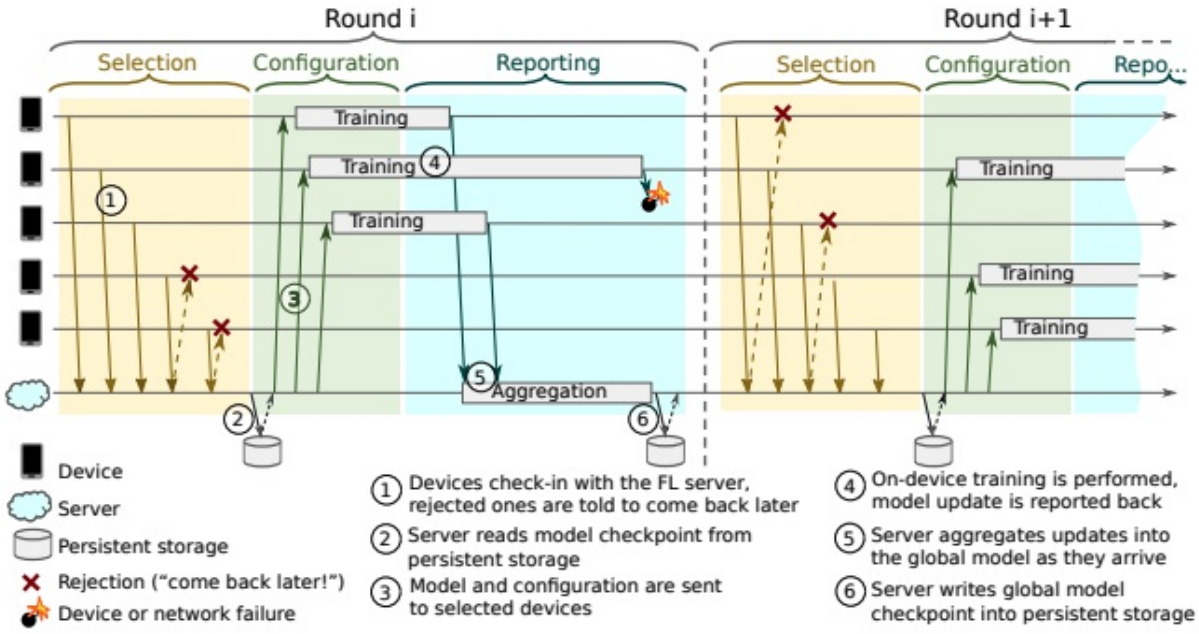
Summary of "Towards Federated Learning at Scale: System Design"

Xiangfeng Zhu(zxfeng), Jiachen Liu(amberljc), Chris Chen(zhezheng)

Problem and Motivation

The goal is to build a system that can train a deep neural network on data stored on the phone which will never leave the device. The weights are combine in the cloud with Federated Averaging, constructing a global model which is pushed back to the phone for inference.

Solution Overview



Device-server protocol

Each round of the protocol contains three phases.

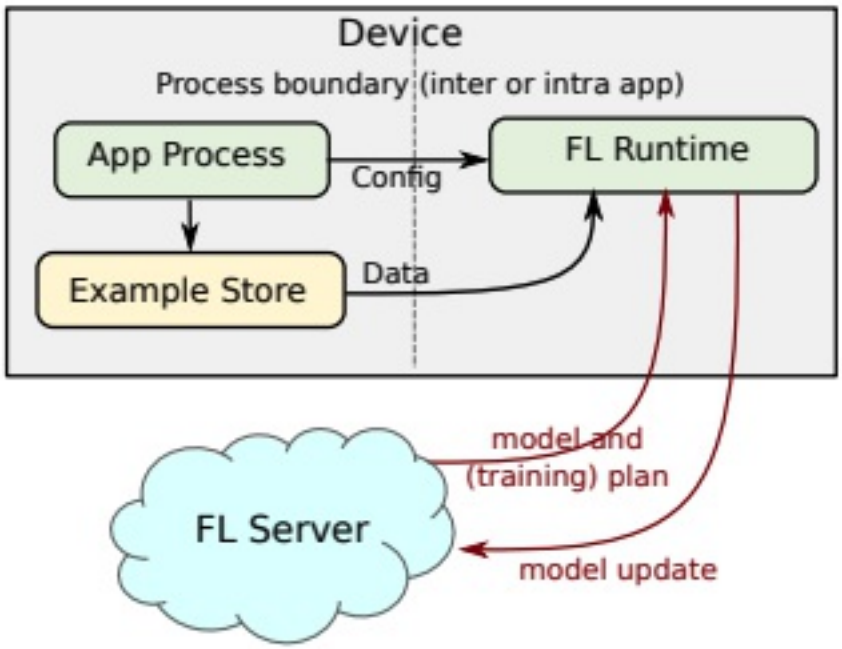
1. Selection: The server(i.e. the coordinator in the server) picks a subset of available devices to work on a specific FL task.

