Assignment 3

Repository URL: https://github.com/Survi15/Fall2018/tree/master/Assignment3
Repository Contents: FitbitServer-1.0-SNAPSHOT.war file, AWS_Lambda_Server folder contains the three lambda function handler classes as .jar files, AWS_Lambda_Client folder contain the Java Client Class used for testing the Lambda Functions, GCP_Client folder contains the Java Client class used for testing the server deployed in Google Compute Engine and Load Balancer. Client.zip contains the archive format of the Project folder with all the client classes(AWS & GCP).

Application Idea :

Server Application - Created as a Maven Project in NetBeans IDE with the following classes

1. DataSource.java - uses a static C3P0 ComboPoolDataSource and has the Database instance login credentials
2. FitbitResource.java - Java class which has the code to handle the endpoints for the assignment
3. UserDAO.java - Java class to implement the logic for each endpoint
4. RESTConfig.java - This has the application path.

I used Glassfish 5.0 for deployment purpose for this assignment.

For the RDS instance I made use of t2.medium which allows 312 threads of connection at a time and free tier AWS Lambda functions. For testing the Client class I made use of a t2.medium EC2 instance.

For GCP I used a compute engine of 1 vCPU and 3.75GB memory and an OS of Ubuntu 18.04 LTS instead of a vanilla Linux box and Cloud SQL storage for persistence.

The Client Application used in the assignment has the same design as previous assignment.

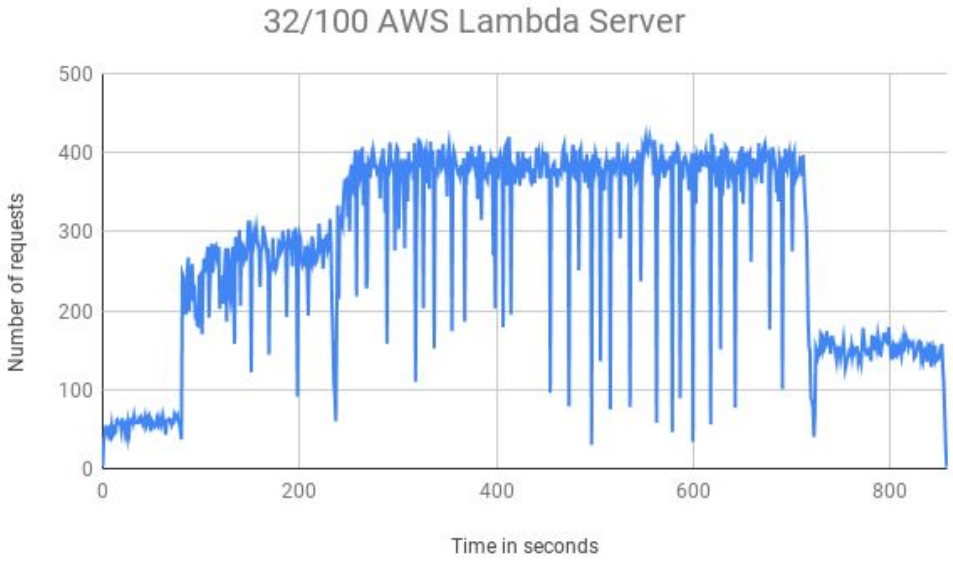
Overview of Client Java class :

Client Application Created as a Maven Project in Eclipse IDE with the following class.
FitbitClient.java - A Java class that implements the logic for a Jersey Client application which will access the web server endpoints. I have made use of Blocking Queue Data Structure to keep the count of number of responses, requests and successful requests. I have used a HashMap<Long,ArrayList<Long>> for tracking all the request latencies as well. Whenever a response for successful request is received by the client , the blocking queue adds an entry to the map with the request timestamp and request latency(both long values). Since this application sends concurrent requests, it's highly likely that any two requests have the same timestamp. Hence the value of the hashmap is a list of all latencies for that timestamp. The

postprocessing of the map is to convert the map to a TreeMap(sorted on key-time in milliseconds) and then making a new TreeMap with buckets of requests latencies map to each second(function MapToSecondsBucket()). I finally write both the Original treemap and the Seconds treemap to output files. This way I was able to retrieve the data for plotting and p99 and p95 as well.

