FedSS: Federated Learning with Smart Selection of clients

Ammar Tahir Yongzhou Chen Prashanti Nilayam

Abstract

Federated learning provides the ability to learn over heterogeneous user data in a distributed manner, while preserving user privacy. However, its current clients selection technique is a source of bias as it discriminates against slow clients. For starters, it selects clients that satisfy certain network and system specific criteria, thus not selecting slow clients. Even when such clients are included in the training process, they either straggle the training or are altogether dropped from the round for being too slow. Our proposed idea looks to find a sweet spot between fast convergence and heterogeneity by looking at smart clients selection and scheduling techniques.

1 Background

Last decade saw an explosion of data driven and machine learning based applications to solve different problems. This naturally lead to a lot of fruitful discussion about ownership and control of access to this data. Users are concerned about privacy of their sensitive data and do not prefer sharing it with the third party organizations. However, the value of models trained on this user data is quite high to both the organizations and end users. Thus, recently we have seen a lot of work on privacy preserving designs of machine learning models, popularly known as federated learning.

Federated learning is a privacy preserving method of distributed learning over heterogeneous user data. Federated learning follows the philosophy of "bringing the code to data, instead of bringing data to code" to address the above mentioned privacy conerns [7, 19]. The architecture presented by Bonawitz et. al. [7] consists of a server responsible for selecting client devices for training from a pool of currently available devices for each round. Server maintains a copy of global model, which it distributes to the selected client devices at start of each round. Each client trains the model on their local data and send the gradients back to the server. The server, after getting gradients from all the clients calculates the average gradient using the FedAvg algorithm. This gradient is used to update the global model, which is then distributed in the next round.

Since gradients can still reveal information about the data, to ensure privacy further the gradient updates sent by client are preserved with secure aggregation [8]. Secure aggregation prevents the server from learning individual clients' gradients, but the server can learn overall aggregate once response from all clients is received. This coupled with already synchronous nature of federated learning, makes the performance of training susceptible to degradation due to stragglers. Thus, it is desirable from the angle of performance that the selected devices are almost homogeneous with respect to the network and compute power. Most of the current designs select clients randomly from a pool of devices that satisfy certain criteria e.g. minimum 2 GB memory, unmetered network etc [7, 16]. This technique of client selection introduces explicit bias in the system since factors like device memory and quality of network are directly linked with socioeconomic status [6, 14, 16]. Thus, it's critical to improve client selection mechanism to build models that are void of this explicit bias.

2 Related Work

There have been a great emphasis on this problem, and there are solutions varying from model compression to different strategies of client selection. In this section, first we discuss different works that highlight and try to address the problem of bias and then we discuss different current client selection strategies.

2.1 Bias Mitigation Strategies

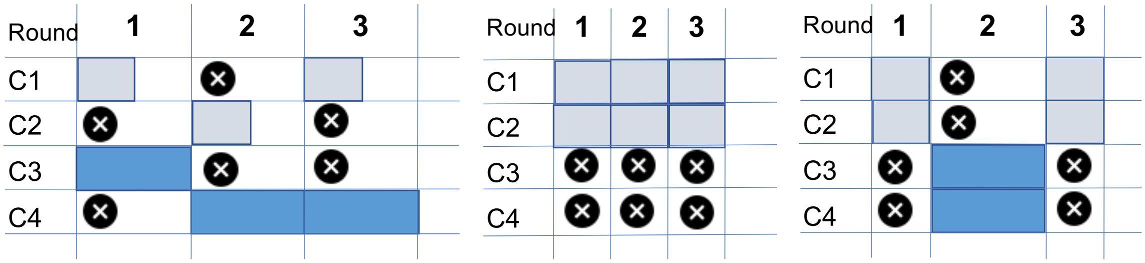
As we discussed, the existing design of federated learning introduces explicit bias since parameters based on which devices are selected are linked with socioeconomic factors [6,14,16]. Bias can also come because of non-IID distribution of data, however this is a deep learning optimization problem and there are some techniques to circumvent it [10, 17, 28]. To mitigate the explicit bias discussed above, we need to include slow devices in the training process. Clearly doing this comes at the cost of slow convergence, owing to the slow clients straggling the process. However, there have been some work on reducing compute and network costs for slow client at the expense of accuracy of the model [15,17,26]. For example Li et. al. [17] allows different devices to perform variable amount of workload depending on their resources, slow clients can run fewer epochs or use lesser input data. Similarly, Konečnỳ et. al. [15] compresses model updates in a lossy fashion to reduce communication costs at the expense of accuracy of the sent parameters. Abay et al [6] propose a fairness aware regularization in the loss function. However, we argue that such techniques do not truly address the bias problem because there is still discrimination in how slow clients are handled. An ideal design should give equal opportunity to contribute to the global model to all clients.

