Midterm Report

Mohammad Maaz Owais, Muhammad Hamza Khawaja, Muhammad Taha, Shazer Ali

1. Introduction

Federated Learning involves training a shared global model using local data and compute on various user devices. Several approaches have been proposed to implement this paradigm starting with FedAvg [2]. However, the system heterogeneity in participating devices poses a significant challenge that needs to be addressed. In developing countries, 57% of population are categorised as low-end users. [4] This has implications for fairness due to introduction of systematic bias, in addition to degradation in model accuracy. Recent works such as FedProx [5] and Hassas [3] have attempted to include slow devices by incorporating partial work and serving a subset model according to device characteristics, respectively. These approaches have mostly been evaluated on simulations using LEAF Benchmark [1]. To the best of our knowledge, none of these works have been evaluated on federated learning systems using real-world devices with a sufficiently large number of users.

To this end, we develop a federated learning system that includes 100+ active real-world users. We achieve this by building a robust FL system using the Flower framework. This deployment will provide a conducive platform to concretely evaluate the robustness of Hassas as well as other FL strategies. Conducting experiments on real-world data in the face of dynamic changes in systems heterogeneity, including state changes, will provide valuable insights that will benefit the community.

2. What approaches did we try?

Intially, we started with comparing the testbed framework with flower by training on each framework with 4 devices. This was done to conduct a cost-benefit analysis of each framework so that we can make an informed decision keeping in mind our goals which include scalability and support to ensure a robust and efficient implementation of our system. Flower is an open-source framework which provides intuitive APIs and an interface to implement custom strategies. However, support for android training in Java and Kotlin is still under development, which is why we implemented our own android application for training purposes.

Our existing implementation in testbed for device information logging requires root access. Since this is not conducive for the remote, continuous infrastructure that we want to deploy, we integrated support for memory profiling, cpu profiling, among other characteristics that do not require root access, using the android Activity Manager API. Due to

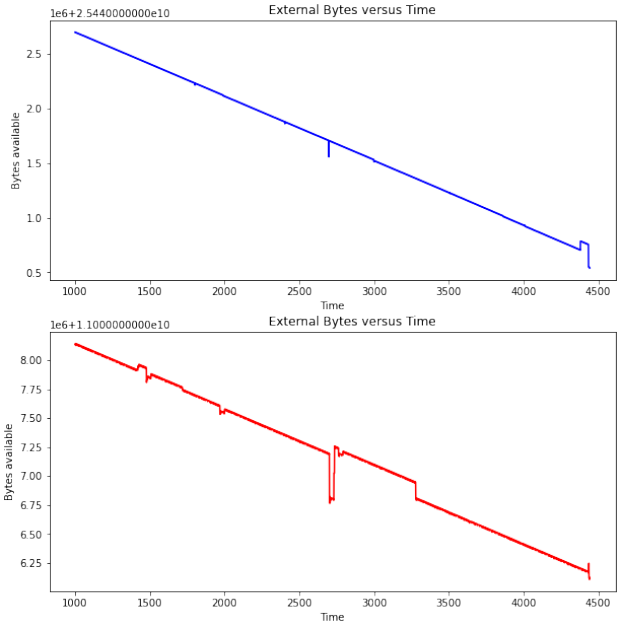

Table 1: External Bytes Available: As shown in Figure 1, since we are logging data every second, this is why the external storage bytes are constantly decreasing.

the remote nature of the deployment, the FL task should be independent of the need for user interference. Specifically, we eliminated the need for manual loading of data from the server and beginning FL task execution.

Keeping in mind android system optimizations, it was important to ensure that our application ran in the foreground so that it is given higher priority and is less likely to be killed in high memory pressure situations. We also anticipated reduced application performance when the system limited its resources like CPU and Memory usage, for example, to conserve battery life. If the application is killed by the Operating System, it will lose its connection to the server, resulting in lost updates and incur an additional overhead of data reloading. This can be a significant issue, particularly in our case, where we expect large number of clients and longer training times.

Due to time constraints, the intial idea of developing an application that would generate real-world data by providing users with an attractive incentive, was postponed. As a result, we decided to use an emulated dataset and partitioning the data among the users. From an implementation perspective, the dataset will be hosted on a centralized server. Clients connecting to the server will be assigned respective client

