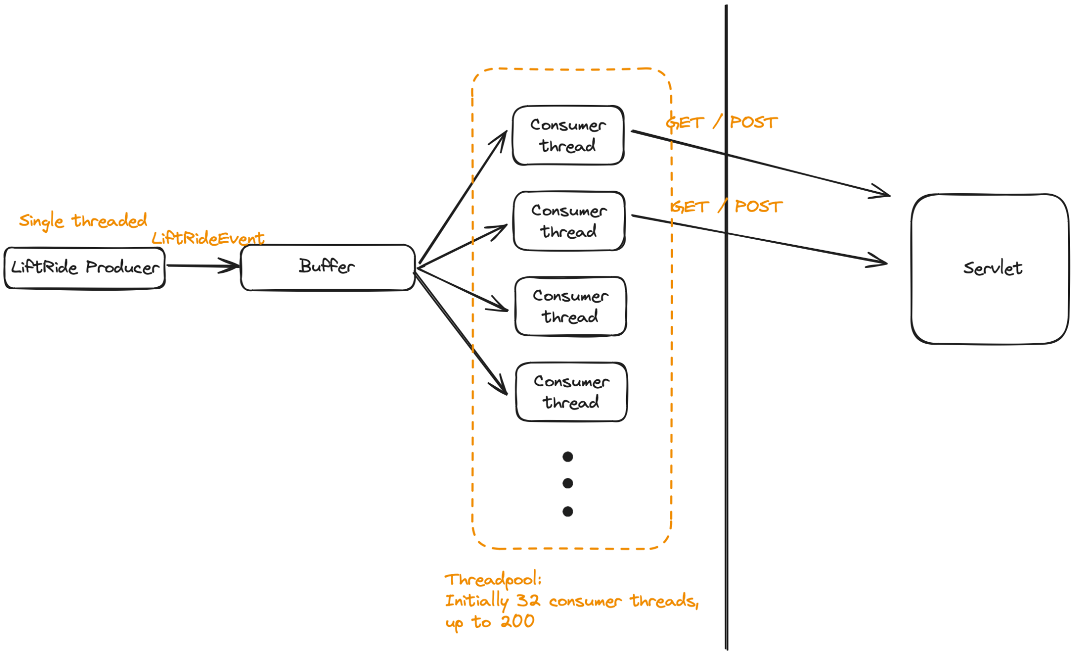
Client url : https://github.com/wakaka1123/SkierClient.git
Servlet url: https://github.com/wakaka1123/SkierServlet.git

High level architecture

The LiftRide producer randomly generates 200k LiftRideEvent objects and push them to the buffer. A thread pool sets up 32 threads initially to listen to the buffer, each of which takes one object from the buffer and make a POST request to the servlet for 1k times. As soon as any of the 32 threads completes its task, it will signal, and the thread pool will scale up to 200 threads to handle the remaining requests.

The thread pool is parameterized with SynchronousQueue and CallerRunsPolicy. SynchronousQueue is a message queue and signals the thread to start running without blocking, and CallerRunsPolicy ensures the threads are not discarding any failed request.

The abovementioned configurations can handle most tps with my laptop.



