Github repository url :https://github.com/qilin7000/CS6650

Client Design Report: Part 1 & Part 2

1. Introduction

This report provides an overview of the design and implementation of both Client1 and Client2, which are responsible for generating and sending 200,000 POST requests to a remote server. Client1 focuses on efficient multithreading to maximize throughput, while Client2 extends the functionality by recording latency metrics and logging performance data for analysis.

Both clients implement a multi-phase workload distribution strategy to efficiently utilize system resources and achieve high throughput. Additionally, the performance of the clients is evaluated using Little's Law predictions and real execution metrics.

2. System Architecture

Both Client1 and Client2 are structured similarly, following a producer-consumer model where a dedicated event generator produces lift ride events, and multiple worker threads consume and process them by making POST requests to the server.

The major components of the clients are:

1.Client (Main Entry Point)

- Initializes and starts the test.
- Conducts a preliminary single-threaded latency test to estimate request delays.
- Generates ski lift ride events.
- Launches multiple worker threads to send POST requests concurrently.
- Calculates performance statistics after execution.

2.PhaseThread (Worker Thread Class)

- Each thread pulls events from a shared queue and sends HTTP POST requests.
- Implements retry logic (up to 5 attempts) in case of failures.

Client2 additionally records request latencies and logs them to a CSV file.

3.EventGenerator (Producer Thread)

- Generates 200,000 lift ride events based on predefined constraints.
- Ensures that the worker threads always have available events to process.

4.LiftRideEvent (Data Model Class)

- Represents a single lift ride event, containing skier ID, lift ID, resort ID, season ID, and timestamp.

For Client2, additional logic is included for latency measurement and CSV logging. This allows for detailed post-processing of response times and performance trends.

3. Execution Flow

Shared Flow for Client1 and Client2:

1.Single-threaded latency test (testLatency)

- Sends 10,000 requests sequentially to estimate the average request latency.
- This helps predict expected throughput using Little's Law.

2.Event Generation

- A separate thread generates 200,000 lift ride events and stores them in a shared queue.

3.Multi-Phase Concurrent Execution

- Phase 1: 32 threads send 1,000 requests each (Total: 32,000 requests).
- Phase 2: The remaining 168,000 requests are dynamically distributed across worker threads.

4.Post-Processing & Output

- Client1 prints successful and failed request counts, total execution time, and throughput.
- Client2 records additional latency statistics, writes them to a CSV file, and computes metrics such as p99 latency, median latency, and min/max latencies.

4. Little's Law Throughput Prediction

From the single-threaded latency test, we measure the average request latency is 0.016867 s.

Therefore, based on the formula: Throughput = Number of threads / Average Latency, we can predict the through put with the latency we got:

Phase 1 expected throughput = 32 / 0.016867 = 1897.19571
Phase 2 expected throughput = 168 / 0.016867 = 9960.27746
Expected throughput = (1897.19571 + 9960.27746) / 2 = 5928.73695

```
[ec2-user@ip-172-31-17-102 ~]$ java -jar client_part1-1.0-SNAPSHOT.jar -nl 40 -nr 10 -ns 10000 -nt 256 -serv
er http://ec2-54-214-18-127.us-west-2.compute.amazonaws.com:8080/my_lab2-1.0-SNAPSHOT
Single-threaded average request latency: 16.867 ms
Total request: 10000
```

The actual throughput is showed as below:

```
Phase one threads: 32
Phase two threads: 168
All requests completed. Total execution time: 33827 ms
Successful requests: 200000
Failed requests: 0
Throughput: 5912.436810831585
```

We can see that the difference is within 5% of the Little Law's prediction.

5.Conclusion

The Client1 and Client2 implementations effectively test the performance of the server by generating high-concurrency HTTP POST requests. Client1 focuses on raw throughput, while Client2 provides a deeper analysis of system behavior by recording latency metrics and logging results to a CSV file.

The results confirm that Little's Law can provide a reasonable throughput estimate, but real-world performance can be affected by network conditions, server load, and error handling overhead. The detailed latency analysis in Client2 helps identify performance issues and optimize request handling.

Client1 screenshot:


```
[ec2-user@ip-172-31-17-102 ~]$ java -jar client_part1-1.0-SNAPSHOT.jar -nl 40 -nr 10 -ns 10000 -nt 256 -serv
er http://ec2-54-214-18-127.us-west-2.compute.amazonaws.com:8080/my_lab2-1.0-SNAPSHOT
Single-threaded average request latency: 16.867 ms
Total request: 10000
```

```
Phase one threads: 32
Phase two threads: 168
All requests completed. Total execution time: 33827 ms
Successful requests: 200000
Failed requests: 0
Throughput: 5912.436810831585
```


Client screenshot:

```
[ec2-user@ip-172-31-17-102 ~]$ java -jar client_part2-1.0-SNAPSHOT.jar -nl 40 -nr 10 -ns 10000 -nt 256 -server http://ec2-54-214-18-127.us-west-2.compute.amazonaws.com:8080/my_lab2_war
CSV log written to: request_log.csv
Throughput: 4376.84648210964 requests/sec
Min latency: 0 ms
Max latency: 772 ms
Mean latency: 6.96026 ms
Median latency: 5 ms
P99 latency: 55 ms
```

Server screenshot: