Gowtham Chandrasekaran

Report for Server Monitoring System

1. generate.py

Note: IPs are generated randomly and the timestamps are generated by adding 24 hours to the current time of generation.

Logic:

The program reads the argument from the command line and creates a folder with that name on the current working directory.

1000 randomly generated server IP addresses are stored in a list.

The start time is set to the time the program is run and the end time is 24 hours added to the start time.

A file is opened and is written with the server information for each CPU_ID (0 and 1) and IP address.

Output:

Run command: python generate.py test

In my case, test is the directory name that I wanted to create.

```
(base) gowthamc@Gowthams-MacBook-Air generate_logs % python generate.py test
Log Start Time: 2022-03-30 07:07:59 -> Log End Time: 2022-03-31 07:07:59
(base) gowthamc@Gowthams-MacBook-Air generate_logs %
```

The directory was created successfully.

```
(base) gowthamc@Gowthams-MacBook-Air generate_logs % ls
generate.py     query.py        test
(base) gowthamc@Gowthams-MacBook-Air generate_logs %
```

The directory had the server_logs.txt file which contained all the server info.

```
(base) gowthamc@Gowthams-MacBook-Air test % ls
server_logs.txt
(base) gowthamc@Gowthams-MacBook-Air test %
```

```
1648710479      6.115.177.29    0       85
1648710479      6.115.177.29    1       35
1648710479      3.36.109.142    0       98
1648710479      3.36.109.142    1       9
1648710479      61.189.154.169  0       20
1648710479      61.189.154.169  1       52
1648710479      151.96.244.200  0       2
1648710479      151.96.244.200  1       45
1648710479      72.92.96.91     0       66
1648710479      72.92.96.91     1       90
1648710479      122.230.43.172  0       20
1648710479      122.230.43.172  1       38
1648710479      202.35.78.217   0       32
1648710479      202.35.78.217   1       41
1648710479      12.29.57.118    0       82
1648710479      12.29.57.118    1       50
1648710479      20.100.84.236   0       10
1648710479      20.100.84.236   1       14
1648710479      62.84.216.140   0       10
1648710479      62.84.216.140   1       36
1648710479      250.56.57.48    0       56
1648710479      250.56.57.48    1       78
(base) gowthamc@Gowthams-MacBook-Air test %
```

2. query.py

Logic:

Note: All timestamps are converted to UTC

A dictionary called ip2usage is created to store the key as a tuple of IP address and CPU_ID along with a value as a list of tuples of timestamp in Unix time and CPU_Utilization. This is done for each file present under the specified directory path. This is the initialization phase.

The query is parsed and the parameters are taken from it to set values of start time, end time, CPU_ID and IP address.

The key which is a tuple of IP address and CPU_IP is searched in the populated dictionary to get the list of values under it.

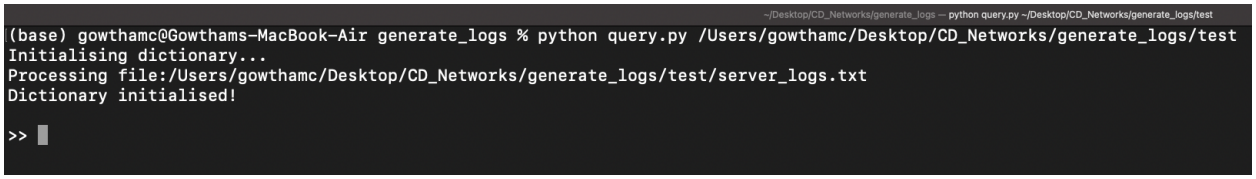
A **binary search** is performed to find the start and end points from this list so that only the values present in between these two points are considered. **i.e.** we are just considering the timestamps that fall between the provided start and end timestamps in the query.

Now print all the values between the start and end timestamps.

Output and Testing Scenarios:

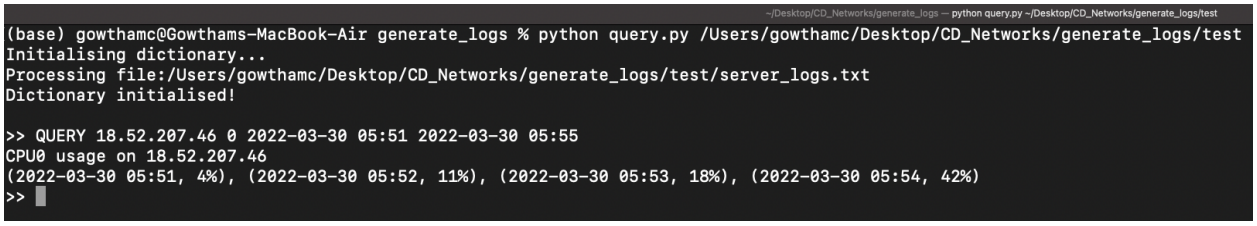
Run the command: python query.py DATA_PATH

In my case DATA_PATH = “/Users/gowthamc/Desktop/CD_Networks/generate_logs/test”



The program iterates through each file in the directory and creates the respective hash map.
Test 1:

Running the query: QUERY 18.52.207.46 0 2022-03-30 05:51 2022-03-30 05:55

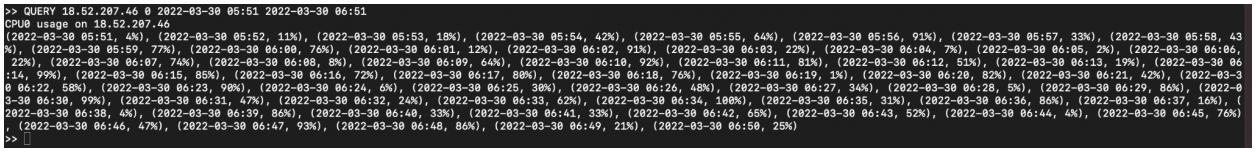


In my test case, the IP address is **18.52.207.46** and CPU_ID is **0**. The start time that I have given is the start timestamp at which the log file was created. The end time is just 4 mins added to the start time. Hence, we are getting 4 pairs printed.