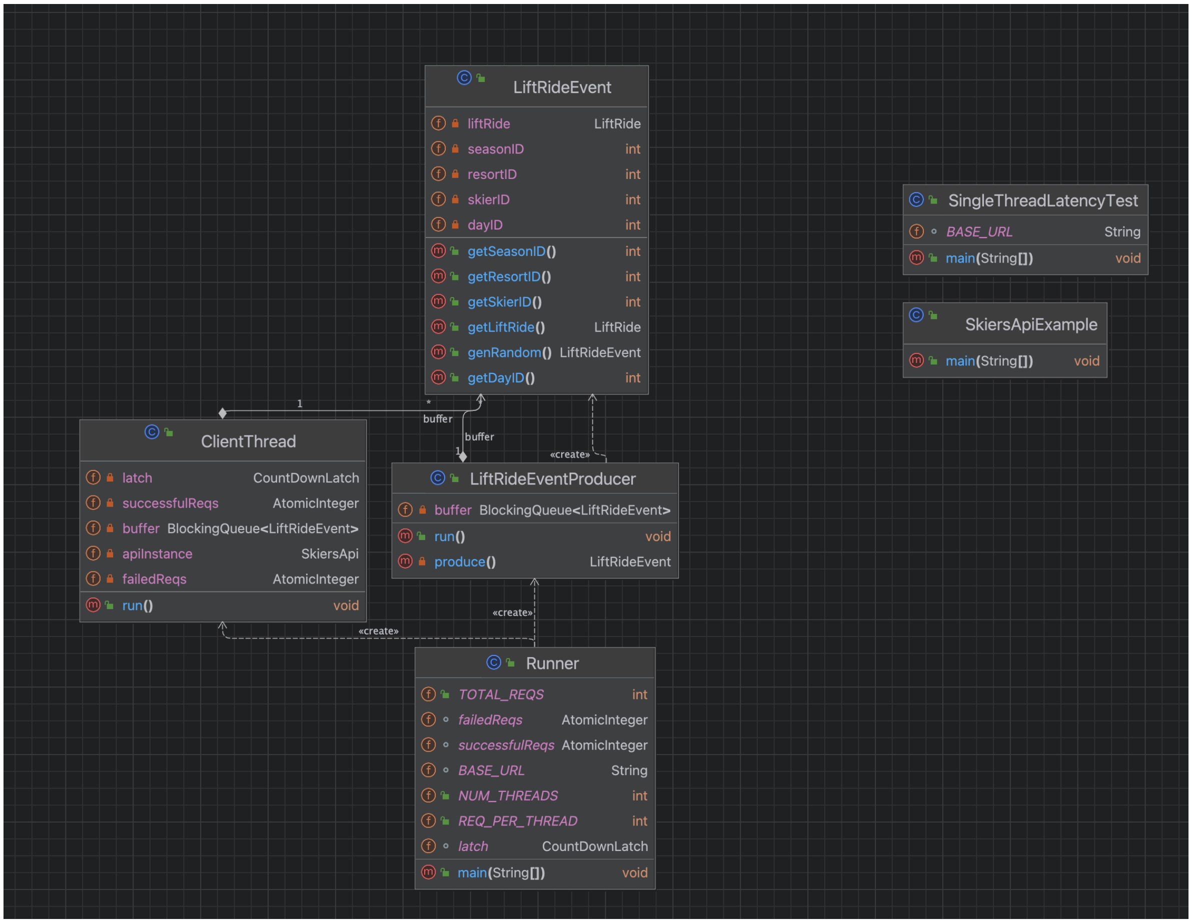
SkierClient Part 1

Run Runner to send 200k POST requests to servlet.

Run SingleThreadLatencyTest to get estimate of latency.

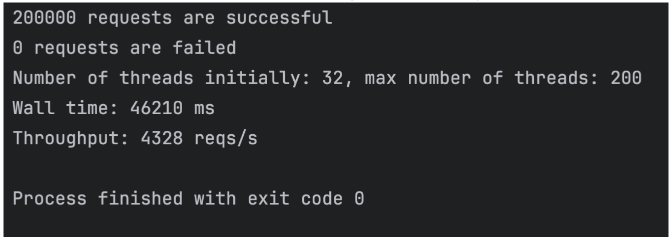
//Run SkierApiExample to send single GET/POST request to test connection with servlet.

Major Classes are: LiftRideEvent, LiftRideProducer, ClientThread, Runner



Part1 Screenshot including wall time and throughput.

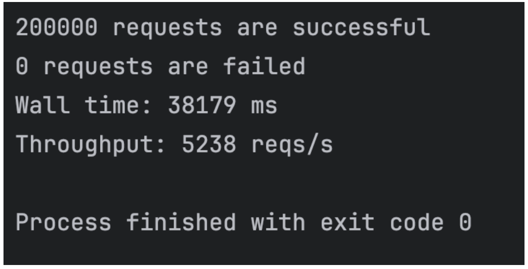
32 to 200 threads, handling 200k requests with config



```
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 46210 ms
Throughput: 4328 reqs/s

Process finished with exit code 0
```

