Student Name: Yi-Cheng Lee

1. GitHub repository

https://github.com/ycleo/cs6650-hw/tree/main/hw1

2. Client Design

Overview:
The client code is designed to simulate a load on a server by generating multiple threads that send POST and GET requests to a specified server endpoint. The client aims to measure and analyze the performance of the server by capturing latency records, calculating throughput, and exporting data for further analysis.
Major Classes and Packages:

- **Main Class (Main.java):**
    - The main class orchestrates the entire process, handling command-line arguments, initializing parameters, and managing the flow of the load testing.
    - It spawns multiple instances of the **AlbumThread** class to simulate concurrent requests.
- **AlbumThread Class (AlbumThread.java):**
    - Represents a thread responsible for sending POST and GET requests to the server.
    - Contains methods to perform HTTP POST and GET operations, record latency, and manage the lifecycle of each thread.
    - Utilizes the Swagger-generated **DefaultApi** class to interact with the server API.
- **LatencyRecord Class (LatencyRecord.java):**
    - A simple data class representing a record of latency for a single request.
    - Stores information such as start time, request type (POST or GET), latency duration, and response code.

Relationships:

- **Main - AlbumThread Relationship:**
    - The **Main** class creates and manages instances of **AlbumThread** based on the specified number of thread groups and threads per group.
    - Utilizes **CountDownLatch** to synchronize the start and completion of threads.
- **AlbumThread - DefaultApi Relationship:**
    - The **AlbumThread** class utilizes the **DefaultApi** class generated by Swagger to make HTTP requests to the server API endpoint.
- **Main - Throughput Calculation Relationship:**
    - After all threads complete, the **Main** class aggregates latency records from all threads.
    - The class then calculates various performance statistics, including mean, median, p99, min, and max response times.
    - It exports throughput data to a CSV file, capturing the number of completed requests per second.

Execution Flow:

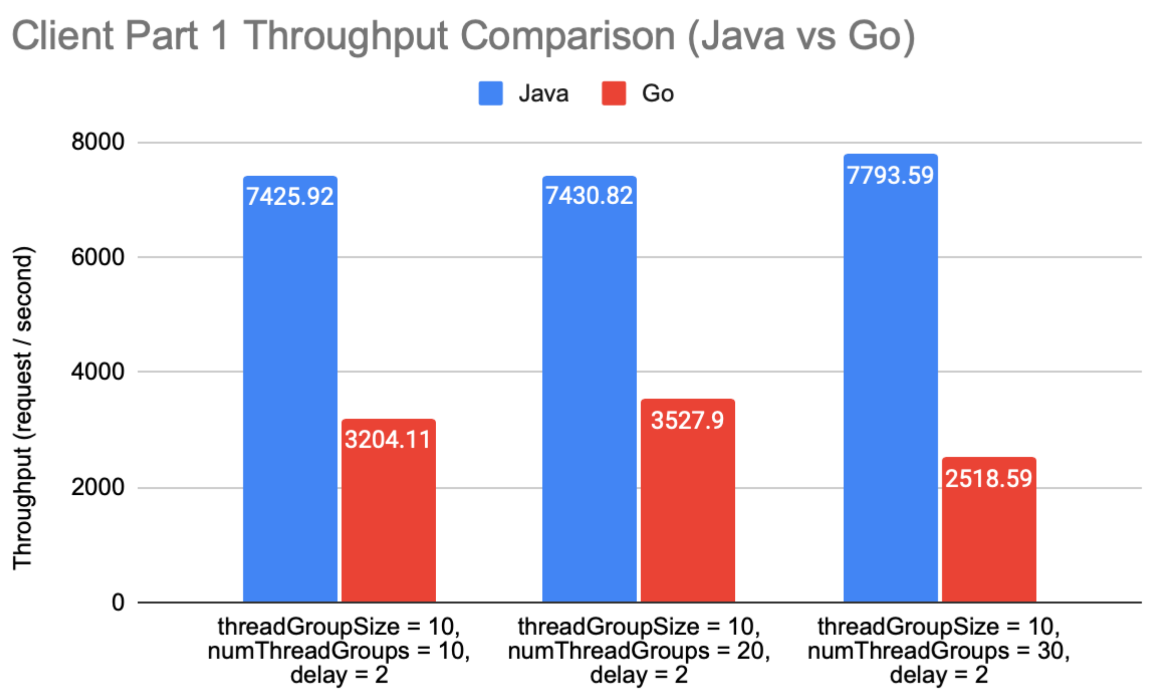
- **Initialization:**

    - Command-line arguments are parsed to determine the test configuration parameters (e.g., thread group size, number of thread groups, delay, IP address).
- **Thread Spawning:**
    - Initial 10 threads are spawned to simulate a warm-up phase, ensuring the server is ready for the load test.
- **Load Testing:**
    - Thread groups are spawned, each executing a specified number of threads.
    - Each thread in a group performs a defined number of POST and GET requests to the server endpoint.
- **Latency Recording:**
    - Latency records are collected during the execution of each thread.
- **Throughput Calculation and Export:**
    - After all threads complete, the main class calculates performance statistics.
    - Throughput data is exported to a CSV file for further analysis.

