Assignment 1 Client/Server Data Model and Transmission

Kasra Aminiyeganeh
Student ID: 40199177.
Fall 2021 COEN 6313 Programming on the Cloud, Professor Yan Liu

***Abstract*—In this report, I review the process I used to achieve data transmission between the client and the server. This report has six parts: I. Data model design, II. Data serialization/de-serialization method, III. Technical implementation of response, request, and (de)serialization, IV. Cloud deployment, V. Instructions on how to run both the client and server applications, VI. Screenshots of running the application, VII. Conclusion.**

***Index Terms*—Data Model, Client Server, Data Transmission, Data Serialization .**

I. Data Model Design

For this assignment, I used JSON data model design. For each row of data in the CSV file, I assigned an ID and then bound the ID to another JSON object which contains the actual data metrics. I then saved the extracted model in a JSON file. As you can see in (Fig. 1) for instance, the row 2000 to 2002 of the CSV file has been modeled like this.

Fig. 1. The Data Model file with ID binded to the actual metrics

Furthermore, this model design was only to capture the data from the CSV file; In (Fig. 2) I have used another model to optimize the transmission between the client and the server(save bandwidth). So when the client asks for a specific metric, that metric would be bound to the same ID used and sent in a JSON format. This way the client can get other metrics from the server and join them with the ID that they have.

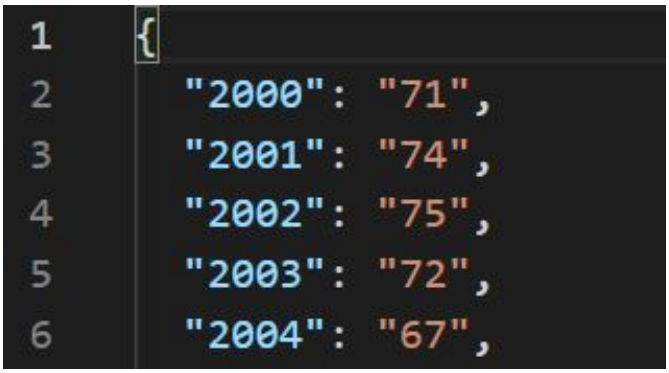

Fig. 2. The Data Model file with ID binded to CPU metrics

II. Data serialization/de-serialization method

For this part, I used JSON and binary serialization to achieve the desired task. I used JSON formatting to send and receive data and communicate between my client and server. Also, to transmit in the network layer, I had to serialize it with a default binary decoding and encoding (utf-8) mechanism to get byte objects. In Python I have done this via default function calls from the built-in libraries.

III. Technical implementation of response, request, and (de)serialization

I used Python 3.7 and some of its built-in libraries for the implementation, which I will go through. I used the socket library to achieve the task of communication between my client and the server. Socket library uses the TCP protocol for data transmission in the network and make a bi-directional connection between client and server which through it, you can stream data. I have also used the json library for Python; this library will serialize and deserialize the Python objects into JSON objects and vice versa. On top of that, I used the CSV Python library to read from the data file provided by the professor a consistent database. At last, I have also used the threading library to make sure I can have multiple clients connected to the server. The advantages of using these libraries are that you do not need to install and learn any new frameworks; second, they are easy and convenient to use; third, they are proven to work fast and without issues since they are built-in solutions. The disadvantage of using TCP/IP protocol was that I had to make sure the data was transmitted fully either by an escape character or by fixing the length. I achieved it by a combination of fixing the size of RFWID and Last Batch ID and considering a timeout for the client receiving function.

The flow of the program is pretty smooth. In the server application, we have three functions; the first is send(), which is responsible for sending messages over the open connection with the client; the second one is handleclient(), which is responsible for handling new client requests and connections; the last one is makebatch(), which is accountable for making batches based on the client request parameters. After computing the batches and serializing them, the server would send back the data with the predetermined format, and then the client would receive them batch by batch.