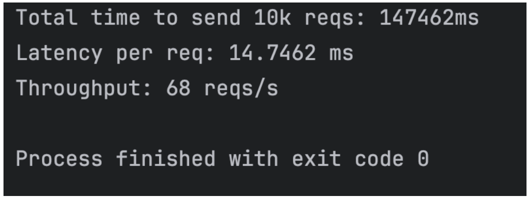
My top performer at around 1am, 32 to 200 threads, handling 200k requests
Although without thread number printing out, you can trust this one



```
200000 requests are successful
0 requests are failed
Wall time: 38179 ms
Throughput: 5238 reqs/s

Process finished with exit code 0
```

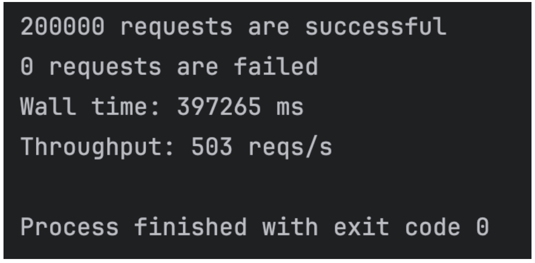
1 thread, handling 10k requests



```
Total time to send 10k reqs: 147462ms
Latency per req: 14.7462 ms
Throughput: 68 reqs/s

Process finished with exit code 0
```

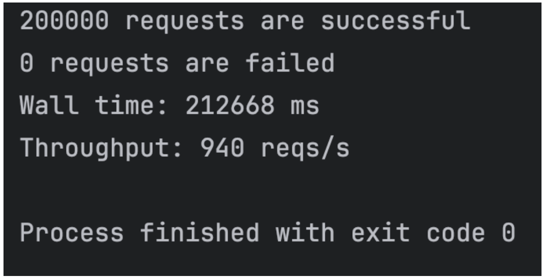
Supplemental tests
10 threads, 200k reqs



```
200000 requests are successful
0 requests are failed
Wall time: 397265 ms
Throughput: 503 reqs/s

Process finished with exit code 0
```

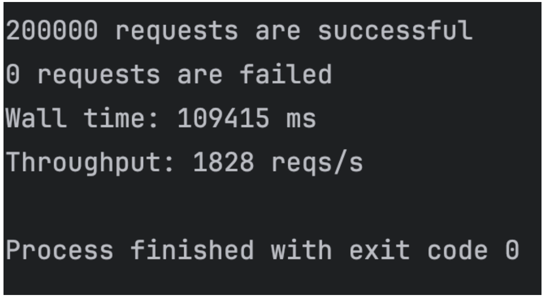
20threads, 200k reqs



```
200000 requests are successful
0 requests are failed
Wall time: 212668 ms
Throughput: 940 reqs/s

Process finished with exit code 0
```

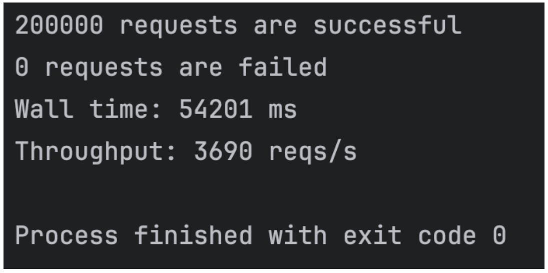
50threads, 200k reqs



```
200000 requests are successful
0 requests are failed
Wall time: 109415 ms
Throughput: 1828 reqs/s

Process finished with exit code 0
```

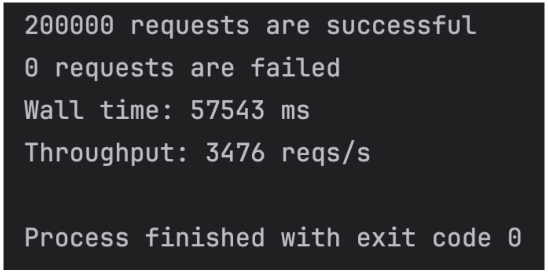
100threads, 200k reqs



```
200000 requests are successful
0 requests are failed
Wall time: 54201 ms
Throughput: 3690 reqs/s

Process finished with exit code 0
```

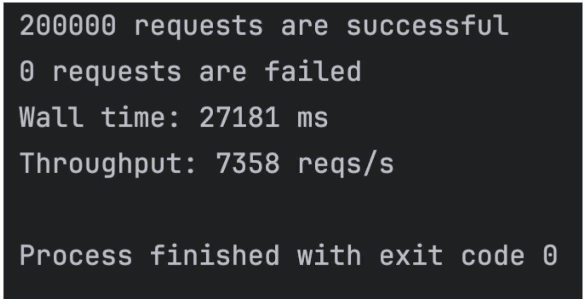
150threads, 200k reqs



```
200000 requests are successful
0 requests are failed
Wall time: 57543 ms
Throughput: 3476 reqs/s

Process finished with exit code 0
```