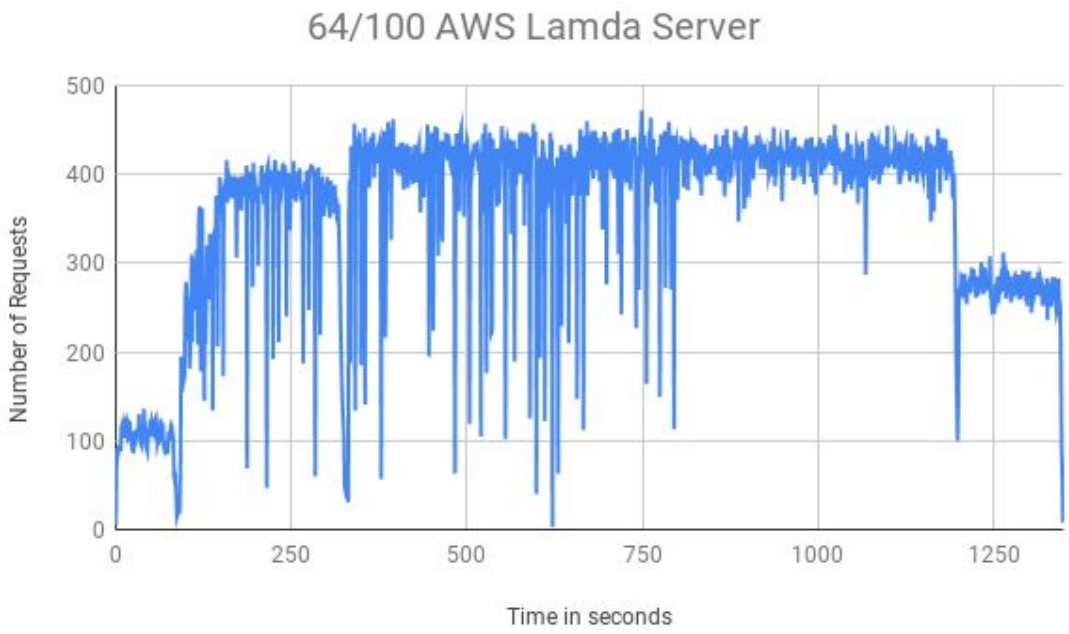
Plots and Statistics:

Default Client Settings: 32 threads in peak phase and 100 iterations:



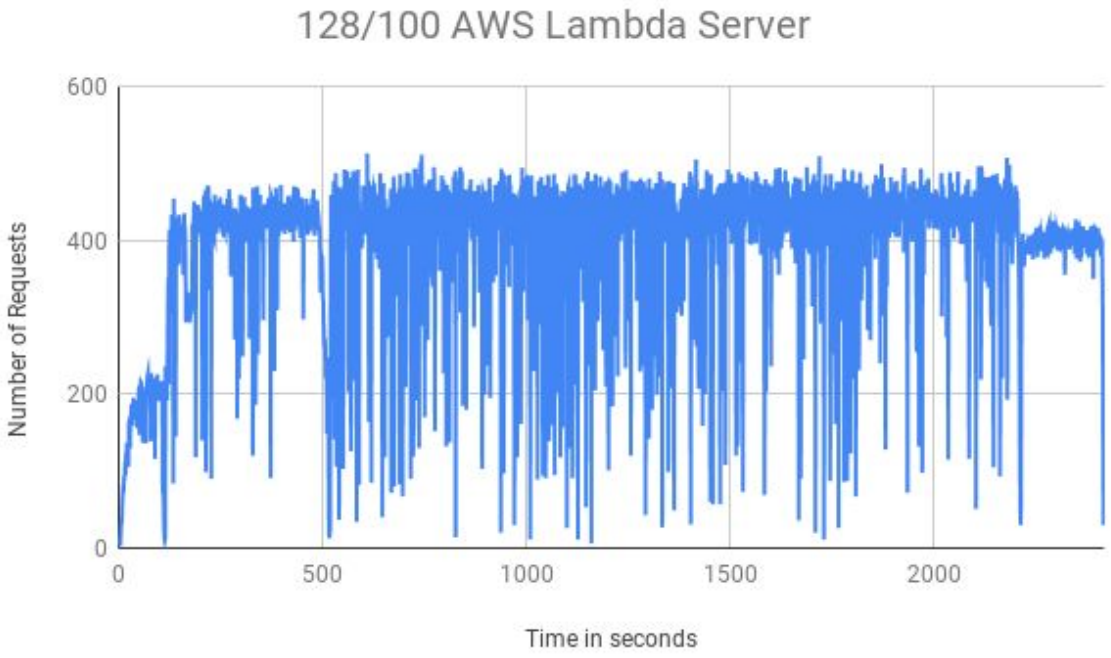

```
Client: Thread Count: 32, Iteration Count: 100
Client starting time: 1542251529755 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 82.282 seconds
Load phase:All threads running...
Load phase complete time: 161.609 seconds
Peak phase:All threads running...
Peak phase complete time: 510.207 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 134.449 seconds
Client end time: 1542252418306 milliseconds
========================================
Total requests sent: 240500
Total response received: 240500
Total wall time: 888.551 seconds
Total number of successful requests: 240500
Overall throughput across all phases: 270.6653789457632 seconds
latency of 95th Percentile: 0.099 seconds
latency of 99th Percentile: 0.071 seconds
```

