GossipScale: Benchmarking Model and System Performance of Gossip Learning at Scale

Jeremy Hsu
Harvard University
jeremyhsu@college.harvard.edu

Kevin Huang
Harvard University
kevin_huang@college.harvard.edu

Steve Li
Harvard University
steveli@college.harvard.edu

***Abstract*—Federated learning is an emerging new paradigm for performing model training. In federated learning, model training and data collection are performed in-situ on client devices and only trained weights are sent to centralized aggregators. The state-of-the-art implementation of federated learning is hierarchical federated learning, where clients feed weight updates to local aggregators, which forward the weights up the tree until they reach the root-level aggregator where final model weights are sent back down to clients to resume training. However, hierarchical federated learning presents a bottleneck at each of the aggregators, which limits client scalability. Gossip learning presents a fully decentralized alternative to hierarchical federated learning. However, there currently are not any standardized testing platforms to benchmark gossip learning implementations at scale. In this project, we show that FedScale, a development and benchmarking suite for federated learning, can be extended to benchmark peer-to-peer learning frameworks such as gossip learning by implementing a naive version of gossip learning using the FedScale framework. We highlight the key differences between traditional federated learning designed around a central set of aggregators compared to a decentralized gossip learning approach. We also show the persistent challenges in comparably benchmarking gossip learning implementations against centralized federated learning implementations. Through our experiments, we show that our implementation of gossip learning achieves comparable results to federated learning under similar conditions.**

1. Introduction

Federated learning [1] is an emerging approach to machine learning model training and testing where local model training and data collection are performed directly on edge devices and centralized actors are used only to aggregate weights across clients. This collaborative approach to machine learning enables edge devices with limited hardware capabilities, lack of data, and concerns about data privacy/data transfer to contribute to machine learning which has seen enormous success with both Apple [2] and Google [3] [4] adopting FL across many different applications.

However, all federated learning designs concentrate a lot of critical and frequent work on a finite number of centralized nodes. This creates a small amount of system-wide bottlenecks, hurting client scalability. In response, alternative distributed learning approaches have been proposed, the most prominent being gossip learning [5]. Gossip learning utilizes ideas from the gossip consensus protocol [6] and provides a fully decentralized, peer-to-peer implementation of distributed model training.

With the rise in both real-world applications for machine learning coupled with growing concerns about the lack of user privacy, researchers and developers are interested in developing privacy-centric distributed training algorithms. An essential step to developing new approaches is directly testing against existing state-of-the-art ones. There exist state-of-the-art evaluation platforms such as FedScale [7] and Flower [8] for federated learning, along with platforms including PeerSim [9] and GoLF [10] for gossip learning. Unfortunately, a standardized, unified platform to compare federated learning and gossip learning implementations directly against each other currently does not exist.

In this paper, we study the challenges involved in designing a gossip learning algorithm within a federated learning testing platform. In particular, we extend the FedScale [7] platform to also include a gossip learning implementation, which we call GossipScale, that can be directly tested against existing federated learning implementations.

2. Background

Federated Learning. Federated learning is an approach to machine learning training that has client-edge devices directly train local models on locally collected data rather than having a centralized set of servers do so. As a result, only weight updates are communicated to a centralized set of nodes, and raw client data can remain strictly on-device. The majority of federated learning implementations involve one centralized aggregator or parameter server [11].

In theory, the current federated learning paradigm should speed up training manyfold as models can be trained simultaneously across many different clients. However, in reality, these approaches suffer from several issues relating to communication overheads for both the parameter-server/centralized-aggregator and clients in the FL fleet.

First, the parameter server must be able to send and receive model parameters (which can be on the order of millions or billions of parameters) to and from clients (which

can also be on the order of millions of devices). No matter how resilient and robust the parameter server implementation is, this still represents a significant bottleneck.

Second, every client must be able to pull the latest model parameters from the parameter server, which can require downstream communication traffic on the order of millions (the number of bytes that must be sent scales linearly in terms of the number of parameters). This also applies to clients pushing their local parameters to the parameters server but this time, it affects upstream communication traffic.

Optimizations for federated learning. Hierarchical federated learning [12] eases the load on the central aggregator by adding multiple layers of intermediate aggregators between the root aggregator and client edge devices, forming a tree structure. Unfortunately, this does not remove the reliance on a centralized set of actors. Furthermore, adding additional intermediate aggregators introduces more points of failure than the single parameter server approach, as all aggregators are still required to perform federated learning.

