase the number of clients will increase the training time per round, but at this point the number of rounds will no longer decrease. So the total time increases at a faster rate.

In term of total communication data load on figure 3.3, in edge pattern, the total communication data load synchronised with the number of rounds. For figure 3.6, in central pattern, the total communication data load increase with the increment of total number of clients, In the case of clients communicating directly with the server, the increase in the number of clients monotonically increases the total communication load.



H

FIGURE 3.2
total time of converge to given accuracy between different number of client simulation machine



H

FIGURE 3.3
total communication data load to given accuracy between different number of client simulation machine

3.3 SCALABILITY OF SIMULATION MACHINES

The scalability of simulation machines is compared under edge pattern.

When simulating the same number of participating training clients on different numbers of client simulation machines, the number of device simulating machine does not have an impact on the number of rounds needed to converge to a specified accuracy. This is consistent with our findings in the previous section, the knowledge and information contained in 100 clients is sufficient to represent all the data. More specifically, in the MNIST handwritten digit recognition task, consider the most extreme cases of Non-IID and imbalance, where only one or two digits are available per client. In this case, the 100 clients involved in each round of training are sufficient to represent the pervasive case for all digits. The result is shown in figure 3.1.

For the result shown in figure 3.2, given a fixed total number of clients participating in each round of training, the time of converging to the given accuracy increases approximately with multiplier relationship as the number of device simulation machine decreases. This is because the training within the machine we mentioned earlier is in a tandem relationship and is not parallel. When all the devices are concentrated on a single machine for simulation, it can be much slower than distributing the devices across multiple machines. If the time used for communication is not taken into account, under the same conditions, the total time used of simulating devices by 4 machines s approximately one quarter of that using one, one half of that using two and 75% of that using three.

For the differences in communication loads illustrated in Figure 3.3, we have shown that the number of communication rounds required for convergence does not vary with the number of device simulating machine. Under edge pattern, because each device simulation machine only sends one model to the server simulation machine with a more or less equal number of rounds, the fewer the number of machines involved in the simulation, the less the total communication load.

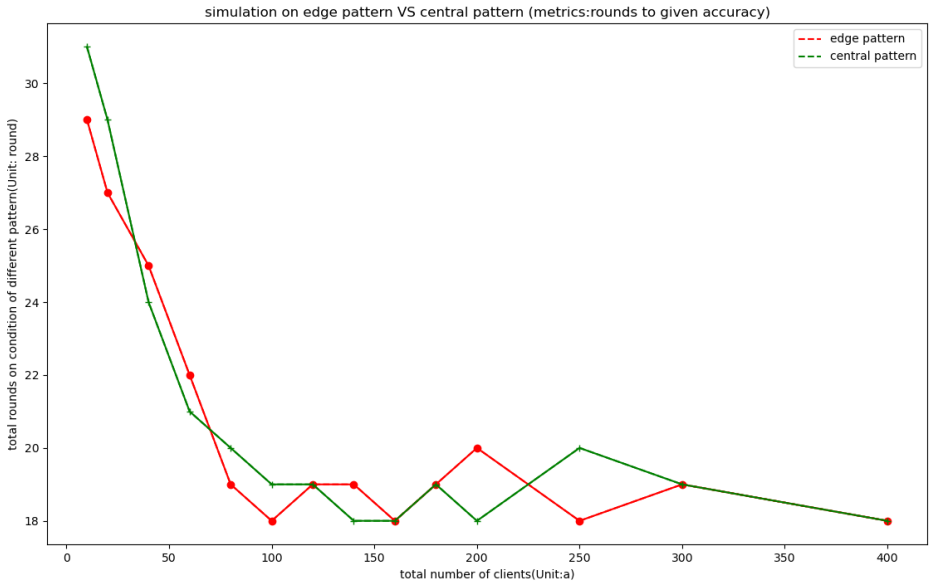
The results of this experiment also illustrate the **advantages of our multi-machine distributed runtime.** In a laboratory environment, our main goal is to speed up the convergence of the model, **multi-machine distributed runtime has a significant speed advantage compared to single machine runtime**, and the more nodes there are, the more significant the acceleration effect. The increase in communication load is acceptable in the context of the acceptable size of the number of parameters of the model.

For larger models, the results are also generally the same, good acceleration can still be achieved in a multi-machine distributed runtime. For central pattern, the results are also generally the same too, the total number of rounds will not be affected, the total time will be decreased and the total communication load will be increased. One thing worth mentioning is that when in central pattern, the speed-up is not only reflected in the reduction in training time, but also in communication, because a large number of models are sent in this pattern and communication of different device simulating machines is in parallel, the speed-up in communication is also significant.