Default Client Settings: 64 threads in peak phase and 100 iterations:




```
Client: Thread Count: 64, Iteration Count: 100
Client starting time: 1542252997639 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 94.192 seconds
Load phase:All threads running...
Load phase complete time: 247.440 seconds
Peak phase:All threads running...
Peak phase complete time: 896.206 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 149.504 seconds
Client end time: 1542254384990 milliseconds
========================================
Total requests sent: 481000
Total response received: 481000
Total wall time: 1387.351 seconds
Total number of successful requests: 481000
Overall throughput across all phases: 346.70391025077566 seconds
latency of 95th Percentile: 0.170 seconds
latency of 99th Percentile: 0.173 seconds
```

Default Client Settings: 128 threads in peak phase and 100 iterations:




```
[ec2-user@ip-172-31-40-255 ~]$ java -jar ClientForLambda.jar 128 1 1000000 100
Client: Thread Count: 128, Iteration Count: 100
Client starting time: 1542313053609 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 146.006 seconds
Load phase:All threads running...
Load phase complete time: 415.888 seconds
Peak phase:All threads running...
Peak phase complete time: 1806.881 seconds
Cooldown phase:All threads running...
^[[A^[[ACooldown phase complete time: 203.104 seconds
Client end time: 1542315625515 milliseconds
========================================
Total requests sent: 962000
Total response received: 962000
Total wall time: 2571.906 seconds
Total number of successful requests: 962000
Overall throughput across all phases: 374.0416631900037 seconds
latency of 95th Percentile: 0.347 seconds
latency of 99th Percentile: 0.338 seconds
```

Comparison of AWS Lambda performance with EC2 :

On comparing the performance of the above tests with Server deployed on EC2 and not load balanced I found the latencies and total wall time in case of Server implemented on Lamba is

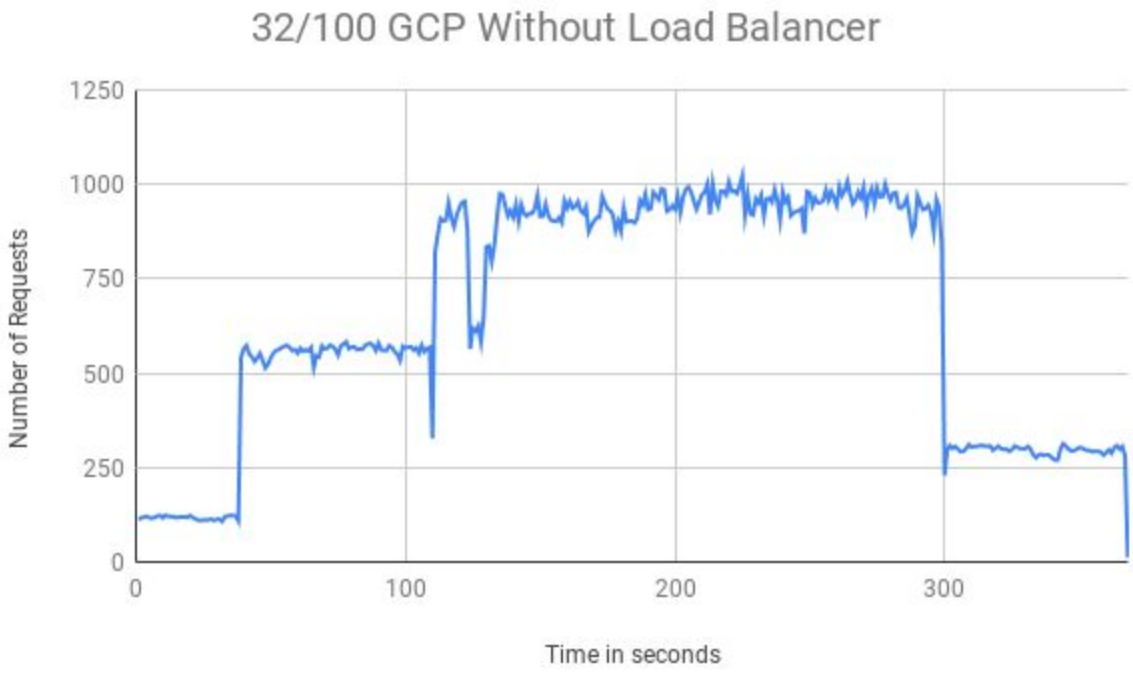
much more than that deployed on EC2. But the throughput across all phases is much less for lambda server i.e. more number of requests are being processed per second in the case of Lambda server than the later. Also the throughput does not vary marginally when load increases from 64/100 to 128/100. But in the case of EC2 the difference is clearly noticeable.
For Load Balanced EC2 instance the throughput is lesser than original EC2 instance but still more than AWS Lambda. So overall AWS lambda has a better performance in case of Throughput. This could be because of better load balancing strategies internally taken care of by AWS Lambda than the CPU utilization rule I used for EC2. The network in which I performed these tests also play a role to such observations.

