Client

Repository link: Concurrency&Client-server

Client Description

Execute instructions

Both part 1 and part 2 can be executed with the same instructions below. To execute this application from IDE, you can create a "Run Configuration". In Intellij, the run configuration needs to use JDK 17 with main class point to org.neu.cs6650.client.Main. Then you need to fill the CLI argument with the following parameters:

```
<Server_ip> <Thread_pool_size> <Load_size>
```

For example, the target server has an ip address 18.237.117.228, and we want to use 350 thread to send 500k request.

```
18.237.117.228 350 500000
```

Note: thread_pool_size=350 give the best performance with around 9000~10000 req/sec.

Major Classes, packages, relationships

Main class under org.neu.cs6650.client is the entry point of this project. It governs the client's life cycle. It will read 3 command line arguments, which are server_ip, thread_pool_size and load_size. The main class inits an ExecutorService instance with the given thread_pool_size in order to control the concurrency.

The Main class inits a HttpService instance that contains an apache http client with connection pool size equals thread_pool_size. The required retry is built within the client. This service class handles all http requests.

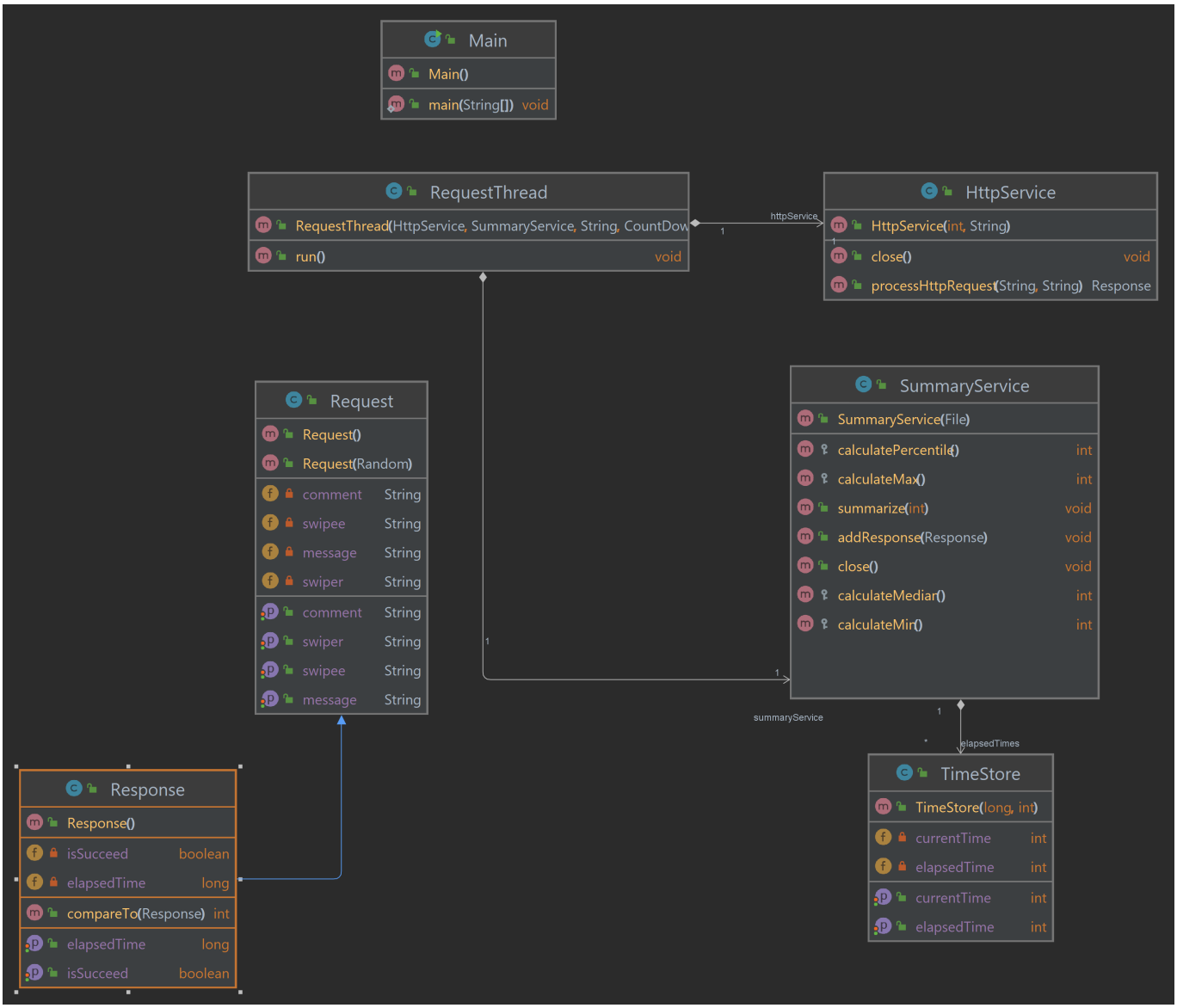
A SummaryService instance is initialized by the main class to handle recording all values and finalizing all statistic analysis at the end.