2.2 Clients Selection Strategies

It's clear that the heterogeneous communication environments and computation resources at clients can hamper the overall training speed. To accelerate the convergence speed under this client heterogeneity, existing work has investigated to make smarter decisions about clients selection to alleviate the communication overhead.

In the prevalent random client selection strategy, all clients participate in the client selection phase in every round. The server uniformly and randomly selects a subset of clients from the pool [18, 20, 24] for training, and the chosen clients do multiple iterations of SGD on the local data, given the ML model and the latest parameters from the server. Finally, the server collects and aggregates the computed gradients to update the global parameters. Li et. al. [18] provided a necessary convergence condition for federated learning on non-iid data with partial clients participation. Ruan et. al. [24] offered a selection scheme that converges even when devices can flexibly join or leave the training.

Some recent work looks to do client selection based on different criteria like higher potential for global model convergence or good network conditions. Cho et. al. [10] revealed that a selection biased towards clients with higher local loss can increase the speed of convergence. This is because higher loss indicates higher potential for model improvement. The proposed POWER-OF-CHOICE algorithm can yield up to 3X faster convergence and 10% higher test accuracy compared with conventional federated learning with



(a) Random Selection (b) FedCS (c) Ours

Figure 1: (a) The round time in the random selection scheme is determined by the straggler. (b) While the round time in FedCS is shortest, the client3 and client4 are always excluded because of their long training time and transmission delay. (c) Our scheme reaches the best trade off between training time and data heterogeneity.

random clients selection. FedCS [22] requests the resource information, such as wireless network bandwidth and computational capacities, from selected clients before the distribution of parameters of the global model. It then only collects the gradients from clients which can update and upload the parameters within a deadline. Although these biased clients selection models can facilitate quicker convergence, it sacrifices the original benefit of federated learning: the heterogeneity of data.

3 Motivation

As evident from the related work, federated learning between clients with heterogeneous data, device and network can result in prolonged convergence time. Appropriate clients selection decisions can result in quick convergence. Convergence time is determined by two factors: *i) number of rounds* until the model convergence condition is reached and *ii) time duration of each round*. The number of rounds can be reduced by selecting clients that add more value to the learning e.g. high losses. Whereas duration of each round is determined by the stragglers, thus selecting devices based on device and network parameters can reduce convergence time.

To avoid the prolonged round duration time caused by stragglers, we can select clients with similar training time in each round. Meanwhile, we can randomly choose clients with different network/computation conditions across rounds to guarantee data heterogeneity. We understand that slow clients might be in minority in some cases, where arranging separate round for them can jeopardize their privacy. To ensure privacy for such rounds we can fill up the round group with faster clients too. We visualize and compare this strategy with random selection in Bonawitz et, al. [7] and FedCS [22] in fig1.

4 System Design

To reduce bias and also ensure faster model convergence, we propose a smart approach of clients selection. We first collect the device's compute captured by FLOPS (Floating point Operations per Second) along with network conditions such as uplink and downlink bandwidth [11] of every connected client.

Based on these collected parameters, we categorize clients into different equal sized clusters, each cluster with clients with similar training time. We also determine optimal number of clusters, *k*, for a given distribution. *k* is optimized to minimize training time, while ensuring higher possible degree of anonymity (high cluster sizes). In every round, the FedSS server chooses a cluster in a round-robin way and selects clients within it randomly to ensure equal opportunity of every client.

There are two basic intuitions behind the approach. 1) Reduce bias induced due to barrier to entry for low bandwidth/ low-end devices, which in turn minimizes socio-economic bias. By giving fair chance to all the clients, model gets a better chance to learn from different data distributions. This also prevents against content farms attacks using uncompromised phones with high availability and bandwidth. 2) Have more coordinated training rounds with similar performing clients grouped together. We are less likely to run into a situation in which overall completion time of a round is longer due to a fraction of low-performing devices.

