**Assignment 1 - Upic**

**Sorry for late submission since I requested a deadline extension from Prof Ian due to personal emergency.**

CS6650, Fall 2022
Xiao Lan

**Github Repository URL**:
https://github.com/elioxiaolan/DistributedSystemsAssignment1

**Client Design**:

**Client Part 1**:

In Client Part 1, I created three classes in the **Model** file at first:

The first one is **LiftRideEvent** which represents a random skier lift ride event. The fields of this class are resortId, seasonId, dayId, skierId, liftId, and time. This class also contains helper methods such as getResortId, getSeasonId, getDayId, getSkierId, getLiftId, getTime.

The second one is **EventGenerator** which represents the generator of random skier life ride event. It can generate a queue of lift ride event based on the number of total request and given parameters.

The third one is **PostProcessor** which can send **POST** requests based on the input lift ride events and calculate the number of successful requests and the failed requests.

Then I created two classes which monitor single thread client and multithread client:

**SingleThreadClient**:

This class monitors a single thread client which processes **1000 POST** requests by using **LiftRideEvent**, **EventGenerator**, and **PostProcessor**, and then calculate number of successful requests sent, number of unsuccessful requests (should be 0), the total run time (wall time) for all phases to complete, and the total throughput in requests per second.

**MultiThreadClient**:

This class monitors a multithread client which processes **20000 POST** requests by using **LiftRideEvent**, **EventGenerator**, and **PostProcessor**, and then calculate number of successful requests sent, number of unsuccessful requests (should be 0), the total run time (wall time) for all phases to complete, and the total throughput in requests per second.

**Client Part 2**:

In Client Part 2, I use the most part of Client Part 1 and make some modifications:

In Model file, I add a new class called **StatisticsGenerator** which can calculate the **mean response time**, the **median response time**, the **p99 (99th percentile) response time**, the **max response time**, and **the min response time**.

Then I also make some modifications on **SingleThreadClient** and **MultiThreadClient**. I add a dependency on pom.xml so that I can use **CSVWriter**. At the end of SingleThreadClient and MultiThreadClient, I utilize CSVWriter to generate a CSV file to record key statistics of this client.

**Little’s Law Prediction**:

Little’s law is universal formula for any system where a queue is present. From a bank to a distributed system. It states:

“The long-term average number of customers in a stable system N is equal to the long-term average effective arrival rate, λ, multiplied by the average time a customer spends in the system, W; or expressed algebraically: **N = λW**.”

- Arrival Rate: The rate at which the customers are entering the system is known as arrival rate.
- Exit Rate: The rate at which the customers are leaving the system is known as exit rate.

Little’s Law assumes a stable system so the arrival rate and exit rate are identical.

We can use Little’s Law in distributed systems to relate the total number of users/requests, server throughput & the average request latency.

Throughput is number of requests processed per unit time. It can be used as the exit rate = arrival rate (λ). Response time – average response time is amount of time a request spends in the System (W).

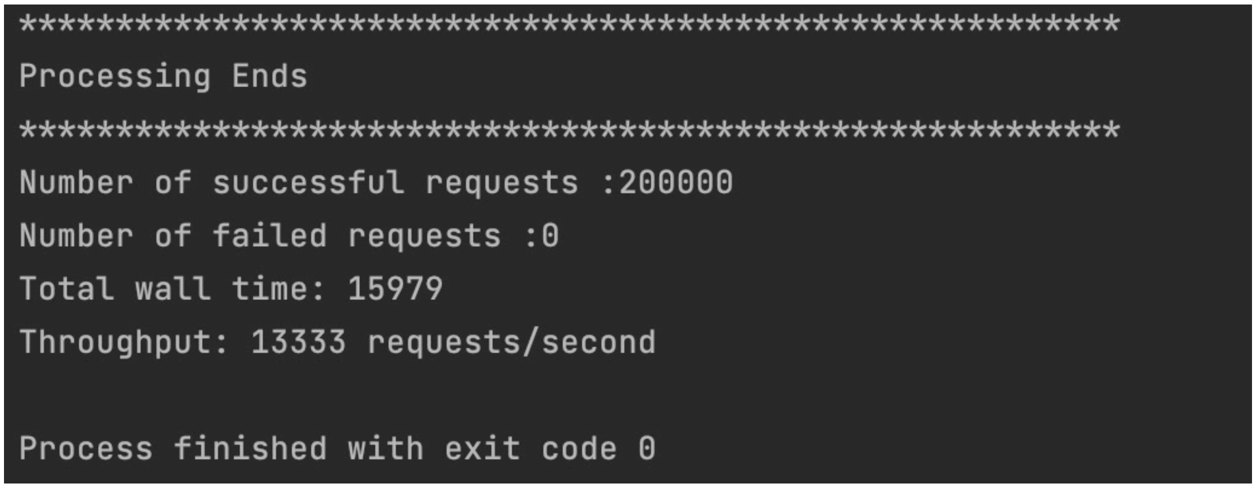
This gives us:

N = Throughput * Response Time
N is basically the amount of concurrency in your server. If the server threads available is less than the client threads, N is the server threads. Otherwise it is the number of client threads.

Some Little’s Law predictions based on 32, 64, and 128 threads which process 200K lift ride events are as followed:

| N (Number of Thread) | λ (Throughput) | W (Average Response Time) |
|---|---|---|
| 32 | 12500 | 2.58715178794699 (ms) |
| 64 | 10526 | 5.90445187840967 (ms) |
| 128 | 10000 | 12.5711985277203 (ms) |

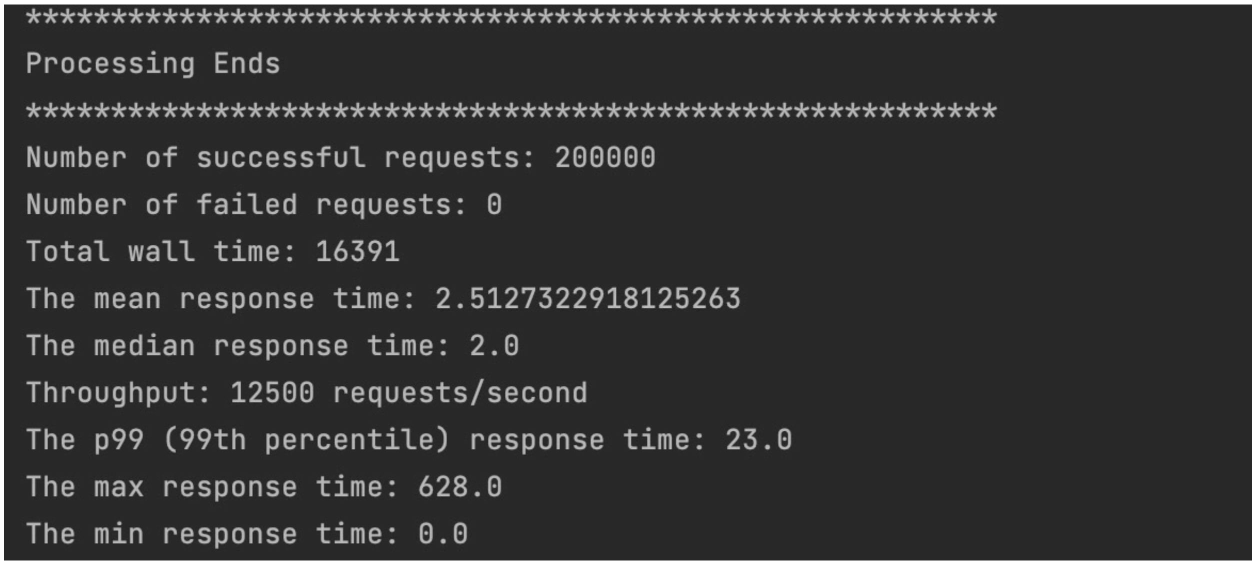
**Client Part1 Screen Shot**:



```
*********************************************************
Processing Ends
*********************************************************
Number of successful requests :200000
Number of failed requests :0
Total wall time: 15979
Throughput: 13333 requests/second


Process finished with exit code 0
```

**Client Part2 Screen Shot (32 Threads)**:



```
*********************************************************
Processing Ends
*********************************************************
Number of successful requests: 200000
Number of failed requests: 0
Total wall time: 16391
The mean response time: 2.5127322918125263
The median response time: 2.0
Throughput: 12500 requests/second
The p99 (99th percentile) response time: 23.0
The max response time: 628.0
The min response time: 0.0
```