Throughput Comparison	32/100 config	64/100 config	128/100 config
AWS Lambda	270	346	374
EC2	560	811	886
EC2 (Load Balancer)	597	593	866

If we talk about wall time, AWS Lambda has more wall time in comparison to both EC2 without Load Balanced and with Load Balanced. One of the reasons for such behaviour could be the network connection of the testing. And the other could be AWS Lambda when used with lower number of threads is leading to underutilization of the compute power available for such operations. The latencies of AWS Lambda Server is also much more. This could also be for the same reasons as explained above.

Step 3: With GCP Compute Engine and Cloud SQL

Default Client Settings: 32 threads in peak phase and 100 iterations:

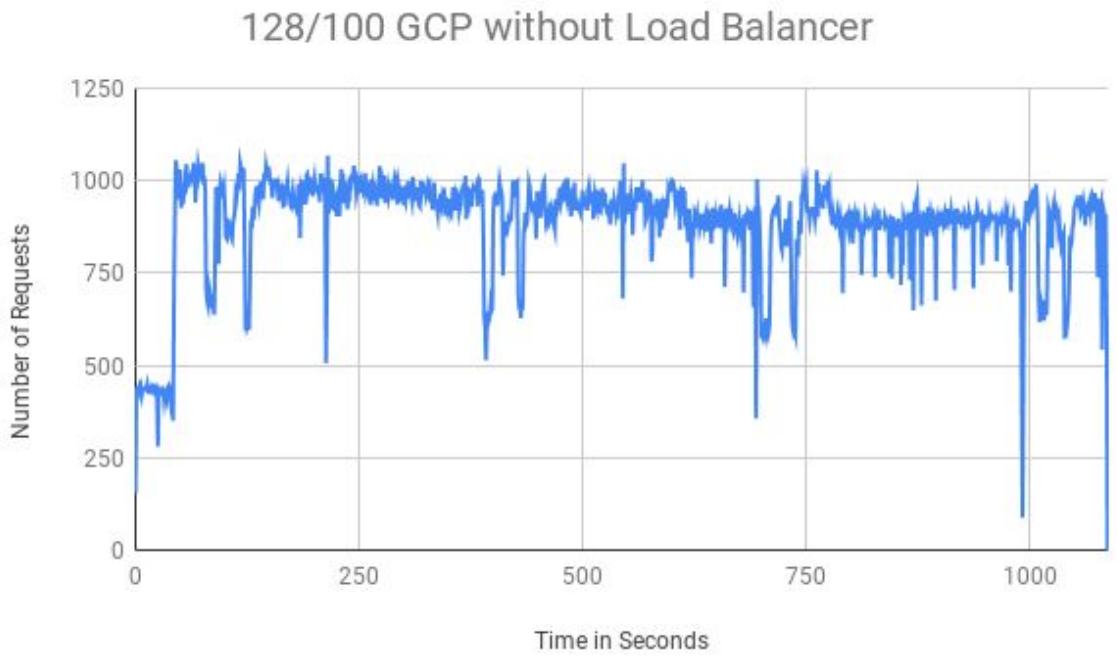



```
Client: Thread Count: 32, Iteration Count: 100
Client starting time: 1542743232452 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 39.256 seconds
Load phase:All threads running...
Load phase complete time: 72.025 seconds
Peak phase:All threads running...
Peak phase complete time: 190.155 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 67.789 seconds
Client end time: 1542743601686 milliseconds
========================================
Total requests sent: 240500
Total response received: 240500
Total wall time: 369.234 seconds
Total number of successful requests: 240498
Overall throughput across all phases: 651.3484518632026 seconds
latency of 95th Percentile: 0.041 seconds
latency of 99th Percentile: 0.031 seconds
```