Model sparsification (in the style of S-FedAvg) [13] proposes to reduce communication traffic by sparsifying the model parameters during model sharing. However, they found that the optimal setup was applying sparsification during client pushes which meant that client pulls and server processing were still major bottlenecks in the process.

Gossip Learning. An alternative framework for federated learning is gossip learning (GL) [5] [14], which addresses the communication bottleneck introduced by the parameter-server/centralized-aggregator paradigm by completely removing the parameter-server from the system design completely. Instead of client edge devices sending their local set of parameters to a parameter server to aggregate and then distribute the new set of updated parameters, client edge devices now share and receive parameters directly with a select subset of neighbor client edge devices. By decentralizing weight aggregation across client edge devices, the parameter-server bottleneck should hypothetically be relieved.

Formally, the optimization problem for a decentralized, gossip learning paradigm is as follows:

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\xi\sim\mathcal{D}_i} F_i(\mathbf{x}_i;\xi)$$

where n is the number of clients in the fleet, $\mathcal{D}_\rangle$ is the data distribution of client i, $\mathbf{x}_i$ is the local model on client i and $F_i(\mathbf{x}_i;\xi)$ is the loss function of client i. The clients collaboratively minimize the global objective function $f(\mathbf{x})$ by minimizing their local loss functions through local updates and exchanging information with neighbors. In a similar way to how the gossip consensus communication protocol [6] is able to converge in a decentralized system, a gossip learning system objective should be able to converge.

3. GossipScale

3.1. FedScale

FedScale [7] is a "scalable and extensible open-source federated learning (FL) engine and benchmark." As a benchmarking suite, it provides data that allows FL developers to investigate how their FL system behaves under several key conditions: "(1) data heterogeneity and (2) device heterogeneity under (3) heterogeneous connectivity and (4) availability conditions at (5) multiple scales on a (6) broad variety of ML tasks." It is the first of its kind to provide such comprehensive coverage and support for these different conditions which are critical for assessing the efficacy of these systems. Figure 1 displays the original FedScale architecture, courtesy of the original authors.

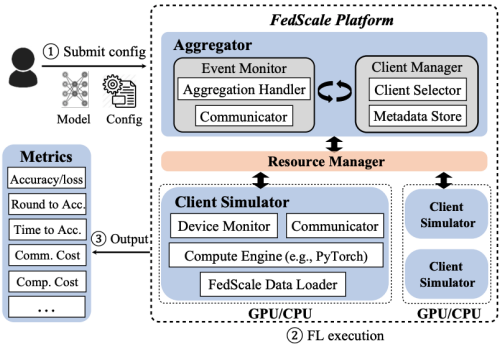


Figure 1: The original FedScale architecture [7], outlining the central aggregator and client paradigm complete with a resource and client manager.

3.2. Architecture

Figure 2 details the high-level architecture of GossipScale. We keep the same general actor setup: a single coordinator (FedScale's term for this actor is *aggregator* but because we are no longer using the centralized server for parameter aggregation, the original term is no longer suitable), multiple executors, and a client co-located with each executor. However, we also made several significant deviations from the FedScale's design. First, we fixed the executor to only house a single client, effectively increasing and shifting the responsibilities of executors to individual clients [1]. Next, we shifted the weight aggregation logic completely away from the coordinator and to the executors. This required us to rewrite substantial portions of the executor's communication layer in order to facilitate peer-to-peer communication as specified by the gossip learning protocol. With these changes, the coordinator's only tasks are to ping all executors to begin and end training. Everything else is handled completely by executors in a decentralized manner. We describe our design in more detail below.

1. With this co-location, we can now use the terms "executor" and "client" interchangeably in the context of GossipScale

3.3. General Workflow

The general workflow of GossipScale is as follows. The *coordinator*, formerly "aggregator," begins accepting gRPC channel connections from participating clients. Once the number of connections successfully initialized by the coordinator reaches a threshold (a user-defined hyperparameter), the coordinator pings all connected clients to initiate gossip learning. Each client will begin training their local model on their data. After a certain amount of local update rounds, each client will select a random subset of neighbors to request weights from and halt local training until it receives weights from a minimum threshold of neighbors and aggregates them with its local weights. The frequency of weight aggregation is also a user-defined parameter.

