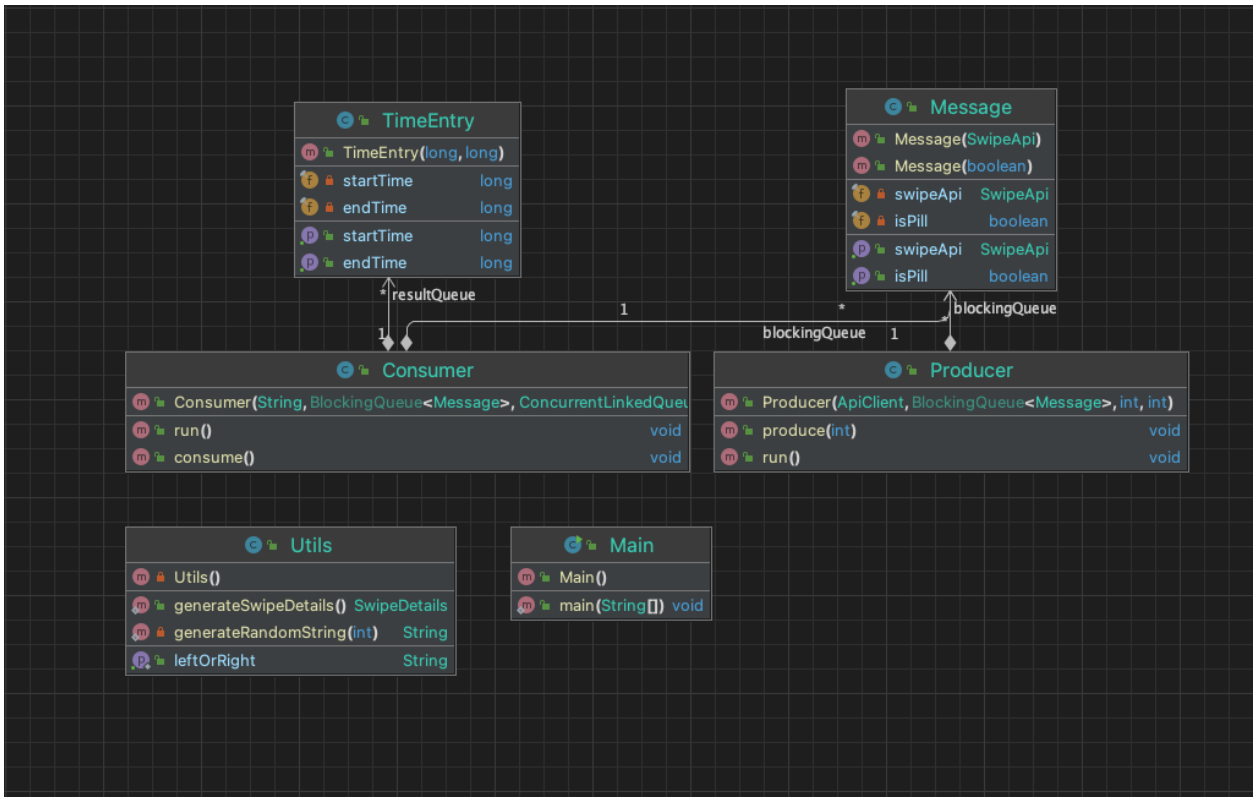
Twinder Client Design Report

CS6650 Distributed System
Zidi Xia

Client design:



This UML shows the Major Classes, Packages, and Relationships of Twinder Client.

Client (Part 1) :

- This should be a screenshot of your output window with your wall time and throughput. Also, make sure you include the client configuration in terms of number of threads used and Little's Law throughput predictions.

Throughput Prediction:
To get our best throughput predation we need to get response time for a single request, then use Little's law to get our throughput.

Step1:
Deploy server to EC2 and run.
Result: total number of threads: 1, total number of tasks: 10000, total wall time: 161642

```
total number of threads: 1, total number of tasks: 10000

number of successful requests sent: 10000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 161642.0
total throughput in requests per second: 61.86510931564816
mean: 16.1414
median: 16.0
throughput: 61.86510931564816 (requests/second)
p99: 29
min: 12
max: 243
```

Step2:
Using a single thread to run a simple test and send 10000 requests.
N = 1, total number of requests: 10000
Response Time = how long a single request takes = time / total number of requests
Total Time = 161642 ms = 161.642 s (convert to second)
Response Time = 161.642 / 10000 = 0.0162 second per request

Step3:
Now we have the response time, we can calculate the throughput.
N = Throughput * Response Time
Throughput = N / Response Time
Throughput = 1/ 0.0162 = 61 requests per second
As a result, our best prediction is 61.

Step4:
N = 1, Throughput = 61
Making prediction by multiplying the single-threaded throughput by the number of client threads, to estimate the multi-threaded throughput. N is basically the amount of concurrency.

Number of threads	Single thread throughput	Total throughput
1	61	61
20	61	1220
50	61	3050
100	61	6100
150	61	9150
200	61	12200

Based on our result, the best throughput is 61.