Note: The parameters of the query will be different at the time of the creation of log files.

Test 2:

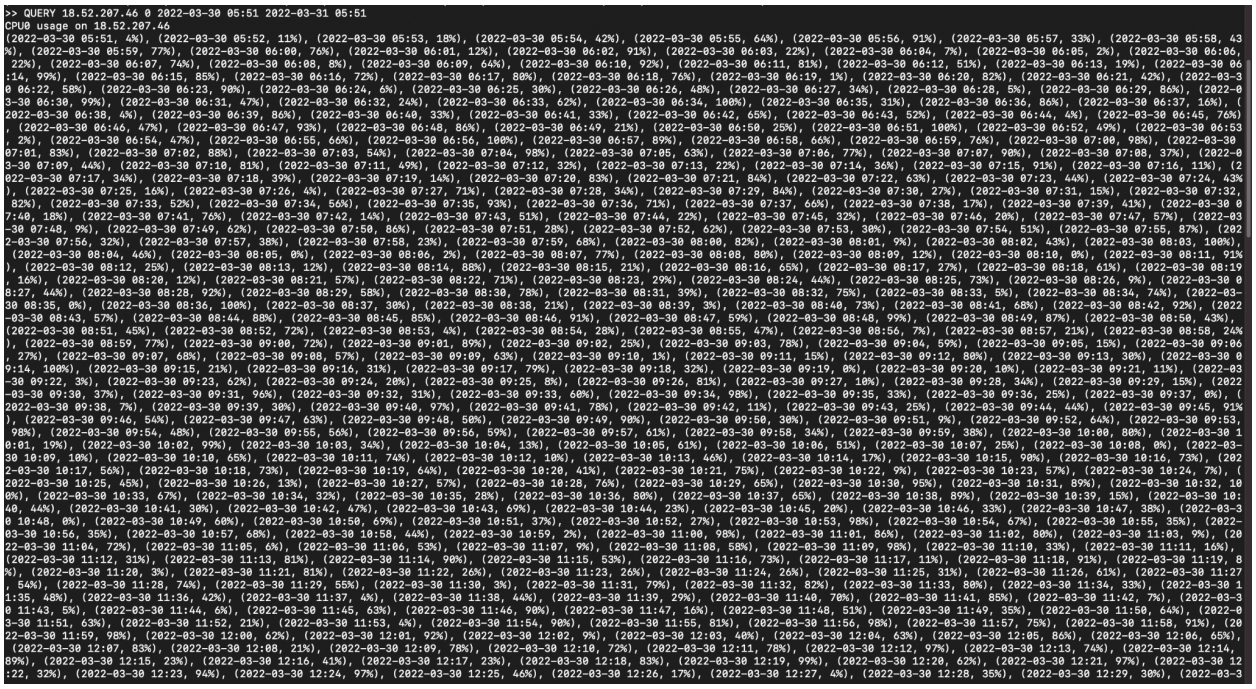
Running the query: QUERY 18.52.207.46 0 2022-03-30 05:51 2022-03-30 06:51



In this test case, I ran it to get CPU usage info for one hour. The output was as expected. I got the logs for each minute between the given start and end times.

Test 3:

Running the query: QUERY 18.52.207.46 0 2022-03-30 05:51 2022-03-31 05:51

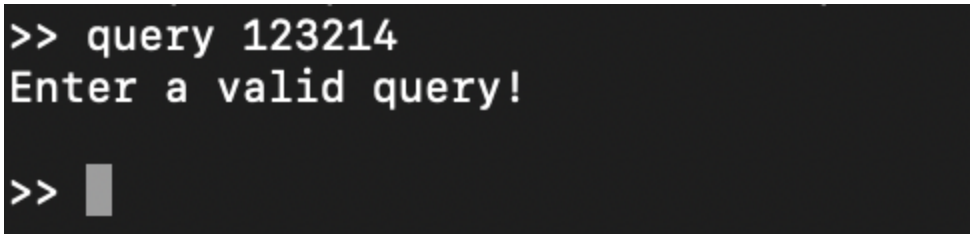


In this test case, I ran it to get CPU usage info for one day. The output was as expected. I got the logs for each minute between the given start and end times.

I noticed that in every test case the response time was all under 1 second. The **binary search logic** in the code boosts the performance of searching and displaying.

Test 4:

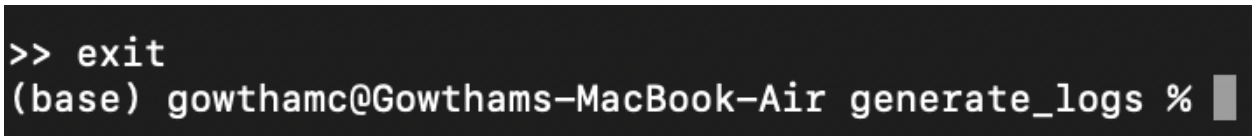
Running invalid query



Printed a message “Enter a valid query” when an invalid query was given.

Test 5:

Running the exit command



The program ended as expected.

Time Spent:

I spent a total of 6 hours. I faced difficulties in understanding the timezone conversions as I was not familiar with them. (Example: Local time to UTC)