3.4. Neighbor Selection

FedScale and other federated learning approaches implement an *all-to-all* communication collective pattern via a centralized set of aggregators, in which client devices send their local weights to the aggregator set, the aggregator set performs aggregation of all received client weights, and then the aggregator set broadcasts the updated set of weights back to the original clients. In gossip learning, aggregation itself is performed on client devices (via the executor) and communication does not pass through any centralized aggregator or coordinator. Instead, each executor will select a subset of all neighbors to request weights from. The intuition of gossip learning follows from the gossip consensus protocol: if each participant communicates a different subset of neighbors, then eventually the entire group will converge and reach a consensus.

In FedScale, the central aggregator maintains a fleet-wide view of all client health states and can filter out offline neighbors for communication. However, an inherent challenge of implementing a peer-to-peer consensus algorithm is not being able to rely on any set of nodes as single sources of truth. To work around this, we have each client select a subset of *all* neighbors, whether they are active or not. The `RequestWeight` RPC call also serves as an unintentional health check for a given neighbor: the given neighbor is online and the connection is good if and only if the RPC can be successfully returned. Thus, the sending client can proceed accordingly based on the type of response it receives from any initially selected neighbor.

3.5. Decentralized Aggregation

While decentralizing aggregation theoretically reduces the number of bottlenecks on centralized aggregators, we also must be more careful while handling RPC and server failures as each client asynchronously performs aggregation. In addition to having to retry logic surrounding every RPC call, we also have each client broadcast more RPCs than it actually requires to serve as a buffer in case some of the RPCs fail or there are stragglers.

A more concrete case is as follows. Suppose a client receives an incoming set of weights from a neighbor, but as the result of an outdated weight request (this is entirely possible because clients only wait for a subset of requested neighbors to send their weights for aggregation before moving onto another round). To filter out these "stale" incoming weights, each client will include its current round number in its `RequestWeights` RPC to its neighbors, and each neighbor responds by sending the original client an `UploadWeights` RPC call will include the same round number in its request. Thus, if a client observes an incoming set of weights in its receiver queue with an outdated round number, it can simply ignore the weights and discard them.

3.6. Evaluating Performance

Evaluating machine learning models in a traditional setting where both the model and data are centralized can be achieved by running an evaluation run every certain number of training iterations/epochs on an isolated training dataset. This procedure is well documented and has been made trivial by existing machine learning and deep learning frameworks, requiring only a few lines of code.

Model evaluations become more challenging in a federated learning context where clients do not share data. As a result, federated evaluation [15] has developed as a subset of the overall federated learning field of study to determine strategies for conducting evaluations of federated learning fleets. For example, the existing FedScale implementation orchestrates a testing round by having the central coordinator first send an update of the most up-to-date model parameters to all clients and then starting a testing job on each client. It then aggregates all the key metrics that each of the clients returns upon completion (round duration, loss, accuracy, etc.) which the coordinator logs. This is even more challenging in a gossip learning context. In particular, while in federated learning, the model that gets tested is the most recent model snapshot captured by the aggregator, for gossip learning, there is no agreed-upon model. As a result, what it means to evaluate the system becomes much more nuanced and is explained more in Section 5.

4. Implementation

We implemented GossipScale as a fork of FedScale, making sure to present the same API as FedScale so we could use the same benchmarking setup and evaluation dataset. FedScale is implemented in Python, uses the gRPC [16] library for inter-node communication, and PyTorch [17] for on-device training.

5. Evaluation

5.1. Experimental and Hardware Setup

We ran our experiments on an Intel machine with an NVIDIA A100 GPU, 64 CPU cores, and anywhere from

