Assignment 1 Write Up
Peter Tsanev
CS66500, Fall 2023
10/10/23

Github Url: https://github.com/tsanevp/CS66500-Assignment1

Client Design

The architecture I implemented for this assignment's client component strongly emphasizes Java's synchronization tools to prevent potential race conditions and deadlock situations. I leveraged classes like Java's CountDownLatch, ExecutorService, and AtomicInteger to effectively manage thread interactions, minimizing contention and maximizing request throughput.

Major Classes

AlbumClient:

The AlbumClient class simulates client-server interactions by initializing and loading a server with a predefined number of threads. It uses Java's synchronization mechanisms, such as CountDownLatch, to control concurrent execution and prevent race conditions. After successfully executing these interactions, the class calculates and prints various performance metrics, including throughput and latency, to evaluate the server's performance.

AlbumThreadRunnable:

The AlbumThreadRunnable class represents a thread that sends a specified number of GET and POST requests to a server, tracking the success, failure, and latency of these requests. It uses the Swagger-generated API client to interact with the server. This class alternates between making POST and GET requests in groups to simulate client interactions and calculates various performance metrics. The measured metrics, including the number of successful and failed requests, request latency, and other statistics, are aggregated and then updated in the AlbumClient class to assess the server's performance.

LoadCalculations:

The LoadCalculations class provides various statistical calculations for a list of latencies. It includes the methods to calculate the mean, median, p99 (99th percentile), minimum, and maximum response times in milliseconds. The class sorts the latencies in ascending order during initialization, ensuring accurate statistical calculations. It provides a convenient way to analyze latency data collected during load testing or performance monitoring.

WriteToCsv:

The "WriteToCsv" class is responsible for managing the writing of load test results to a CSV file. It uses Apache POI, a popular library for working with Microsoft Office documents, to create and write data to an Excel file (XLSX format). It ensures that data is written in a structured manner and multiple threads can safely contribute their results to the same file without conflicts. After the server load test is completed, all results are written to an Excel file.

Packages

As already mentioned, the client imports and utilizes existing Java packages. The Gson library converts Album and Image data into JSON format for JSON serialization. Next, the Apache POI library writes the server load test results to Excel sheets for data collection. Lastly, the Swagger auto-generated client library Ian provided us creates a simple API client that is used to make thread-safe GET and POST calls.

Results

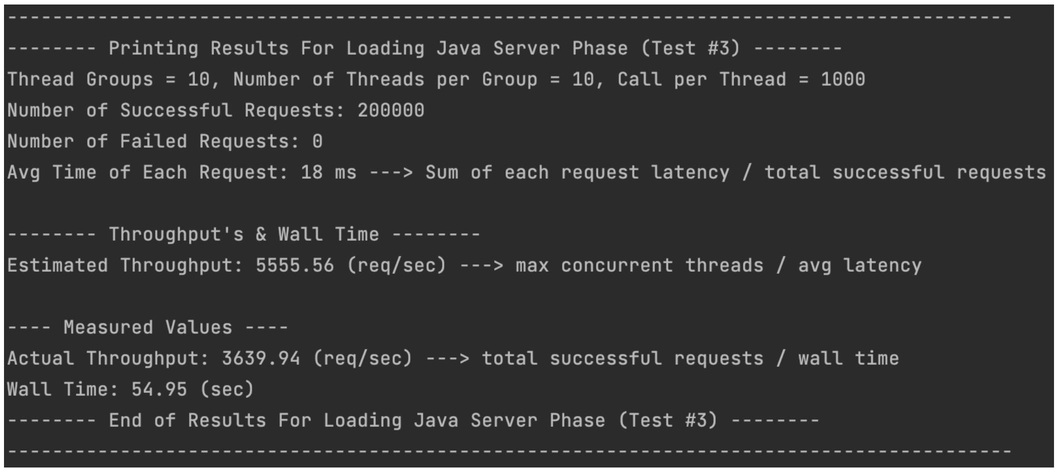
My results are interesting for both part 1 and 2. I struggled to collect results because my home network only has a max upload speed of 11 Mbps. When loading a server and sending thousands of small images, response times become relatively slow for POST requests. Because of this, I had to travel and commute to multiple locations to utilize the home networks of friends with a faster upload speed. However, this led to results not always being consistent. Almost all my tests and results are collected on the same house network with an upload speed of 350+ Mbps, but my Go Server tests with a thread group size of 30 are on a slower network with upload speeds of 150 Mbps. The decrease in network upload speed caused my throughput results for my Go server thread group size 30 to be less than expected. This is why my plots for my Go server become closer horizontal when going from a thread group size of 20 to a size of 30.

Also, I misread the instructions and thought we had to test each thread group size three times. That is why each server has so many images and Excel sheets of different group sizes. For the main plots, I take the average of the three tests for my throughput and wall time plots. For individual thread group result screenshots, I included the best throughput results.

Client Part 1 Results

Through testing, I found a difference between server load tests when comparing the Go and Java servers. Both servers saw increased throughput as more thread groups were called and ran. Overall, the Java server ran faster for all tests.