Client (Part 2):

- Run the client as per Part 1, showing the output window for each run with the specified performance statistics listed at the end.

Run 500K requests using 20, 50, 100, 150, and 200 threads:

Now let's compare if the overall throughput for 500K requests is close to the Little's Law estimate.

total number of threads: 20, total number of tasks: 500000, throughput = 1180

```
total number of threads: 20, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 423448.0
total throughput in requests per second: 1180.7825281970868
mean: 16.913424
median: 16.0
throughput: 1180.7825281970868 (requests/second)
p99: 31
min: 11
max: 481
```

total number of threads: 50, total number of tasks: 500000, throughput = 2278

```
total number of threads: 50, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 219466.0
total throughput in requests per second: 2278.2572243536583
mean: 21.921686
median: 21.0
throughput: 2278.2572243536583 (requests/second)
p99: 44
min: 11
max: 531
```

total number of threads: 100, total number of tasks: 500000, throughput = **2291**

```
total number of threads: 100, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 218173.0
total throughput in requests per second: 2291.759291938049
mean: 43.619158
median: 50.0
throughput: 2291.759291938049 (requests/second)
p99: 67
min: 13
max: 532
```

total number of threads: 150, total number of tasks: 500000, throughput = 2275

```
total number of threads: 150, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 219724.0
total throughput in requests per second: 2275.58209389962
mean: 65.901762
median: 52.0
throughput: 2275.58209389962 (requests/second)
p99: 110
min: 13
max: 585
```

total number of threads: 200, total number of tasks: 500000, throughput = 2287

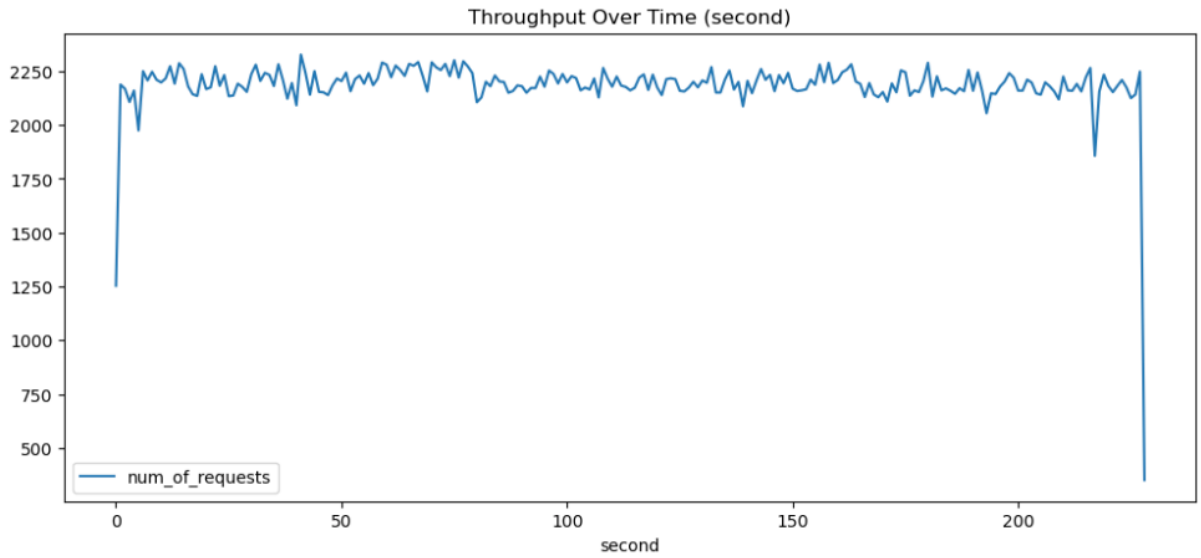
```
total number of threads: 200, total number of tasks: 500000

number of successful requests sent: 500000
number of unsuccessful requests: 0
the total run time (wall time) for all threads to complete: 218542.0
total throughput in requests per second: 2287.8897420175526
mean: 87.40672
median: 73.0
throughput: 2287.8897420175526 (requests/second)
p99: 153
min: 13
max: 681
```

Number of threads	predicted total throughput with 10k requests	total throughput with 500k requests
20	1220	1180
50	3050	2278
100	6100	**2291**
150	9150	2275
200	12200	2287

Conclusion :
Based on our tests, sending 500k requests with 100 threads has the best total throughput.
By comparing predicted throughput and actual throughput, the result seems close when the number of threads is 20. When the number of threads is higher than 50, the predicted throughput increases sharply. However, the actual throughput tends to be stable.

The plot of throughput over time



Comparison with Spring server:

I implemented both the SpringBoot server and java plain servlet to make the comparison. Spring is a framework that also uses the servlet underlying. After testing with 500K requests and 100 threads, the result shows that Java plain servlet is faster than the Spring server. One of the main reasons is that they use different serialization libraries. Servlet uses Gson while Spring uses Jackson, and Gson is faster than Jackson mapper. GSON is not capable of parsing well when it does not know the data type which it is mapping while Spring is easier to use and has better compatibility.