4.1 Dynamic Client Environment Tracking

From figure 2, the steps in one round of training in FedSS comprise: the smart clients selection, the distribution of models, the parallel clients training procedure, the collection of updated models, and the aggregation. Except the first and the last steps, the time it costs in all other steps is decided by the clients' environments. The network conditions, such as bandwidth and propagation delay, determine the time taken to distribute model and collect gradients. Meanwhile, the client's local computing power, reflected by available CPUs, GPUs and memory, determine the time taken to finish the training procedure. Therefore, FedSS dynamically tracks and records those parameters of every connected client, and predicts the approximate time it takes to accomplish one round.

Let's assume the model size is M, and total number of floating point operations in the model is $Flops$. Given the measured network uplink bandwidth UL_i, downlink bandwidth DL_i, client's FLOPS rate $FlopsRate_i$ and the number of samples at client S_i for client c_i, the overall round time T_i for that client is:

$$T_i = M/UL_i + S_i \cdot Flops/FlopsRate_i + M/DL_i$$

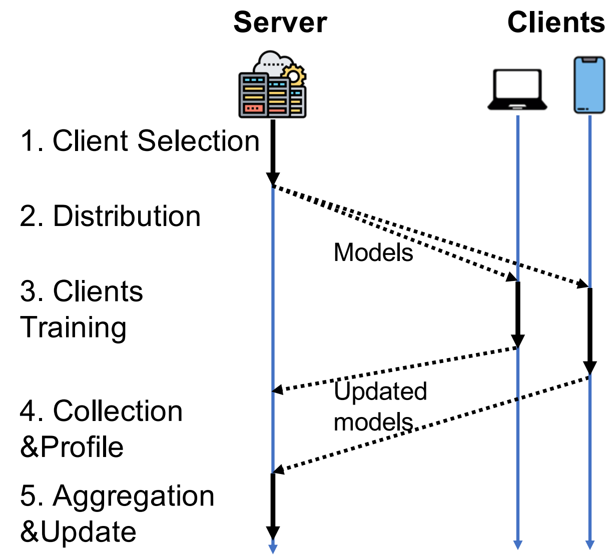


Figure 2: The workflow to finish one round of synchronous training in FedSS . The bottleneck includes models distribution, models collection and parallel clients training, which majorly depend on the client's computation speed and network conditions.

There are different faithful methods to measure bandwidth and FLOPS. Measuring bandwidth has been an active area of research for a long time. There are active ways to measure bandwidth such as speed tests and probe trains [11]. There are passive mechanisms as well which estimate bandwidth based on network usage, e.g. client hints [1]. Similarly, there are different benchmarks to measure FLOPS as a proxy of compute power e.g. Linpack [4] and HPL [3]. Additionally, these estimates of compute power can be improved with real time observations of time taken to run a training round. The mechanisms to track mentioned metrics is not the contribution of our work, in fact, we rely on existing work for this.

4.2 Smart Client Selection

After getting the client round time estimate, T_i for most of the clients, we are ready to run our clustering algorithm. The clustering algorithm takes the number of clusters as an input, and based on it, decide the percentiles of distribution. For example, if number of clusters is 3, selected percentiles would be 25, 50 and 75. Clients are sorted into these clusters based on their T_i's mean-squared distance from percentile values. Since the purpose of this clustering is to group together clients with almost similar training times, equidistant percentile based centroids serve this purpose reasonably well.

The clusters constructed this way tend to have uneven distribution of clients. The clusters with smaller average training time tend to have most number of clients compared to clusters with higher average training time. This means if we have a round robin pattern of switching between clusters for each training round, the clients in larger clusters have smaller probability of selection. Our earlier solution to address this problem was to construct a weighted round robin pattern where clusters appeared in the pattern based on their sizes. For example, if cluster sizes were (50%, 25%, 25%), the weighted pattern would be (1,1,2,3), where the first cluster is selected for 2 rounds compared to the other two.

However, based on feedback during midterm report, we realise that having different sized clusters also means that clients have varying degree of privacy. We use k-Anonymity metric to argue about degree of privacy. k-Anonymity is a metric used to measure the degree of anonymity in anonymised datasets. Given some identifiers (internal and external), one cannot narrow down some datapoint to fewer than k individuals. In our case, if we have varrying cluster sizes, clients in different clusters have varying degree of k-anonymity. Thus, to be fair to all clients, it is desirable to have clusters of equal size.

