

# Assignment 3

Shufan Xing & Wen Zhang

## Step 1 : Run on AWS Lambda

### Development Environment:

- Client:  
AWS EC2 instance (t2.micro)
- Server:  
AWS Lambda
- Database:  
AWS RDS PostgreSQL in db.t2.medium (max connections: 312)
- language: Java, Python
- tools: Postman, Eclipse, PowerShell

### Source Code Repos:

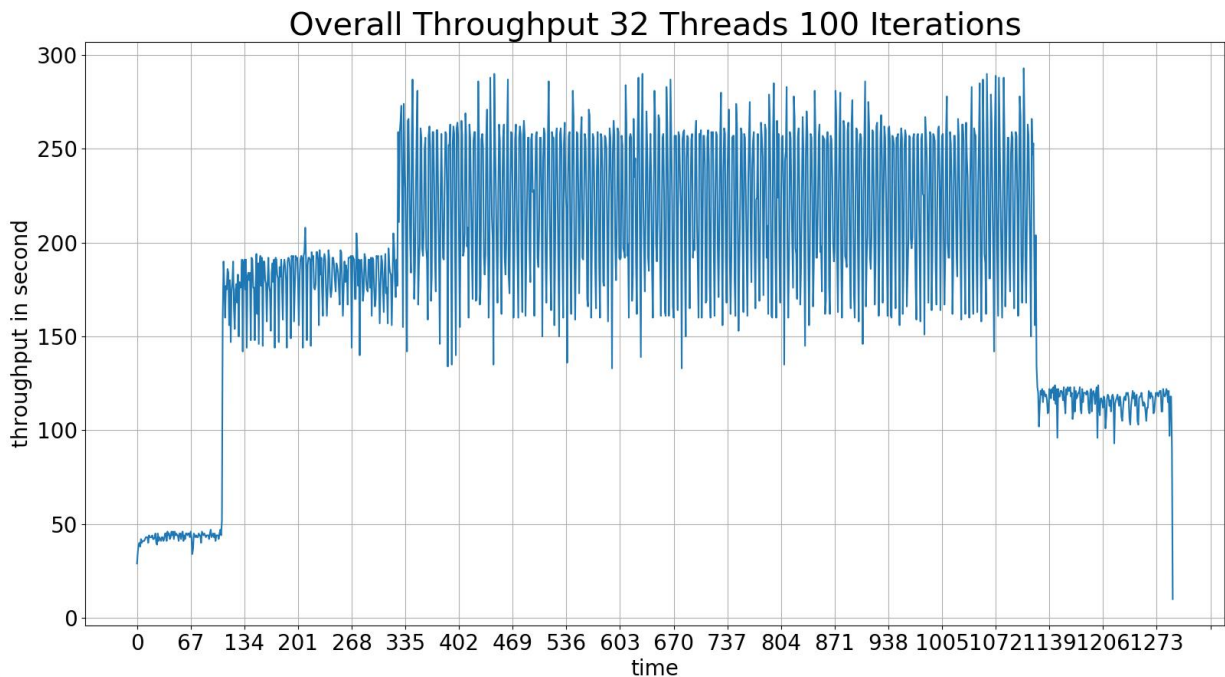
Server: [https://github.com/shufanxing/wearable\\_lambda.git](https://github.com/shufanxing/wearable_lambda.git)

Client: <https://github.com/shufanxing/wearableDeviceClient.git>

### Test Results:

Statistic and plots for 32, 64, 128, 256 max threads respectively

## 1. 32 max threads:



### - Performance statistics

```
url: https://9329js3tmh.execute-api.us-west-2.amazonaws.com/Prod
client maxthread: 32;
numberOfTestsPerPhase: 100
dayNumber: 1
userPolulation: 1000000
deleted old records in table:{"message":"281"}
Client start..... Time: 1543306385324
Warmup phase: All threads running...
Warmup phase complete: Time 104.574 seconds
Loading phase: All threads running...
Loading phase complete: Time 221.288 seconds
Peak phase: All threads running...
Peak phase complete: Time 793.052 seconds
Cooldown phase: All threads running...
Cooldown phase complete: Time 174.508 seconds

=====
Total number of requests sent: 240500
Total number of Successful responses: 240500
Test Wall Time: 1293.424 seconds
Overall throughput accross all phases:185.9
Mean latency for all requests: 128.1 milliseconds
Median latency for all requests: 119.0 milliseconds
99th percentile latency for all requests: 290 milliseconds
95th percentile latency for all requests: 253 milliseconds
```

wall time: 1293.4 s

Failure: 0

AVG throughput: 185.9/second

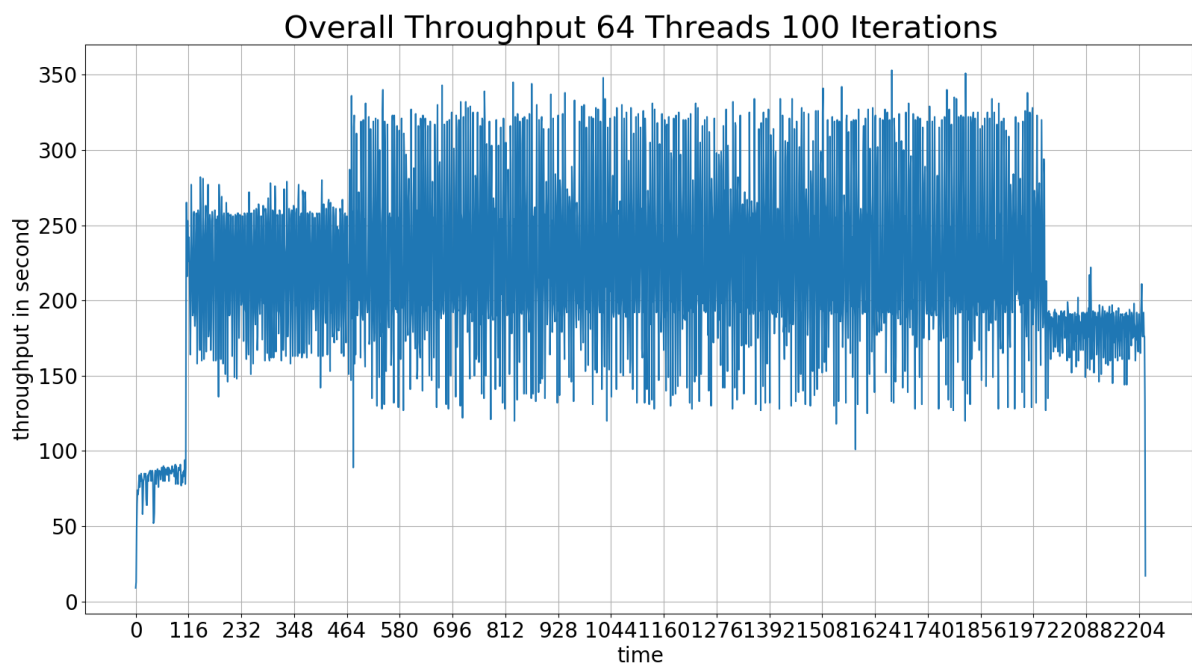
mean latency: 128.1 ms

median latency: 119.0 ms

p99: 290 ms

p95: 253 ms

## 2. 64 max threads



- Performance statistics

```
url: https://9329js3tmh.execute-api.us-west-2.amazonaws.com/Prod
client maxthread: 64;
numberOfTestsPerPhase: 100
dayNumber: 1
userPolulation: 1000000
deleted old records in table:{"message":"264562"}
Client start..... Time: 1543461438279
Warmup phase: All threads running...
Warmup phase complete: Time 108.425 seconds
Loading phase: All threads running...
Loading phase complete: Time 360.883 seconds
Peak phase: All threads running...
Peak phase complete: Time 1521.766 seconds
Cooldown phase: All threads running...
Cooldown phase complete: Time 227.579 seconds
=====
Total number of requests sent: 481000
Total number of Successful responses: 481000
Test Wall Time: 2218.655 seconds
Overall throughput accross all phases:216.8
Mean latency for all requests: 236.8 milliseconds
Median latency for all requests: 215.0 milliseconds
99th percentile latency for all requests: 553 milliseconds
95th percentile latency for all requests: 441 milliseconds
```

wall time: 2218.6s

Failure: 0

AVG throughput: 216.8/second

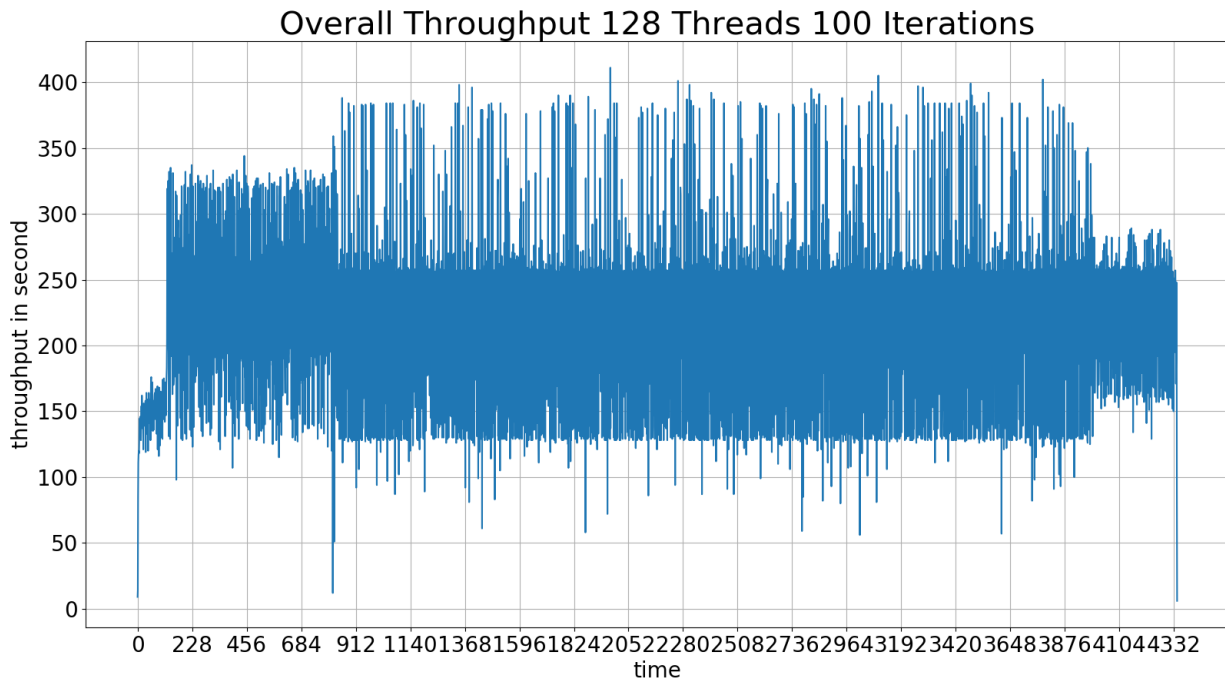
mean latency: 236.8 ms

median latency: 215.0 ms

p99: 553 ms

p95: 441 ms

### 3. 128 max-threads



#### - Performance statistics

```
url: https://9329js3tmh.execute-api.us-west-2.amazonaws.com/Prod
client maxthread: 128;
numberOfTestsPerPhase: 100
dayNumber: 1
userPolulation: 1000000
deleted old records in table:{"message":"188165"}
Client start..... Time: 1543387141702
Warmup phase: All threads running...
Warmup phase complete: Time 121.135 seconds
Loading phase: All threads running...
Loading phase complete: Time 690.562 seconds
Peak phase: All threads running...
Peak phase complete: Time 3164.544 seconds
Cooldown phase: All threads running...
Cooldown phase complete: Time 369.094 seconds

=====
Total number of requests sent: 962000
Total number of Successful responses: 961999
Test Wall Time: 4345.338 seconds
Overall throughput accross all phases:221.4
Mean latency for all requests: 482.7 milliseconds
Median latency for all requests: 483.0 milliseconds
99th percentile latency for all requests: 1094 milliseconds
95th percentile latency for all requests: 857 milliseconds
```

wall time: 4345.3s

Failure: 0

AVG throughput: 221.4/second

mean latency: 482.7 ms

median latency: 483.0 ms

p99: 1094 ms

p95: 857 ms

## Analysis:

1. While the thread number increases:

1) wall time increases

2) Average throughput slightly increases, though the latency increases

3) when the max thread is 32, the loading stage is obvious; when 64, 128, it's not, it's almost the same as that in peak phase. My best guess is when the thread number in loading stage is high, the throughput is limited by other hardware conditions, such as the creation of database connections, and single CPU and 1G memory in t2.micro.

4) Based on the plots, when the thread number is under 32, the throughput increases while the increasement of thread number. When the thread number is higher than 32, the throughput doesn't increase anymore while the thread number increases.

2. Compared with EC2 server, wall time, throughput and latency increases are as following table:

	wall time(ms)				throughput				mean latency			
Thread number	32	64	128		32	64	128		32	64	128	
EC2 server	167	236	503		1435	2034	1909		14	23	52	
lambda	1293	2218	4345		185	216	221		128	236	482	

*Table: wall time, throughput and mean latency in EC2 server and AWS Lambda*

In the table, EC2 server performance is 8-10 times faster than AWS lambda. The main reason is that AWS Lambda can't share connections by using connection pool in Java, so every Lambda function query needs to create a new connection to database and it's time consuming (usually it takes more than 100 milliseconds according some tests ).

