October 2021 Review

Emmanouil Petrakos

Federated Learning loop

Initially, server announces a global modal to all connected clients. Then, each client creates a new local model using the received global model and local data. These new models are sent to the server and used to create a new global model. This process continues till requirements, such as accuracy of the global model or a set number of epochs, have been met. Communication between server and clients is done with messages that contain the trainable variables of the models or their deltas in relationship with the global model, accompanied with metadata like epoch and loss.

Server Overview

Before anything else, the server must set up a socket to receive incoming connections. This socket is set in TCP non-blocking mode and used to create new sockets when connection requests are accepted. The next required task is to create a structure to monitor the sockets. This is achieved with the *poll(2)* system call and it's accompanying structure containing the file descriptors of the sockets currently used. The program sleeps till an event is raised. When this occurs, the *poll* call returns the number of elements whose event value indicate event or error. In order to find the appropriate file descriptors, all of them are checked.

If the event was raised by the file descriptor of the listening socket, a new client is trying to connect. The server accepts the connection and sends the current global model to allow the new client to start working immediately. If the event was raised by a non-listening socket, a client is trying to send data. Due to the size of the models, the messages can be larger than the windows of the sockets set by the operating system and multiple send events will occur. Therefore, a mechanism is required which collects all incoming data from clients and concatenates them to complete messages. This is achieved by storing the received data in buffers coordinated by the file descriptors and tracking their size for the current epoch. In case a message is not completed, the program returns to the *poll()* call and sleeps till a new event occur. When a message is completed, the model is accumulated to a temporary one. An event can also be raised by a socket when the other end shuts down. Then, there is no reason to keep the socket alive. The socket is closed and its file descriptor removed from the poll structure.

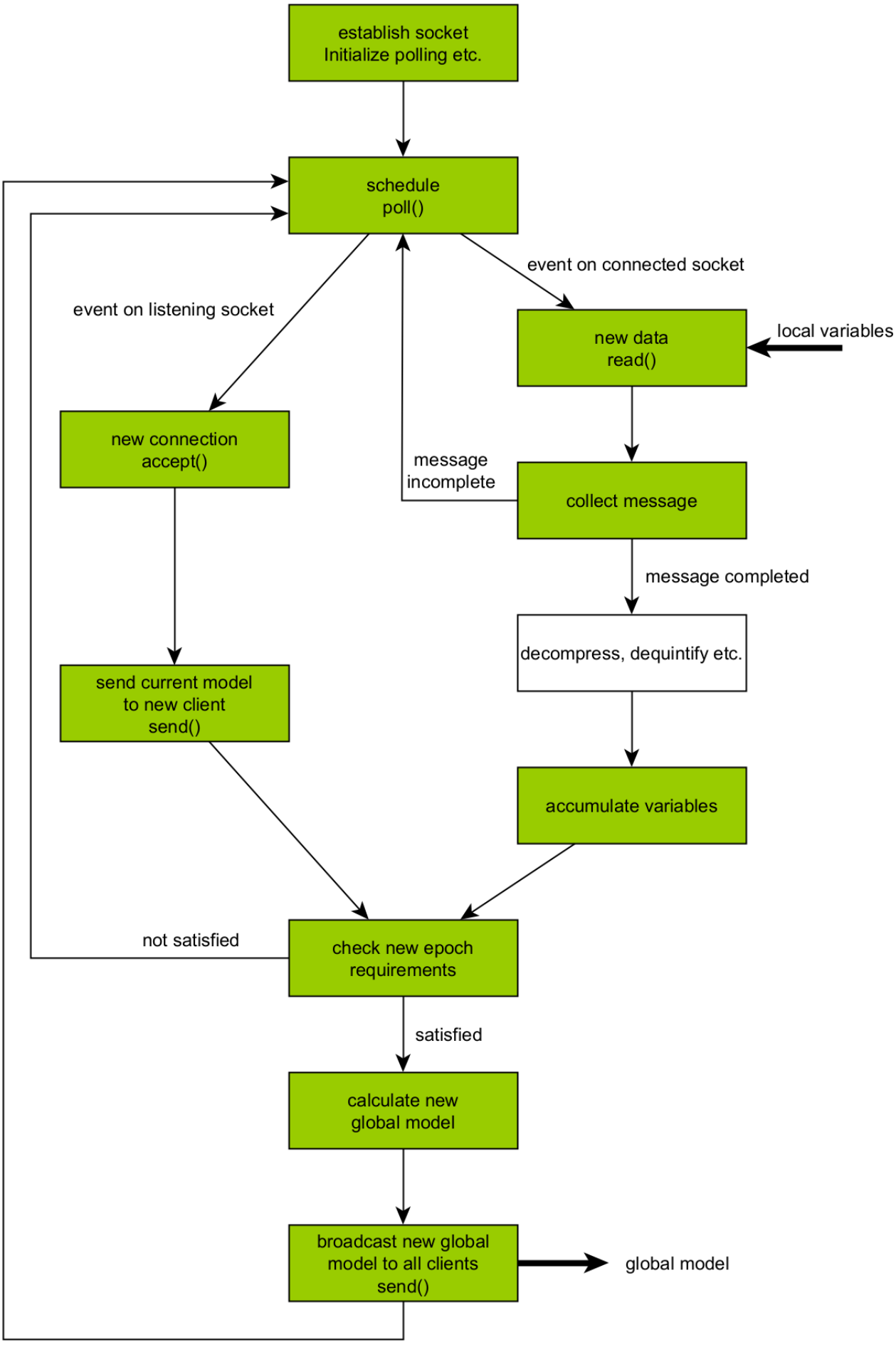
Following a new connection or a completed message, the requirements for a new epoch are checked. This requirements can be a certain amount of messages have been received, all connected clients have stopped working or a certain amount of time has passed. Provided the requirements are satisfied, a new global model is produced and broadcasted to all connected clients. The new global model is created by dividing each variable of the cumulative temporary model by the number of the of the local models used to create it.