3.4 EDGE PATTERN VS CENTRAL PATTERN

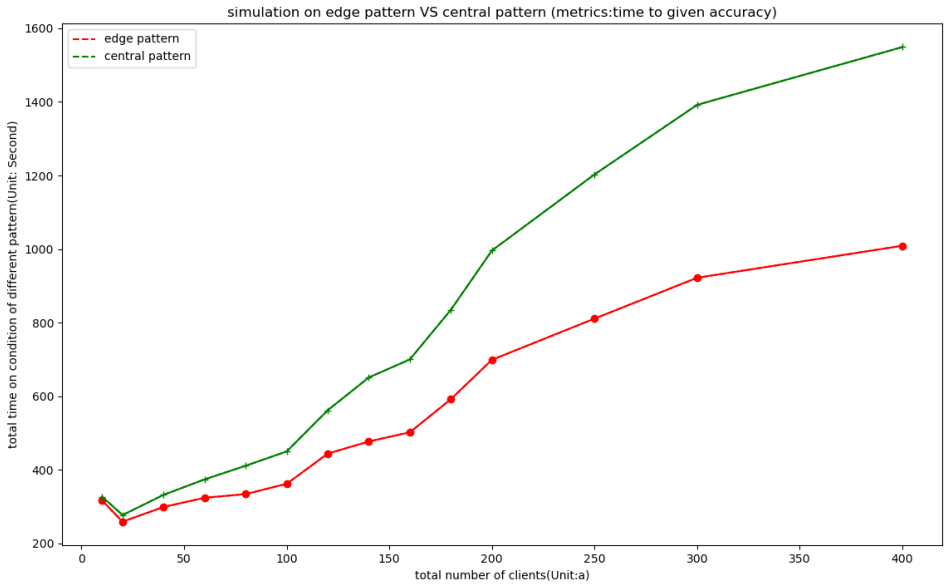
The definitions of the edge and centre patterns have already been described. From figure 3.4 we can see, the number of rounds required for model convergence does not vary with pattern change, this point is also in accordance with our previous discussion.

Since in edge pattern, the device simulating machine sending only one model to the server, this saves significantly on the time required for communication and the total time, accelerating the convergence of the model. The result can be seen at figure 3.5.



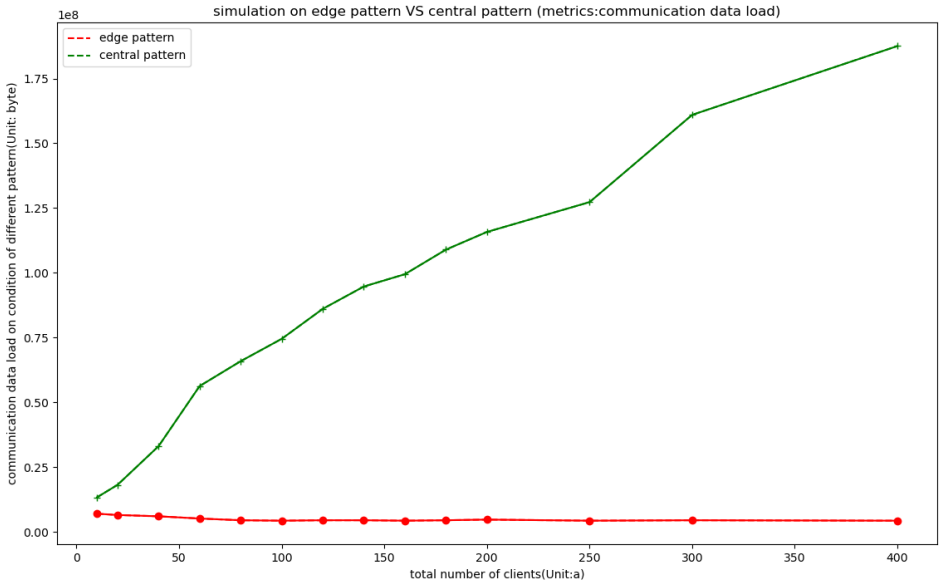
H

FIGURE 3.4
rounds of converge to given accuracy between edge pattern and central pattern



H

FIGURE 3.5
total time of converge to given accuracy between edge pattern and central pattern



H

FIGURE 3.6
total communication data load of converge to given accuracy between edge pattern and central pattern

About the communication data load, we can clearly see in Figure 3.6 the huge difference in communication load between the two patterns. The edge pattern is superior to the central pattern in terms of both time consuming and communication load.

CHAPTER 4

4-DISCUSSION AND CONCLUSION

The core purpose of both the related work and my experiments is to investigate the factors that affect the performance of federated learning and how to improve it. Our research focuses on two scenarios: real-life applications and laboratory simulation environments. The main limitations and challenges of federated learning are described in Section 1.1, most of optimisation for federated learning is around these points.

There are some differences between real-life applications and laboratory simulation environments. The first one is real-life applications will be under very strict communication restrictions, there are few opportunities for communication between the device and the server. But in laboratory simulation environments, communication can take place in a nearly real-time and big bandwidth manner. So the main goal of real-life Federated learning applications is reducing communication rounds, leverage each communication round as much as possible to optimise the model, but for laboratory simulation environments, We wanted to accelerate the simulation as much as possible from a time perspective to get the experimental results we wanted, in the mean time, involving more simulating clients into experiment. The second one is in real-life Federated learning applications, the devices is in both asynchronous and parallel relation during training and communication. But in laboratory simulation environments, due to the limitations of the framework, the relationship between devices is not necessarily parallel and asynchronous/synchronous depends on the situation. The similar point is both of them want reduce the communication data load.

Impact of fraction(C) for the fraction(C), both related work and my experiment show the fraction(C) mainly affect the rounds required to model convergence. Increasing fraction(C) to reduce total communication rounds is meaningful both to real-life Federated learning applications and laboratory simulation environments. But because of the decreasing marginal benefit of increasing fraction(C) and the increasing consumption of computing and communication resources when increasing fraction(C). So a suitable trade-off needs to be found for the fraction(C). Through related work and my experiment, set C as 0.1(10% of all clients) is a reasonable choice.

Increasing computation per client we can also conclude from other related papers that for the urgent need to reduce the number of communication rounds for real-life Federated learning applications, increasing computation on each client is the most useful way. A simple way which already be verified multiple times already is to increase the number of epochs for training process per client. Selecting a proper local minibatch size is also a useful way to reduce the total communication rounds. The same methods under laboratory simulation environments can also reduce the total communication rounds, however, it is unproven whether the time required for the model to converge can be reduced.

Initialisation and modeling There are two other factors that affect the speed of model convergence and the number of communication rounds required - the initialisation method and the modelling method.

Initialisation method will affect the convergence of the model. Different modeling approaches have different levels of adaptation to different datasets.

Global model and local data Even if our global model has been optimised extremely well, due to the great variability of local data, A model that is particularly well optimised overall may also have a large loss and very low accuracy on the local data of a particular device. This bad phenomenon is mitigated when overall each device has more data.

Edge pattern and Central pattern In real-life Federated Learning applications, Federated learning is well suited for edge computing applications and can leverage the the computation power of edge servers and the data collected on widely dispersed edge devices. If not in this pattern, the computing pressure and communication pressure on the central server can be very high. The edge server always geographically close to devices, who has abundant computing resources and high bandwidth communicating with end nodes. In the mean time, edge computing also has many other advantages, such as robustness, when an edge server goes down or suffers an attack, the overall functionality and data security is not seriously affected. In the other hand, the elasticity and scalability of the entire network will also be greatly enhanced[7][8].

Under laboratory simulation environments, my experiment also prove using edge pattern can reduces the time required for model convergence and the communication load.

CHAPTER 5

5-FUTURE WORK

The first future work I want to do is improving the ease of use of my existing TFF code, put parameter modification and new code copy generating into script, continue to improve the running speed of the code. Then on the basis of improved code, evaluating the effect of three factors on the performance of Federated Learning: number of epoch, different initialization methods, and different averaging methods(such as FSVRG).

The second is as close as possible to simulate real-world Federated Learning. Some main implement directions are: 1. Simulating different devices with different computing power and communication ability, not all devices are same. 2. Simulating real-world situations in which communication occurs asynchronously. 3. Complex network graph topology.

The third is exploring methods such as gradient qantization and gradient sparsification to make communication more efficient, network quantization and network pruning to make computation more efficient[7].

BIBLIOGRAPHY

[1] Jakub Konecný et al. 'Federated Optimization: Distributed Machine Learning for On-Device Intelligence'. In: *ArXiv* abs/1610.02527 (2016).

[2] Tian Li et al. 'Federated Learning: Challenges, Methods, and Future Directions'. In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: 10.1109/MSP.2020.2975749.

[3] Peter Kairouz et al. 'Advances and Open Problems in Federated Learning'. In: *Found. Trends Mach. Learn.* 14 (2021), pp. 1–210.

[4] H. Brendan McMahan et al. 'Communication-Efficient Learning of Deep Networks from Decentralized Data'. In: *AISTATS*. 2017.

[5] Sebastian Caldas et al. 'LEAF: A Benchmark for Federated Settings'. In: *ArXiv* abs/1812.01097 (2018).

[6] Adrian Nilsson et al. 'A Performance Evaluation of Federated Learning Algorithms'. In: *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning* (2018).

[7] Qi Xia et al. 'A Survey of Federated Learning for Edge Computing: Research Problems and Solutions'. In: *High-Confidence Computing* 1 (Mar. 2021), p. 100008. DOI: 10.1016/j.hcc.2021.100008.

[8] Wei Yang Bryan Lim et al. 'Federated Learning in Mobile Edge Networks: A Comprehensive Survey'. In: *IEEE Communications Surveys & Tutorials* 22 (2020), pp. 2031–2063.