We use a simple trick to even out clusters created by our algorithm. As we observed that the slow client clusters tend to have fewer clients. We pick slowest client from fast client clusters and add them to the small cluster until sizes are almost evened out. The inverse of this can also be done if fast client cluster is smaller, but there we are careful so that shifting clients from slower cluster does not have a very drastic effect.

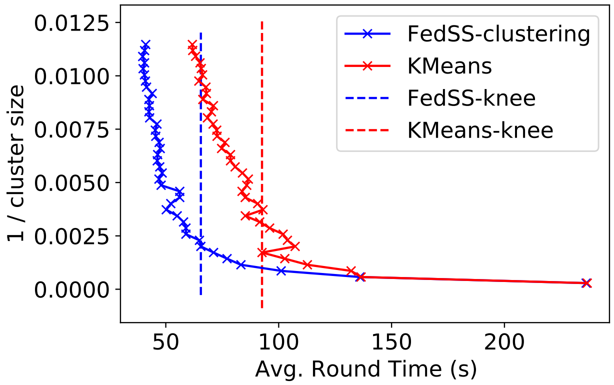


Figure 3: Visualization of the process behind finding the optimal number of clusters (6 in both cases). Also worth noticing is superior efficiency of FedSS 's clustering algorithm compared to KMeans.

This process of evening out has an additional benefit, data heterogeneity amongst clusters. This reduces the socio-economic data homogeneity that could arise due to having similar type of clients in same clusters.

4.3 Optimal Number of Clusters

As we pointed out in 4.2, standalone client selection algorithm requires the number of clusters as an argument. However, it can be difficult for operator to know this. Thus, we solve for optimal value of number of clusters. Having fewer clusters is desirable from the perspective of privacy, whereas more clusters results in shorter average round time. However, average round time has diminishing returns as we keep increasing number of clusters. The optimal number of clusters would be the point after which we see diminishing returns on improvement in average round time.

Thus, to find optimal number of clusters, we simulate training process given round time estimates of clients with different value of clusters. This gives us average round time for the given number of clusters. Then, we use Kneedle [25] to find the optimal point on the curve of average round time vs. 1/cluster size. The figure 3 shows the curve between average round time and 1/cluster size along with the knee point found by Kneedle algorithm.

Figure 3 also compares the effectiveness of our algorithm as compared to KMeans. This is simulation of training with 10000 clients and 1000 rounds of training. We also compared it against DBScan and KDE clustering, which had almost similar curve as KMeans. The reason our clustering algorithm does better is because it is optimised to reduce round time. Whereas Kmeans and other clustering algorithms are specialised to just cluster data, these clusters may not be optimal with respect to reducing round time. Moreover, it is difficult for clustering algorithms to construct equal sized clusters.

4.4 Overhead

This grouping algorithm has a time complexity of $NlogN$, where N is the total number of client devices. Finding optimal number of clusters can be costly if the number of clients is quite high. But this computation can be bounded by limiting maximum number of clusters to try e.g. for 100 clients, it is not desirable to have tens of clusters. Moreover, the whole algorithm can be run in parallel with model training and new schedule can be used from next round onwards. Network conditions as well as training time can vary over time depending on factors like competing traffic, memory or compute back-pressure. Therefore, it is desirable to run grouping algorithm periodically on updated estimates of T_i.

In our experimental implementation, we did not implement mechanisms to measure bandwidth and profile FLOPS because our experiments are simulation based. However, existing mechanisms to do so are very lightweight and do not have significant overhead. Similarly, the profiling of FLOPS can be done by measuring time to train a round.

