CS6650 Assignment 1 Report

Shengdi Wang - wang.shengd@northeastern.edu

1. GitHub Repo URL

https://github.com/leowang396/distributed-music-service

2. Client Design

The client program is designed to evaluate the performance of the music-service server under stress. Though the design is different between part 1 and part 2, part 2 design builds on the part 1 code by modifying existing classes and adding a new class.

In part 1, the design consists of 2 classes, where LoadTestOrchestrator contains the main method, and ConcurrentLoadTestRequestDispatcher contains **static utility methods** which are called by LoadTestOrchestrator.

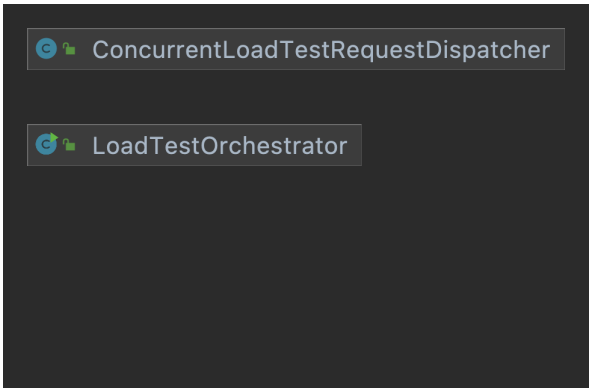


Figure 1: UML Diagram for Client Part 1

1. **LoadTestOrchestrator**:
    a. This class contains only the *main(String[] args)* method, which orchestrates the load test based on command-line inputs which define the number of concurrent request threads in a thread group (stored as *threadGroupSize*), the number of such groups (stored as *numThreadGroups*), a delay between groups (stored as *delay*), and the target server's IP address (stored as *iPAddress*).
    b. It calls the static utility method *dispatchConcurrentPostAndGetRequests(int numThreadGroups, int threadGroupSize, int apiCallCount, String serverAddress, long delay)* in *ConcurrentLoadTestRequestDispatcher* class to issue concurrent requests to the server.
    c. An initial batch of API calls is made as a "startup phase" using predetermined parameters (*threadGroupSize*=10, *numThreadGroups*=1, *delay*=0). Subsequently, the main load test is initiated based on command line input.
    d. At the conclusion of the test, it reports the wall time and throughput by printing to standard output.
2. **ConcurrentLoadTestRequestDispatcher**:

a. This class dispatches concurrent requests to the music service using the *DefaultApi* class, which is the Swagger-generated client, and user-specified parameters such as *threadGroupSize, numThreadGroups, apiCallCount, delay.*
b. Since no states need to be maintained across test batches, I made the methods and a few member variables static to reduce memory footprint of creating objects.
c. The 2 requests are adding a new album (POST request) and retrieving an album by its key (GET request). If a request fails with a specific range of HTTP status codes, it is retried up to a limit (configured via the static variable *MAX_RETRIES*, which is set to 5 for now).

In part 2, the design is modified as follows, where MetricReporter is a new class that contains the static utility method for reporting latency metrics of POST/GET requests.

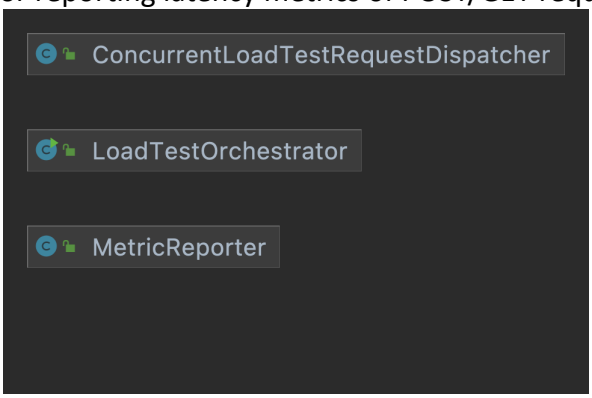
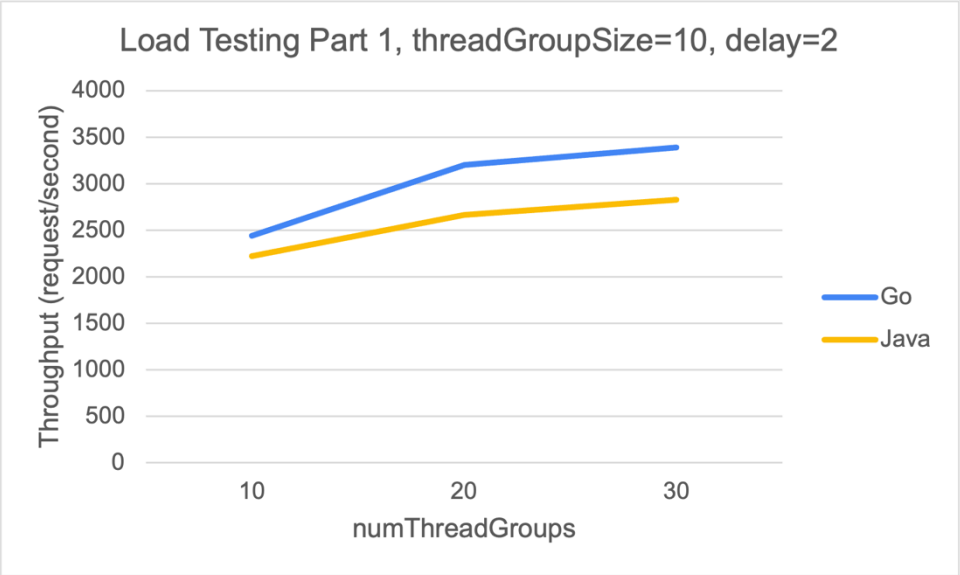