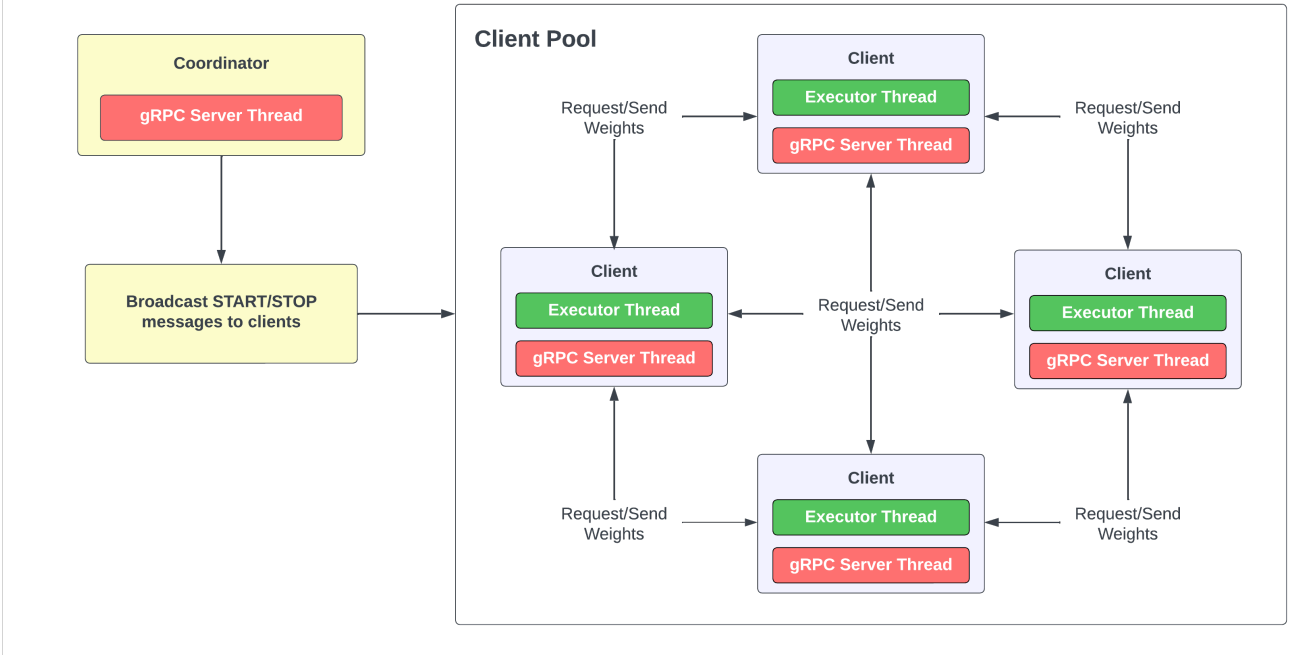


Figure 2: High-level overview of GossipScale's architecture.

64GB to 150GB of RAM depending on the task. RAM usage was high due to the number of threads required for each client: at minimum 1 for the training loop, 1 for the gRPC server, and 1 for any wandb logging and handling. We ran each experiment for 200 training rounds and 40 clients. We chose the CIFAR-10 [18], FEMNIST [19], and Google Speech Commands [20] datasets for our experiments.

CIFAR-10: The CIFAR-10 dataset [21] consists of 60000 32x32 color images across 10 classes, with a 50000-10000 train test split. We used ResNET-50 [22] as our benchmarking model.

FEMNIST: FEMNIST [23], or Federated Extended MNIST, is a modified version of Extended MNIST by partitioning the data based on the writer of the digit or character. We utilized ResNET-18 [22] for our benchmarks.

Google Speech Commands: The Google Speech Commands dataset [24] is an audio dataset of spoken words designed to help train and evaluate keyword spotting systems. With over ten thousand clips of one-second-long duration, each clip contains one of 35 common words spoken by thousands of different people. We benchmarked this dataset using ResNET-34 [22].

Accuracy was determined by taking the most recently reported accuracy score by any client. Note that model weights might not correspond to the same training or testing round – one could imagine that a client A might lag behind client B and as a result, finish testing round i much later than client B. Gossip learning is in part resilient to this type of large because even if client A lags behind client B, when it requests weights from client B, it will receive an up-to-date snapshot of client B's weights. We chose to do this for simplicity reasons, but a more principled approach might be to have the coordinator orchestrate testing events across the entire fleet every N minutes and aggregate the results.

Additionally, the time statistic used in the TTA calculation was taken to be the total of all training round durations. This calculation was based on FedScale's own `global_virtual_clock` calculation which is the total of all training round durations. However, FedScale also has some additional computations that might explain the some of the differences concerning TTA, outside of system efficiency.