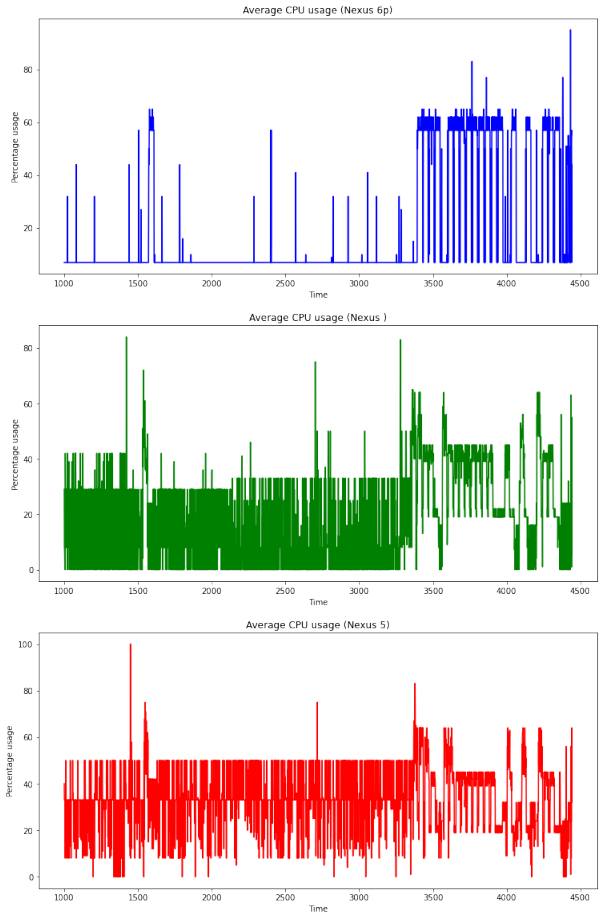

Table 2: CPU Profile: In the first round, the training completes for each device for the set number of epochs. However, due to a systematic error, the client thread blocks and the server waits for the client parameters so no CPU is utilized for the FL task. When one of the devices was switched to the foreground, the blockage is removed and the task resumes. This is a bug in our system which we are working on fixing.

IDs which will be used by the clients to fetch their respective data partitions from the central server. Due to similar reasons as above, the implementation of Hassas was also postponed. Instead, we are conducting evaluations using the Federated Averaging algorithm.

In order to access the performance and impact of Flower on low end phones, we mainly aimed at creating a sufficient user pool of mainly android phones with 2 to 3 GB RAM. We are approaching the LUMS student body, PDC staff, and MBM workers. However, we have found most devices to be mostly 4GB and above. Hence, we explored options outside LUMS in our individual capacity. As a result of our efforts, we were able to gather around 10 devices (< 4GB RAM) and 5 devices (> 3GB RAM). We expect to gather a much larger number of fast devices, but intend to maintain an appropriate ratio between the two. We are also reaching out to mobile phone

RAM (GB)	Quantity
1	1
2	13
3	8
4	5

Table 3: Device Characteristics (By RAM)

vendors who sell used phones. If we are able to reach some arrangement with them, we will be able to leverage many devices with diverse characteristics.

(Metrics) cpu profile, memory profile, os interaction (timer interrupts, context switches), os heterogeneity (base profiling across devices)

From our earlier experience of running FedAvg on Flower in a controlled setup, we postulate that the FL background process can be permanently interrupted at the client side due to mainly two reasons: the device is powered off, the process is killed by the system, or if it crashes due to some reason. Device state changes will be very frequent with real users. Hence, we are trying to add a restart mechanism which will essentially relaunch the process after device reboots and if the process is killed. To better analyse and inspect the reason for OS killing the App, we need to understand the OS interaction with App by performing crash analytics. Our crash analytics will include the context switching frequency, interupt frequency, and I/O Wait Time. Context switching frequency will tell us how difficult it is for the app to simultenously train data and log API metrics in the local storage. On the other hand, interupt frequency and I/O wait time will provide us with the insight of how much the App is competing for resources such as memory and CPU to carry out its tasks.

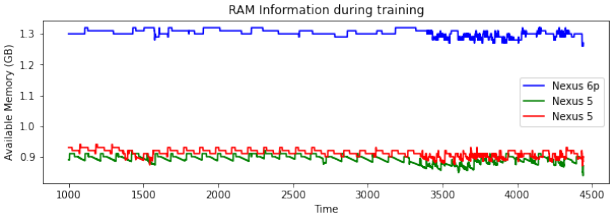

Table 4: Free Memory

Once, we have sucessfully evaluated Federated Averaging process on the users, we aim to implement Hassas using the strategy interface in Flower.

References

[1] CALDAS, S., WU, P., LI, T., KONEČNÝ, J., MCMAHAN, H. B., SMITH, V., AND TALWALKAR, A. LEAF: A benchmark for federated settings. *CoRR abs/1812.01097* (2018).

[2] MCMAHAN, H. B., MOORE, E., RAMAGE, D., AND Y ARCAS, B. A. Federated learning of deep networks using model averaging. *CoRR abs/1602.05629* (2016).

[3] MUNIR, M. T., SAEED, M. M., ALI, M., QAZI, Z. A., AND QAZI, I. A. Fedprune: Towards inclusive federated learning. *CoRR abs/2110.14205* (2021).

[4] NASEER, U., BENSON, T. A., AND NETRAVALI, R. Webmedic: Disentangling the memory-functionality tension for the next billion mobile web users. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2021), HotMobile '21, Association for Computing Machinery, p. 71–77.

[5] SAHU, A. K., LI, T., SANJABI, M., ZAHEER, M., TALWALKAR, A., AND SMITH, V. On the convergence of federated optimization in heterogeneous networks. *CoRR abs/1812.06127* (2018).