Default Client Settings: 64 threads in peak phase and 100 iterations:

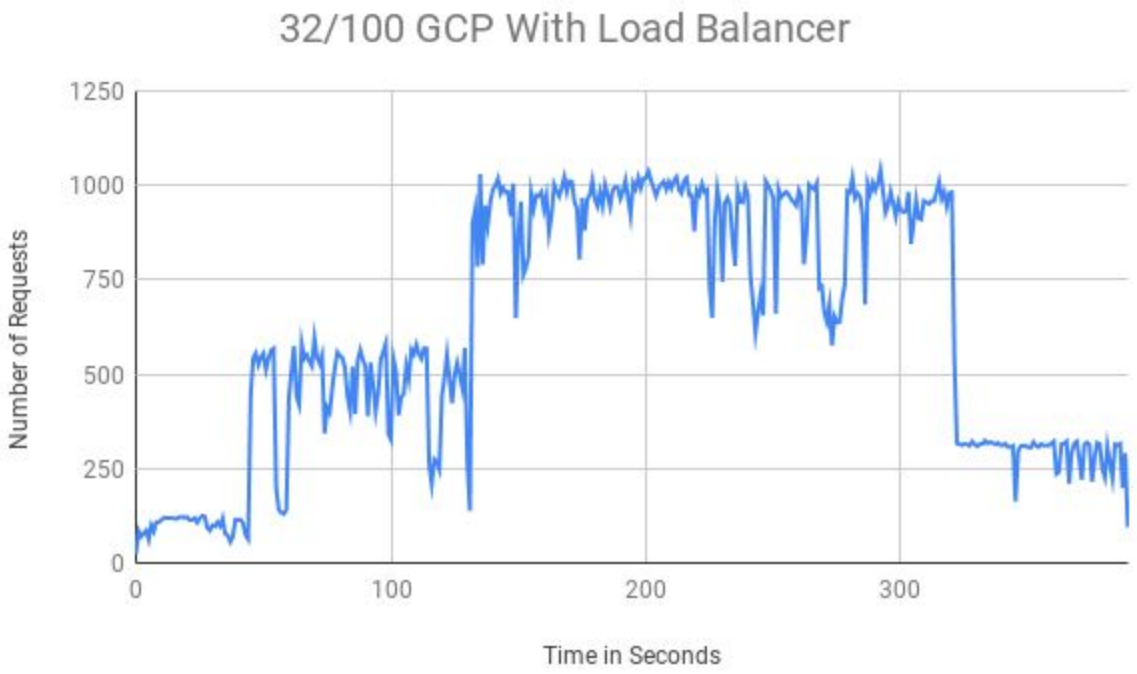

```
Client: Thread Count: 64, Iteration Count: 100
Client starting time: 1542745792128 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 40.490 seconds
Load phase:All threads running...
Load phase complete time: 81.898 seconds
Peak phase:All threads running...
Peak phase complete time: 372.404 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 71.507 seconds
Client end time: 1542746358431 milliseconds
========================================
Total requests sent: 481000
Total response received: 481000
Total wall time: 566.303 seconds
Total number of successful requests: 480996
Overall throughput across all phases: 849.3686564403769 seconds
latency of 95th Percentile: 0.072 seconds
latency of 99th Percentile: 0.069 seconds
```

Default Client Settings: 128 threads in peak phase and 100 iterations:



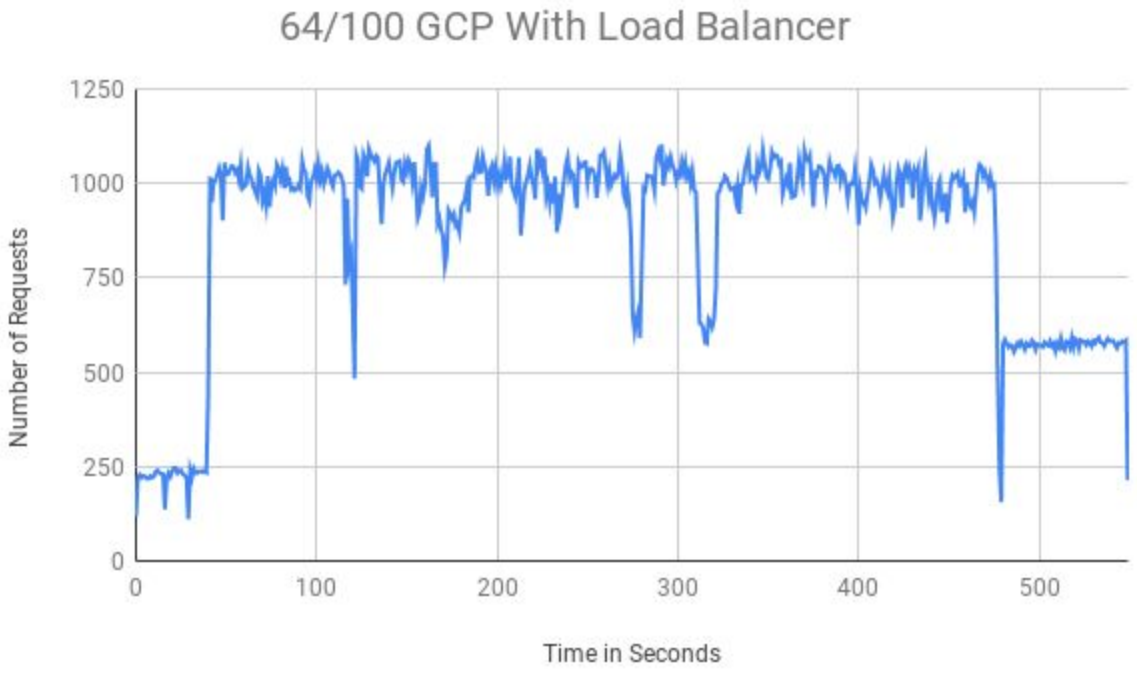

```
Client: Thread Count: 128, Iteration Count: 100
Client starting time: 1542746908680 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 42.972 seconds
Load phase:All threads running...
Load phase complete time: 171.264 seconds
Peak phase:All threads running...
Peak phase complete time: 781.320 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 94.028 seconds
Client end time: 1542747998272 milliseconds
========================================
Total requests sent: 962000
Total response received: 962000
Total wall time: 1089.592 seconds
Total number of successful requests: 961994
Overall throughput across all phases: 882.8992538375238 seconds
latency of 95th Percentile: 0.143 seconds
latency of 99th Percentile: 0.146 seconds
```

Default Client Settings: 32 threads in peak phase and 100 iterations: (With Load Balancer)



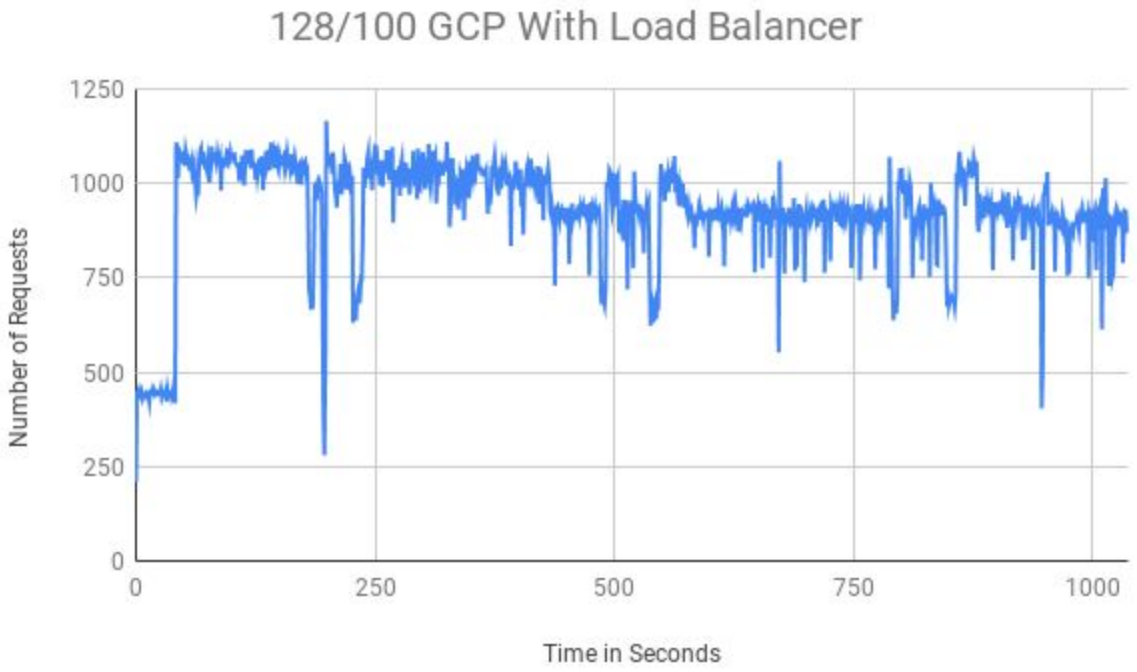

```
Client: Thread Count: 32, Iteration Count: 100
Client starting time: 1542751386293 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 45.517 seconds
Load phase:All threads running...
Load phase complete time: 87.057 seconds
Peak phase:All threads running...
Peak phase complete time: 190.766 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 67.771 seconds
Client end time: 1542751777407 milliseconds
========================================
Total requests sent: 240500
Total response received: 240500
Total wall time: 391.114 seconds
Total number of successful requests: 240498
Overall throughput across all phases: 614.9102092817554 seconds
latency of 95th Percentile: 0.023 seconds
latency of 99th Percentile: 0.024 seconds
```

Default Client Settings: 64 threads in peak phase and 100 iterations: (With Load Balancer)




```
Client: Thread Count: 64, Iteration Count: 100
Client starting time: 1542751965727 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 40.672 seconds
Load phase:All threads running...
Load phase complete time: 80.831 seconds
Peak phase:All threads running...
Peak phase complete time: 359.572 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 70.129 seconds
Client end time: 1542752516937 milliseconds
========================================
Total requests sent: 481000
Total response received: 481000
Total wall time: 551.210 seconds
Total number of successful requests: 481000
Overall throughput across all phases: 872.6256432686213 seconds
latency of 95th Percentile: 0.031 seconds
latency of 99th Percentile: 0.034 seconds
```

Default Client Settings: 128 threads in peak phase and 100 iterations: (With Load Balancer)

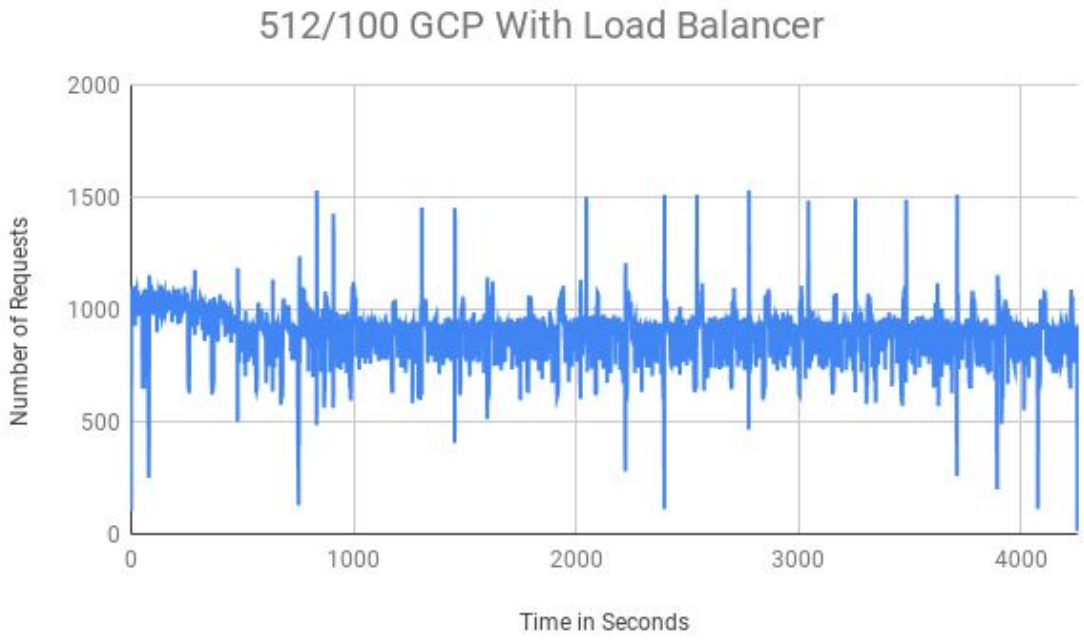



```
Client: Thread Count: 128, Iteration Count: 100
Client starting time: 1542752672377 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 41.444 seconds
Load phase:All threads running...
Load phase complete time: 156.800 seconds
Peak phase:All threads running...
Peak phase complete time: 752.691 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 90.468 seconds
Client end time: 1542753713785 milliseconds
========================================
Total requests sent: 962000
Total response received: 962000
Total wall time: 1041.408 seconds
Total number of successful requests: 961997
Overall throughput across all phases: 923.749421829062 seconds
latency of 95th Percentile: 0.135 seconds
latency of 99th Percentile: 0.125 seconds
```