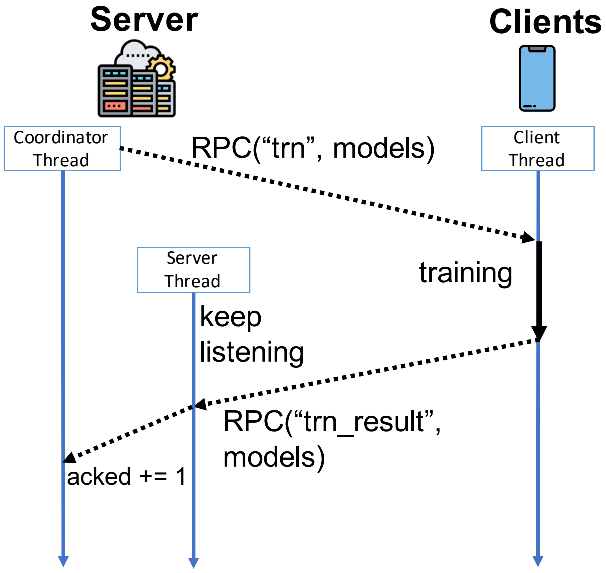


Figure 4: The RPC-based communication protocol

5 Implementation

Based on an open-source federated learning benchmark system leaf [9], we implemented FedSS by writing a RPC-based communication protocol and scheduling logic. We emulate the real federated learning by doing distributed computations in multiple processes on a single machine.

Figure4 illustrates the communication protocol in detail. The coordinator thread in the server asynchronously sends RPC commands to K selected clients for training with its local data, while the server thread keeps listening clients' requests to upload the updated models. After the client finishes training, the server thread receives results and increments a shared variable *acked* by one. The coordinator thread waits until *acked* equals K to continue the aggregation step and finish this round.

To emulate the real variable network conditions in clients, initially we planned to utilize Mahimahi [21] to emulate a trace-based network. However, the training time can take 10s of hours to run one baseline using the emulated

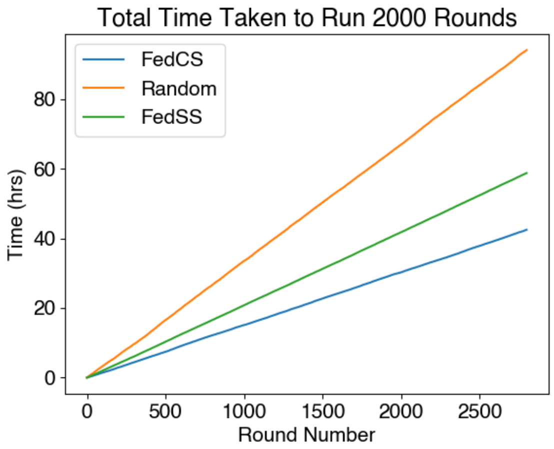


Figure 5: The total time taken by FedSS , FedCS and Random Selection to train 2800 rounds.

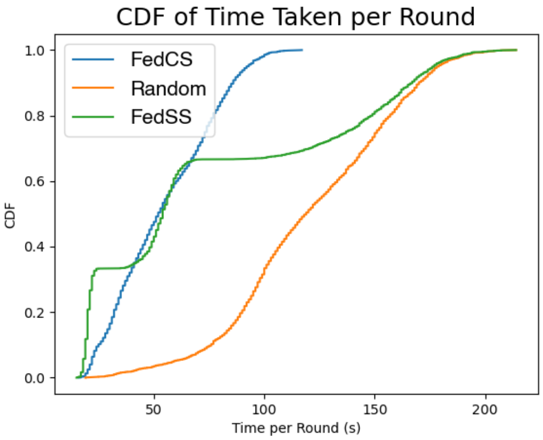


Figure 6: CDF of time taken per round for FedSS , FedCS and Random Selection.

network, which is intolerable given the time and experiments resource limitation. Thus we simulate the network delays based on the internet speeds data from World Population Review [2].

Since we neither own the required number of mobile phones, nor find an appropriate mobile-device emulator for experiments, we simulate the training speed based on the benchmarked FLOPS [5] of top 20 most sold mobile phones in 2020 [27].

6 Evaluation

We evaluate FedSS against our implementation of FedCS and random client selection. We train a CNN model on subset of Femnist dataset, distributed in a Non-IID manner between 20 clients. Initially, we wanted to run our experiments on a much larger number of clients, but it's not possible to run so many instances of concurrent CNN models in our machine (neither given VMs). For FedSS and random client selection, we set number of clients per round to be 5. For FedCS, we select 8 clients per round but aggregate updates with first 5 responses. Also note that for system and network heterogeneity, we simulate delays for different clients. However, to ensure fairness between our system and benchmarks, same random delays are used for clients across all our experiments. Similarly, same distribution of dataset between clients is used across our experiments.

We explored different metrics to evaluate our system. Since our trade-off is between training time and bias, we primarily focus on metrics to capture these two.