Java Server - Thread Group Size 10

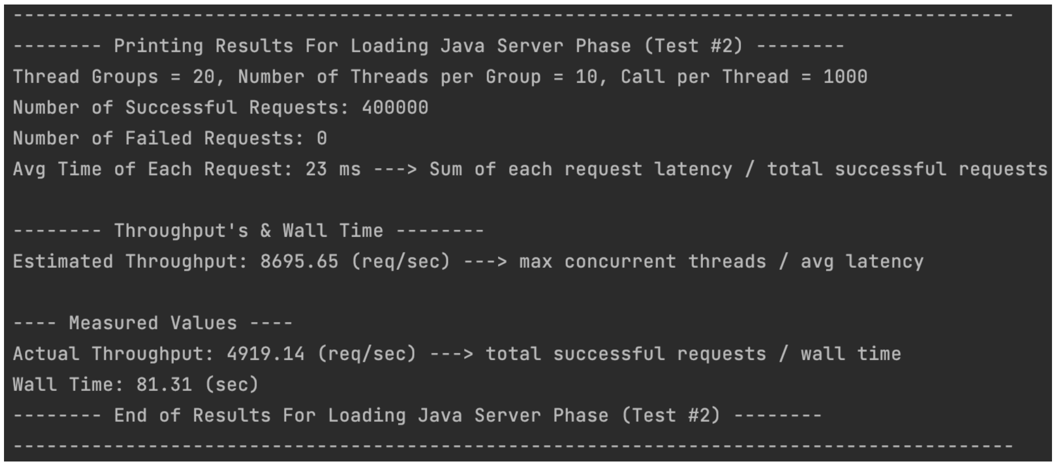


```
-----------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #3) --------
Thread Groups = 10, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 200000
Number of Failed Requests: 0
Avg Time of Each Request: 18 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 5555.56 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 3639.94 (req/sec) ---> total successful requests / wall time
Wall Time: 54.95 (sec)
-------- End of Results For Loading Java Server Phase (Test #3) --------
-----------------------------------------------------------------------------------------
```

Java Server - Thread Group Size 20

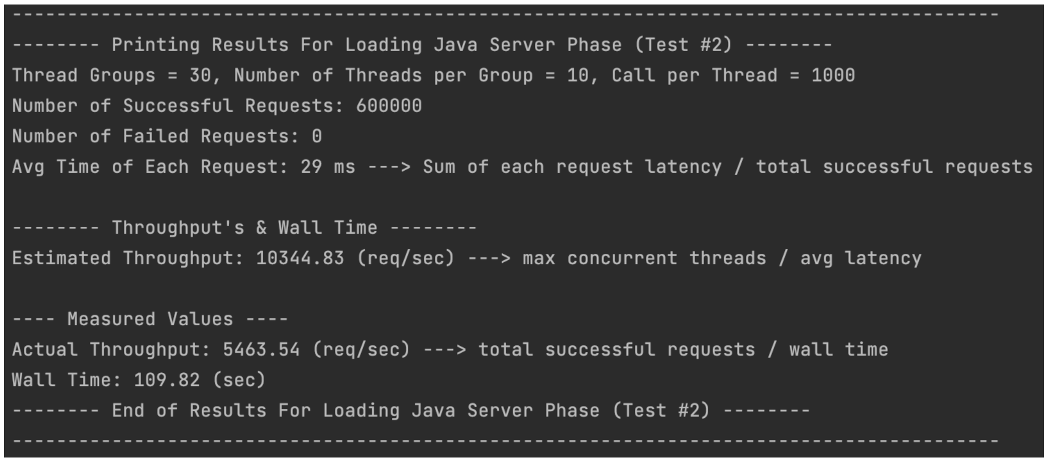


```
-----------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #2) --------
Thread Groups = 20, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 400000
Number of Failed Requests: 0
Avg Time of Each Request: 23 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 8695.65 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 4919.14 (req/sec) ---> total successful requests / wall time
Wall Time: 81.31 (sec)
-------- End of Results For Loading Java Server Phase (Test #2) --------
-----------------------------------------------------------------------------------------
```

