1 System architecture

1.1 Overview

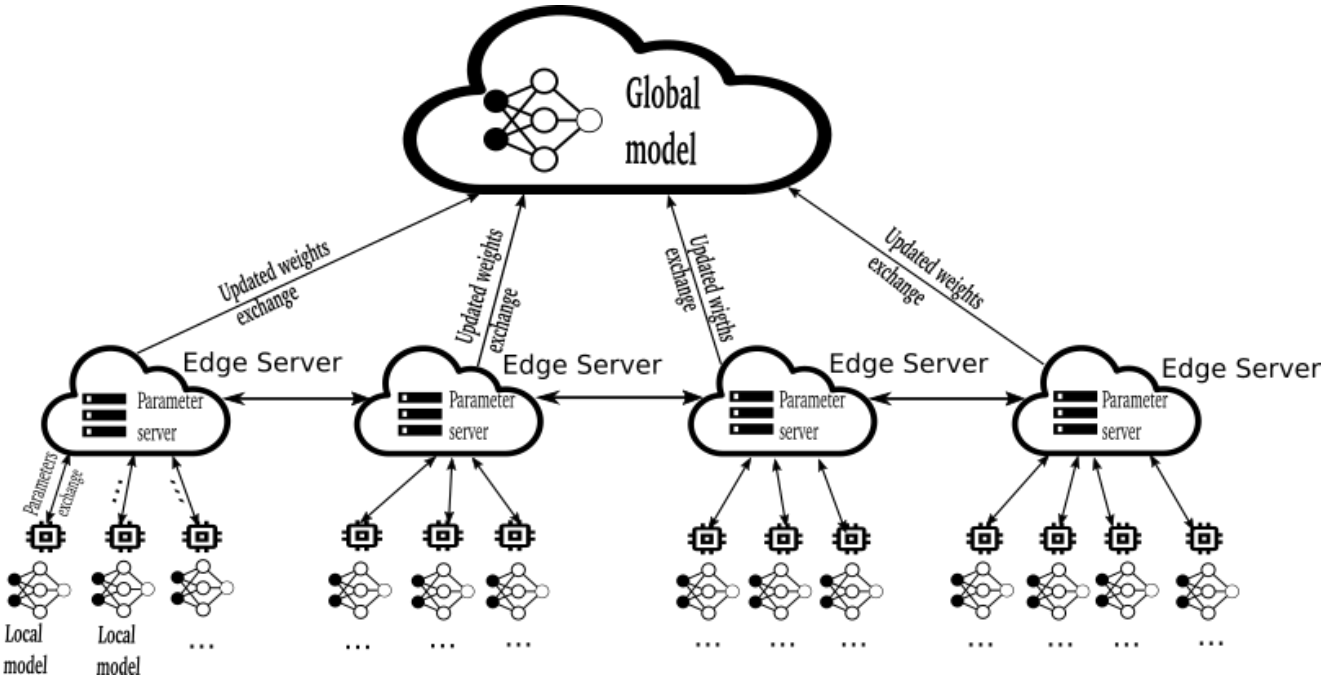

Figure 1.1: Overview of system architecture

The system consists of 3 layers:

- Device layer: this layer plays an important role in the proposed system. It is responsible for sensor data generation, pre-processing (data filtering and normalization) and the local model training / retraining. The IoT nodes in this layer are constituted by heterogeneous resource-constrained devices which are responsible to replay the data in time series for simulating the real world.
- Edge layer: the end devices are grouped (by their location in real case but here in simulation we just assume that they are associated logically to a particular edge node). At the local model training / retraining process, the gradients and then the weights of the model are recomputed at the edge device. These parameters will then be sent to their associate edge device. Each edge device is responsible for collecting updated parameters (weights) from local IoT devices within its region, averages them in order to update the weight of the "local partial model". The averaged weights are then updated to the parameter server for end devices to update their model. After n rounds of local - regional updates, at the round $n+1$, the selected edge devices will now forward their received weight updates to the cloud server for updating the global model. The weight update process is based on averaging like the local - regional update. The updated weights are then either forwarded to the cloud for the global aggregation, or sent back to the end devices.