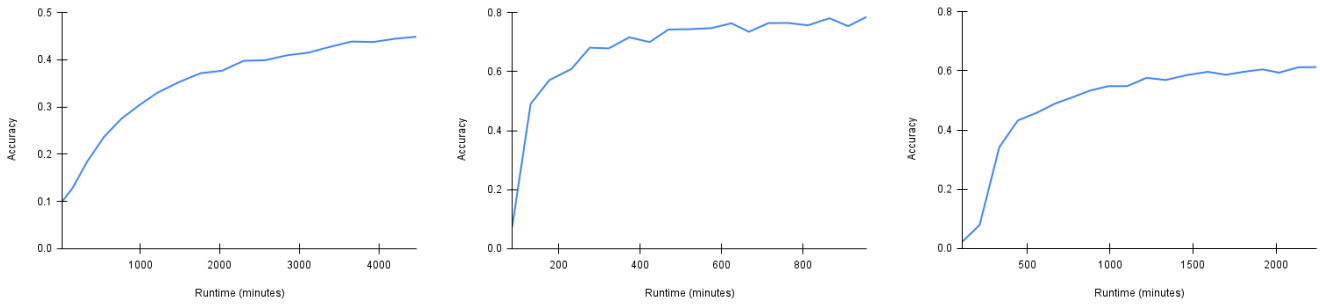
With regards to the discussion on alternative approaches to accuracy and discrepancies in TTA, more explanation is in Section 6.

Dataset	Model	Type 1 Accuracy	TTA (minutes)
CIFAR-10	ResNET-50	38%	1000
Google Speech Commands	ResNET-34	42%	6000
FEMNIST	ResNET-18	69%	3000

TABLE 1: Gossip Learning performance on CIFAR-10, Google Speech Commands, and FEMNIST. Time to accuracy is calculated approximately in minutes.

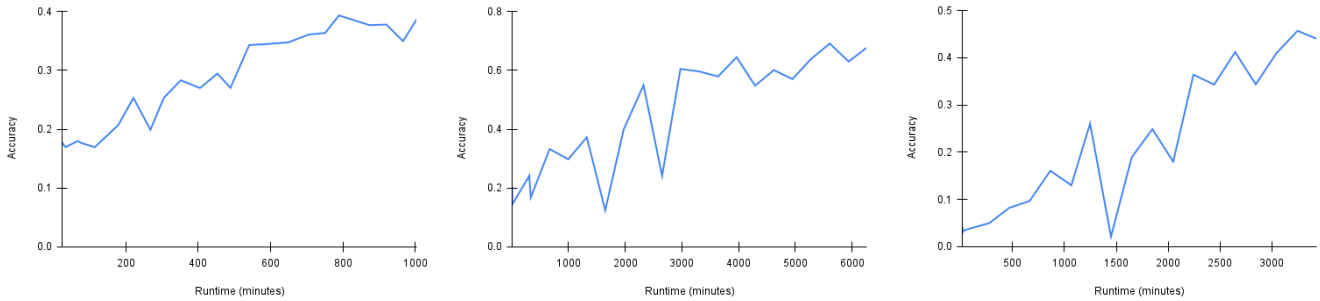
Dataset	Model	Type 1 Accuracy	TTA (minutes)
CIFAR-10	ResNET-50	45%	4474
Google Speech Commands	ResNET-34	62%	2248
FEMNIST	ResNET-18	79%	1000

TABLE 2: FedScale performance on CIFAR-10, Google Speech Commands, and FEMNIST. Time to accuracy is calculated approximately in minutes.



(a) FedScale accuracy convergence on CIFAR-10 (b) FedScale accuracy convergence on FEMNIST (c) FedScale accuracy convergence on Google speech

Figure 3: FedScale benchmarking results on different datasets



(a) GossipScale accuracy convergence on CIFAR-10 (b) GossipScale accuracy convergence on FEMNIST (c) GossipScale accuracy convergence on Google speech

Figure 4: GossipScale benchmarking results on different datasets

5.2. Analysis and Takeaways

Figure 4 shows the accuracy and loss convergence of our three evaluation datasets and the time it took to complete 200 training rounds for both gossip and federated learning. Table 1 and Table 2 show the final accuracy values. Across the board, we see accuracies steadily increase over time for gossip learning. Given that neither value plateaus during the 200 training rounds, we believe that even better performance can be achieved with more training steps. We plan to improve our testing infrastructure and implementation to efficiently run for longer training periods, as discussed in Section 6. The accuracy graphs for FedScale follow a smoother curve because they report testing results more frequently than we do.

