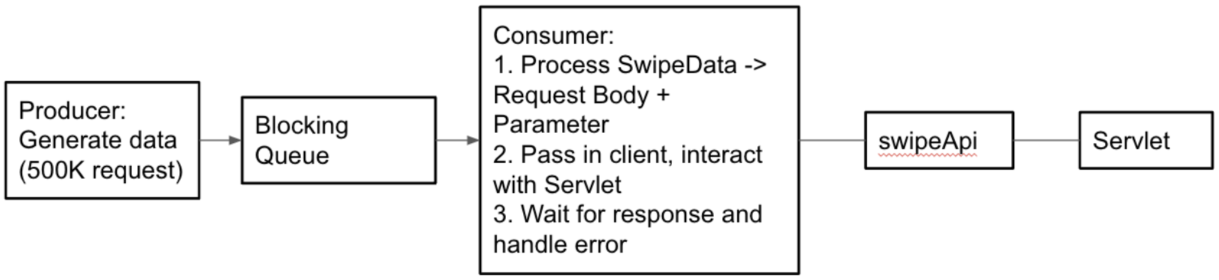
Name: Junyao Feng
Git repo: https://github.com/june616/CS6650

Client design description

In the project of Swipe Servlet, a POST API is implemented following the rules and gets tested with POSTMAN.

In the project of client 1, the major classes include Swipe Data, Producer, Thread Single, and Thread Pool (act as Consumer, every thread in the thread pool will perform the tasks defined in Thread Single). The main idea of the project is shown as follows:


Producer: Generate data (500K request)
Blocking Queue
Consumer:
1. Process SwipeData -> Request Body + Parameter
2. Pass in client, interact with Servlet
3. Wait for response and handle error
swipeApi
Servlet

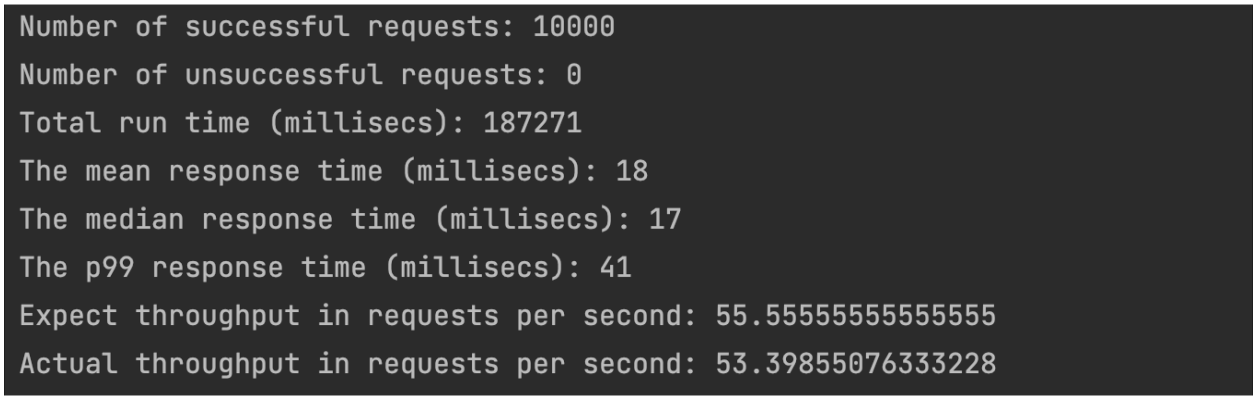
(1) Create a blocking queue shared by Producer and Consumer.
(2) Create a producer to generate 500k Swipe Data objects followed by rules and store them in the blocking queue
(3) Create a thread pool with 50 threads and acts as a consumer: take the Swipe Data objects from the blocking queue, interact with the Servlet through the swipe API using the objects as parameters (handle error: retry 5 times)
(4) At the same time, use an AtomicInteger counter to count the number of successful requests; take a time stamp before/after any POST requests start/finish to calculate the wall time.

In the project of client 2, in addition to what is mentioned in client 1, we also need to keep track of some properties of every request in a CSV file and calculate some metrics related to response time. Therefore, we introduce new classes including Record, Record List Process, and Write Csv.
(1) Create a thread-safe list to store the record objects.
(2) In Thread Single, take a time stamp before/after every POST request starts/finishes to calculate the latency, package required properties as a Record object, and store them in the record list.
(3) In Record List Process, create a list storing the response time for every record object, and calculate the required metrics (mean, median, p99, min and max).
(4) In Write Csv, create a new CSV file, iterate the record list, convert record properties into strings, and write into the file using csvWriter.