2. Configuration: The server sends the FL plan(execution plan) and the FL checkpoint(i.e. a serialized state of a Tensorflow session) with the global model to each of the chosen

devices. When receiving a FL task, the FL runtime will be responsible for performing local training.

3. Reporting: The server waits for the participating devices to report updates. The round is considered successful if enough devices report in time. (The update to the model is often sent to the server using encrypted communication.)

As an analogy, we can interpret the FL server as the reducer, and FL devices as mappers.

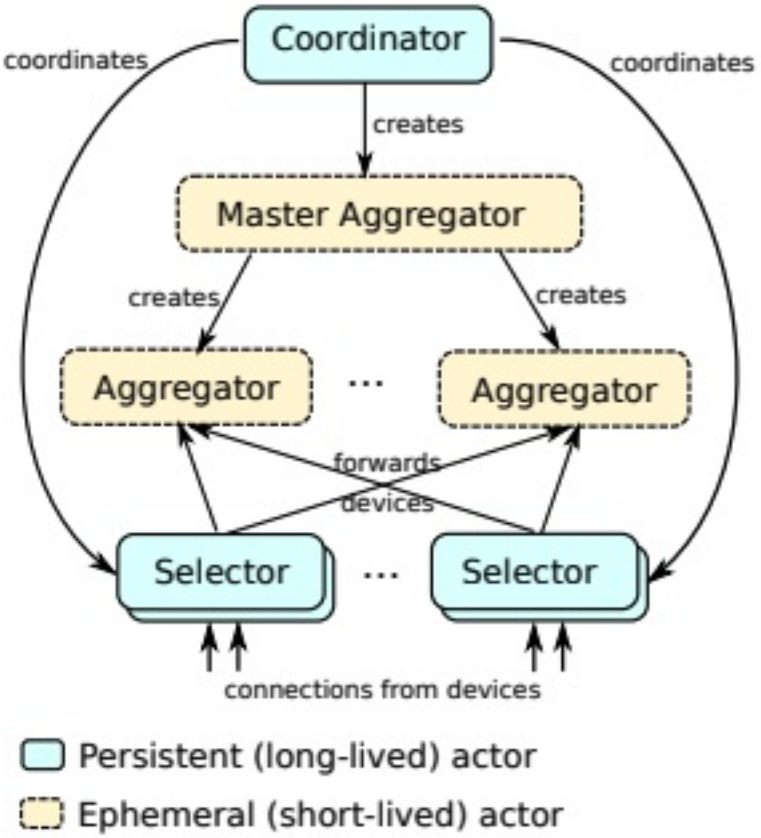


Device

The device should maintain a repository of locally collected data for model training and evaluation. Applications are responsible for making their data available to the FL runtime as an example store(e.g. an SQLite database recording action suggestions show to the user and whether or not these suggestions were accepted). When a task arrived at the device, the FL runtime will access an appropriate example store to compute model updates.

Two things to note here: 1. We need to avoid any negative impact on the user experience. Thus, the FL runtime will only start the task when the phone is idle, connected to the WiFi/power etc. 2. FL plans are not specialized to training, but can also encode evaluation tasks.

For other details, such as multi-tenancy and attestation, please refer to the paper.