Java Server - Thread Group Size 30

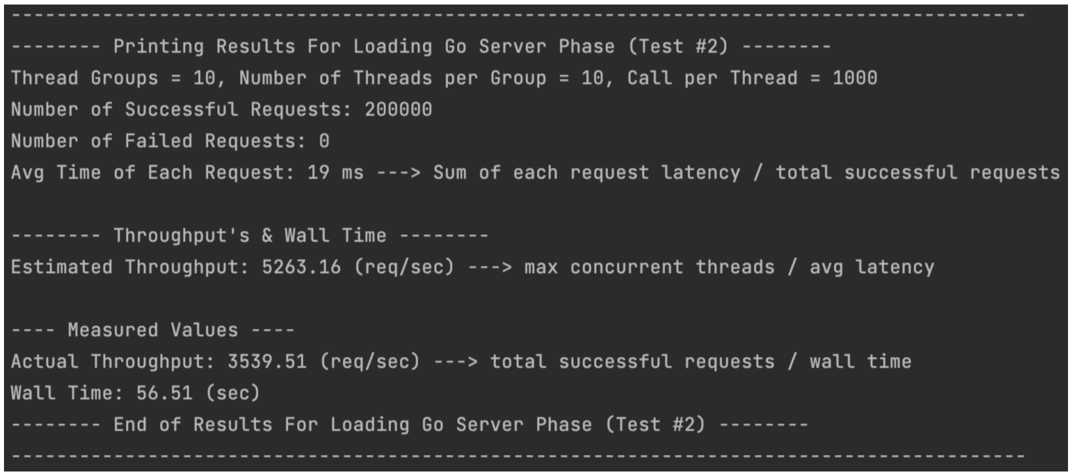


```
-----------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #2) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 600000
Number of Failed Requests: 0
Avg Time of Each Request: 29 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 10344.83 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 5463.54 (req/sec) ---> total successful requests / wall time
Wall Time: 109.82 (sec)
-------- End of Results For Loading Java Server Phase (Test #2) --------
-----------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 10

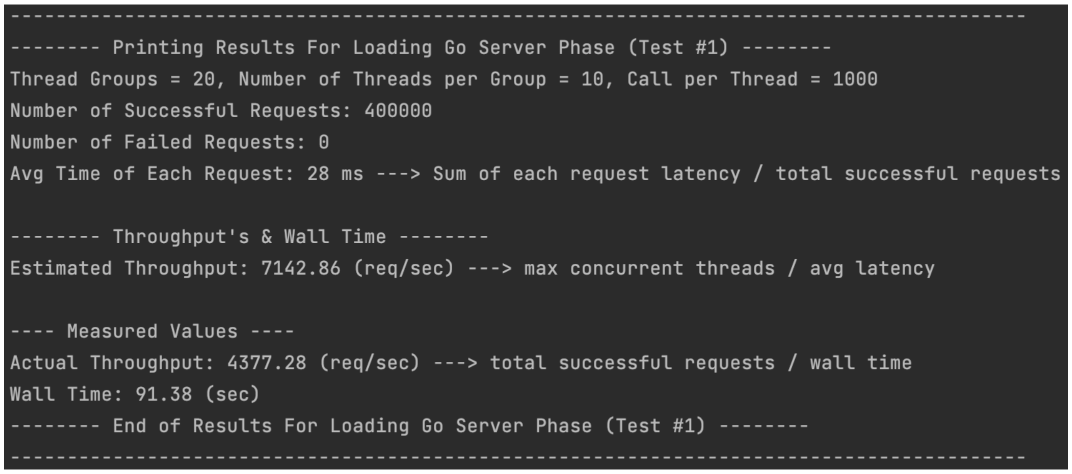


```
----------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #2) --------
Thread Groups = 10, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 200000
Number of Failed Requests: 0
Avg Time of Each Request: 19 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 5263.16 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 3539.51 (req/sec) ---> total successful requests / wall time
Wall Time: 56.51 (sec)
-------- End of Results For Loading Go Server Phase (Test #2) --------
----------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 20

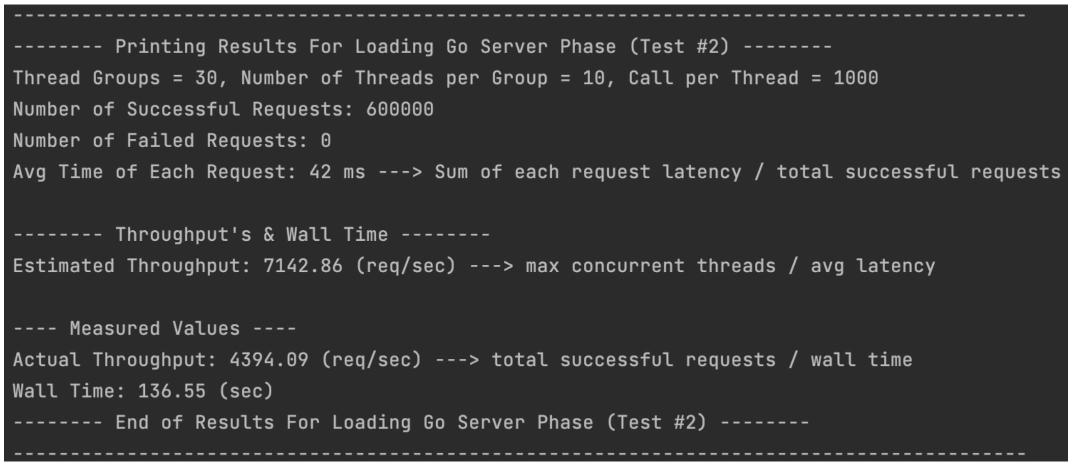


```
----------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #1) --------
Thread Groups = 20, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 400000
Number of Failed Requests: 0
Avg Time of Each Request: 28 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 7142.86 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 4377.28 (req/sec) ---> total successful requests / wall time
Wall Time: 91.38 (sec)
-------- End of Results For Loading Go Server Phase (Test #1) --------
----------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 30



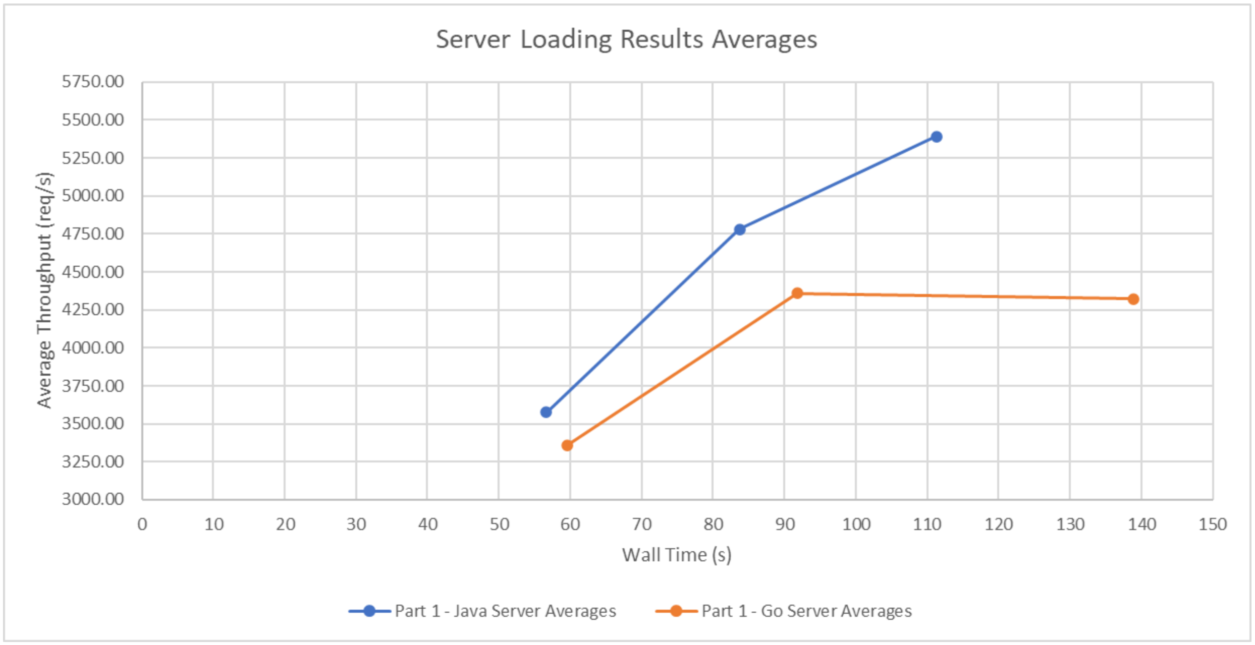
```
----------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #2) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 600000
Number of Failed Requests: 0
Avg Time of Each Request: 42 ms ---> Sum of each request latency / total successful requests