6.1 Training Time

Measuring training time is easy, we can individually measure the time taken to complete one round of training at the server. This time would include time taken to send model to all clients concurrently, time taken by the clients to train and send back the gradients. This process is bottlenecked by the slowest client accepted by the server for aggregation. Thus, time per each round is measured by the time taken by the slowest client to receive, train and send back the model.

Figure 5 shows the total time taken by FedSS and benchmarks to run 2800 rounds of training. FedCS beats our system by approximately 40%, because in each round it only waits for the fastest clients to send back the gradients.

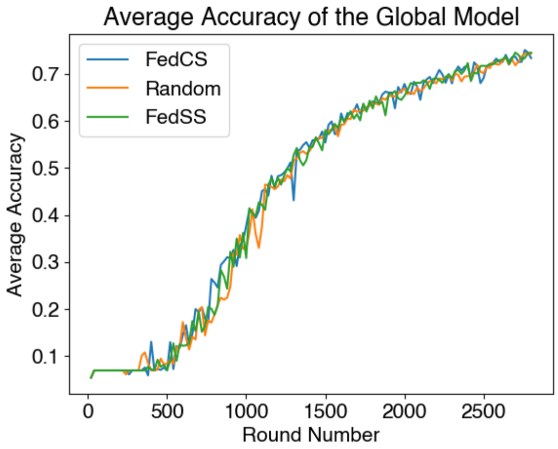


Figure 7: The average accuracy of the global model for FedSS , FedCS and Random Selection.

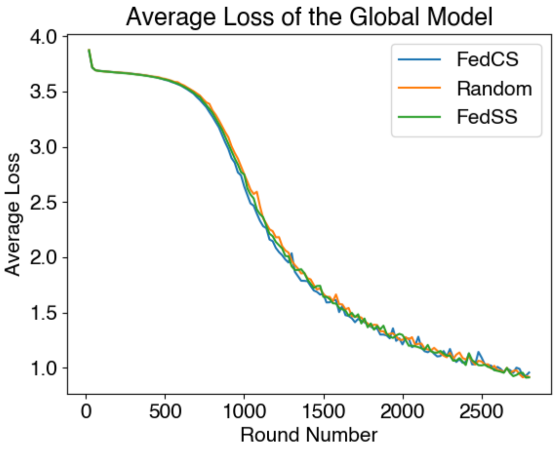


Figure 8: The average loss of the global model for FedSS , FedCS and Random Selection.

This can be validated by looking at the CDF of time taken per round by FedCS in figure 6. On the other hand Random Selection has the longest training time, since time taken per round in this case would be straggled by the slowest client. Again this can be validated by looking at CDF for Random Selection in figure 6, which is shifted to right. In comparison, FedSS which selects clients with similar training times together is able to run more than 80% of the rounds much faster saving us about 40 hours in training.

6.2 Bias

Before measuring bias, let's take a look at the performance of global model after 2800 rounds for each of the selection strategies. There are many well established evaluation measures for classification models including precision, recall, accuracy, f1-score, etc. For our evaluation, we have selected accuracy and f1-score as the measures.

6.2.1 Accuracy

Accuracy is the measure to determine the correct prediction ratio for the given dataset. It is calculated as :

$$\frac{TP+TN}{TP+TN+FP+FN}$$

Figure 7 & 8 respectively show the average accuracy and loss of global model across all clients. From the first look, it seems if all the strategies have similar performance with respect to model validity. However, a deeper look at the model's metrics show a clearer picture.

Table 1 shows the breakdown of accuracy of the model across different kinds of clients. We notice that while FedCS has an average accuracy as good as Random Selection and FedSS , it has lower accuracy for slower clients. On the other hand, on faster clients, its accuracy is higher than both Random Selection and FedSS . This difference of performance on slow clients captures the bias in FedCS's client selection strategy. This difference arises from the fact that faster clients end up getting a lot more say in the training than the slow clients.