Client (Part 1)

Little's Law throughput analysis:

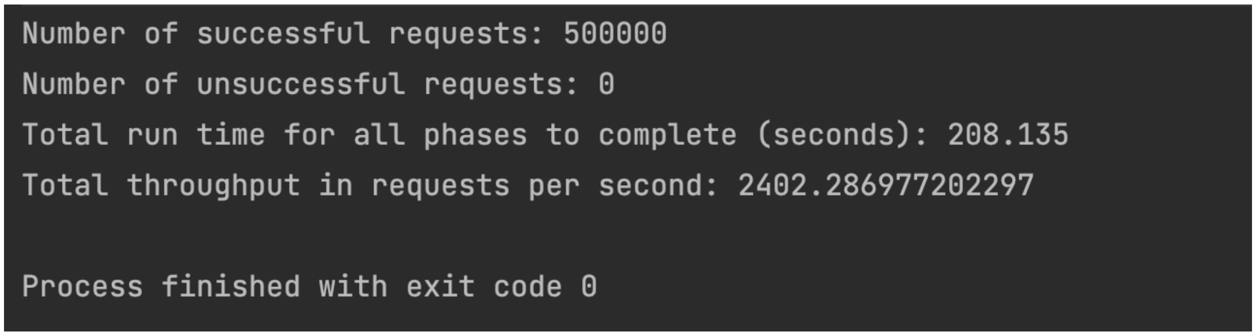
When sending 10k requests with 1 thread, the mean response time is 18 ms (0.018s).



```
Number of successful requests: 10000
Number of unsuccessful requests: 0
Total run time (millisecs): 187271
The mean response time (millisecs): 18
The median response time (millisecs): 17
The p99 response time (millisecs): 41
Expect throughput in requests per second: 55.55555555555555
Actual throughput in requests per second: 53.39855076333228
```

Therefore, the expected throughput will be 50/0.018=2777.77 when sending 500k requests with 50 threads (implemented by a thread pool) in client 1 and client 2.

In client 1, the actual throughput is 500000/208.135=2402.28, which is reasonable.

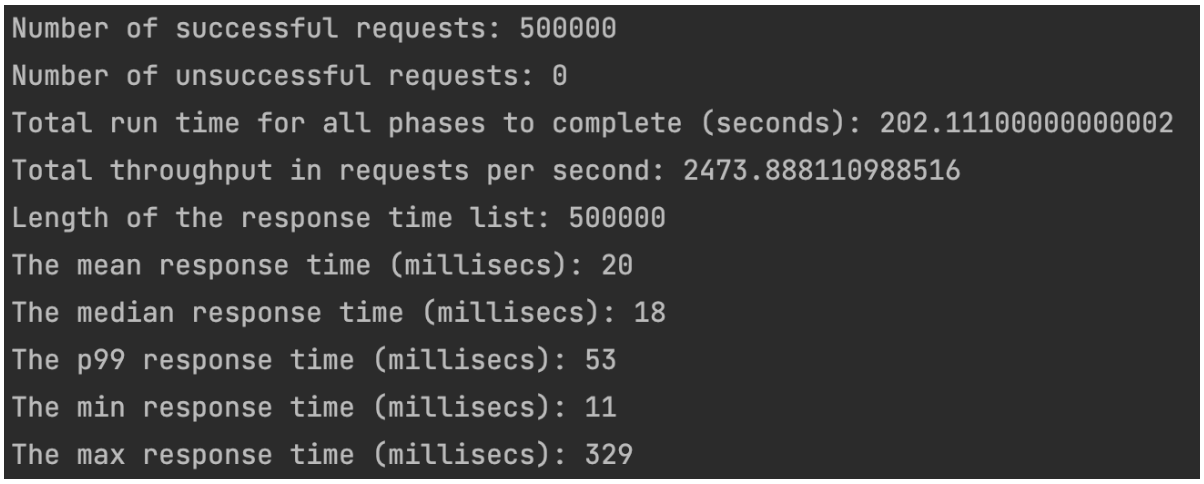


```
Number of successful requests: 500000
Number of unsuccessful requests: 0
Total run time for all phases to complete (seconds): 208.135
Total throughput in requests per second: 2402.286977202297

Process finished with exit code 0
```

Client (Part 2)