-------- Throughput's & Wall Time --------
Estimated Throughput: 7142.86 (req/sec) ---> max concurrent threads / avg latency

---- Measured Values ----
Actual Throughput: 4394.09 (req/sec) ---> total successful requests / wall time
Wall Time: 136.55 (sec)
-------- End of Results For Loading Go Server Phase (Test #2) --------
----------------------------------------------------------------------------------------
```

Plot of Load Results

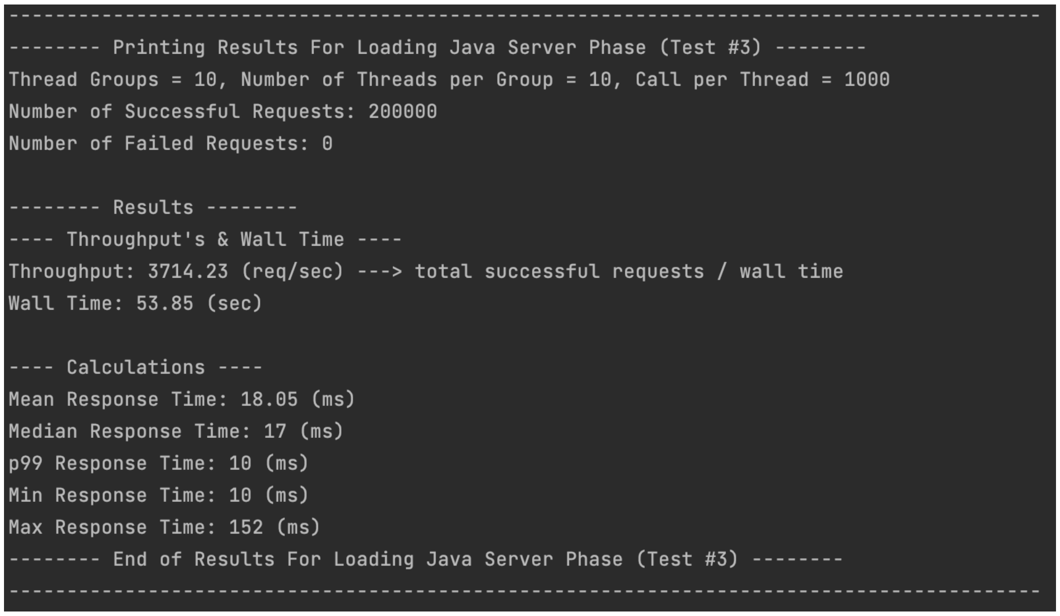
Since I ran three trials for each group size, I took each server's average throughput and wall time. See below.



Client Part 2 Results

Like Client Part 1, the Java server produced faster throughputs and wall times for me. See the results below. All my throughput results between thread group sizes on my servers for Part 1 and Part 2 were within 5% of each other.

Java Server - Thread Group Size 10

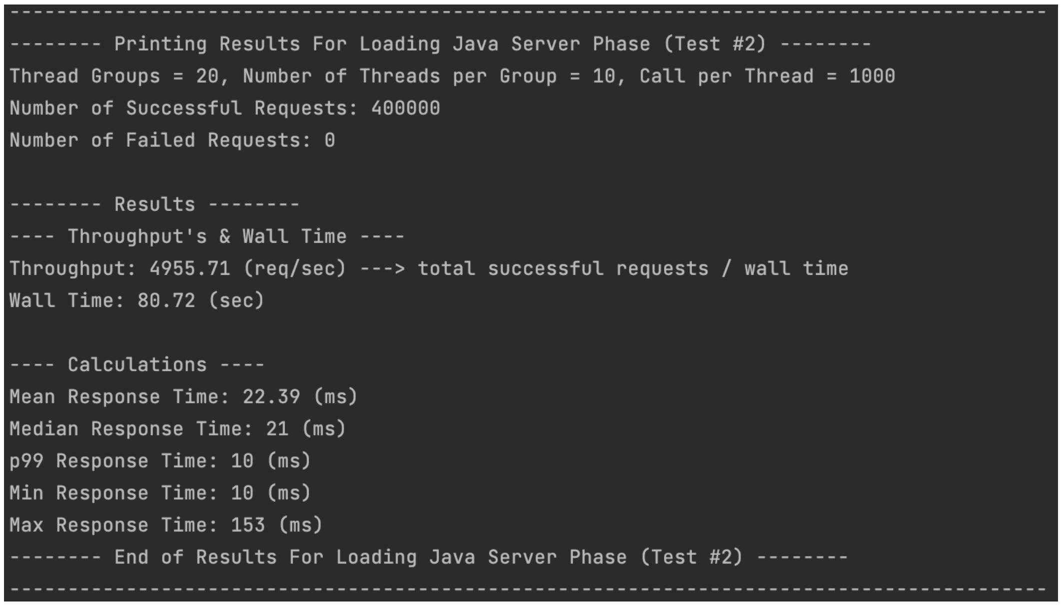