Conclusion:
The client code provides a modular and scalable approach to load testing, allowing for easy customization of test configurations. It effectively utilizes multi-threading to simulate concurrent requests and captures detailed latency information, enabling in-depth analysis of server performance. The generated throughput data in CSV format facilitates external visualization tools for charting and graphing.

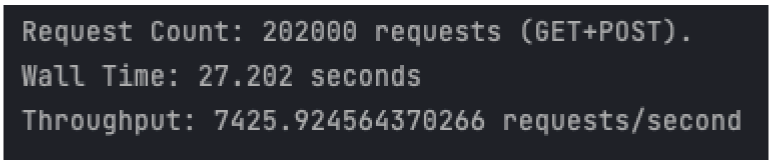
3. Client (Part 1)

Result:



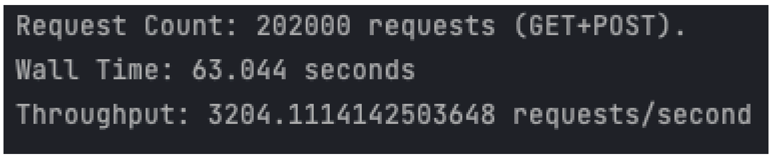
Screenshots:

- threadGroupSize = 10, numThreadGroups = 10, delay = 2
    - Java



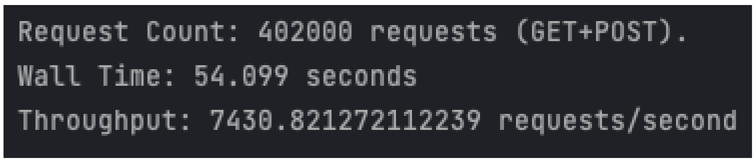
```
Request Count: 202000 requests (GET+POST).
Wall Time: 27.202 seconds
Throughput: 7425.924564370266 requests/second
```

    - Go



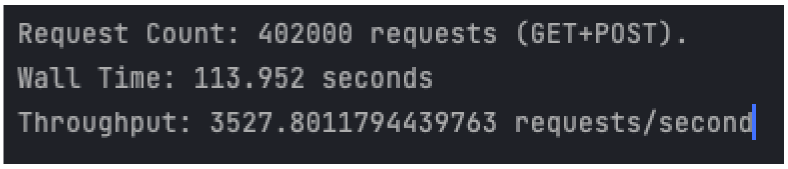
```
Request Count: 202000 requests (GET+POST).
Wall Time: 63.044 seconds
Throughput: 3204.1114142503648 requests/second
```

- threadGroupSize = 10, numThreadGroups = 20, delay = 2
    - Java



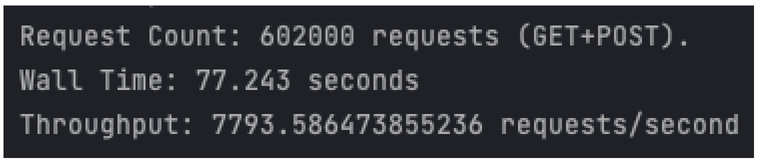
```
Request Count: 402000 requests (GET+POST).
Wall Time: 54.099 seconds
Throughput: 7430.821272112239 requests/second
```

    - Go



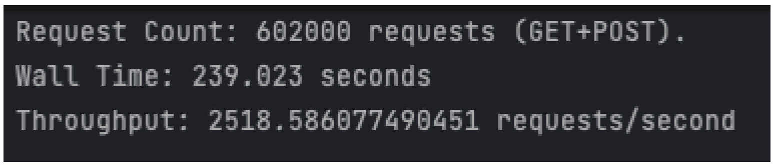
```
Request Count: 402000 requests (GET+POST).
Wall Time: 113.952 seconds
Throughput: 3527.8011794439763 requests/second
```

- threadGroupSize = 10, numThreadGroups = 30, delay = 2
    - Java



```
Request Count: 602000 requests (GET+POST).
Wall Time: 77.243 seconds
Throughput: 7793.586473855236 requests/second
```