The 1.5% difference of accuracy for FedSS , although negligible, can be explained from data heterogeneity point of view. Recall that Random Selection by design does not have any explicit bias due to device heterogeneity. This is because Random Selection selects clients randomly and then waits for all of them to respond before ag-

Accuracy for 4 Slowest Clients				Accuracy for 4 Fastest Clients			
Client	FedCS	Random Selection	FedSS	Client	FedCS	Random Selection	FedSS
1	68.76	80.12	79.81	1	83.28	85.92	86.80
2	72.34	76.17	76.59	2	55.97	55.97	60.37
3	56.68	67.51	71.33	3	80.07	76.56	78.90
4	67.32	71.25	70.86	4	84.88	79.42	78.13
Average	66.27	73.76	74.65	Average	76.05	74.47	76.05

Table 1: A breakdown of accuracy of model across 4 slowest and 4 fastest clients for each of the selection strategies.

gregating. Thus, the 1% difference in its accuracy when compared on faster vs slower clients, can solely be attributed to data heterogeneity. The fact that FedSS does as good as Random Selection while reducing the training time by a huge margin shows promise that FedSS can reduce bias and amortize the cost of slow nodes.

6.2.2 F1 Score

F score is a measure to determine model performance by using both, precision and recall values. Precision is the measure to determine the percentage of correct results out of all the results for a given class. It is given as :

$$TP/(TP+FP)$$

Recall is the measure to determine the percentage of correct results out of all the true results for a given class. It is given as :

$$TP/(TP+FN)$$

F1 score, also known as balanced F score, is the harmonic mean of precision and recall.

$$2 * \frac{precision * recall}{precision + recall}$$

For a multi-class classification, F1 score is given as the average of F1 score of each class based on averaging scheme. For our experiments, we have selected **weighted** averaging to deal with class imbalance issue due to non-iid distribution of data across multiple clients.

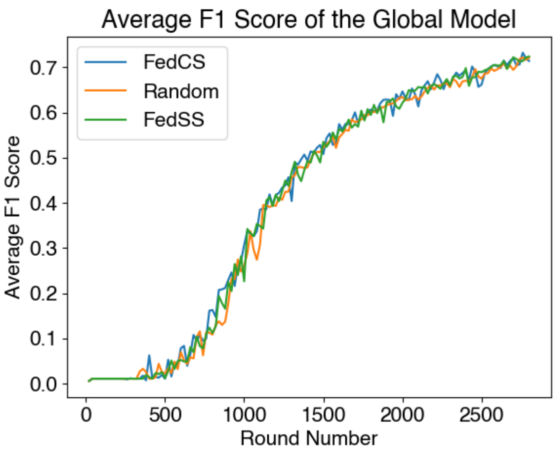


Figure 9: The average F1 score of the global model for FedSS , FedCS and Random Selection.

Table 2 shows the breakdown of f1-score for the slowest and the fastest clients trained during our evaluation. The results are consistent with the accuracy scores. FedSS achieves results similar to FedCS for the fastest clients and FedSS improves the results by 10% for the slowest clients as compared to FedCS. Similar to accuracy, the results from FedSS matches to random selection while taking considerably lesser amount of time to train.

F1 score for 4 Slowest Clients				F1 score for 4 Fastest Clients			
Client	FedCS	Random Selection	FedSS	Client	FedCS	Random Selection	FedSS
1	67.10	78.08	77.22	1	81.48	83.72	84.45
2	69.00	72.58	73.51	2	56.34	56.86	60.85
3	54.04	66.32	69.97	3	77.50	73.60	75.64
4	62.28	68.02	67.69	4	85.13	79.39	78.14
Average	62.60	71.25	72.09	Average	75.11	73.39	74.77

Table 2: A breakdown of F1 score of model across 4 slowest and 4 fastest clients for each of the selection strategies.

7 Future work

We discuss the limitations of FedSS and the future work in this section. While FedSS achieves the best trade-off between training time and data heterogeneity in our testbed, we have some restrictions in experiments which need more work and real world experimentation.

Experimental Setup: Currently, we are bound to work on our local setup, which restricts us to be able to run experiments with only 20 clients. Running with more clients results in system crashing. We believe better performance on larger number of clients as this improves clustering. Also, it would be really valuable to be able to run our setup with some real devices, along with implementations to track client environment.

Handling Churn: Currently FedSS assumes that all clients join in the beginning and maintain available during the whole training procedure, which may not be true in real-world federated learning. Recalling that it takes a few rounds for FedSS to measure and profile the new client's network condition and computation capability, it can be hard for FedSS to track this information accurately in the scenario where clients join and leave frequently.

To mitigate the possible performance degradation due to the measurement and profiling delay of new connected clients, we propose to collect the hardware specifications of clients, such as processor types and memory size, to initialize the grouping based on some regression models. The rationale behind it is that the clients with similar hardware should have similar computation capability under most circumstances (unless many other background tasks compete for resources). Whenever a new client joins, FedSS takes one RTT to collect this information and determine its initial group based on the model, which is still better than random initialization.