In terms of accuracies, our gossip learning implementation was able to achieve similar results to the federated learning approach, with gossip learning type 1 accuracy on FEMNIST being 69% compared to 79% for federated learning while at the same time achieving 38% compared to 45% on CIFAR-10. While the accuracy for gossip learning on Google Speech falls short of federated learning by 20%, we believe that our experiments show that event with a naive implementation, gossip learning has the potential to perform the same if not better than FedScale.

Unfortunately, the same cannot be said about TTA. FedScale has a significantly lower TTA than gossip learning, save for CIFAR-10, and we attribute this to their executor-aggregator setup as they have multiple clients being handled on a single executor, meaning that simulation for multiple clients happens on one thread. In contrast, we have each client being trained on its *own* thread, which can slow down runtimes when simulating multiple clients. We address some of our compute and memory issues with our approach in Section 6.

6. Limitations and Future Work

While we were able to develop a functional gossip learning implementation on top of FedScale, several limitations exist that we wish to address in future work beyond this course.

Limitation 1: limited access to compute and storage. Full federated and gossip learning simulation requires extensive computation and storage resources. Due to our finite access to compute and server storage, the amount of datasets and clients that we were able to simulate was limited. Several datasets provided by FedScale, such as OpenImages [25] and Reddit [26], are nearly a 100GB or larger in size, much more than what's allotted in the FASRC

cluster or AWS compute. FedScale evaluated their system on 10 NVIDIA Tesla P100 GPUs and likely up to 10 times more CPU cores than we had available, making replicating their results difficult. A more extensive evaluation of our system on different tasks and models, such as NLP-related tasks, would strengthen our results.

Another consequence of being constrained to a finite amount of compute per round is that simulating multiple clients requires multiplexing all of them across double the amount of threads on a single GPU (recall that each client uses an executor and gRPC thread). Gossip learning imposes a much higher RAM overhead on individual edge clients. As a result, using too many logical threads causes contention on GPU resources, leading to unintended system effects. For instance, with 50 clients, we observed gRPC servers shutting down. We plan to address these issues in further iterations of our work.

Limitation 2: lack of a centralized method to measure TTA in gossip learning. A future area for exploration is how to determine accuracy and the overall TTA. We currently use the heuristic that the most recently reported test accuracy score is representative of the most up-to-date model and as a result, we report the most recently reported accuracy score by any client as the model accuracy. As mentioned earlier, a more principled approach would be to use the coordinator to orchestrate fleet-wide testing rounds and aggregate all the results together. However, FedScale's implementation of evaluating system heterogeneity complicates this approach. Specifically during simulation and benchmarking, the FedScales' calculation of the duration of a training round is not the actual wall-clock time but rather a function of computation latency and communication latency derived from device trace data that FedScale has available (i.e., `virtual_clock ! = wall_clock`). Therefore, even if in real time two clients are on the same round, one client's simulated time might be much further behind the other client. To accurately simulate gossip learning, the client at simulated time a should request weights from its neighbor from when it was at simulated time a. This requires each client to save a snapshot of the model for every conceivable timestamp which is not possible. Alternatives include introducing sleeps in each client such that the `wall_clock == virtual_clock`, storing a snapshot for every N iterations and only as far back as the slowest running client needs but both have their strengths and weaknesses and require further exploration

Limitation 3: difficulty with real-world testing at scale. Finally, we would like to have extended GossipScale to real-world settings. For example. FedScale provides infrastructure for running their benchmarks on Android devices. Given our limited access to any fleet of edge devices, we were unable to properly expand into this setting.

7. Related Work

PeerSim [9] is a scalable peer-to-peer simulator implemented in Java for testing a wide range of peer-to-peer algorithms and protocols (including distributed learning algorithms such as gossip learning). Gossip Learning Framework (GoLF) [10] is an open-source benchmarking framework built on top of PeerSim allowing users to develop and test their own peer-to-peer algorithms. As with PeerSim, GoLF provides a realistic simulation of an unreliable large-scale network by allowing developers fine-grained customization options. Both of these platforms are excellent and have seen widespread use by peer-to-peer algorithm developers. However, they do not provide an easy way to also test more centralized federated learning approaches.

Flower [8] is another federated learning framework that provides a realistic platform to develop and test federated learning implementations. Similar to FedScale, Flower presents an easy-to-integrate API and provides users with the option to simulate device heterogeneity and networks with different capabilities. However, like FedScale, Flower does not have support for testing truly peer-to-peer algorithms such as gossip learning, as its internal architecture is set up for primarily testing hierarchical and/or centralized federated learning implementations.