- Cloud layer: this layer will mainly act as a central point for collecting updated weights from the edge nodes, perform global aggregation using weighted averaging then sent back to the edge nodes.

The basic idea of the whole system design comes from these articles: [?][?][?][?]

1.2 Devices layer

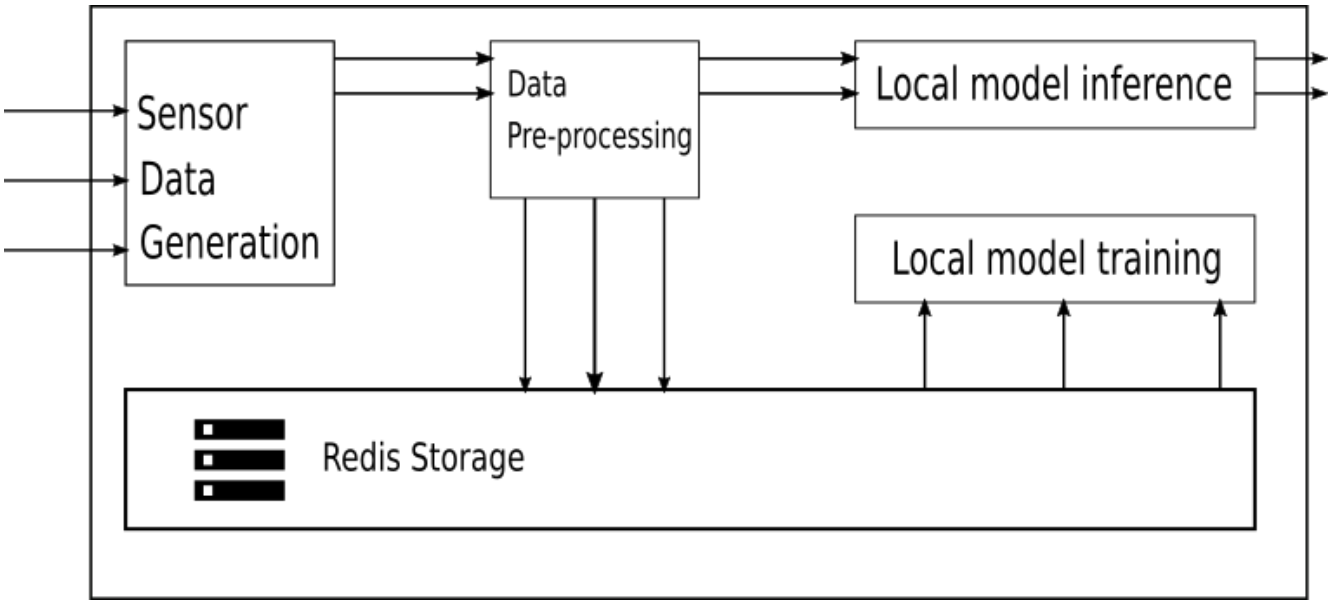

Figure 1.2: End device components

Technical detail for virtualizing device, replaying data and supporting federated training / re-training at device nodes:

- I used multiple docker containers for simulating the end devices in the system. The containers are grouped into smaller groups. Each node in a group will be fully connected via the docker bridge network. To represent the real world case, where devices are grouped into region, and each region would definitely have its own WLAN or PAN, each group of simulated devices will operate on a different local network (i.e. a bridge network with different IP). The chief node (edge server node) in each group connect with the cloud server via the backbone network (internet).
- For performing local model training and validating, I used TensorFlow
- For model weights exchange between client nodes and edge node, I used Python SSL socket. In addition, the weights being sent will be hashed together with a secret key. The purpose is to ensure the information confidentiality and integrity
- Using MQTT protocol for message exchanging between client node and server node. The current message types (topics) are:
  - Publish message to topic *client/join*: whenever it wants to join a group

- Subscribe to topic *training/start*: to receive the signal from edge server whenever it triggers the training phase
- Subscribe to topic *training/shuffle*: to receive the signal from edge server whenever it wants to receive the weights from that client node. To be more specific, at the beginning of each communication round (after *r1* local updates), the edge server will select randomly a number of client nodes to be aggregated by publishing a message including the list of *device id* of chosen clients to topic *training/shuffle*. Then when a client receive a message published to this topic, it checks if its device id in the list. If yes, the client will be triggered to send their local update to edge node. Otherwise, it only receives the averaged weights from the edge node.