Figure 2: UML Diagram for Client Part 2

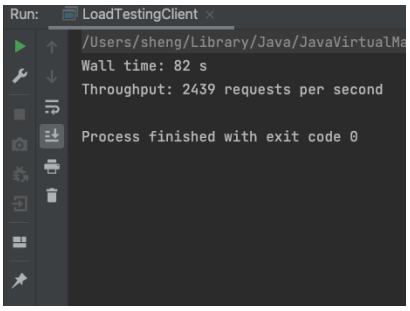
1. **LoadTestOrchestrator**:
    a. While this class still contains only the *main(String[] args)* method, it now uses a thread-safe list to store measurement data for individual requests. It then generates a metrics CSV file name based on command line input and calls the static method in the new *MetricReporter* class to report on the measurement data.
2. **ConcurrentLoadTestRequestDispatcher**:
    a. The static method now measures the start time, latency, and response status code of the individual requests. It also takes an additional parameter *metrics*, which is a thread-safe list data structure, and stores the measurement data within it.
3. **MetricReporter**:
    a. This is the new class added in part 2, and its main method is the static method *reportInMemoryList(String outputName, List<String[]> metrics)*. It also contains a private helper method *computeMeanOfList(List<Integer> list)*.
    b. It takes a list of String arrays, computes the required static like mean, median, 99th percentile, min and max latencies for both POST and GET requests, and writes to an output CSV using the user-specific name string.

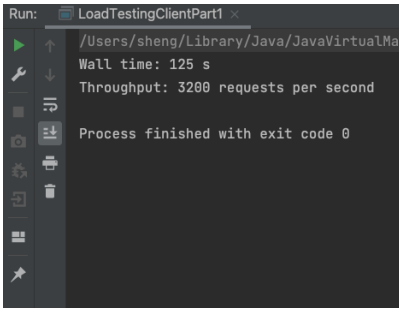
3. Client (Part 1) Measurement

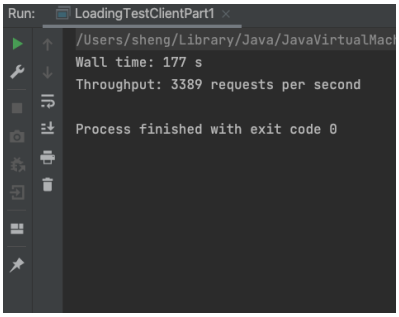
Part 1 Throughput Plot



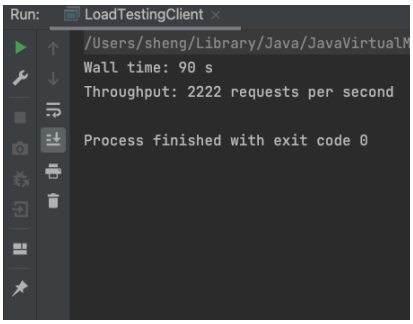
Go Server Screenshots (Part 1)

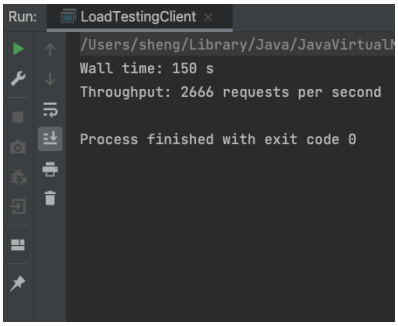


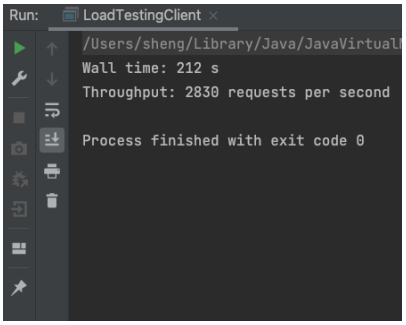




Java Server Screenshots (Part 1)