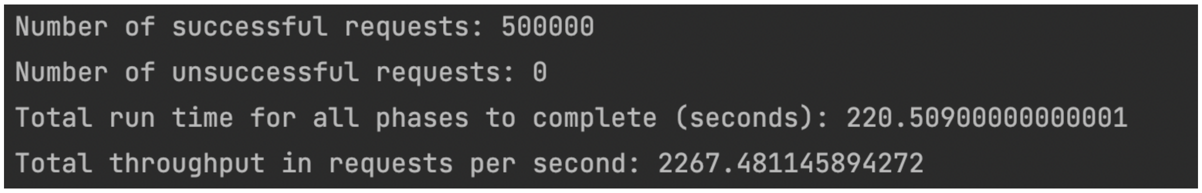
Here is the output of client 2.



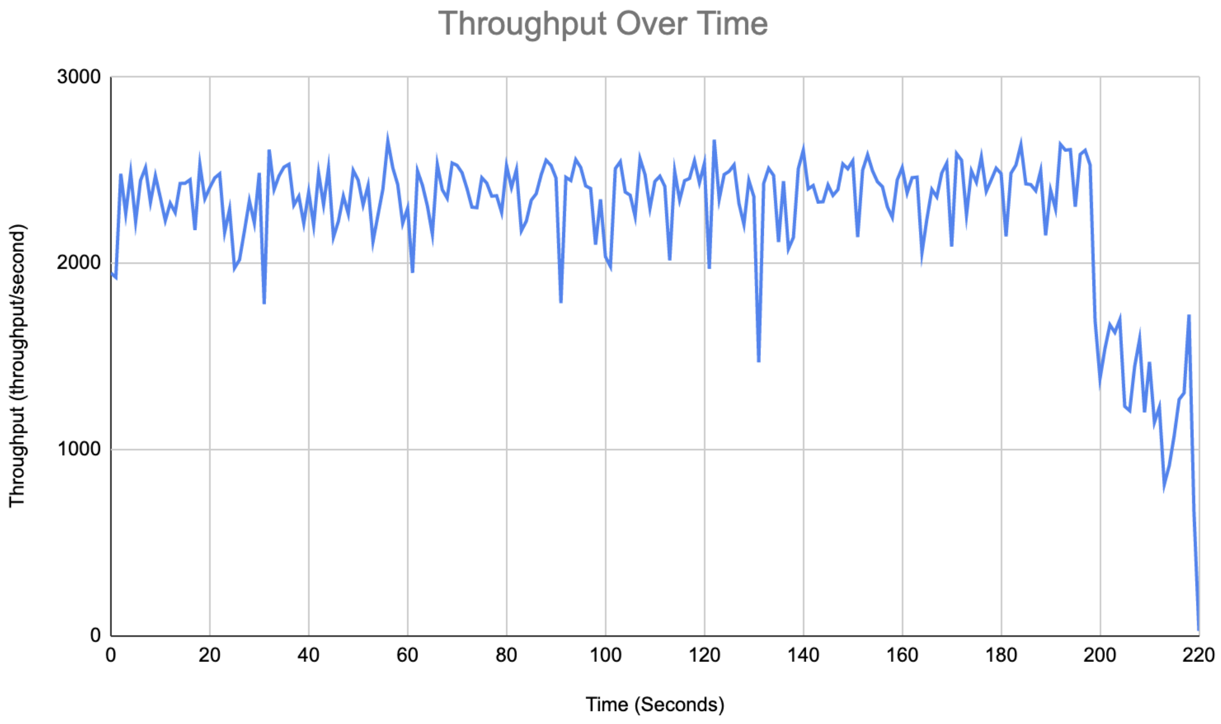
```
Number of successful requests: 500000
Number of unsuccessful requests: 0
Total run time for all phases to complete (seconds): 202.11100000000002
Total throughput in requests per second: 2473.888110988516
Length of the response time list: 500000
The mean response time (millisecs): 20
The median response time (millisecs): 18
The p99 response time (millisecs): 53
The min response time (millisecs): 11
The max response time (millisecs): 329
```

A plot of throughput over time

When generating the needed data points for the plot, the run time is around 220 seconds.



```
Number of successful requests: 500000
Number of unsuccessful requests: 0
Total run time for all phases to complete (seconds): 220.50900000000001
Total throughput in requests per second: 2267.481145894272
```


Throughput Over Time
Throughput (throughput/second)
Time (Seconds)