1.3 Edge layer

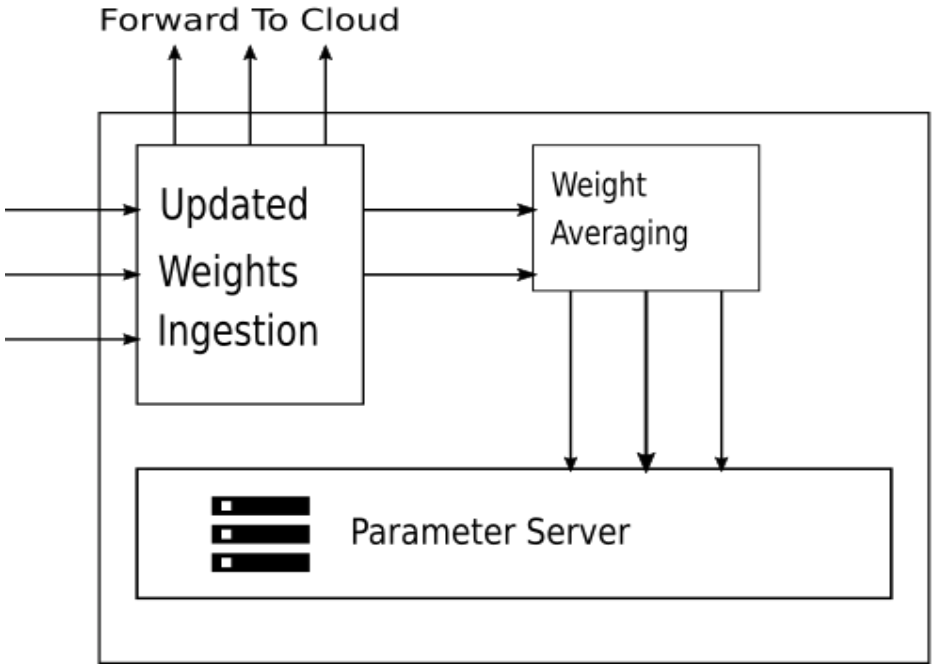

Figure 1.3: Typical edge node components

Technical details for supporting federated learning at edge nodes:

- Like client nodes, TensorFlow is used for local model training and validating
- The weights are exchanged via SSL socket, combining with secret key hashing
- Using MQTT protocol for message exchange. The current message types (topics) are:
  - Publish message to topic *training/start*: whenever it wants to trigger the training phase
  - Publish message to topic *training/shuffle*: performed at the beginning of each communication round to notify the selected clients that their weights are going to be aggregated

– Subscribe to topic *client/join*: to receive a message whenever a new client join the group. The message contains context information of the newly joined device, currently including: it's status (indicating that the device is capable of training or not), it's public socket IP.

1.4 Cloud layer

Technical options: The common architecture is the same as traditional cloud server. Amazon Web Service can be utilized to build the distributed, auto-scaling cloud server. A Python lambda function could be used for deploying and running the Python script for weights averaging.

Bibliography

[1] A. Akbar, A. Khan, F. Carrez, and K. Moessner. Predictive analytics for complex iot data streams. *IEEE Internet of Things Journal*, 4(5):1571–1582, 2017.

[2] M. A. Alsheikh, D. Niyato, S. Lin, H. Tan, and Z. Han. Mobile big data analytics using deep learning and apache spark. *IEEE Network*, 30(3):22–29, 2016.

[3] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll. Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. *Future Generation Computer Systems*, 87:198–212, 2018.

[4] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang. Incorporating intelligence in fog computing for big data analysis in smart cities. *IEEE Transactions on Industrial Informatics*, 13(5):2140–2150, 2017.