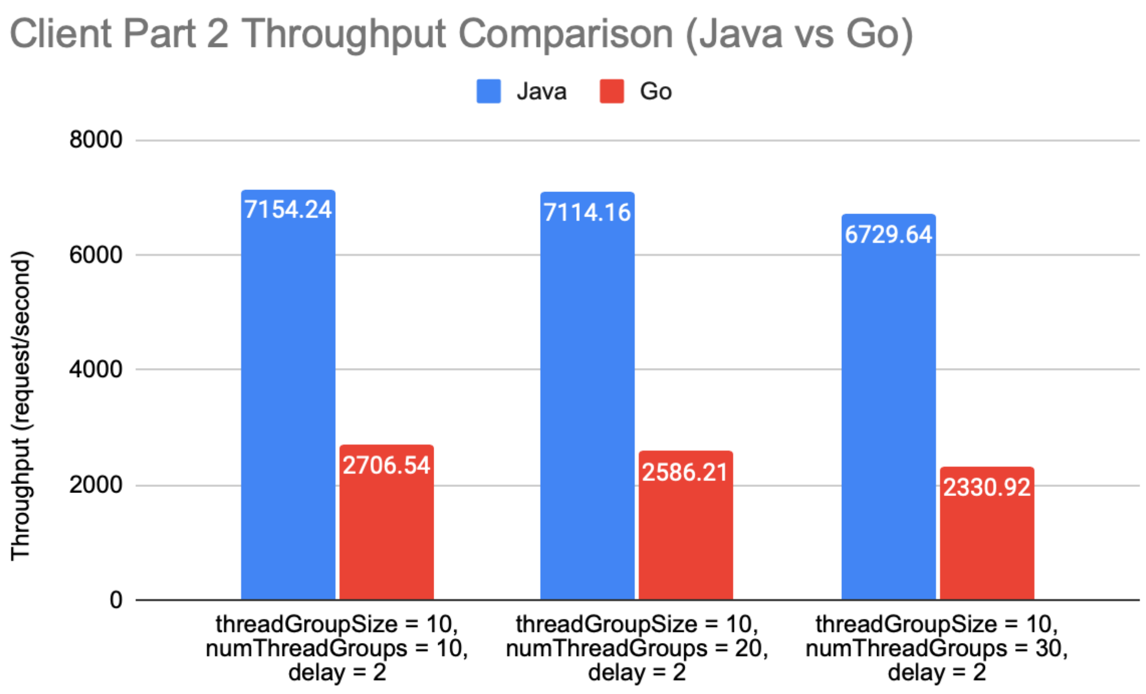
    - Go



```
Request Count: 602000 requests (GET+POST).
Wall Time: 239.023 seconds
Throughput: 2518.586077490451 requests/second
```

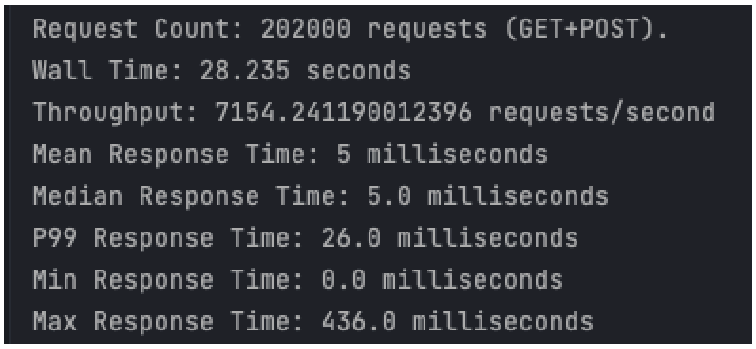
4. Client (Part 2)

Result:



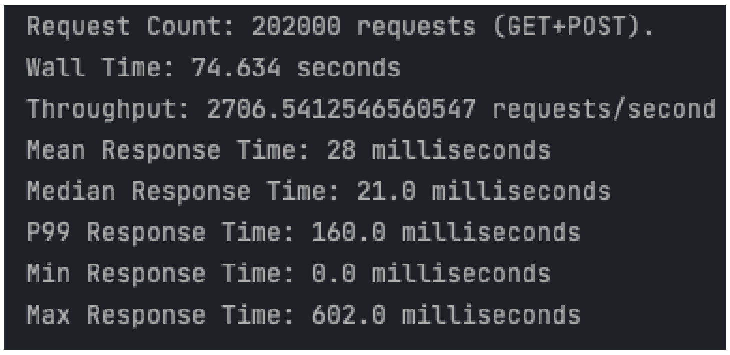
Screenshots:

- threadGroupSize = 10, numThreadGroups = 10, delay = 2
    - Java



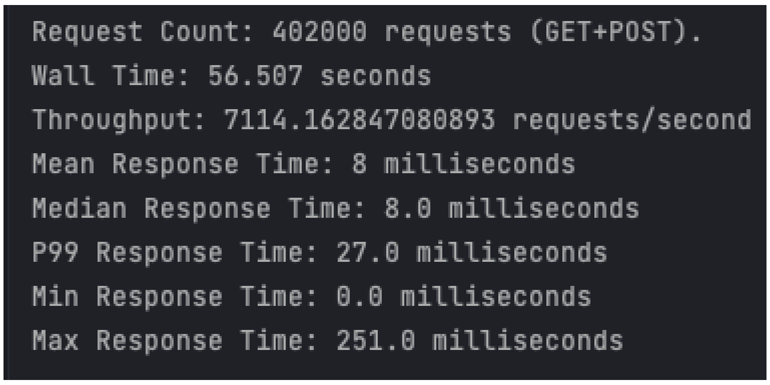
```
Request Count: 202000 requests (GET+POST).
Wall Time: 28.235 seconds
Throughput: 7154.241190012396 requests/second
Mean Response Time: 5 milliseconds
Median Response Time: 5.0 milliseconds
P99 Response Time: 26.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 436.0 milliseconds
```

    - Go



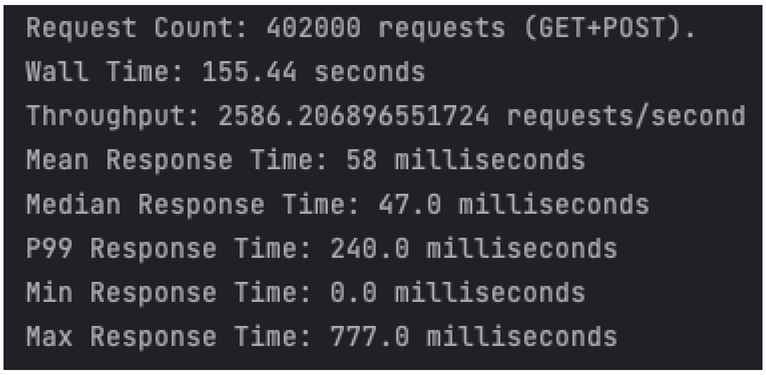
```
Request Count: 202000 requests (GET+POST).
Wall Time: 74.634 seconds
Throughput: 2706.5412546560547 requests/second
Mean Response Time: 28 milliseconds
Median Response Time: 21.0 milliseconds
P99 Response Time: 160.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 602.0 milliseconds
```

- threadGroupSize = 10, numThreadGroups = 20, delay = 2
    - Java



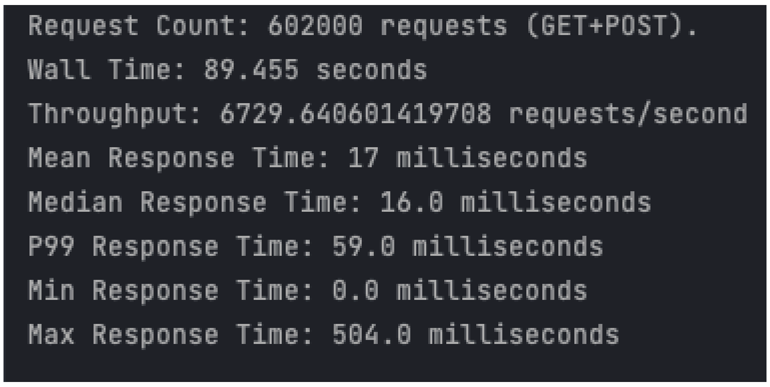
```
Request Count: 402000 requests (GET+POST).
Wall Time: 56.507 seconds
Throughput: 7114.162847080893 requests/second
Mean Response Time: 8 milliseconds
Median Response Time: 8.0 milliseconds
P99 Response Time: 27.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 251.0 milliseconds
```