Server

The FL server is designed around the Actor Programming model. The main actors include:

- Coordinators are Top-level actors(one per population) which enable global synchronization and advancing rounds in lockstep. As previously mentioned, The Coordinator receives information about how many devices are connected to each Selector and instructs them how many devices to accept for participation, based on which FL tasks are scheduled.

- Selectors are responsible for accepting and forwarding device connections. After the Master Aggregator and set of Aggregators are spawned, the Coordinator instructs the Selectors to forward a subset of its connected devices to the Aggregators, allowing the Coordinator to efficiently allocate devices to FL tasks regardless of how many devices are available

- Master Aggregators manage the rounds of each FL task. In order to scale with the number of devices and update size, they make dynamic decisions to spawn one or more Aggregators to which work is delegated.

Federated Averaging

FedAvg is a variation of traditional Stochastic gradient descent(SGD) algorithm, which combines local SGD on each client with a server that performs model averaging.

At the beginning of each round, a random fraction C of clients is selected, and the server sends the current global algorithm state to each of these clients (e.g., the current model parameters). We only select a fraction of clients for efficiency, as the experiments show diminishing returns for adding more clients beyond a certain point. Each selected client then performs local computation based on the global state and its local dataset, and sends an update to the server. The server then applies these updates to its global state, and the process repeats.

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

The amount of computation is controlled by three key parameters: C, the fraction of clients that perform computation on each round; E, the number of training passes each client makes over its local dataset on each round; and B, the local minibatch size used for the client updates.

However, the paper does not provide any theoretical convergence guarantee and the experiments were not conducted in a network setting.

Comparison between Parameter Server and FL:

Federated Learning protocol is very similar to the traditional parameter server protocol. The

main differences are:

- In data center setting, shared storage is usually used, which means the worker machine do not keep persistent data storage on their own, and they fetch data from the shared storage at the beginning of each iteration.

- In FL, the data, and thus the loss function, on the different clients may be very heterogeneous, and far from being representative of the joint data.(e.g. the data stored on each client may be highly non-IID)

- In FL, the server never keeps track of any individual client information and only uses aggregates to ensure privacy.
  Because of the high churn in FL setting, only a small subset of the devices are selected by the server in each round.

Applications

In general, FL is most appropriate when:

- On-device data is more relevant than server-side proxy data
- On-device data is privacy sensitive or large
- Labels can be inferred naturally from user interaction

Advantages

- Highly efficient use of network bandwidth: Less information is required to be transmitted to the cloud.
- Privacy: As described above, the raw data of users need not be sent to the cloud. With guaranteed privacy, more users will be willing to take part in collaborative model training and so, better inference models are built.

Challenges and Limitations

- Does it work? And if so, why?
  - We can prove FL works for linear models and a couple of other special cases, but we cannot prove it works for more complicated things like neural networks unless we train the model in a non-federated way and demonstrate that it gets almost the same performance.
- Security
  - Recent study shows that a malicious participant may exist in FL and can infer the

information of other participants from shared parameters. As such, privacy and security issues in FL need to be considered.

- Statistical and System heterogeneity
  - In a large and complex mobile edge network, the heterogeneity of participating devices in terms of data quality, computation power, and willingness to participate have to be well managed from the resource allocation perspective.
- Slow, unstable and limited communication
  - Due to the high dimensionality of model updates and limited communication bandwidth of participating mobile devices, communication costs remain an issue.
  - One potential mitigation is that we can select N devices in each round and proceed with K response(K <= N).

Summary of Class Discussion

How to reduce the communication cost between the server and the clients?

One popular way of reducing the communication bandwidth of federated learning is compression. There are several different compression objectives of practical value.

- Gradient compression: reduce the size of the model updates communicated from clients to server
- Model broadcast compression: reduce the size of the model broadcast from server to clients

In general, gradient compression is more promising than model broadcast compression because edge devices' connections generally have slower upload than download bandwidth.

However, one caveat of compression is its compatibility with differential privacy and secure aggregation. Many algorithms used in FL such as Secure Aggregation and differential privacy are not designed to work with compressed or quantized communications.

Are global FL-trained models always better than local models?