IV. Cloud Deployment

I used Google cloud services to deploy my python app. In (Fig. 3)

I created a virtual machine instance on the google compute service and then tried to connect to it via ssh for the file transmission and then transferred my python files and data files with FileZila which is a FTP app. But to access my VM I had to generate public/private keys from PUTTY. After I deployed my files on the VM I accessed the VM via SSH windows provided by Google Cloud two times to first run the server.py and then run the client.py and you can see the results that are running on the cloud. (Fig. 4)

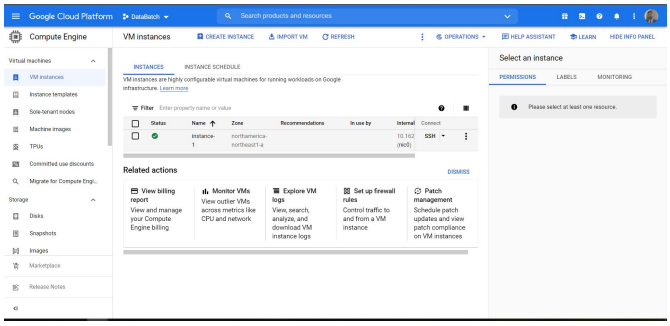

Fig. 3. My VM instance on the Google Cloud

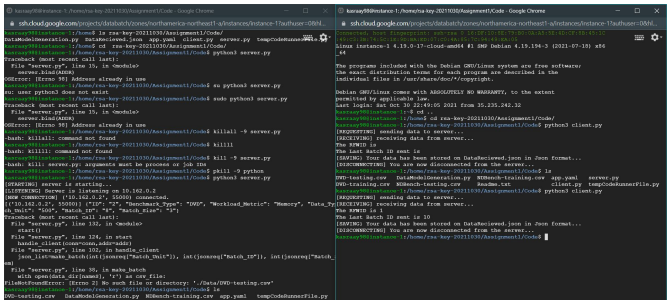

Fig. 4. The Cloud Deployment and Running

V. Instructions on how to run both the client and server applications

To run the program, you need first to run server.py in a command prompt with the python server.py command, and wait until the server is up and running, so you will see the message that the server is listening on some IP address like (Fig. 5). Then you need to run the client.py on another command prompt opened up with the python client.py command. After running the client.py, a JSON file with the name of DataReceived.JSON can be found in the directory afterward. Note that the relative addresses of the primary CSV files are important since everything is based on them. So please be careful not to change any name or location and only unzip and run.

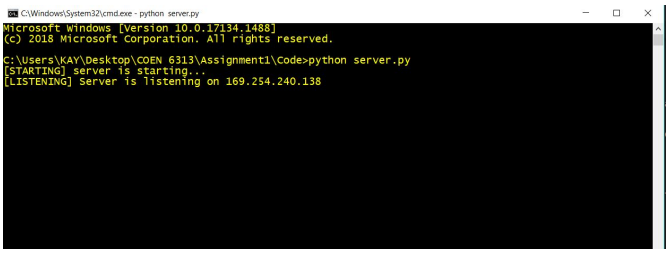

Fig. 5. The Server running

In client.py, you can find the request dictionary (Fig. 6) ; if you want to change any argument to send and get a different result, you can change it here. BatchID, BatchSize, etc., can be modified here. Note that for the kind of metrics to get, you should use one of the following exactly: “CPU,” “NetIn,” “NetOut,” “Memory” as the Workload Metric.

Fig. 6. The Request dictionary to change in client.py

VI. Screenshots of running the application

After running the server we can run the client multiple times with different arguments in the request dictionary to get results. For instance, here (Fig. 7) the client ran 2 times with different but related requests; each time the server responded with the RFWID and Last Batch ID first and then sent the requested data which is stored on a JSON file. In this example, the batch unit is 500 and we will change it to 1000 in the next one; also the batch size and ID is different in each example.

Here in another example, you can see that based on the client’s request, which has the batch ID 2 and Batch size of 2 and Batch unit of 1000, we should return CPU util data from row 2000 (Fig. 9) to 3999 (Fig. 11) in 2 batches (Fig. 10). This is precisely what happened and stored in a file

DataRecived.Json. You can find all of the files and the results in the source folder.