    - Go



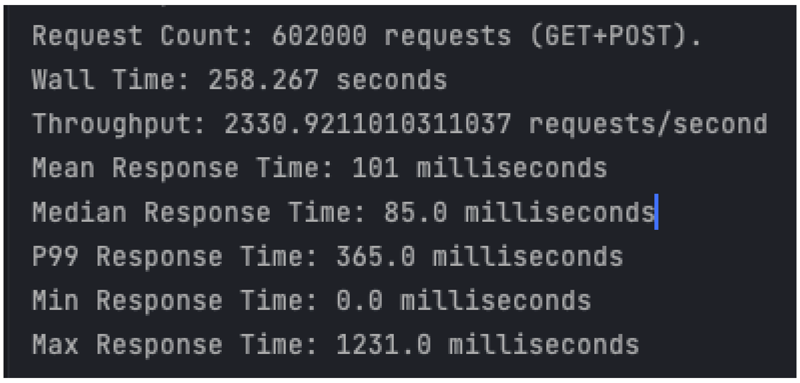
```
Request Count: 402000 requests (GET+POST).
Wall Time: 155.44 seconds
Throughput: 2586.206896551724 requests/second
Mean Response Time: 58 milliseconds
Median Response Time: 47.0 milliseconds
P99 Response Time: 240.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 777.0 milliseconds
```

- threadGroupSize = 10, numThreadGroups = 30, delay = 2
    - Java



```
Request Count: 602000 requests (GET+POST).
Wall Time: 89.455 seconds
Throughput: 6729.640601419708 requests/second
Mean Response Time: 17 milliseconds
Median Response Time: 16.0 milliseconds
P99 Response Time: 59.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 504.0 milliseconds
```

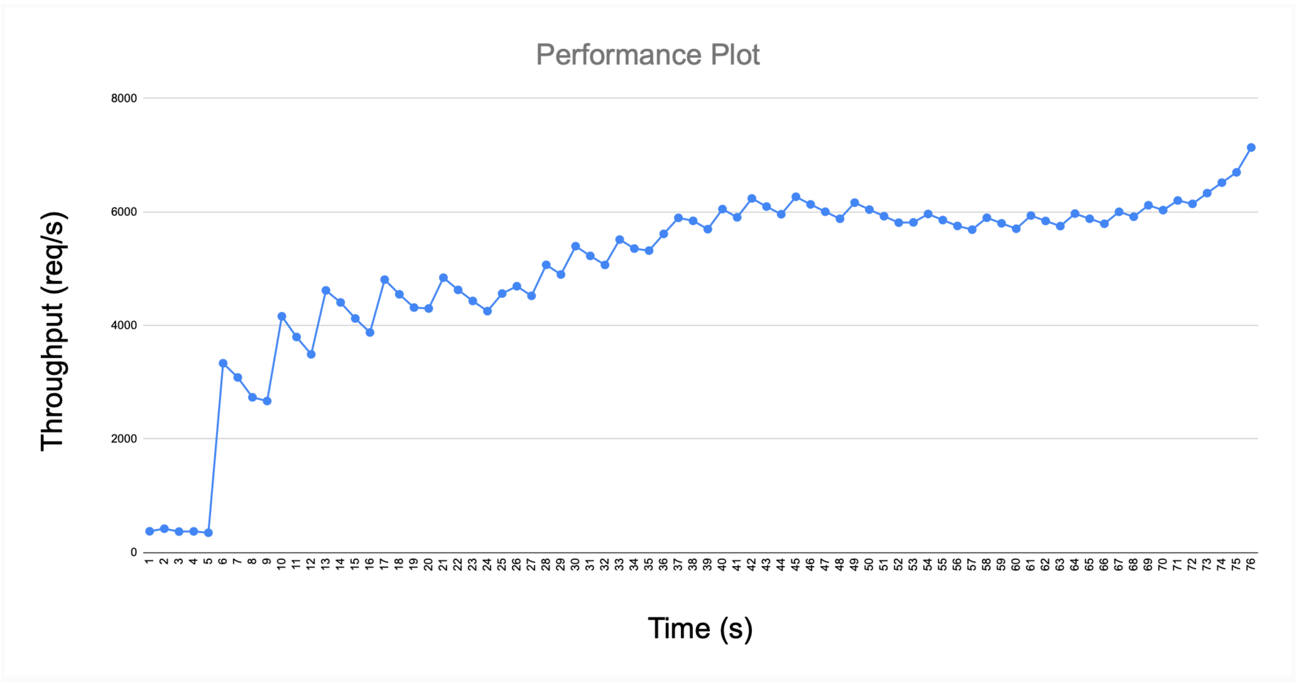
    - Go



```
Request Count: 602000 requests (GET+POST).
Wall Time: 258.267 seconds
Throughput: 2330.9211010311037 requests/second
Mean Response Time: 101 milliseconds
Median Response Time: 85.0 milliseconds
P99 Response Time: 365.0 milliseconds
Min Response Time: 0.0 milliseconds
Max Response Time: 1231.0 milliseconds
```

5. Performance Plot

Java Server (threadGroupSize = 10, numThreadGroups = 30, delay = 2)


Performance Plot
Throughput (req/s)
Time (s)