```
------------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #3) --------
Thread Groups = 10, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 200000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 3714.23 (req/sec) ---> total successful requests / wall time
Wall Time: 53.85 (sec)

---- Calculations ----
Mean Response Time: 18.05 (ms)
Median Response Time: 17 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 152 (ms)
-------- End of Results For Loading Java Server Phase (Test #3) --------
------------------------------------------------------------------------------------------
```

Java Server - Thread Group Size 20

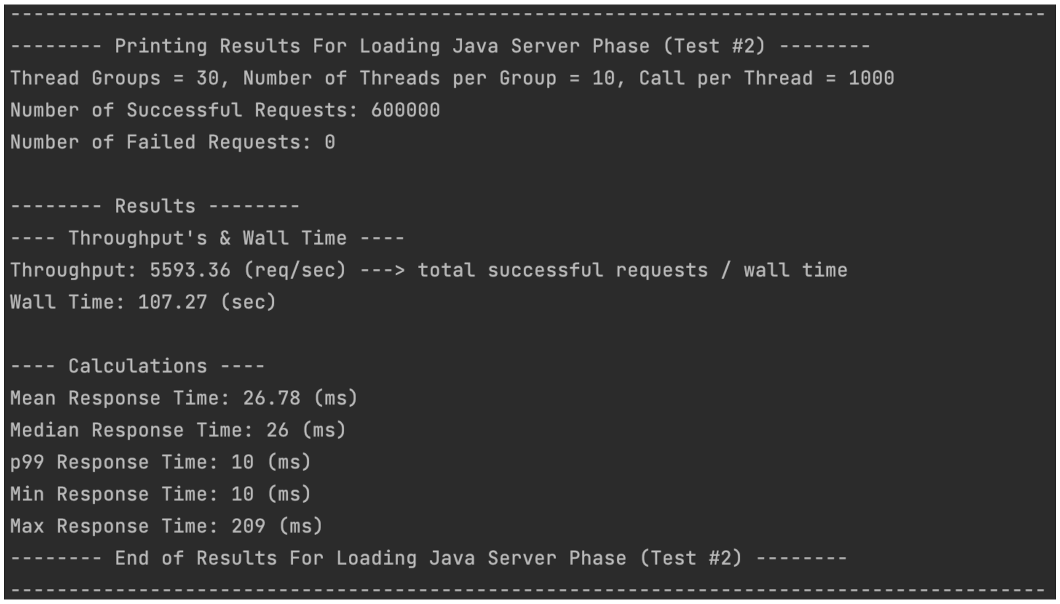


```
------------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #2) --------
Thread Groups = 20, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 400000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 4955.71 (req/sec) ---> total successful requests / wall time
Wall Time: 80.72 (sec)