# Step 2 : Run on Google Cloud Platform (GCP)

## Implementation:

1. Server:
  - HTTP Requests handlers are implemented in JAX-RS.
  - The server-database communication is implemented by JDBC and C3P0 connection pool.
  - Server is deployed to GCP Compute Engine VM instances using tomcat server.
  - DataBase server: GCP MySQL Instance (2nd Gen 5.7).
2. Client:
  - Option Class: to store the clients' requirements from the console.
  - OptionParser: to parse the console input arguments and store the values into an Option Object.
  - Worker Class: to implement Runnable, which is responsible to implement the execution of each thread.
  - JfreeChart Class: to plot the final throughput chart.
  - WDDMClient Class: main thread, to execute four phases based on input configuration, and generate system performance analysis.
  - Details: Client application saves every latency and timestamp into an output file for later performance evaluation.
  - Four phases in my implementation are Non-overlapping, and I use count down latch for execution synchronization.

## Source Code Repos:

<https://github.com/wen-ivy-zhang/NEUCS-Distributed-System/tree/master/Assignment3>

## Instances:

1: Compute Engines:

VM instances						
<div><div>CREATE INSTANCE</div><div>IMPORT VM</div><div>REFRESH</div><div>START</div><div>ST</div></div>						
<div><div>Filter VM instances</div></div>						
<input type="checkbox"/>	Name ^	Zone	Recommendation	Internal IP	External IP	Connect
<input type="checkbox"/>	<input checked="" type="checkbox"/> tomcat-1-vm	us-west1-a		10.138.0.3 (nic0)	35.233.174.106 <a href="#">↗</a>	SSH ▾ ⋮
<input type="checkbox"/>	<input checked="" type="checkbox"/> tomcat-2-vm	us-west1-a		10.138.0.4 (nic0)	35.247.106.208 <a href="#">↗</a>	SSH ▾ ⋮
<input type="checkbox"/>	<input checked="" type="checkbox"/> wen-instance1	us-west1-b		10.138.0.2 (nic0)	None	SSH ▾ ⋮

## 2: MySQL Database:

Google Cloud Platform

Wen Zhang WDDM Project

SQL

Instances

CREATE INSTANCE

MIGRATE DATA

Filter instances

Instance ID	Type
<input checked="" type="checkbox"/> wendb	MySQL 2nd Gen 5.7

## 3: Load Balancer:

Google Cloud Platform

Wen Zhang WDDM Project

Network services

Load balancing

Cloud DNS

Cloud CDN

Cloud NAT

Load balancing

CREATE LOAD BALANCER

REFRESH

DELETE

Load balancers

Backends

Frontends

Filter by name or protocol

Name	Protocol	Backends
<input checked="" type="checkbox"/> wenlb	HTTP	1 backend service (1 instance group, 0 network endpoint groups)



## Test Results:

Single compute engine instance: Performance Statistics and plots of four different max threads values:  
32, 64, 128, 256.

### 1. Performance Statistics: maxThread = 32

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Nov 25, 2018, 7:09:12 PM)
url: http://35.233.174.106:8080/homework2/webapi/hw2server
Client starting.....Time: 2018/11/25 19:09:12
Warmup phase: All threads running.....
Warmup phase complete: Time 46.284 seconds
Loading phase: All threads running.....
Loading phase complete: Time 78.214 seconds
Peak phase: All threads running.....
Peak phase complete: Time 173.579 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 71.717 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 369.826 seconds
Total number of request sent: 240500
Total number of Successful responses: 240500
Overall throughput across all phases: 650.306 requests per second
Mean latencies for all requests: 30 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 47 MilliSeconds*
99th percentile latency for all requests: 78 MilliSeconds*
```

client setting: **MaxThreads: 32**; Day number:1; User population: 100,000; Number of tests per phase: 100.

XY-LineChart of overall throughput:



Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $3 + 16 + 32 + 8 = 59$ ).

## 2. Performance Statistics: maxThread = 64

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Nov 25, 2018, 7:21:34 PM)
url: http://35.233.174.106:8080/homework2/webapi/hw2server
Client starting.....Time: 2018/11/25 19:21:34
Warmup phase: All threads running.....
Warmup phase complete: Time 44.588 seconds
Loading phase: All threads running.....
Loading phase complete: Time 82.044 seconds
Peak phase: All threads running.....
Peak phase complete: Time 228.007 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 78.41 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 433.062 seconds
Total number of request sent: 481000
Total number of Successful responses: 481000
Overall throughput across all phases: 1,110.695 requests per second
Mean latencies for all requests: 37 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 56 MilliSeconds*
99th percentile latency for all requests: 289 MilliSeconds*
```

client setting: **MaxThreads: 64**; Day number:1; User population: 100,000; Number of tests per phase: 100.

### XY-LineChart of overall throughput:



*Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $6 + 32 + 64 + 16 = 118$ ).*

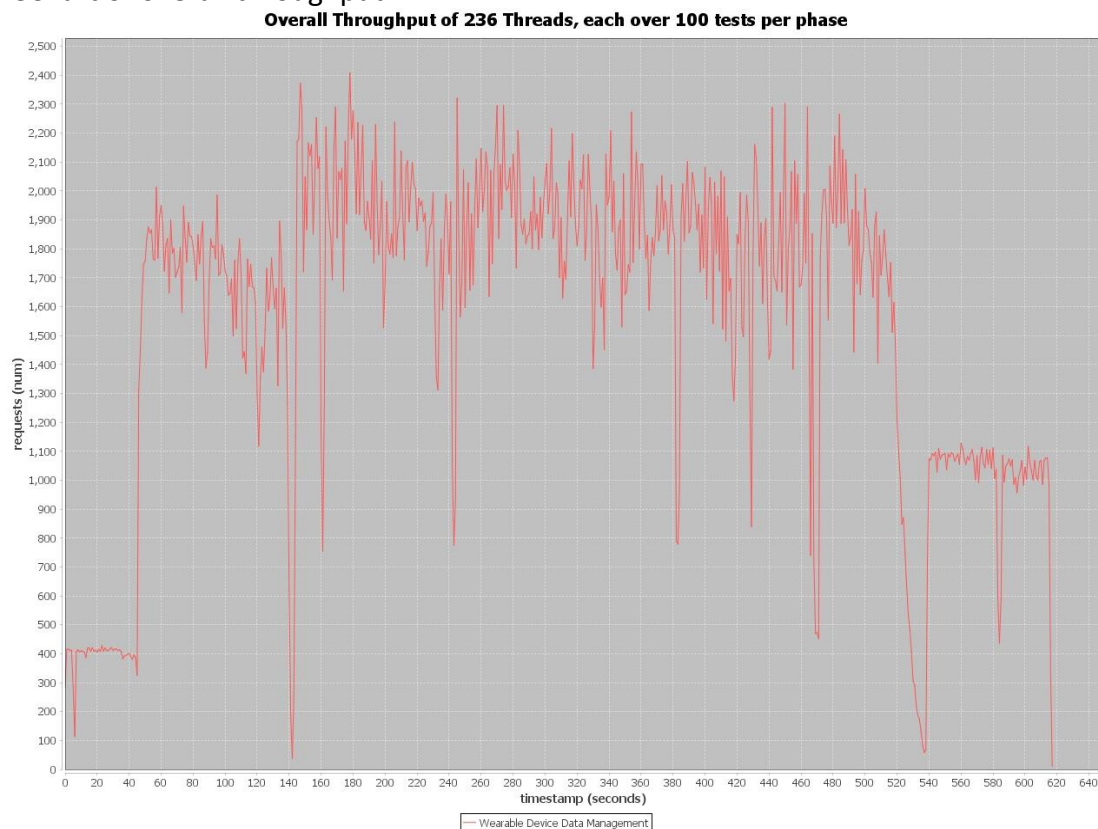
### 3. Performance Statistics: maxThread = 128

