CS6650 Assignment 1 Report
Jingyi Mao

GitHub Link:

https://github.com/jm624461806/DistributedSystemSkiResort

Client Design and How it works:

There are total three packages included in the "client"

@Package Model:

- Record:
    1. Used to store the record of latency and HTTP method of each request in ClientPart2.
- HttpMethod:
    1. Enum class for HttpMethod POST and GET.

@Package ClientPart1:

- ClientPart1:
    1. Parse and validate the command line.
    2. Main function that generates all the threads and call the threads which are the instances of ClientThread class to send requests to the server.
    3. Use CountDownLatch.await() to control the process of different phases.
    4. Call the ClientUtil instance to generate the report of part1.
    5. Get the walltime by calling System.currentTimeMillis().
- ClientThread:
    1. Implement the Runnable interface which represents the thread that sends the requests to the server.
    2. Utilize CountDownLatch to track the completeness of threads and further trigger phase two, phase three and main thread.
    3. Utilize AtomicInteger to track the total number of successful/unsuccessful requests.
- ClientUtil:
    1. Util method to validate a number in a specific range.
    2. Util method to create and start the ClientThread thread in part one and also the ClientThreadPart2 thread in part two.
    3. Util method to print out the data after all the phases complete.

@Package ClientPart2:

- ClientPart2:
    1. Parse and validate the command line.

2. Main function that generates all the threads and call the threads which are the instances of ClientThreadPart2 class to send requests to the server.
3. Call the RecordUtil instance to generate the report of part2.
4. Get the walltime by calling System.currentTimeMillis().
5. Generate the consumer which is an instance of RecordWriter to write record of latency and HTTP method of each request.

- ClientThreadPart2:
    1. Implement the Runnable interface which represents the thread that sends the requests to the server.
    2. Utilize CountDownLatch to track the completeness of threads and further trigger phase two, phase three and main thread.
    3. Utilize AtomicInteger to track the total number of successful/unsuccessful requests.
    4. Get the latency by calling System.currentTimeMillis().
    5. Generate the Record and write to the BlockingQueue<Record> to let the RecordWriter consumes.
- RecordUtil:
    1. Read the csv file and put the latency into a list.
    2. Sort the latency list in ascending order.
    3. Get mean, median, max, min and 99 percentile response time from the sorted list.
    4. Generate the report from the request latency data.
- ChartUtil:
    1. Get the list of record and sort the record by starting time. Generate a list of long[] which consists of second interval and mean response time.
    2. Generate csv file from the list of long[].
- RecordWriter:
    1. Implement the Runnable interface and create a csv file to write all the records of each request.
    2. Construct the consumer – producer structure and act as a consumer to consume the records ClientThreadPart2 threads produce. Meanwhile, add the record into a list for further mean response time vs time analysis.

@Package TestClient

- TestClient:
    1. Main function that calls one thread to send 10000 POST requests to the server.
    2. Get the walltime by calling System.currentTimeMillis().
- SingleThreadForLittlesLaw:
    1. Send 10000 requests to the server.

Little's Prediction:

Single Thread:

```
Wall Time for 10000 requests single thread: 249448
```

Prediction:

32-threads: 32/ (0.0249) = 1285 req/sec
64-threads: 64/ (0.0249) = 2579 req/sec
128-threads: 128/ (0.0249) = 5140 req/sec
256-threads: 256/ (0.0249) = 10281 req/sec

Screenshots of the reports generated from ClientPart1 and ClientPart2 when running 32, 64, 128, 256 threads on 20000 skiers and 40 lifts:

I put the fail counter inside the catch(ApiException e) to catch the exception that swagger did not successfully send out the request. Since the swagger does not have the retry as HttpClient, I got a few fails when the thread reaches 256 which is shown below.