---- Calculations ----
Mean Response Time: 22.39 (ms)
Median Response Time: 21 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 153 (ms)
-------- End of Results For Loading Java Server Phase (Test #2) --------
------------------------------------------------------------------------------------------
```

Java Server - Thread Group Size 30

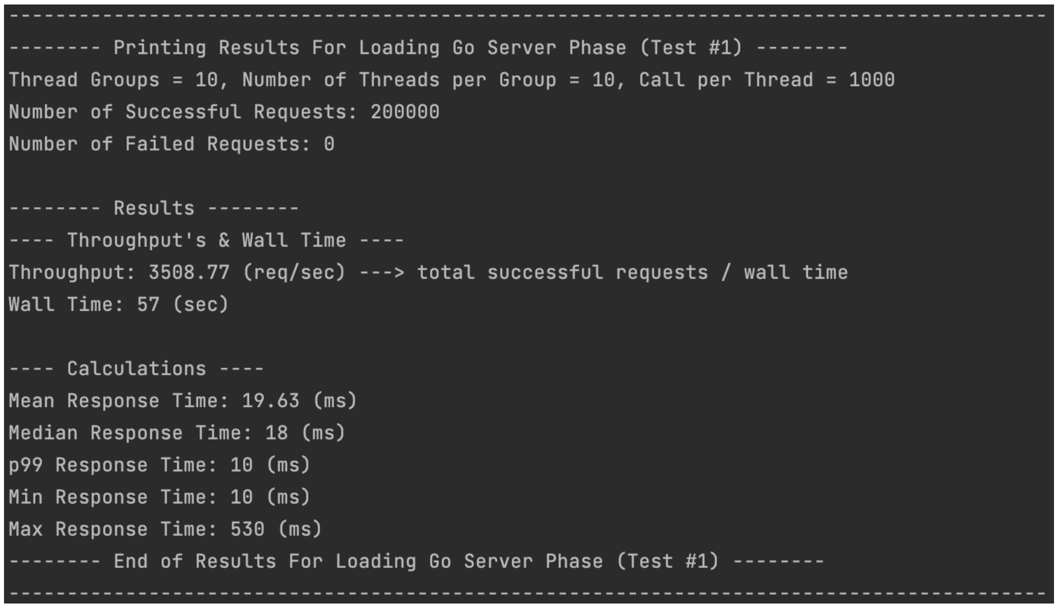


```
------------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase (Test #2) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 600000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 5593.36 (req/sec) ---> total successful requests / wall time
Wall Time: 107.27 (sec)

---- Calculations ----
Mean Response Time: 26.78 (ms)
Median Response Time: 26 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 209 (ms)
-------- End of Results For Loading Java Server Phase (Test #2) --------
------------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 10

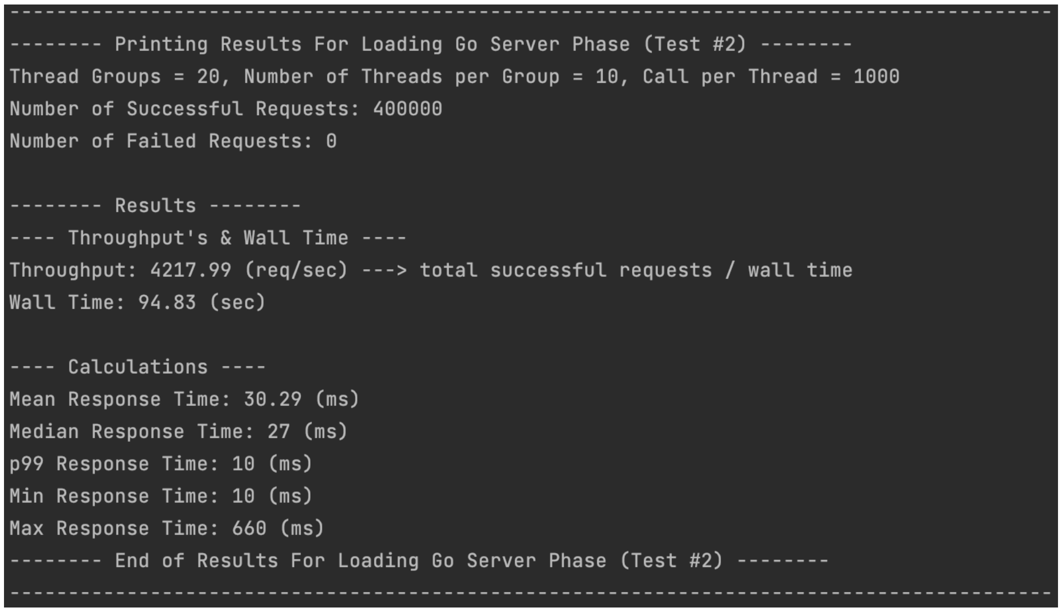


```
------------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #1) --------
Thread Groups = 10, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 200000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 3508.77 (req/sec) ---> total successful requests / wall time
Wall Time: 57 (sec)

---- Calculations ----
Mean Response Time: 19.63 (ms)
Median Response Time: 18 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 530 (ms)
-------- End of Results For Loading Go Server Phase (Test #1) --------
------------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 20

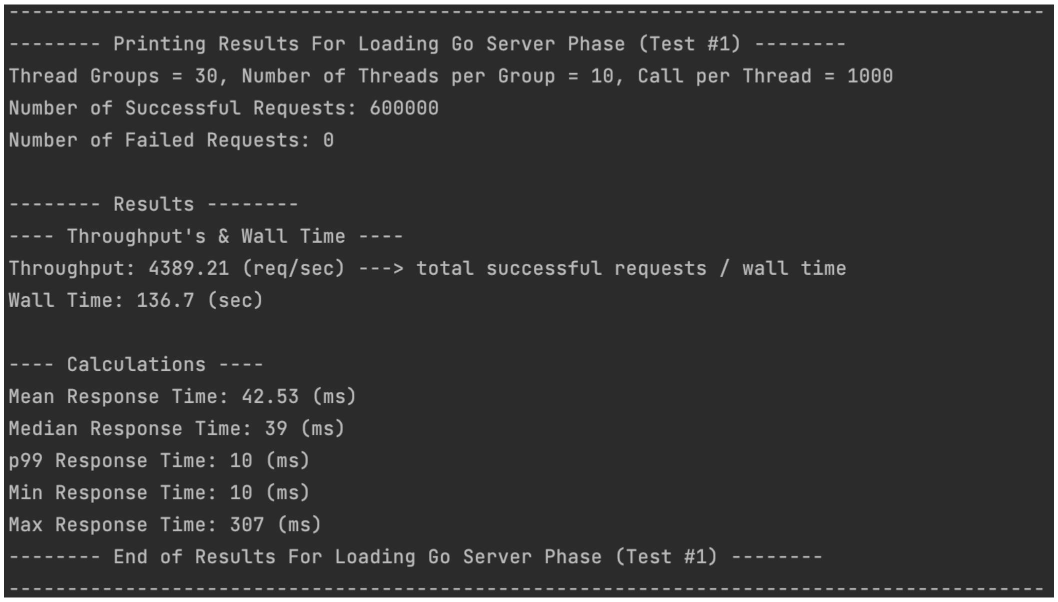