200threads, 200k reqs



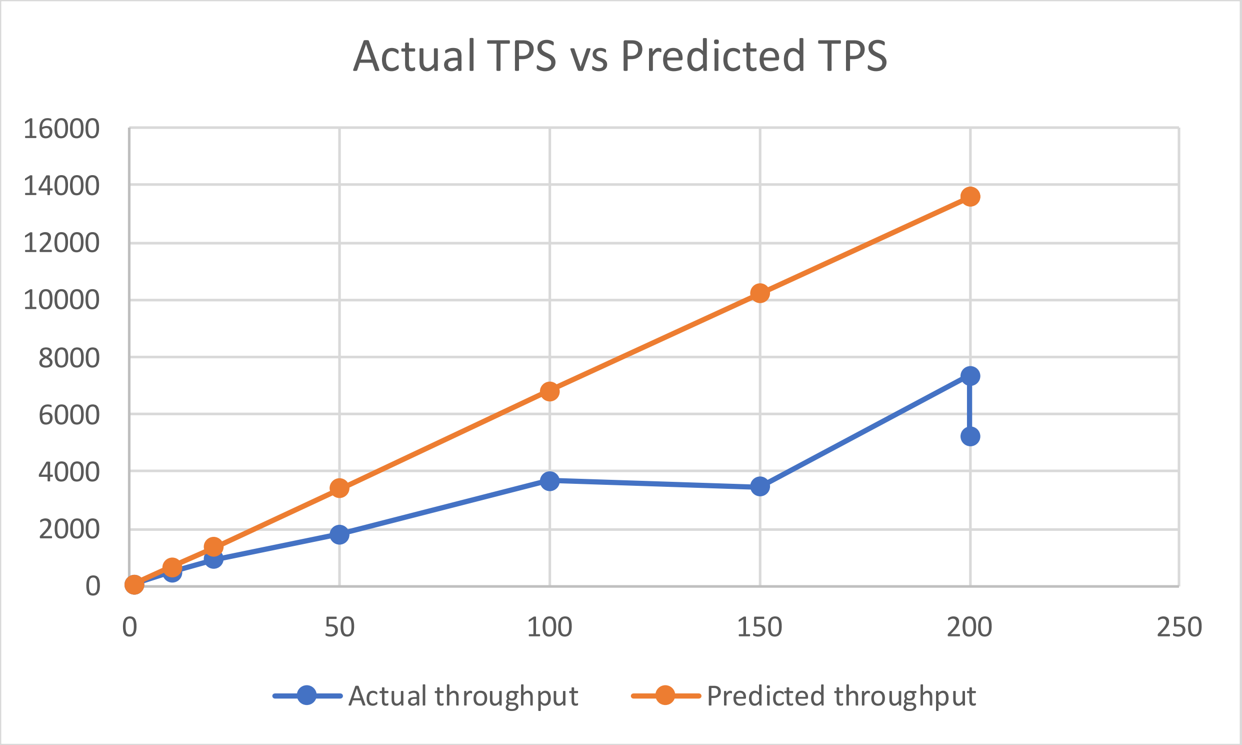
```
200000 requests are successful
0 requests are failed
Wall time: 27181 ms
Throughput: 7358 reqs/s

Process finished with exit code 0
```

Little’s Law predictions

$$Throughput = \frac{work\ in\ progress}{response\ time}$$

Initial number of threads	Max number of threads	Total requests	Reqs per thread	Actual throughput (reqs/s)	Predicted throughput (reqs/s)	Comments
1	1	10k	10k	68	68	Benchmark
10	10	200k	20k	503	680	Supplemental
20	20	200k	10k	940	1360	Supplemental
50	50	200k	4k	1828	3400	Supplemental
100	100	200k	2k	3690	6800	Supplemental
150	150	200k	1.33k	3476	10200	Supplemental
200	200	200k	1k	7358	13600	Supplemental
32	200	200k	~1k	5238	13600	Client Part 1