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Nov 25, 2018, 7:35:36 PM)
url: http://35.233.174.106:8080/homework2/webapi/hw2server
Client starting.....Time: 2018/11/25 19:35:36
Warmup phase: All threads running.....
Warmup phase complete: Time 45.694 seconds
Loading phase: All threads running.....
Loading phase complete: Time 97.581 seconds
Peak phase: All threads running.....
Peak phase complete: Time 395.615 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 78.41 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 617.324 seconds
Total number of request sent: 962000
Total number of Successful responses: 962000
Overall throughput across all phases: 1,558.339 requests per second
Mean latencies for all requests: 59 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 290 MilliSeconds*
99th percentile latency for all requests: 574 MilliSeconds*
```

client setting: **MaxThreads: 128**; Day number:1; User population: 100,000; Number of tests per phase: 100.

### XY-LineChart of overall throughput:



Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $12 + 64 + 128 + 32 = 236$ )

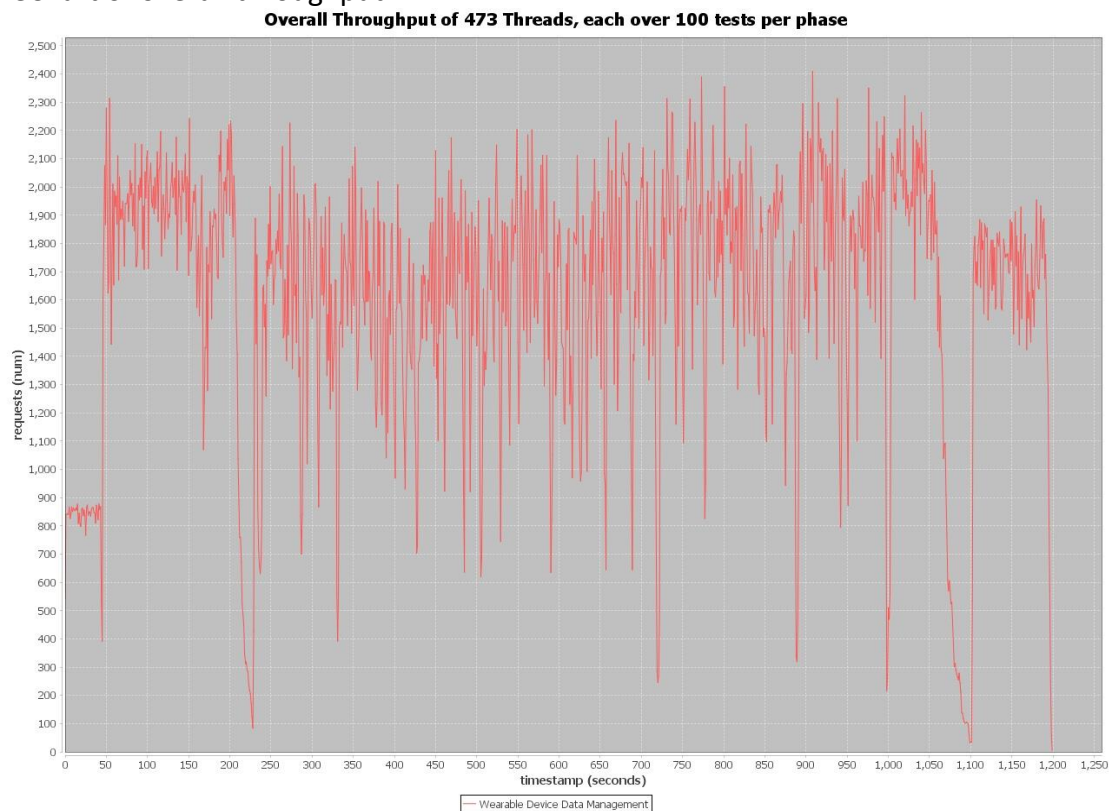
#### 4. Performance Statistics: maxThread = 256

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Nov 25, 2018, 7:55:49 PM)
url: http://35.233.174.106:8080/homework2/webapi/hw2server
Client starting....Time: 2018/11/25 19:55:50
Warmup phase: All threads running....
Warmup phase complete: Time 45.209 seconds
Loading phase: All threads running....
Loading phase complete: Time 183.881 seconds
Peak phase: All threads running....
Peak phase complete: Time 873.053 seconds
Cooldown phase: All threads running....
Cooldown phase complete: Time 96.982 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 1199.16 seconds
Total number of request sent: 1925500
Total number of Successful responses: 1925500
Overall throughput across all phases: 1,605.707 requests per second
Mean latencies for all requests: 121 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 532 MilliSeconds*
99th percentile latency for all requests: 1760 MilliSeconds*
```

client setting: **MaxThreads: 256**; Day number:1; User population: 100,000; Number of tests per phase: 100.

#### XY-LineChart of overall throughput:



Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $25 + 128 + 256 + 64 = 473$ ).



## GCP Load Balancer against 2 same compute engine instances: Performance Statistics and plots of four different max threads values: 32, 64, 128, 256.