```
------------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #2) --------
Thread Groups = 20, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 400000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 4217.99 (req/sec) ---> total successful requests / wall time
Wall Time: 94.83 (sec)

---- Calculations ----
Mean Response Time: 30.29 (ms)
Median Response Time: 27 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 660 (ms)
-------- End of Results For Loading Go Server Phase (Test #2) --------
------------------------------------------------------------------------------------------
```

Go Server - Thread Group Size 30



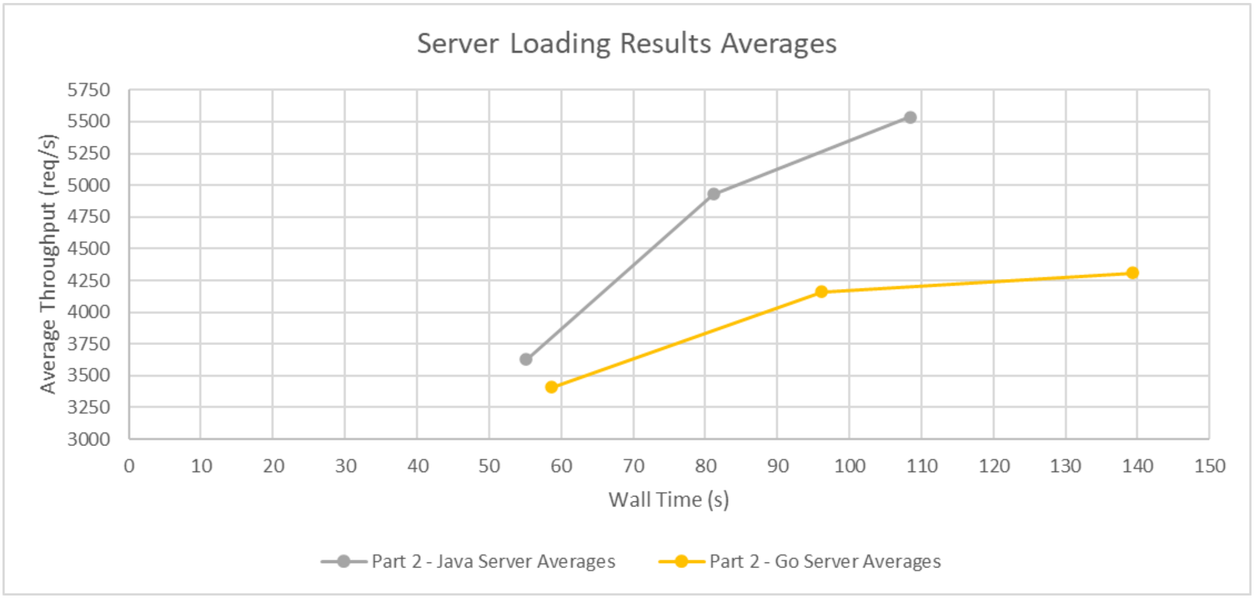
```
-----------------------------------------------------------------------------------------
-------- Printing Results For Loading Go Server Phase (Test #1) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 1000
Number of Successful Requests: 600000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 4389.21 (req/sec) ---> total successful requests / wall time
Wall Time: 136.7 (sec)