And you can see the server reciving the client request on which port and IP. (Fig. 8)

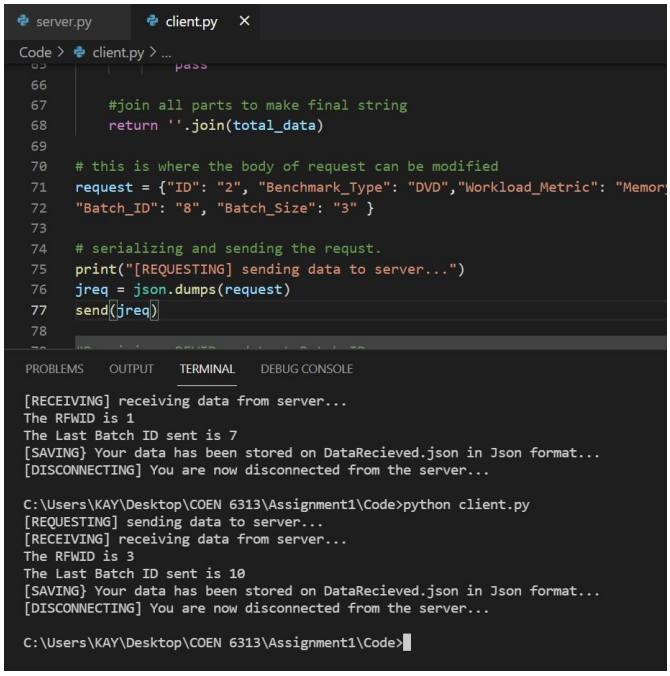

Fig. 7. The Client Receiving Data

Fig. 8. Server's getting requests from client the message details

Fig. 9. The start of Data records is 2000

VII. Conclusion

In conclusion, data modeling is necessary for transmission purposes, and I think this method needs a massive revise in terms of huge datasets. Also, choosing a right protocol for the transmission is vital.

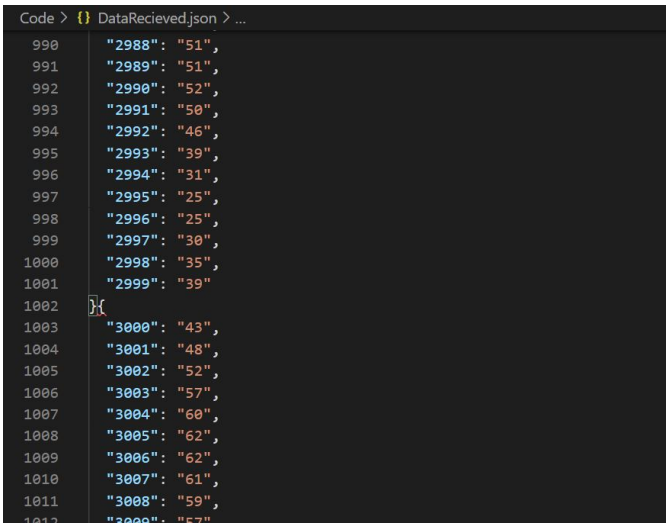

Fig. 10. The batch conjunction which means it is seant in 2 batches

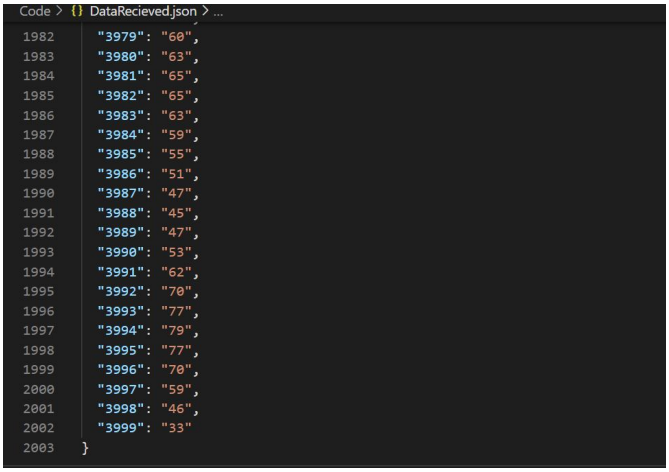

Fig. 11. The end of last batch ID is 3999