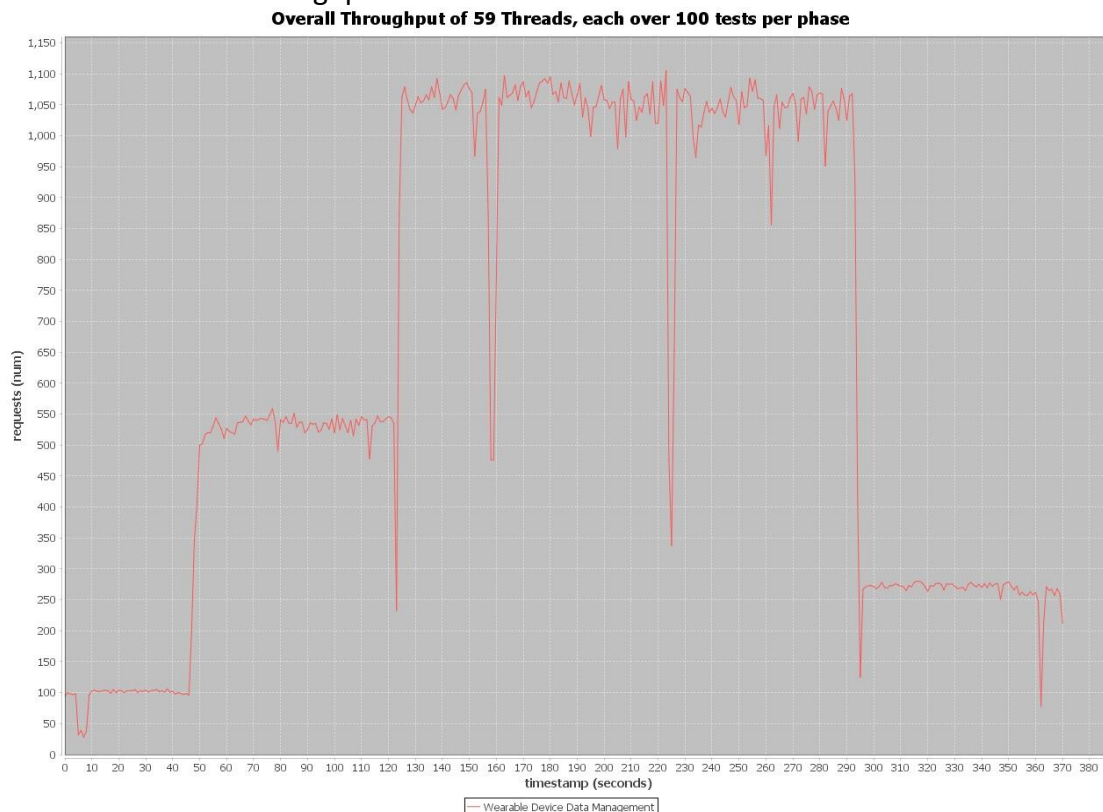
### 1. Performance Statistics: maxThread = 32

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Dec 1, 2018, 6:20:39 PM)
url: http://35.244.218.112:8080/homework2/webapi/hw2server
Initial POST request succeeds : Initial WDDM Database connected!
Client starting.....Time: 2018/12/01 18:20:40
Warmup phase: All threads running.....
Warmup phase complete: Time 47.185 seconds
Loading phase: All threads running.....
Loading phase complete: Time 76.711 seconds
Peak phase: All threads running.....
Peak phase complete: Time 171.952 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 75.139 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 371.003 seconds
Total number of request sent: 240500
Total number of Successful responses: 240500
Overall throughput across all phases: 648.243 requests per second
Mean latencies for all requests: 30 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 41 MilliSeconds*
99th percentile latency for all requests: 58 MilliSeconds*
```

client setting: **MaxThreads: 32**; Day number:1; User population: 100,000; Number of tests per phase: 100.

### XY-LineChart of overall throughput:



Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $3 + 16 + 32 + 8 = 59$ ).

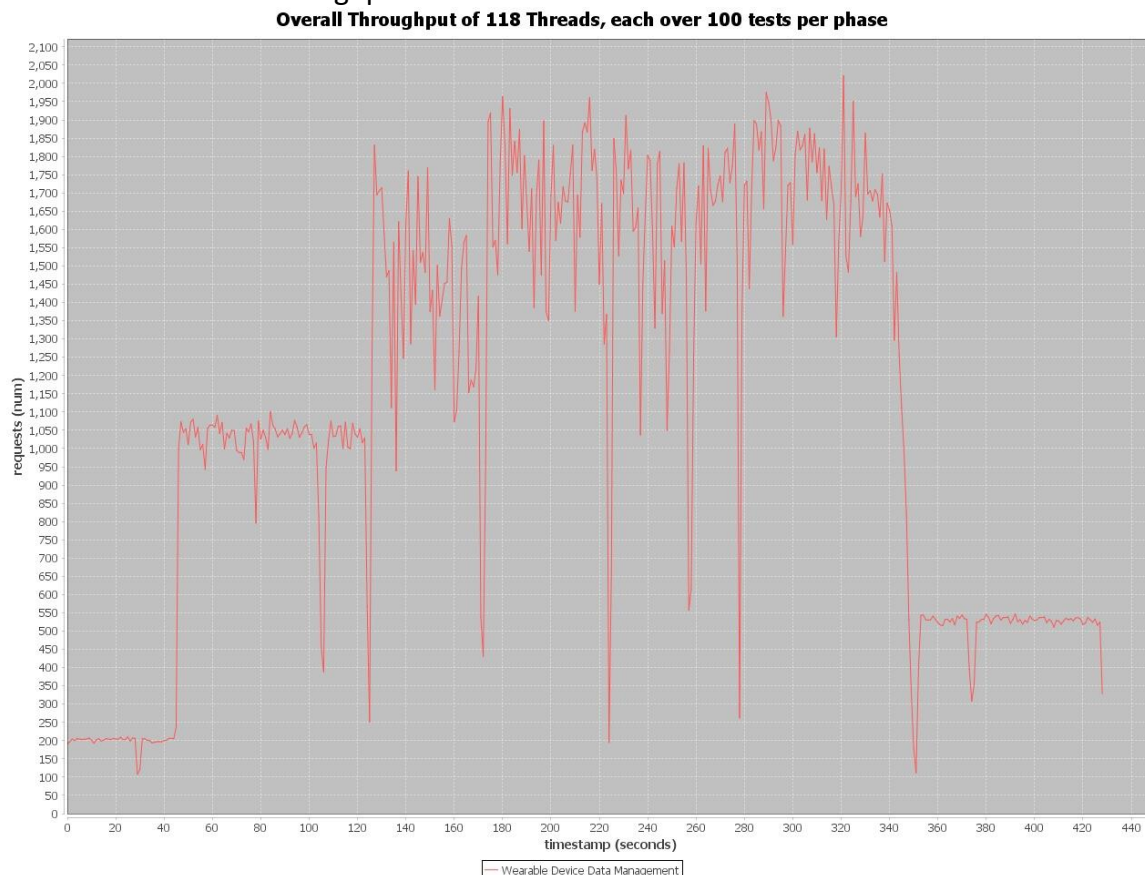
## 2. Performance Statistics: maxThread = 64

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Dec 1, 2018, 6:29:03 PM)
url: http://35.244.218.112:8080/homework2/webapi/hw2server
Initial POST request succeeds : Initial WDDM Database connected!
Client starting.....Time: 2018/12/01 18:29:04
Warmup phase: All threads running.....
Warmup phase complete: Time 45.595 seconds
Loading phase: All threads running.....
Loading phase complete: Time 79.865 seconds
Peak phase: All threads running.....
Peak phase complete: Time 226.68 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 76.835 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 429.001 seconds
Total number of request sent: 481000
Total number of Successful responses: 481000
Overall throughput across all phases: 1,121.21 requests per second
Mean latencies for all requests: 37 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 53 MilliSeconds*
99th percentile latency for all requests: 297 MilliSeconds*
```

client setting: **MaxThreads: 64**; Day number:1; User population: 100,000; Number of tests per phase: 100.

### XY-LineChart of overall throughput:



*Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $6 + 32 + 64 + 16 = 118$ ).*

### 3. Performance Statistics: maxThread = 128

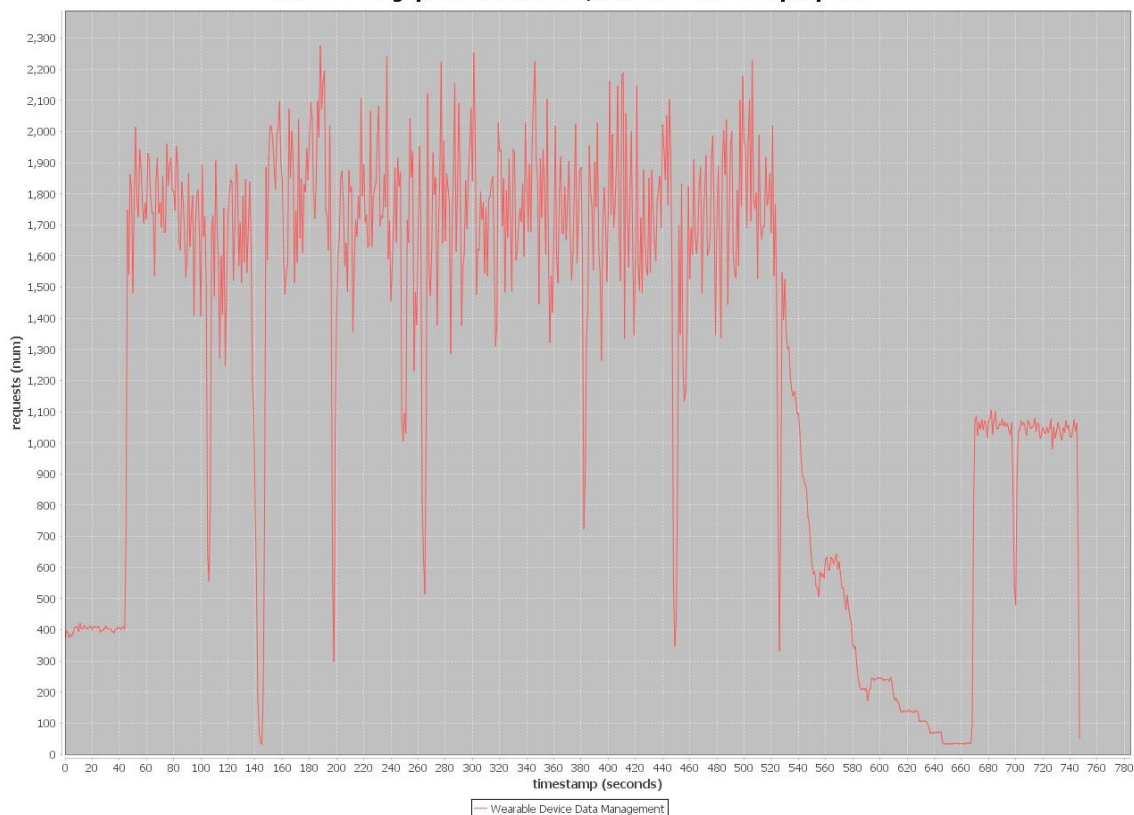
```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Dec 1, 2018, 6:39:47 PM)
url: http://35.244.218.112:8080/homework2/webapi/hw2server
Initial POST request succeeds : Initial WDDM Database connected!
Client starting.....Time: 2018/12/01 18:39:48
Warmup phase: All threads running....
Warmup phase complete: Time 44.947 seconds
Loading phase: All threads running....
Loading phase complete: Time 101.454 seconds
Peak phase: All threads running....
Peak phase complete: Time 522.338 seconds
Cooldown phase: All threads running....
Cooldown phase complete: Time 78.556 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 747.314 seconds
Total number of request sent: 962000
Total number of Successful responses: 962000
Overall throughput across all phases: 1,287.277 requests per second
Mean latencies for all requests: 61 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 78 MilliSeconds*
99th percentile latency for all requests: 734 MilliSeconds*
```

client setting: **MaxThreads: 128**; Day number:1; User population: 100,000; Number of tests per phase: 100.

### XY-LineChart of overall throughput:

Overall Throughput of 236 Threads, each over 100 tests per phase



Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $12 + 64 + 128 + 32 = 236$ )