Within Main class, it will create a number(load_size) of RequestThread instances and submit them into the ExecutorService instances for processing. Within a RequestThread instance, it will call HttpService to send one request and record the elapsedTime into a SummaryService instance.

After all RequestInstances are submitted, main will shut down the ExecutorService and use a CountDownLatch instance to block the process until all request are processed. At the end of main, the SummaryService instance will perform statistic analysis and print out the result to console.

All model classes are within org.neu.cs6650.model package and all service classes are within org.neu.cs6650.service package.

UML Document



Client statistics

Part 1

To calcualte norminal elapsed time for a http request. I used one thread to process 5000 request.The result screen shot is as follows:

```
==================Summary==================
Succeed count: 5000. Failure count: 0
Overall elapsed time: 102579 ms
Throughput: 48 req/sec
Mean response time: 20.455000 ms
Median response time: 20 ms
P99 response time: 43 ms
Min response time: 14 ms
Max response time: 108 ms

Process finished with exit code 0
```

The mean response time is 20.4 ms. Consider 200 threads in tomcat server, the theoratical throughput calcualted by the little's law is 8333 req/sec.

With a few trails, I found that running 250 threads and 250 HTTP connections in the client provide similar result. Here's the screen shot:

```
==================Summary==================
Succeed count: 500000. Failure count: 0
Overall elapsed time: 59868 ms
Throughput: 8351 req/sec
```

Despite the thread I used is over 200, the theoretical throughput calculated above still holds true because tomcat is using 200 threads. I think the additonal thread in client is just compensating the context switching overhead within remote server.

Part 2

Running client with 240 threads and 240 HTTP connections. Here's the example screen shot:

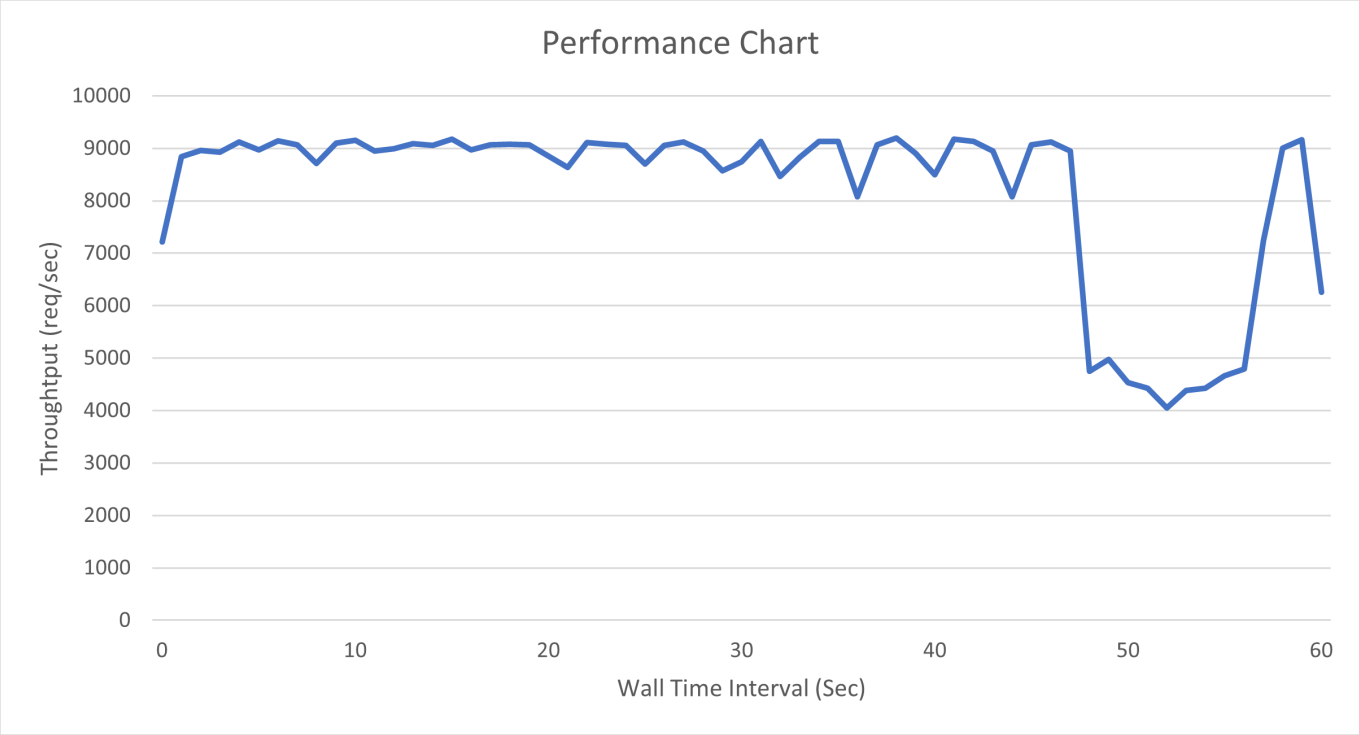
```
==================Summary==================
Succeed count: 500000. Failure count: 0
Overall elapsed time: 59868 ms
Throughput: 8351 req/sec
Mean response time: 28.662162 ms
Median response time: 24 ms
P99 response time: 90 ms
Min response time: 12 ms
Max response time: 15139 ms
```

Statistics for multiple trails:

Overall elapsed time(Sec)	**Throughtput (req/sec)**	**Mean response time(ms)**	**Median response time(ms)**	**P99 Response time(ms)**	**Min response time(ms)**	**Max response time(ms)**

Overall elapsed time(Sec)	Throughtput (req/sec)	Mean response time(ms)	Median response time(ms)	P99 Response time(ms)	Min response time(ms)	Max response time(ms)
59.868	8351	28.61	24	90	12	15139
61.324	8153	29.96	24	247	12	4548
60.621	8247	29.24	24	249	12	3074

Performance Plot



Spring boot result

Here's performance when I use spring boot to build the remote server.

```
==================Summary==================
Succeed count: 500000. Failure count: 0
Overall elapsed time: 103208 ms
Total recorded stored: 500000
Throughput: 4854 req/sec
Mean response time: 269.072836 ms
Median response time: 52 ms
P99 response time: 302 ms
Min response time: 13 ms
Max response time: 21157 ms

Process finished with exit code 0
```

Spring boot performance result is similar to the performance result when I use 120 threads in my local client with customized servlet on remote server. Here's my client's performance result with 120 threads:

```
==================Summary==================
Succeed count: 500000. Failure count: 0
Overall elapsed time: 102106 ms
Total recorded stored: 500000
Throughput: 4901 req/sec
Mean response time: 224.466762 ms
Median response time: 23 ms
P99 response time: 50 ms
Min response time: 12 ms
Max response time: 7093 ms

Process finished with exit code 0
```