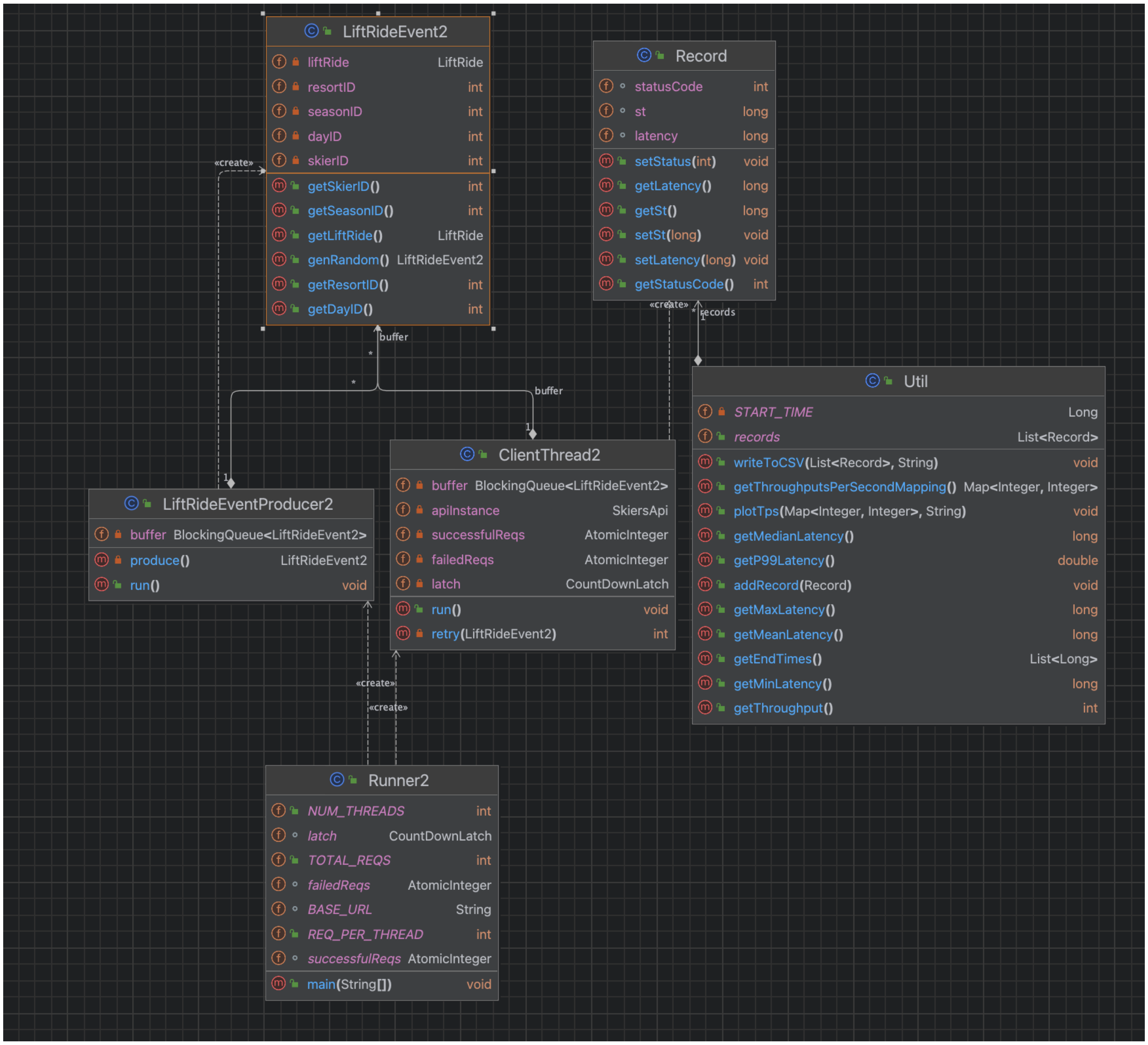


The last point of our running case(SkierClient part 1) falls lower than the last supplemental test as our client start with only 32 threads and scales up to 200 when 1k requests is completed. The top TPS(7358 reqs/s) shall not grow higher as tomcat can handle 200 threads as default.