32 Threads:

```
Part1 Report:
================================================
Number of successful requests sent:160003
Number of unsuccessful requests sent:0
Wall time: 233830
The total throughput in requests per second: 686.7081545064377(req/sec)
```

```
Part 2 Report:
================================================
Mean response time: 25.78 ms
Median response time: 24.0 ms
Wall time: 238769 ms
The total throughput in requests per second: 672.281512605042(req/sec)
The p99 response time: 51 ms
Max response time: 2375 ms
Min response time: 11 ms
```

64 Threads:

```
Part1 Report:
================================================
Number of successful requests sent:160006
Number of unsuccessful requests sent:0
Wall time: 123029
The total throughput in requests per second: 1300.861788617886(req/sec)
```

```
Part 2 Report:
================================================
Mean response time: 29.88 ms
Median response time: 25.0 ms
Wall time: 125749 ms
The total throughput in requests per second: 1280.048(req/sec)
The p99 response time: 149 ms
Max response time: 885 ms
Min response time: 14 ms
```

128 Threads:

```
Part1 Report:
================================================
Number of successful requests sent:159948
Number of unsuccessful requests sent:0
Wall time: 77866
The total throughput in requests per second: 2077.246753246753(req/sec)
```

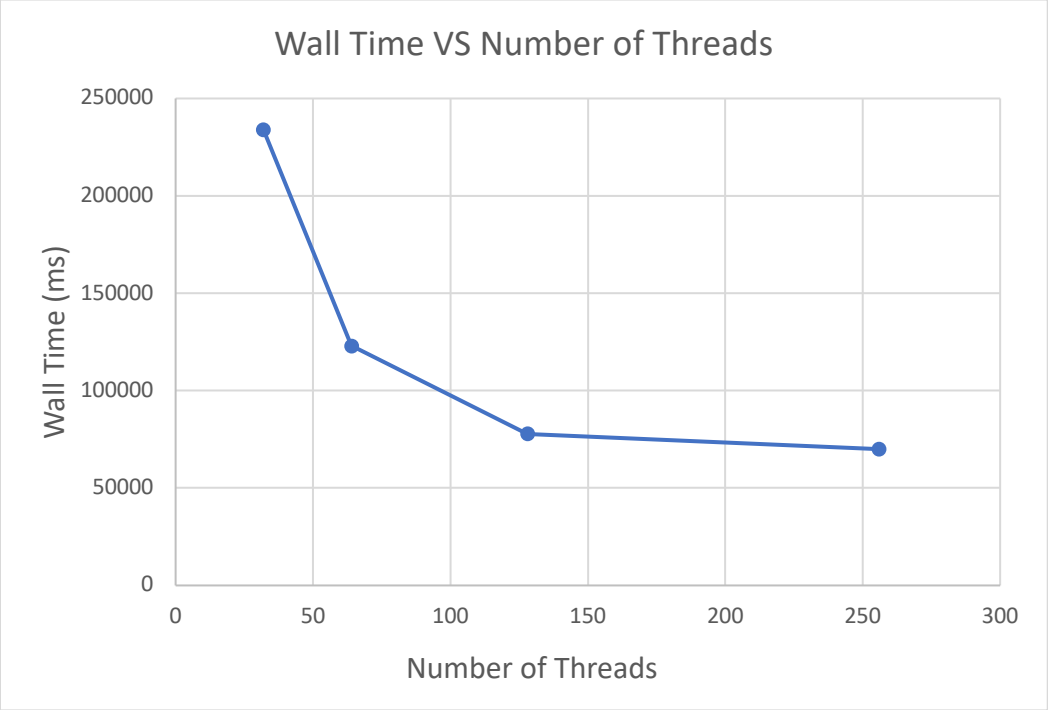
```
Part 2 Report:
================================================
Mean response time: 39.81 ms
Median response time: 27.0 ms
Wall time: 79393 ms
The total throughput in requests per second: 2024.6582278481012(req/sec)
The p99 response time: 311 ms
Max response time: 3025 ms
Min response time: 12 ms
```