In a setting that the size of local datasets are in the same ballpark and the data is IID, FL clearly has an edge. However, given that in real-world, the data is almost always Non-IID, local models might do much better than the global model. Thus, finding under what conditions the global model is better than the local modes is an interesting question.

I think one promising way is to begin with federated learning of a single global model and, before the model is used to render predictions, the model is personalized by additional training on the local held-out dataset.

Summary of "Scaling Video Analytics on Constrained Edge Nodes"

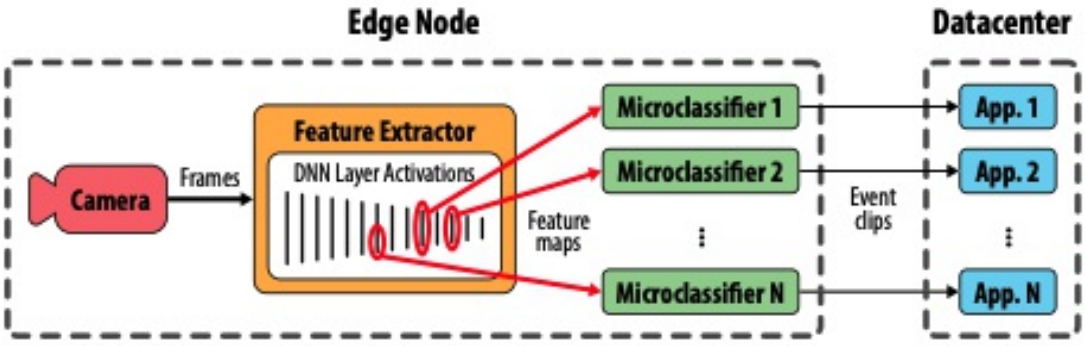
Problem and Motivation

The scenarios that motivate FilterForward include remote “Internet of Things” monitoring and “smart city” deployments of tens or hundreds of thousands of wide-angle, fixed-view cameras. However, there are a number of challenges:

- Limited Bandwidth: Each camera in large-scale deployments may only have few hundred kbp, while each stream coming in may be several orders of magnitude greater than our available uplink bandwidth. This bandwidth gap, exacerbated by the requirement for high-quality data, necessitates an edge-based decision about which frames to send to the datacenter
- Scalable Multi-tenancy: In real-world deployments, cameras observe scenes containing diverse objects and activities. Different applications are simultaneously interested in all of this information, and more. Given edge nodes’ limited compute resources, scaling to multiple applications naturally poses a performance challenge.

Solution Overview

To address the above challenges, FilterForward uses a feature extractor and microclassifier to provide highly accurate, multi-tenant video filtering for bandwidth-constrained edge nodes.



Feature Extractor

In FF, microclassifiers reuse computation by taking as input feature maps produced from the intermediate results (activations) of a single reference DNN, called base DNN. The component that evaluates the base DNN and produces feature maps is called the feature extractor. As

prior work observes, activations capture information that humans intuitively desire to extract from images, such as the presence and number of objects in a scene, and outperform handcrafted low-level features.

Although feature extraction is computationally intensive phase, its results are reused by all of the MCs, amortizing the per-frame, upfront overhead once the number of MCs passes a break-even point.

Microclassifier

Microclassifiers are lightweight binary classification neural networks that take as input feature maps extracted by the base DNN and output the probability that a frame is relevant to a particular application.
Choosing which base DNN layer to use as input to each microclassifier is critical to their accuracies. Too late a layer may not be able to observe small details (because they have been subsumed by global semantic classification). Too early a layer could be computationally expensive due to the large size of early layer activations and the amount of processing still required to transform low-level features into a classification. The authors discuss some microclassifier architectures in the paper.

Advantages

- Edge-to-cloud design: this hybrid design provides a way to satisfy the need to retrive high-fidelity data from bandwidth-limited edge nodes.

- Shared feature extractors + lightweight classifiers: this two-level design allows scalable multi-tenant ML task executions in compute-constrained settings.

Summary of Class Discussion

Does FilterForward require cameras to have computational power?

Not necessarily. The paper assumes the cameras are connected to some local compute nodes.

However, there are cameras empowered with more onboard compute resource, such as AWS deeplens, and can run complex deep learning models
locally.