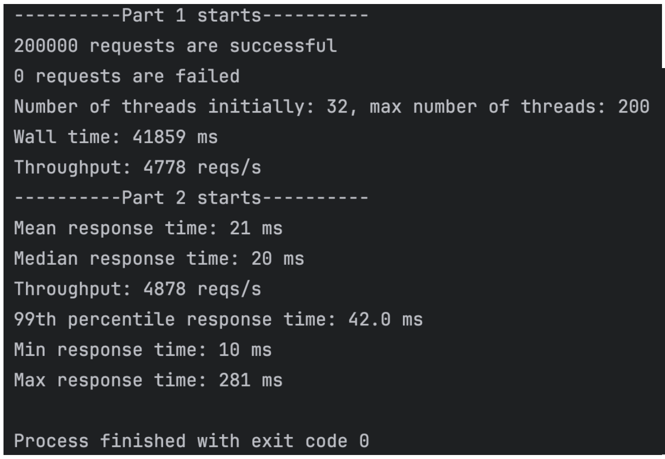
SkierClient Part2

Run Runner2 to send 200k POST requests to the servlet and prints out performance statistics

Major classes: LiftRideEvent2, LiftRideEventProducer2, Record, Util, ClientThread2, Runner2

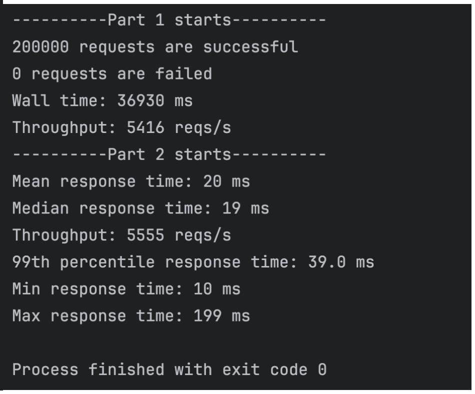


Performance Statistics (two runs)



```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 41859 ms
Throughput: 4778 reqs/s
----------Part 2 starts----------
Mean response time: 21 ms
Median response time: 20 ms
Throughput: 4878 reqs/s
99th percentile response time: 42.0 ms
Min response time: 10 ms
Max response time: 281 ms

Process finished with exit code 0
```

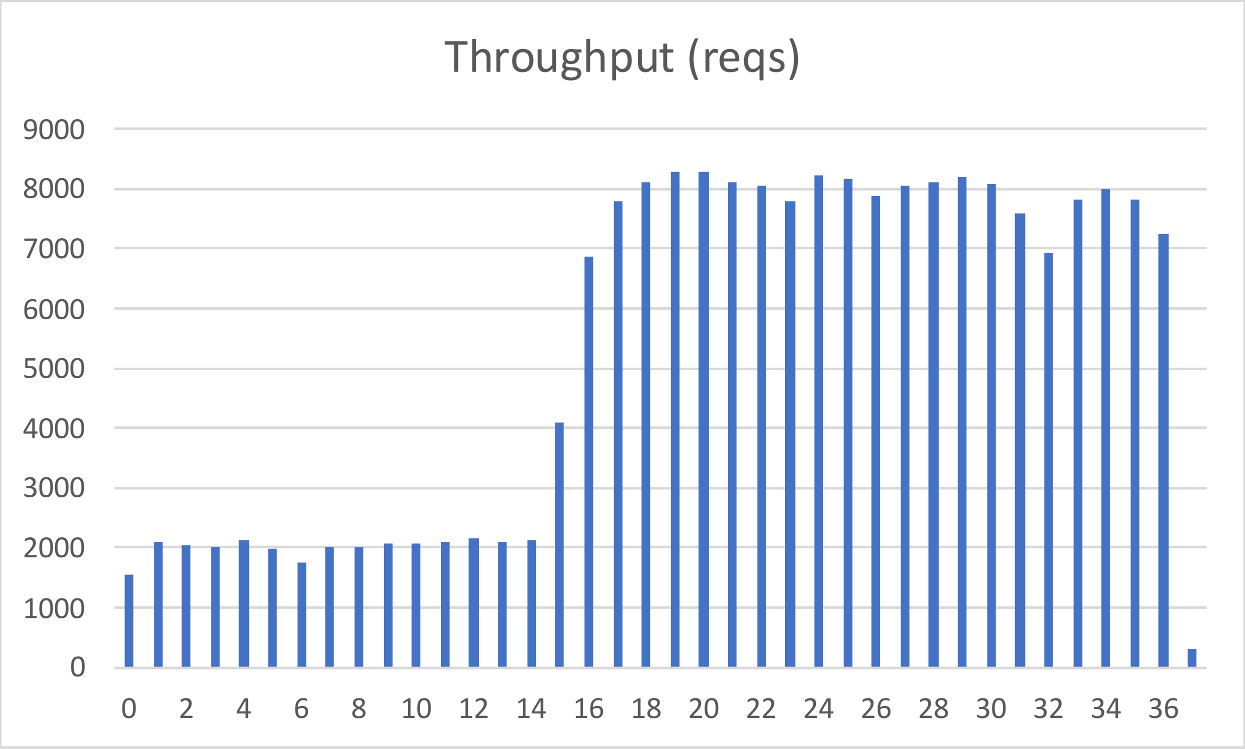


```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Wall time: 36930 ms
Throughput: 5416 reqs/s
----------Part 2 starts----------
Mean response time: 20 ms
Median response time: 19 ms
Throughput: 5555 reqs/s
99th percentile response time: 39.0 ms
Min response time: 10 ms
Max response time: 199 ms

Process finished with exit code 0
```