8. Conclusion

In this paper, we discussed the motivation behind different distributed learning frameworks that keep models and data on client edge devices, namely federated learning and gossip learning. We highlighted the challenges in evaluating gossip learning implementations with respect to established federated learning algorithms and the current lack of unified benchmarking infrastructure. In response, we have developed GossipScale, a fork of the popular FedScale platform for federated learning, which aims to implement an extensible version of gossip learning within the FedScale engine for direct evaluation against federated learning implementations. GossipScale is publically available on GitHub [27]. We were able to demonstrate results from GossipScale that show that gossip learning has the potential to be a more effective approach to distributed model training than federated learning. However, we also observed persistent challenges that must be solved going forward in order to fully develop a unified testing platform, namely determining a standardized system-wide accuracy equation for both gossip learning and federated learning. With that said, we are confident in the potential for the future of gossip learning, which will be accelerated by providing more standardized platforms to test it against alternative approaches.

Acknowledgments

We would like to thank Prof. Minlan Yu and the course staff of CS 243 for providing an excellent seminar on the current state of research at the intersection of computer networks and machine learning, and for their guidance and advice throughout the duration of this project.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2023.

[2] M. Paulik, M. Seigel, H. Mason, D. Telaar, J. Kluivers, R. van Dalen, C. W. Lau, L. Carlson, F. Granqvist, C. Vandevelde, S. Agarwal, J. Freudiger, A. Byde, A. Bhowmick, G. Kapoor, S. Beaumont, Áine Cahill, D. Hughes, O. Javidbakht, F. Dong, R. Rishi, and S. Hung, "Federated evaluation and tuning for on-device personalization: System design & applications," 2021.

[3] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019.

[4] "Federated learning on google cloud," 2022. [Online]. Available: https://cloud.google.com/architecture/federated-learning-google-cloud

[5] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, p. 556–571, May 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.2858

[6] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, jul 1982. [Online]. Available: https://doi.org/10.1145/357172.357176

[7] F. Lai, Y. Dai, X. Zhu, H. V. Madhyastha, and M. Chowdhury, "Fedscale: Benchmarking model and system performance of federated learning," in *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, ser. ResilientFL '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–3. [Online]. Available: https://doi.org/10.1145/3477114.3488760

[8] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2022.

[9] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, Seattle, WA, Sep. 2009, pp. 99–100.

[10] R. Ormándi, I. Hegedűs, and M. Jelasity, "Asynchronous peer-to-peer data mining with stochastic gradient descent," in *17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011)*, ser. Lecture Notes in Computer Science, vol. 6852. Springer-Verlag, 2011, pp. 528–540.

[11] C. He, A. D. Shah, Z. Tang, D. F. N. Sivashunmugam, K. Bhogaraju, M. Shimpi, L. Shen, X. Chu, M. Soltanolkotabi, and S. Avestimehr, "Fedcv: A federated learning framework for diverse computer vision tasks," 2021.

[12] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[13] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2017.

[14] I. Hegedundefineds, G. Danner, and M. Jelasity, "Gossip learning as a decentralized alternative to federated learning," in *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference, DAIS 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2019, p. 74–90. [Online]. Available: https://doi.org/10.1007/978-3-030-22496-7_5

[15] B. Soltani, Y. Zhou, V. Haghighi, and J. C. S. Lui, "A survey of federated evaluation in federated learning," 2023.

[16] "grpc: A high performance, open source universal rpc framework." [Online]. Available: https://grpc.io

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744

[19] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.

[20] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[21] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[23] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.

[24] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[25] I. Krasin, T. Duerig, N. Alldrin, A. Veit, S. Abu-El-Haija, S. Belongie, D. Cai, Z. Feng, V. Ferrari, V. Gomes, A. Gupta, D. Narayanan, C. Sun, G. Chechik, and K. Murphy, "Openimages: A public dataset for large-scale multi-label and multi-class image classification." *Dataset available from https://github.com/openimages*, 2016.

[26] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire, and J. Blackburn, "The pushshift reddit dataset," 2020.

[27] J. Hsu, K. Huang, and S. Li, "Gossipscale." [Online]. Available: https://github.com/hsujeremy/FedScale