---- Calculations ----
Mean Response Time: 42.53 (ms)
Median Response Time: 39 (ms)
p99 Response Time: 10 (ms)
Min Response Time: 10 (ms)
Max Response Time: 307 (ms)
-------- End of Results For Loading Go Server Phase (Test #1) --------
-----------------------------------------------------------------------------------------
```

Plot of Load Results

Since I ran three trials for each group size, I took each server's average throughput and wall time. See below.



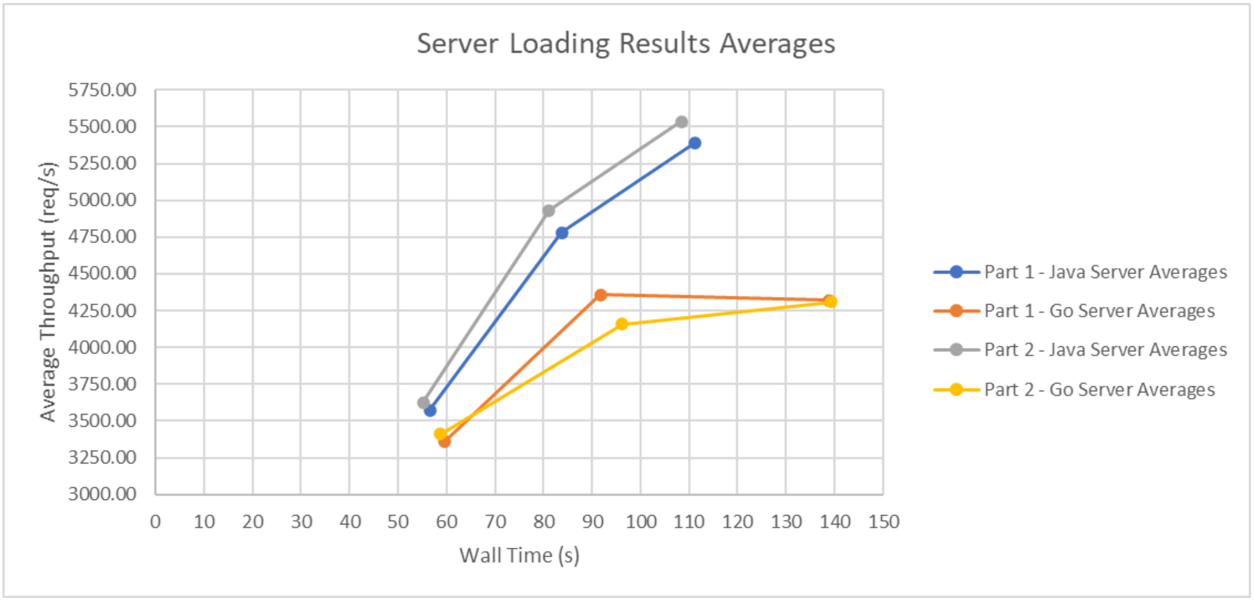
Plots

Plot of Loads on Each Server

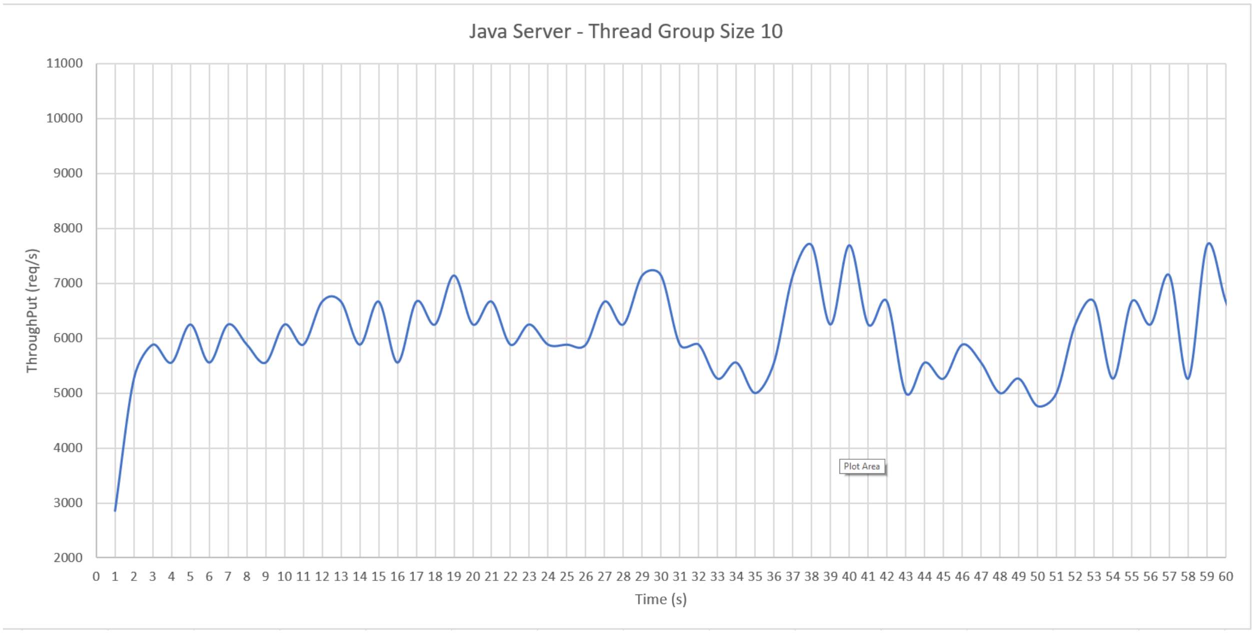
This plot shows the throughput vs. time for each server for the different-sized thread groups. As the throughput increases, so does the thread group size. Each point represents a thread group size.

Since I accidentally recorded three runs for each thread group size, each point represents the average results for that thread group.

The plot below shows that results from Part 1 and Part 2 are very close for each server. However, this plot makes it clear that for my server load results, the Java server runs faster than the Go server.



Plot ThroughPut Over Time


Java Server - Thread Group Size 10
ThroughPut (req/s)
Time (s)
Plot Area