Client Overview

At start up, each client need to perform two tasks. First, create a socket and establish a connection with the server. Second, prepare the environment were the training of the model will take place. As the training has been developed with TensorFlow in Python and the client program in C++, the Python interpreter must be embedded. Furthermore, the model is compiled and the input data preprocessed. Lastly, the data that is sent and received are stored in C-like arrays to minimize network overhead. In order for the arrays to be modifiable in the Python environment without any data copying, NumPy array metadata is created around them.

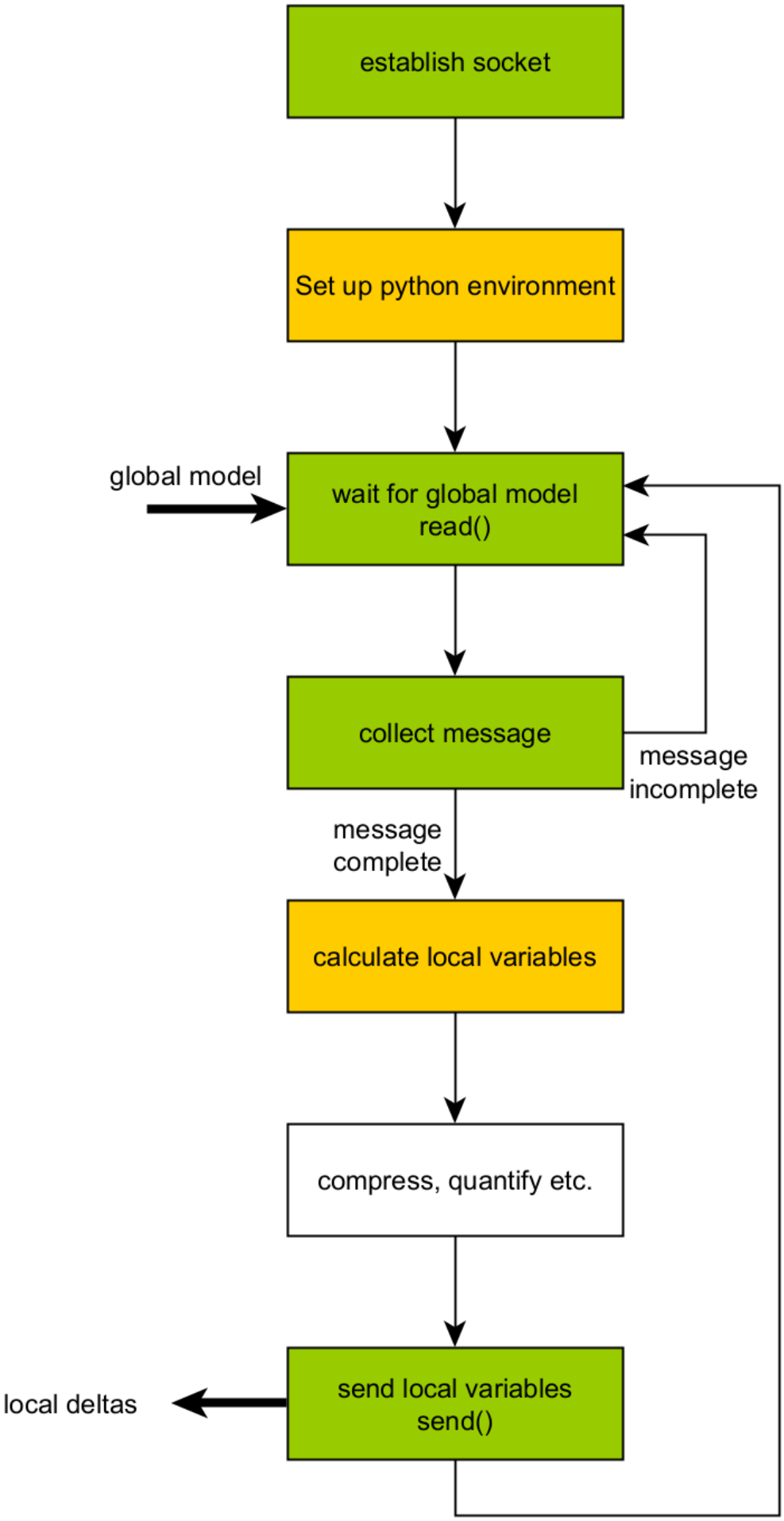
After the necessary initializations are completed, the program enters the main loop. The client waits for the global model and collects incoming data like the server. When the global model is received in its entirety, it passes in the Python environment and assigned on the trainable variables of the local model. After a few epochs of local training the produced trainable weights are returned to the C++ environment and sent to the server. This operation continues till the connection with the server shuts down or the server gives such an order.

Server Block Diagram



Green: Implemented, White: Unimplemented

Client Block Diagram



Green: Implemented, White: Unimplemented, Orange: Implemented in TensorFlow