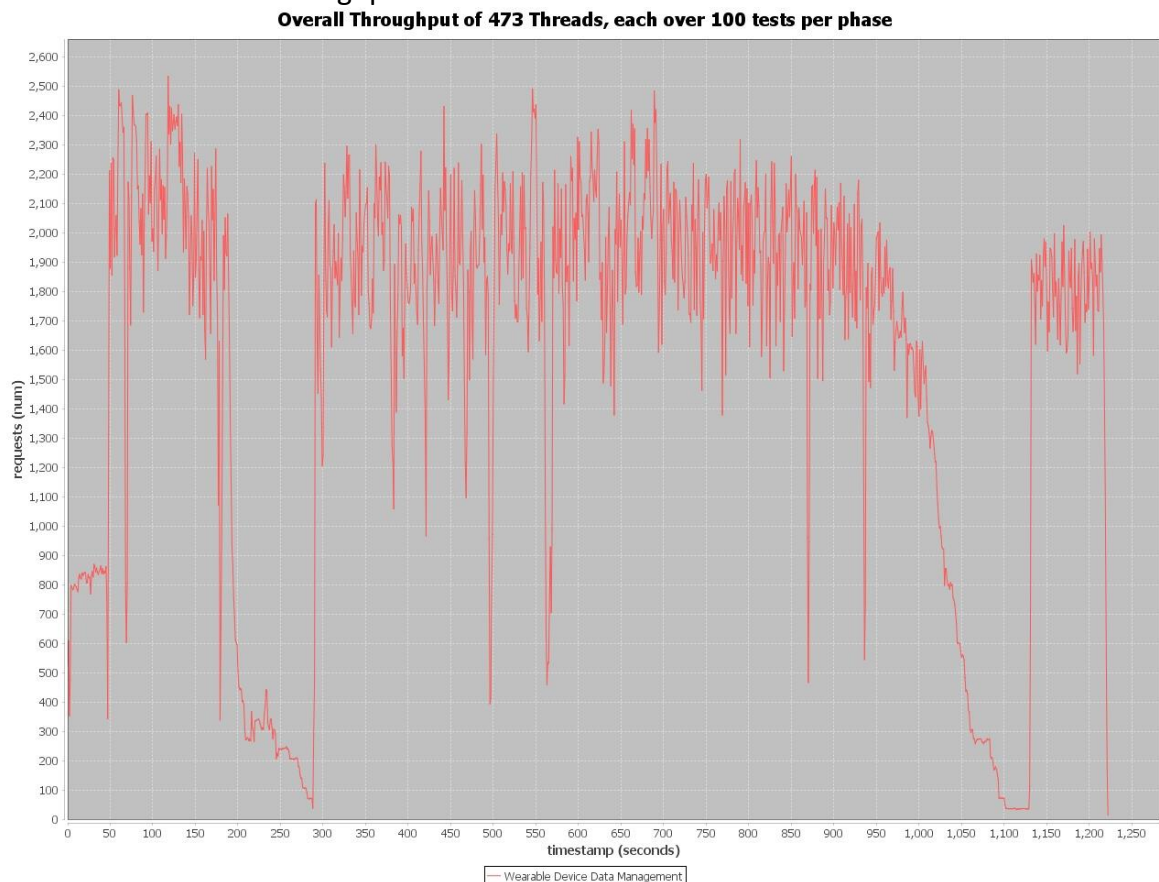
#### 4. Performance Statistics: maxThread = 256

```
Markers Properties Servers Data Source Explorer Snippets Console Search
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Dec 1, 2018, 6:57:34 PM)
url: http://35.244.218.112:8080/homework2/webapi/hw2server
Initial POST request succeeds : Initial WDDM Database connected!
Client starting.....Time: 2018/12/01 18:57:35
Warmup phase: All threads running.....
Warmup phase complete: Time 47.404 seconds
Loading phase: All threads running.....
Loading phase complete: Time 242.234 seconds
Peak phase: All threads running.....
Peak phase complete: Time 840.815 seconds
Cooldown phase: All threads running.....
Cooldown phase complete: Time 91.909 seconds
Client Execution Complete.

=====Execution Summary=====
Generating XYLineChart for the overall throughput...
XYLineChart WDDMThroughput.jpeg generated successfully.
Test Wall Time: 1222.405 seconds
Total number of request sent: 1925500
Total number of Successful responses: 1925500
Overall throughput across all phases: 1,575.174 requests per second
Mean latencies for all requests: 98 MilliSeconds*
Median latencies for all requests: 31 MilliSeconds*
95th percentile latency for all requests: 97 MilliSeconds*
99th percentile latency for all requests: 1282 MilliSeconds*
```

client setting: **MaxThreads: 256**; Day number:1; User population: 100,000; Number of tests per phase: 100.

#### XY-LineChart of overall throughput:



*Note: The thread number in the above chart is the total number of threads generated during all four phases (e.g:  $25 + 128 + 256 + 64 = 473$ ).*



## Performance comparison between AWS and GCP:

Below tables summarize the running tests data between Google Cloud Platform and Amazon Web Service.

### Single Client-Server:

Perf Threads	Wall time (seconds)		Throughput (Req per Sec)		Mean Latency (MilliSeconds)		95 percentile (MilliSeconds)		99 percentile (MilliSeconds)	
	GCP	AWS	GCP	AWS	GCP	AWS	GCP	AWS	GCP	AWS
32	370	379	650	632	30	32	47	48	78	131
64	433	429	1111	1120	37	37	56	62	289	312
128	617	561	1558	1712	59	52	290	71	574	565
256	1199	1150	1606	1674	121	112	532	331	1760	1641

### Client-Load Balancer-Servers:

Perf Threads	Wall time (seconds)		Throughput (Req per Sec)		Mean Latency (MilliSeconds)		95 percentile (MilliSeconds)		99 percentile (MilliSeconds)	
	GCP	AWS	GCP	AWS	GCP	AWS	GCP	AWS	GCP	AWS
32	371	361	648	665	30	30	41	47	58	98
64	429	408	1121	1178	37	35	53	57	297	280
128	747	635	1287	1514	61	50	78	69	734	487
256	1222	1157	1575	1663	98	101	97	174	1282	974

We can see from the above two tables that AWS performances are slightly better than GCP for all the test cases except for the one of 32 threads that run against single server. I think the reasons could be manifold:

1: Network speed. For both assignments, I'm using the same computer running the same client program through home WIFI connection. The internet speed difference between the dates of these two assignments could directly affect the differences in the running results. This factor is been controlled by ISP, weather and many other factors and is not controllable among the tests.

2: Sample size. Even though for each assignment, I ran different test cases against different threads, for each thread configuration however, I only executed test once. The sample size is pretty small and can't be reliable enough to determine which cloud service has better performance.

3: Instance locations. The compute and database instances are created and deployed by cloud service platforms. Even though I chose to deploy all my instances in US-west regions for both assignments, the physical location differences of the datacenters that runs these instances between these two cloud services could also cause the result difference.

4: Software differences. The different operating systems, different database and web server applications, and other software differences of the physical host machines and virtual instances could all contribute to the result difference. This means that if I deploy my instances on other environment using different software or OS versions, I might also get different results.

5: Hardware differences. The different hardware resources used in these two assignments could also contribute to the result difference. For example, different CPU, memory, network adapters of the physical host machines could all affect the execution results of the virtual instances.

Based on all these factors, it's NOT accurate to directly declare that AWS performance is better than GCP for this assignment implementation even though it has slightly better test results, especially because of first and second factors. I personally think their performances for this assignment would be almost the same.