Data Heterogeneity: FedSS treats all clients equally in every selection, which may not be the optimal solution. Although the selection is unbiased, the data heterogeneity itself among clients is another important source of bias [13, 14, 28]. If a subset of clients with similar kind of data are picked most of the times, model will converge faster but can have severe bias to the data distribution of that subset and may not be able to capture the true global data distribution.

To overcome this issue, we want to explore the possibility to introduce the training loss of individual clients into client selection process. Specifically in the every round, if the training loss for a particular client is non-significant, we exclude it from selection for next few rounds. This will improve the probability of other clients contributing towards the global model and also prevent model askew.

Scalability and Fault-tolerance FedSS applies only one server in the system, which induces communication bottleneck and a single point of failure. We plan to leverage locality-aware multi-server architecture to enhance the scalability, fault-tolerance and even training performance in the future.

Firstly, a locality-aware server tends to schedule and disseminate its model to local clients with better network service, which alleviates the communication bottleneck at the server side and reduces the ratio of stragglers resulting from network. Secondly, multiple servers can apply a consensus algorithm to replicate the training state of each other to tolerate failure. Lastly, for geographically distributed applications such as next work prediction [12] and geo-local language translation [23], a locality-aware server can captures and preserves those geo-local characteristics better.

8 Conclusion

Federated learning due to its distributed nature of training machine learning model suffers from the issues of data and device heterogeneity. When we talk about device heterogeneity, there exists a trade-off between small training time and bias, as existing schemes end up dropping slower clients. We show this trade-off by comparing existing mechanisms of client selection. We then argue that to eliminate bias, it is necessary to make slow clients part of training. We present FedSS which finds a sweet spot between small training time and handling device heterogeneity by performing smart selection of clients.

References

[1] https://developers.google.com/web/fundamentals/performance/opt-imizing-content-effici- ency/client-hints.

[2] https://worldpopulationreview.com/country-rankings/internet-speeds-by-country.

[3] http://www.netlib.org/benchmark/hpl/.

[4] http://www.netlib.org/linpack/.

[5] Passmark software - cpu benchmarks https://www.cpubenchmark.net/.

[6] Annie Abay, Yi Zhou, Nathalie Baracaldo, Shashank Rajamoni, Ebube Chuba, and Heiko Ludwig. Mitigating bias in federated learning. *arXiv preprint arXiv:2012.02447*, 2020.

[7] K. A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In *SysML 2019*, 2019. To appear.

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[9] S. Caldas, Peter Wu, Tian Li, Jakub Konecný, H. McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *ArXiv*, abs/1812.01097, 2018.

[10] Yae Jee Cho, Jianyu Wang, and Gauri Joshi. Client Selection in Federated Learning: Convergence Analysis and Power-of-Choice Selection Strategies. *arXiv e-prints*, October 2020.

[11] Bong Dae Choi, Doo Il Choi, Yoonju Lee, and Dan Keun Sung. Priority queueing system with fixed-length packet-train arrivals. *IEE Proceedings-Communications*, 145(5):331–336, 1998.

[12] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[13] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479*, 2018.

[14] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[15] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[16] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[17] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[18] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the Convergence of FedAvg on Non-IID Data. *arXiv e-prints*, July 2019.

[19] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data, Apr 2017.

[20] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.

[21] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 417–429, Santa Clara, CA, 2015.

[22] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, May 2019.

[23] Christiane Nord. *Text analysis in translation: Theory, methodology, and didactic application of a model for translation-oriented text analysis*. Number 94. Rodopi, 2005.

[24] Yichen Ruan, Xiaoxi Zhang, Shu-Che Liang, and Carlee Joe-Wong. Towards

Flexible Device Participation in Federated Learning for Non-IID Data. *arXiv e-prints*, June 2020.

[25] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE, 2011.

[26] Zirui Xu, Zhao Yang, Jinjun Xiong, Jianlei Yang, and Xiang Chen. Elfish: Resource-aware federated learning on heterogeneous edge devices. *arXiv preprint arXiv:1912.01684*, 2019.

[27] Yordan. Top 20 most popular phones in 2020, Dec 2020.

[28] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.