256 Threads:

```
Part1 Report:
================================================
Number of successful requests sent:159818
Number of unsuccessful requests sent:15
Wall time: 69970
The total throughput in requests per second: 2316.4202898550725(req/sec)
```

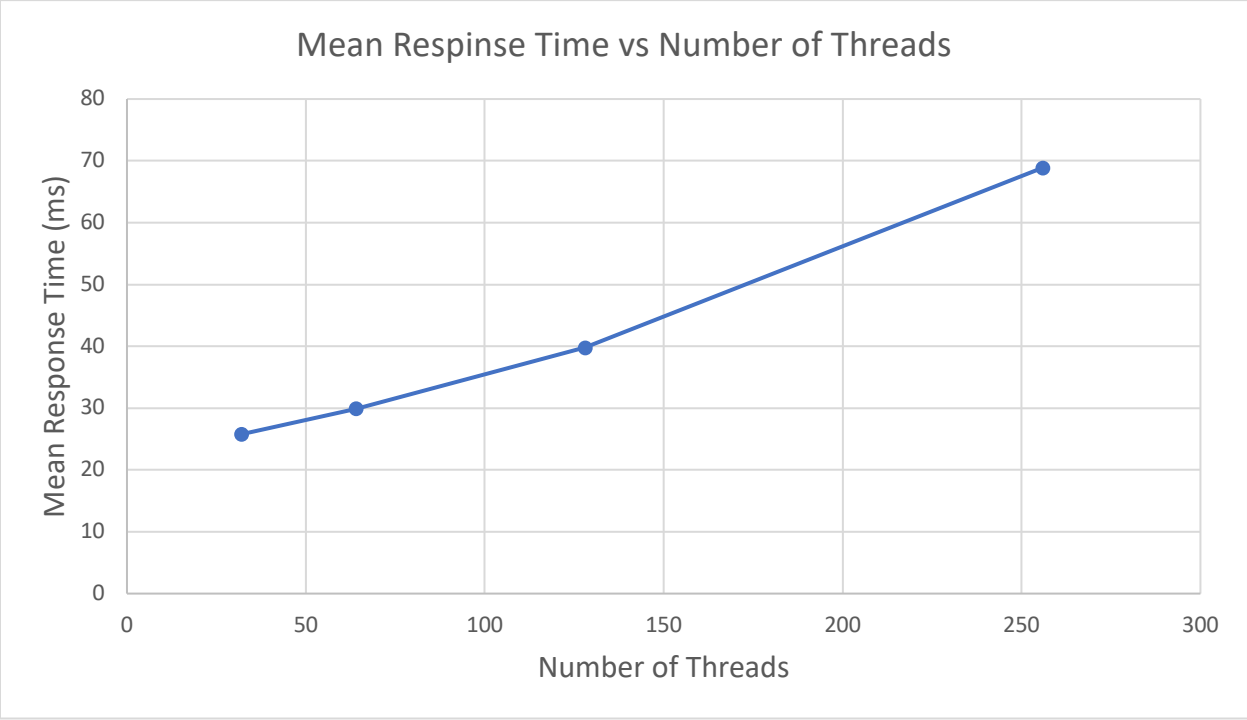
```
Part 2 Report:
================================================
Mean response time: 68.9 ms
Median response time: 30.0 ms
Wall time: 70436 ms
The total throughput in requests per second: 2283.3285714285716(req/sec)
The p99 response time: 685 ms
Max response time: 9694 ms
Min response time: 14 ms
```

Client 1 Plot:



I tried different approaches like eliminate the thread bottleneck but still did not get the linear wall time. Would appreciate any thoughts on possible “outside” reasons or “inside” reasons on my codes side.

Client 2 Plot:



Bonus Charting:

The mean response time was calculated by first sort the record list and then record the number of request within 1000 ms and sum up the latency, then use sum/number to get the mean response time.