During the running of all the tests against single compute engine and load balancer, I captured their corresponding CPU utilization. The results are below:

GCP resource utilizations:

Single CE		
Threads	CPU %	DB %
32	51%	68%
64	68%	82%
128	74%	86%
256	75%	87%

Load Balancer		
Threads	CPU %	DB %
32	42%, 35%	83%
64	55%, 45%	88%
128	58%, 46%	91%
256	63%, 54%	93%

AWS resource utilizations (from previous assignment):

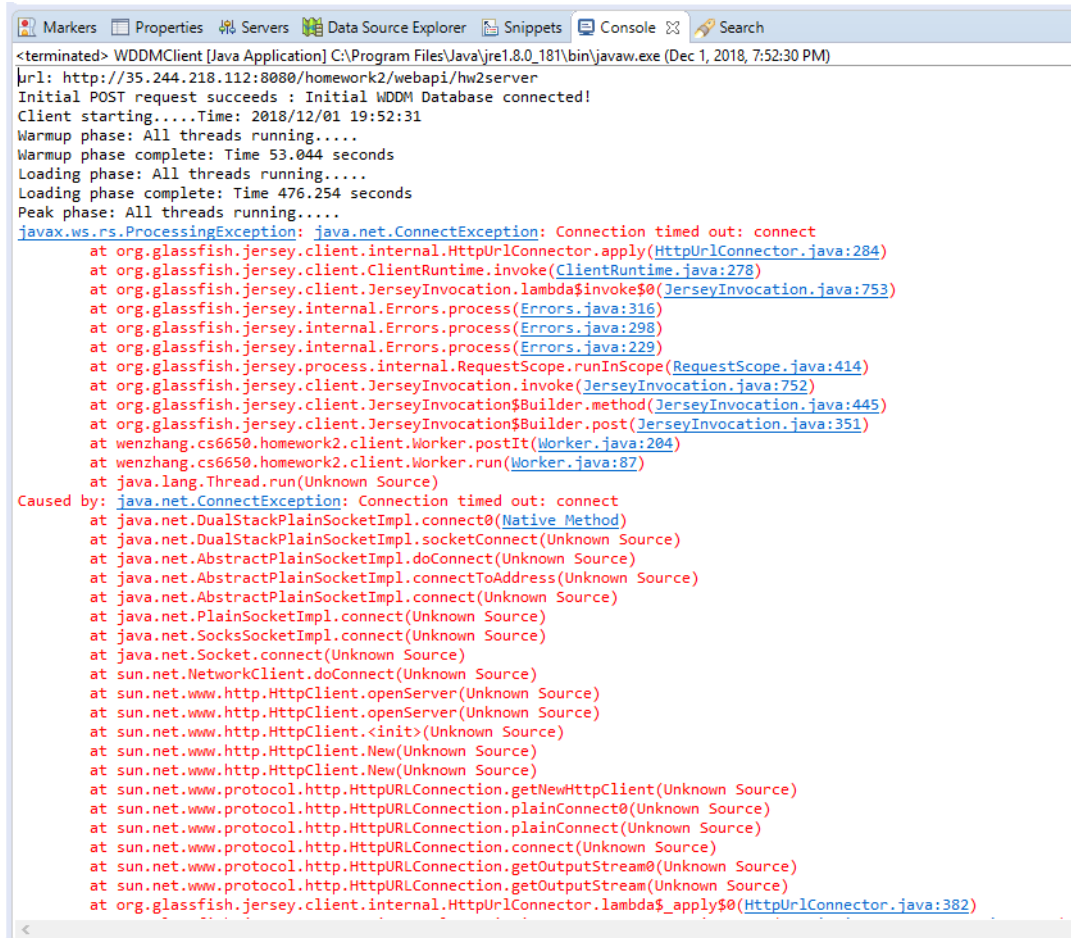
Single EC2		
Threads	CPU %	DB %
32	28%	34%
64	43%	46%
128	75%	53%
256	69%	54%

Load Balancer		
Threads	CPU %	DB %
32	11%, 10%	35%
64	20%, 26%	44%
128	35%, 38%	51%
256	35%, 37%	49%

From the above tables we can see that, with the same test loads, the CPU utilizations of GCP are much higher than that of AWS, especially for local balancer cases. The utilization differences of Database instances might not apply here since for AWS, I used medium tier which has 2 CPUs, while for GCP I only used 1 CPU for database. However, every compute instance has only 1 CPU, and we can see that under the same load, GCP instances CPU utilizations are much higher. This could probably cause by physical hardware differences used by two cloud services.

## Bonus step, Scaling up

I tried 512 maxThreads, 100 iterations setting against load balancer. However, during the test, I saw a lot of time out exceptions. Below is the running result:



```
<terminated> WDDMClient [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Dec 1, 2018, 7:52:30 PM)
url: http://35.244.218.112:8080/homework2/webapi/hw2server
Initial POST request succeeds : Initial WDDM Database connected!
Client starting.....Time: 2018/12/01 19:52:31
Warmup phase: All threads running.....
Warmup phase complete: Time 53.044 seconds
Loading phase: All threads running.....
Loading phase complete: Time 476.254 seconds
Peak phase: All threads running.....
javax.ws.rs.ProcessingException: java.net.ConnectException: Connection timed out: connect
    at org.glassfish.jersey.client.internal.HttpUrlConnector.apply(HttpUrlConnector.java:284)
    at org.glassfish.jersey.client.ClientRuntime.invoke(ClientRuntime.java:278)
    at org.glassfish.jersey.client.JerseyInvocation.lambda$invoke$0(JerseyInvocation.java:753)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:316)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:298)
    at org.glassfish.jersey.internal.Errors.process(Errors.java:229)
    at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:414)
    at org.glassfish.jersey.client.JerseyInvocation.invoke(JerseyInvocation.java:752)
    at org.glassfish.jersey.client.JerseyInvocation$Builder.method(JerseyInvocation.java:445)
    at org.glassfish.jersey.client.JerseyInvocation$Builder.post(JerseyInvocation.java:351)
    at wenzhang.cs6650.homework2.client.Worker.postIt(Worker.java:204)
    at wenzhang.cs6650.homework2.client.Worker.run(Worker.java:87)
    at java.lang.Thread.run(Unknown Source)
Caused by: java.net.ConnectException: Connection timed out: connect
    at java.net.DualStackPlainSocketImpl.connect0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketConnect(Unknown Source)
    at java.net.AbstractPlainSocketImpl.doConnect(Unknown Source)
    at java.net.AbstractPlainSocketImpl.connectToAddress(Unknown Source)
    at java.net.AbstractPlainSocketImpl.connect(Unknown Source)
    at java.net.PlainSocketImpl.connect(Unknown Source)
    at java.net.SocksSocketImpl.connect(Unknown Source)
    at java.net.Socket.connect(Unknown Source)
    at sun.net.NetworkClient.doConnect(Unknown Source)
    at sun.net.www.http.HttpClient.openServer(Unknown Source)
    at sun.net.www.http.HttpClient.openServer(Unknown Source)
    at sun.net.www.http.HttpClient.<init>(Unknown Source)
    at sun.net.www.http.HttpClient.New(Unknown Source)
    at sun.net.www.http.HttpClient.New(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.getNewHttpClient(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.plainConnect0(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.plainConnect(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.connect(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.getOutputStream0(Unknown Source)
    at sun.net.www.protocol.http.HttpURLConnection.getOutputStream(Unknown Source)
    at org.glassfish.jersey.client.internal.HttpUrlConnector.lambda$apply$0(HttpUrlConnector.java:382)
```

When running 512 maxThreads, I saw a lot of connection time out at Socket level. This means that my local machine cannot hold 512 maxThreads simultaneously, so the bottleneck of the system is my local machine.