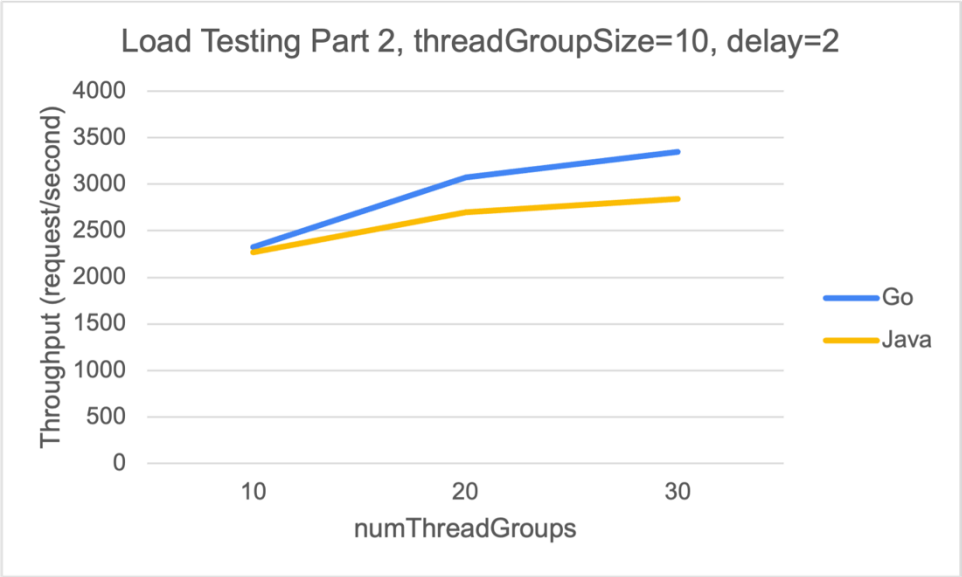




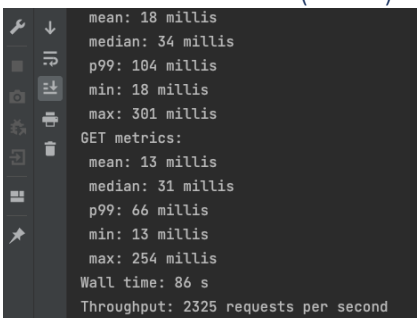


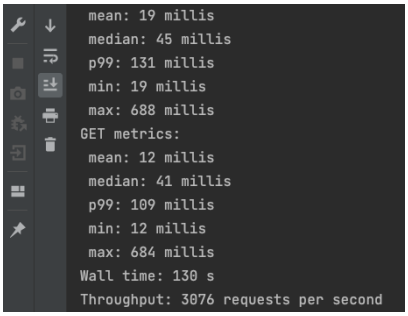
4. Client (Part 2) Measurement

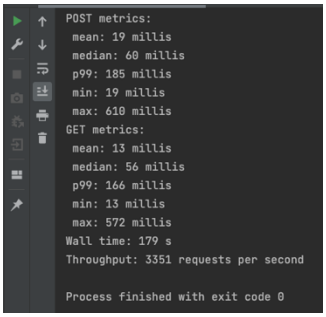
Part 2 Throughput Plot



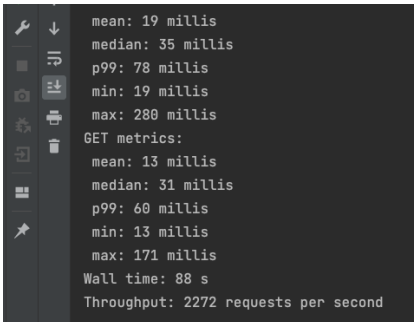
Go Server Screenshots (Part 2)

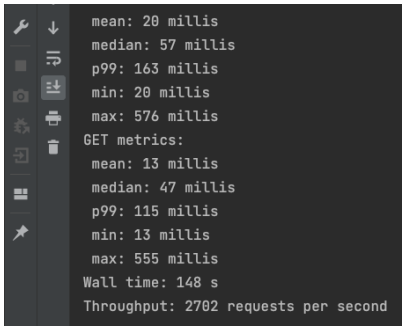


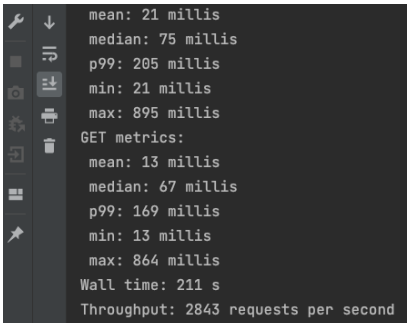




Java Server Screenshots (Part 2)







5. Throughput Over Time for Go Server, numThreadGroups=30