FC Prototyping Assignment - Documentation

Kapp, Thomas Delzer, Ole

July 2022

1 Client-Side (Edge Device)

The client on the edge node has a dedicated thread to continuously gather weather data from a sensor and store it at its local database (listing 1). At the same time, the client fetches the data from its local database and sends it to the server. The initial request to the server is a JSON-Object {'send_timestamp': 1} (listing 2 line 2). The server always responds with the timestamp of the newest record that it knows. The client then deletes all records older than the newest known to the server (listing 2 line 13). Afterwards, it fetches the data newer than the received timestamp and (listing 2 line 21) and forwards them to the server (listing 2 line 9). The server again responds with the timestamp of the newest record it knows, which now should belong to one of the records it just received from the client. This ensures that the client only deletes records once the server confirmed it received them, which prevents loss of data in case the server crashes or the connection fails.

If the client cannot reach the server, the sending (listing 2 line 9) will block until the connection is re-established. Should the sending succeed and the server becomes unavailable before the "recv"-method (listing 2 line 10, listing 3 line 1) completes, it will time out and the client will try to reconnect to the server (listing 3 line 9). Afterwards, the initial request is sent to the server again (listing 2 16). This loop is repeated until the "recv"-method does not time out anymore and the normal data transfer can proceed.

2 Server-Side (Cloud Node)

The server waits for incoming messages (weather data) from the client (listing 4 line 3). If the incoming request is empty (such as the "initial request" from the client) (listing 4 line 8) or the received record is older than the records known to the server (listing 4 line 14), the server only respond with its latest known timestamp. Otherwise, it stores the new records in its database (listing 4 line 18) and responds with the timestamp of the new record.

If the client crashes or the connection fails before the server could respond, the response is discarded. This is not a problem, because the client will request

the latest timestamp after the connection is re-established (i.e. send the the "initial request" again).

See the README for a more detailed documentation.[1]

Listing 1: client.py: gathering and storing data

```
def gather_data(db: cldb):
    gd = Gather_data()
    while 1:
        time.sleep(0.1)
        db.insert(gd.gather())
```

Listing 2: client.py: sending data to the server

```
response = None
initial_request = {'send_timestamp': 1}
request = initial_request

while 1:

    # send data and receive timestamp
    while 1:
        client.send(request)
        response = client.recv()
        if response:
            print("erase data since {}...".format(
                response['time']))
            db.erase_data_starting_from(response['time'])
            break
        else:
            request = initial_request

    # aquire data from database
    while 1:
        print('get data from db...')
        data = db.get_data_latter_than(response['time'])
        if data:
            print("got {} from db...".format(data[0]))
            request = data[0]
            break
        print('No new data in database. Waiting...')
        time.sleep(0.2)
```

Listing 3: client.py: reconnect after failure

```
def recv(self):
```

[1] https://github.com/OleDe/fogcomputing-prototyping_assignment/blob/main/README.md

```
2       item = self.poller.poll(self.timeout)
3       response = None
4       if item:
5           response = self.socket.recv_pyobj()
6           print('received response: {}'.format(response))
7       else:
8           print("Server is not reachable. Reconnecting...")
9           self.reconnect()
10      return response
```

Listing 4: server.py: receive and store data

```
1   while True:
2       #  Wait for message from client
3       message = socket.recv_pyobj()
4       print("Received request: {}".format(str(message)))
5
6       # initial message
7       stamp = db.get_latest_timestamp()
8       if 'send_timestamp' in message:
9           print('Sending latest timestamp: {}...'.format(
                stamp))
10          response = {'time': stamp}
11
12      # process consecutive messages
13      else:
14          if message['time'] <= stamp:
15              print('deprecated data {} <= {}'.format(
                    message['time'], stamp))
16              response = {'time': stamp}
17          else:
18              db.insert({'air_pressure': message['
                    air_pressure'], 'air_temperature': message
                    ['air_temperature'], 'time': message['time
                    ']})
19              response = {'time': message['time']}
20
21      # send request
22      socket.send_pyobj(response)
```