Default Client Settings: 256 threads in peak phase and 100 iterations: (With Load Balancer)


```
Client: Thread Count: 256, Iteration Count: 100
Client starting time: 1542753881470 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 45.000 seconds
Load phase:All threads running...
Load phase complete time: 326.127 seconds
Peak phase:All threads running...
Peak phase complete time: 1562.703 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 178.974 seconds
Client end time: 1542755994315 milliseconds
=========================================
Total requests sent: 1925500
Total response received: 1925500
Total wall time: 2112.845 seconds
Total number of successful requests: 1925494
Overall throughput across all phases: 911.3304699110133 seconds
latency of 95th Percentile: 0.374 seconds
latency of 99th Percentile: 0.265 seconds
```

(Bonus Part) Default Client Settings: 512 threads in peak phase and 100 iterations: (With Load Balancer)




```
Client: Thread Count: 512, Iteration Count: 100
Client starting time: 1542756270600 milliseconds
Warmup phase:All threads running...
Warmup phase complete time: 80.766 seconds
Load phase:All threads running...
Load phase complete time: 675.448 seconds
Peak phase:All threads running...
Peak phase complete time: 3155.147 seconds
Cooldown phase:All threads running...
Cooldown phase complete time: 361.514 seconds
Client end time: 1542760543520 milliseconds
========================================
Total requests sent: 3852500
Total response received: 3852500
Total wall time: 4272.920 seconds
Total number of successful requests: 3852489
Overall throughput across all phases: 901.6082843671651 seconds
latency of 95th Percentile: 0.592 seconds
latency of 99th Percentile: 0.580 seconds
```

For 1028 threads I got Connection Timed Out exception multiple times. When I improved the timeout config in the load balancer from 30 seconds to 300 seconds, that somehow froze the execution of my client application in the Peak phase of 1024/100. So could not complete the execution.

Comparison of Google Compute Platform performance with EC2 :

If we look at the table below we can see GCP has more througput time than EC both for with and without load balancer. This means the performance of EC2 is better in processing the same number of requests. But one of the factors of such varied result could be the Autoscale rule I used in GCP was CPU utilization>=60 % which was CPU utilization>=40 % for scale out in EC2. So the same configuration of requests could be possible being proccessed in lesser number of instances in GCP than EC2.

Throughput Comparison in seconds	32/100 config	64/100 config	128/100 config	256/100 config
GCP without Load Balancer	651	849	882	-
GCP with Load Balancer	614	872	923	911
EC2 without Load Balancer	560	811	886	1256
EC2 with Load Balancer	597	593	866	907

From P99 comparison below we can see For lower number of threads(32,64) GCP with Load Balancer has better performance than EC2 with Load Balancer meaning the Computing power of Google's compute engine has a better performance than an EC2 instance, but with more number of threads the performance is almost the same if not marginally lesser. This scaling factor explained previously could also be the reason for such anomaly.

P99 Comparison in seconds	32/100 config	64/100 config	128/100 config	256/100 config
GCP without Load Balancer	0.031	0.069	0.146	-
GCP with Load Balancer	0.024	0.034	0.125	0.265
EC2 without Load Balancer	0.026	0.054	0.154	0.263
EC2 with Load Balancer	0.042	0.098	0.074	0.168

Overall, Google Cloud Platform was much easy to use and learn than AWS mostly because of better documentation and guided examples. But having said that I feel more biased towards AWS because I faced most of the mistakes in AWS and learned from them and then applied my sense of understanding in GCP. So it is very likely possible ease of use with Google Cloud Platform will play a major role in increased customer base for Google!