Federated Learning with a Highly-Scalable Pipeline

Sebastian Alcaino Jaque
sia14@sfu.ca
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

Amir Mousavi
amir_mousavi@sfu.ca
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

ABSTRACT

Federated Learning (FL) is a machine learning paradigm that lately has been getting popular due to known data privacy protection by training models on local data in own devices without user data ever entering a centralized server [2]. One of the advantages of a distributed approach to machine learning training is that it allows for a large number of end nodes to work on local data, effectively distributing computing resources and storage that otherwise would prove infeasible to do in a classic machine learning environment. A challenge in this type of training is how to communicate the training results in between devices and a 'global' model in a way that can allow to scale and be fault tolerant.

The objective of this project is to implement a highly scalable federated learning framework with a centralized averaging and global model that could be able to handle millions of update messages from clients training on their own local data. As a proof of concept, we propose to achieve this by deploying our FL model using a Kafka cluster as the network infrastructure supporting the communication channel in a large-scale scenario. Deployment of the Kafka cluster will be in Google Cloud Platform (GCP).

This would prove a viable solution for a FL algorithm by leveraging GCP's reliable infrastructure, Kafka's high fault tolerance based on log replication and distributed messaging, this approach allows to have an asynchronous connection between the averaging server and the clients, surpassing the usual connectivity instability that edge devices may have.

Finally we show our system testing results by simulating a scenario where 20 clients are training at the same time with their own local data and updating a common global model.

KEYWORDS

Distributed Systems; Cloud; Federated Learning; GCP; Kafka

ACM Reference Format:
Sebastian Alcaino Jaque and Amir Mousavi. 2021. Federated Learning with a Highly-Scalable Pipeline. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

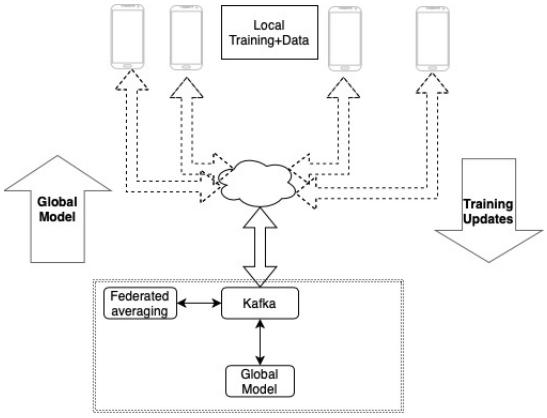

Figure 1: Figure depicts high level view of proposed architecture.

Federated Learning has recently been getting traction because of its privacy-preserving promises. Since this is an exciting and new field in the world of distributed computing and machine learning, there are some main challenges that face FL, namely (1) expensive communication, (2) systems heterogeneity, (3) data heterogeneity, and (4) privacy shortcomings [2, 10]. While the field has made much progress in all of above barriers and continues to require contributions from researchers, we have identified the expensive communication aspect as an area that can be improved using existing open-source frameworks.

Communication remains a huge bottleneck for a large FL infrastructure, where a large number of devices have to communicate with a central server by receiving instructions on edge devices and later sending back the resulting updates after training. Existing FL solutions to reduce communication costs are divided into two approaches: (1) reducing the number of exchanged messages, and (2) minimizing the size of the exchanged messages (i.e. message compression) [10]. Federated Averaging (FedAvg) [11] - an algorithm which combines gradients from all edges and only sends their average to the server - is an example of the former, while Golomb lossless encoding [12] is an example of a compression algorithm.

However, in our opinion, another possible avenue of achieving better communication-efficiency and fault tolerance is taking advantage of a robust and proven distributed message broker like Kafka [10]. Kafka has proven itself as the de-facto stream processing framework capable of handling millions of messages a second [8], it offers solid horizontal scalability so that we can accommodate large sets of clients, in addition to being replicated to provide fault-tolerance as well.

In the scope of this project, we will use the existing FedAvg algorithm and will use a compression schema that leverages Kafka's

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

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

compression features to transmit data, this would help us in decreasing the size of the transmitted messages smaller.

1 SYSTEM DESIGN

Our implementation of the solution will be based on a federated learning python framework called Flower [1]. This framework uses Google's Remote Procedure Call (gRPC) [5] to act as the network pipeline supporting it. Our goal is to extend Flower architecture, replacing the usage of gRPC as the network layer and integrate Kafka messaging as an asynchronous pipeline based in a message queueing system. By integrating Kafka as the message processing pipeline into the federated learning system, we are hoping to see a more resilient and fault-tolerant system while at the same time enabling the system to scale up to a large number of clients.

Why Kafka? Kafka is one of the most powerful streaming platforms and it is open-source, it offers horizontal scalability, high reliability due to replication, and excellent parallelism[13]. Kafka uses a distributed commit-log in its data structure, this means that is writing all messages to disk, and since the commit-log is sequential, the writing operations are linear in time too. Additionally, Kafka uses optimizations like having a binary data structure that allows the use of zero-copy where the message received gets written directly to its memory address in disk without going through Kafka first - this gives Kafka fast performance and allows disk operations near network speeds[7].

For our proof of concept implementation we will focus on a simple ML model: hand-written digit classification based on the MNIST dataset [9], where we will send tensors and parameters from clients to the server and back from the server using our network layer running on Kafka.

First, the device has to schedule or detect the best time to start training without interrupting the user and ensuring not to affect the device's normal use performance. This aspect of the client implementation is left out of the scope of this project and we assume a device is ready to start training as soon as it registers itself with our client manager server. Registration is done by the client sending a registration message to the server using a Kafka topic. Once the registration message is received by the server, it will create the objects required to interact with normal Flower workflow.

The server instance uses a Kafka consumer querying a designated Kafka topic which all clients send messages to. When clients send messages to this topic, the server forwards them to bridge objects between the server and the federated averaging strategy object that handles the different steps involved in a round, such as initialization, tensors distribution and evaluation. When a message needs to be sent to the client, the bridging objects receive a request from the averaging handler which in turn is sent to a specific client on a separate Kafka topic used by the server to push messages to a client identified by a client id received from the client at registration time. As mentioned, when a client starts, it is assumed to be ready to work, so it sends a registration message to the server with its unique id. After this, clients will start a Consumer object to receive messages on a Kafka topic associated with its unique id - this is where the server will push messages to it. When the client receives a message, it pushes it to the local client that is implementing the machine learning steps. These messages for example include those for requesting a round of training or model weights to be sent.

The following sequence diagrams cover the workflow for each use case described above.

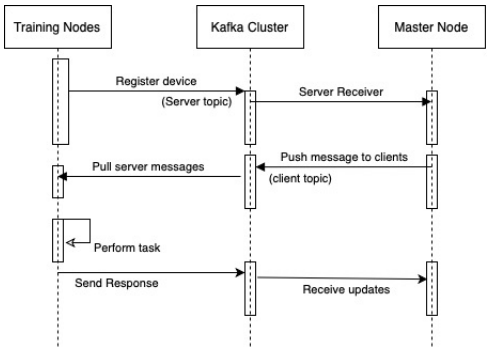

Figure 2: Figure describes the workflow of the solution proposed.

2 IMPLEMENTATION

Our implementation will comprises of a Kafka cluster, a central server to orchestrate training and evaluation rounds, and the required edge devices (e.g. GCP instances which act as a client). The code repository is publicly available on GitHub [1].

2.1 Components

Our cluster is currently deployed in Google Cloud with the following resources:

(1) One server instance orchestrating the training rounds and aggregating all weights from the clients based on a strategy. [2]
(2) One instance acting as the client that will run the actual training of the model. This instance will be replicated and tuned so that can work for our load testing.
(3) One Kafka server running in a n1-standard-2 instance.
(4) Cloud function acting as client for tests

2.2 Central Server

We have set up an n1-standard-2 instance as our server that will be running federated averaging algorithm. This server will communicate with client instances through the Kafka cluster, and will be responsible for (1) initiating training jobs, (2) receiving updates from clients, and (3) propagating updates to the rest of the clients.

The following shows how a training round works in our solution. Once the server has confirmed the minimum number of clients are registered, it initiates the training round by obtaining the initial weights from a random client, it runs evaluation on these to have a base and then pushes the initialized weights to clients along with a 'FIT' request. This signals the clients to use these weights to start the training round. Once the clients have finished local training, they will send the updated weights in a response to the server.

[1] FDxKafka Repository
[2] Flower Strategies

The server will average the results according to the averaging strategy and then requests evaluations of these results performed by the clients (Evaluations can also be performed on the server-side). Afterwards, a new round begins following the same procedure until the training has been completion. Upon completion, the server will send a disconnect message to clients and report back the configured metrics (e.g. loss, total samples, etc.)

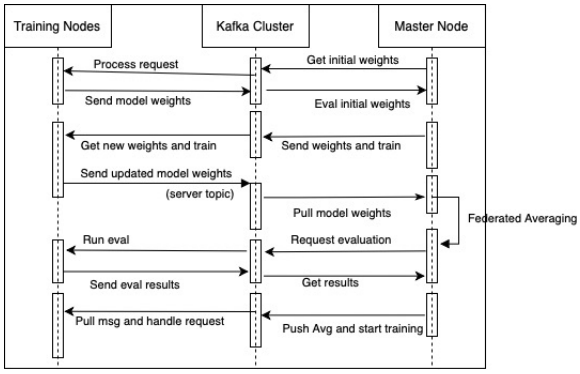

Figure 3: Figure describes the workflow of a training round.

2.3 Kafka: Summary

Both the central server and edge devices are communicating via Kafka consumers/producers. Upon initialization of a training round, the server will publish a message with the training metadata (e.g. weights, batch size, properties, etc) to clients. On the other side, clients will publish their messages to the topic associated with its client id after having processed the requested action by the server, and this cycle will continue until the required number of rounds have passed. Each message will contain the task type, for example, the server can issue "FIT", or "EVALUATE" commands, while the client can return messages about the progress with "RES_WEIGHTS" or "RES_EVALUATION" back to the client. For more details about the specific integration of Kafka, feel free to check out our implementation or Flower documentation found in references section.

3 EVALUATION PLAN

For evaluating the system performance and correctness, we implemented client simulation scripts that act as what a real client would. In order to have multiple clients connected and training at the same time, we created Cloud Functions[4] that would run the client code to simulate clients, each http request done to this function starts a new client that connects to the server to start training. We then used scripts to make many requests at the same time so that we would have a load test in the system.

In order to evaluate our implementation, we need to verify if it improves performance in handling the updates from the client nodes and adds fault-tolerance to the system compared to a base benchmark. For the base benchmark, we used a normal Flower client/server that will keep count of the number of messages and length of the training round.

When executing the evaluation we started with running 100 clients, soon we realized that Cloud functions have a much lower limit for simultaneous requests, forcing us to decrease the size of the test. Regardless of this, the scripts produced to test the system allow to test any number of clients that will only be limited by the available resources for the test. The evaluation we ran had 20 clients connected at the same time and training on local data, this proved the concept that Kafka can be used as a network pipeline for training in a distributed way. Our baseline took 37.77 seconds, and our implementation with Kafka took 42.5 seconds. We believe that for a test with a relatively small number of clients, the advantages of using Kafka are not so evident. Our approach would show more impact when the number of clients connected increases, or when clients do not have a stable connection that would benefit from Kafka's fault-tolerance.

4 AREAS OF IMPROVEMENT

There are some areas of the code where we had to take the shortest path in terms of getting the proof of concept implementation running. One example would be the serialization and deserialization we do to wrap a Flower message into a Kafka message. This could be avoided by modifying lower level code in Flower to only add a single client id field, and would result in faster encoding of messages for transmission. Cloud resources for developing and testing our approach were limited our ability to truly test our system. Having these resolved would also allow for an improved comparison in between the base implementation of Flower and our Kafka approach.

5 CONCLUSION

Federated Learning is a technology being developed and has shown an increase in popularity due to its desirable advantages, we specifically focus in a specific case where the data defining the optimization are distributed over an extremely large number of nodes[6], but the goal to train a high-quality centralized mode taking advantage of the large number remains. Apache Kafka has shown to be reliable, fault tolerant and more importantly for our goal, highly scalable, where clients can talk to multiple brokers at the same time even to the same topic as is the case of the server due to its topic partitioning [3]. We propose an alternative to allow growing to a large scale federated learning, where the amount or nature of data being trained on makes it impossible to have a centralized classical machine learning algorithm. This machine learning paradigm could have an immense impact in how we collect data to train and what other type of information we could learn without the restrictions of sharing private data. In the near future, we hope to polish up our code and propose it as an open-source contribution to the Flower project.

REFERENCES

[1] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro P. B. de Gusmão, and Nicholas D. Lane. 2021. Flower: A Friendly Federated Learning Research Framework. arXiv:2007.14390 [cs.LG]

[2] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. arXiv:1902.01046 [cs.LG]

[3] Apache Software Foundation. [n.d.]. Apache Kafka. https://kafka.apache.org/documentation/#intro_guarantees.

[4] Google. [n.d.]. GCP: Cloud Functions. https://cloud.google.com/functions.

[5] Google. [n.d.]. gRPC. https://grpc.io/about/.
[6] Jakub Konečný, Brendan McMahan, and Daniel Ramage. 2015. Federated Optimization:Distributed Optimization Beyond the Datacenter. arXiv:1511.03575 [cs.LG]
[7] Stanislav Kozlovski. [n.d.]. A Thorough Introduction to Apache Kafka. https://accu.org/journals/overload/28/158/kozlovski/.
[8] Jay Kreps. 2011. Kafka : a Distributed Messaging System for Log Processing.
[9] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (2010). http://yann.lecun.com/exdb/mnist/
[10] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (May 2020), 50–60. https://doi.org/10.1109/msp.2020.2975749
[11] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. arXiv:1602.05629 [cs.LG]
[12] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. 2019. Robust and Communication-Efficient Federated Learning from Non-IID Data. arXiv:1903.02891 [cs.LG]
[13] Bargunan Somasundaram. [n.d.]. All about Apache Kafka – An evolved Distributed commit log. https://www.linkedin.com/pulse/all-apache-kafka-evolved-distributed-commit-log-bargunan-somasundaram.