We have retained the TPS from part 1 calculation and avg TPS from part 2 as above.
4778 * 1.05 = 5016 > 4878
Our avg TPS falls in the 105% range of TPS calculation by total requests and wall time.
RHS is top performer without thread config but you can trust.

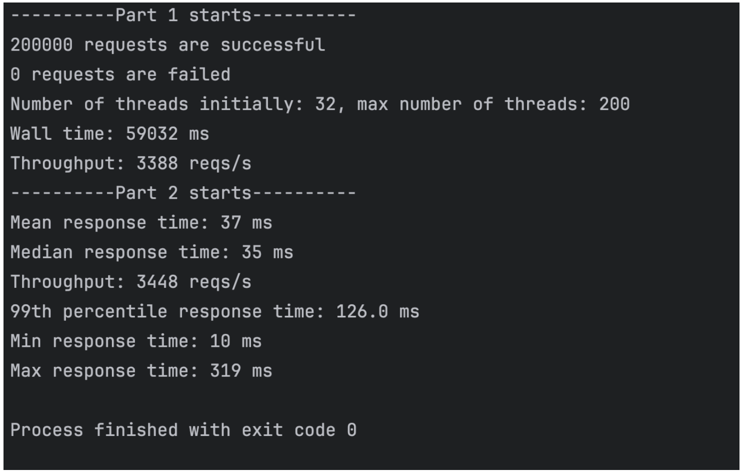
Plot of tps over time



As we recall our single threaded test with 10k reqs, the benchmark tps is 68 req/s. If we want to send 1k reqs, it will take t = 1000 / 68 = 14.7 sec which matches the graph above that throughput merges at around 15sec on X coordinate.

SpringBoot Server

Change base Url and run Runner2 static String BASE_URL = "http://54.189.174.77:8080";



```
----------Part 1 starts----------
200000 requests are successful
0 requests are failed
Number of threads initially: 32, max number of threads: 200
Wall time: 59032 ms
Throughput: 3388 reqs/s
----------Part 2 starts----------
Mean response time: 37 ms
Median response time: 35 ms
Throughput: 3448 reqs/s
99th percentile response time: 126.0 ms
Min response time: 10 ms
Max response time: 319 ms

Process finished with exit code 0
```

SpringBoot server does not show better performance than servlet. I am testing with Springboot server during the day, it is probably the traffic that causes Springboot handling lower throughputs than servlet(3.4k TPS vs 5.2k TPS).

I think it is because I am implementing a simple Springboot server without fully implementing async response. If both client and server are implemented to start threads to send http requests and receive responses in an async way, Springboot should beat servlet.

Appendix

The recorded csv file of 20k post requests and its relevant TPS plotting raw data are enclosed in the git repo. You can find them with the following link:
https://github.com/wakaka1123/SkierClient/blob/main/SkierClient/records-200k-32.csv
https://github.com/wakaka1123/SkierClient/blob/main/SkierClient/tps-200k-32.csv