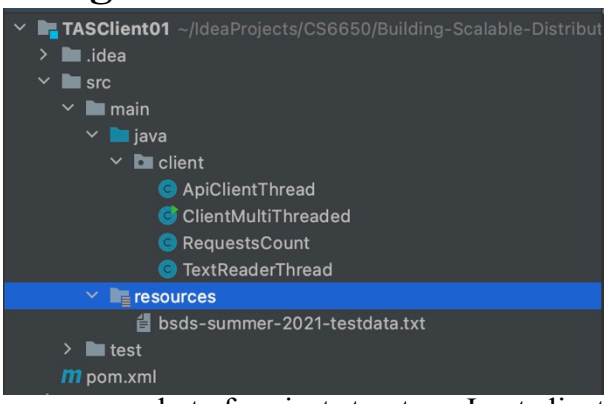
Client Design Description

GitHub Link: https://github.com/guojunseven/Building-Scalable-Distributed-System/tree/master/Assignment01

1. Client1 Design



As you can see from my screenshot of project structure, I put client related code in **client** package (src/main/java/), There are four class : ***ApiClientThread, ClientMultiThreaded, RequestsCount, TextReaderThread***. The **ClientMultiThreaded** creasts multiple **ApiClientThread**, **TextReaderThread** to send requests and read file. It uses **RequestsCount** to count different type of requests.

1.1 ClientMultiThreaded

This is the start class for configure the multiple threads for sending requests and report the running statistics. I initialize all configuration variable in this class and it provides a main method to run the client program.

```java
/**
 * This class generates multiple api client threads to read line from a given input
 * and send them to server for analysis. The main thread will wait for all threads to complete.
 */
public class ClientMultiThreaded {

    private static String input;
    private static int maxThreads;
    private static String localPath = "http://localhost:8080/TextProcessor";
    private static String basePath = "http://ec2-54-91-96-97.compute-1.amazonaws.com:8080/TextProcessor";
    private static String function = "wordcount";

    /**
     * This main method takes two parameters: one is the path of text input file the other is the MAX_Threads
     * used to process the text.
     * @param args the input file and max_threads
     */
    public static void main(String[] args) throws IOException, InterruptedException, BrokenBarrierException {
        // check the arguments
        checkArgs(args);
```

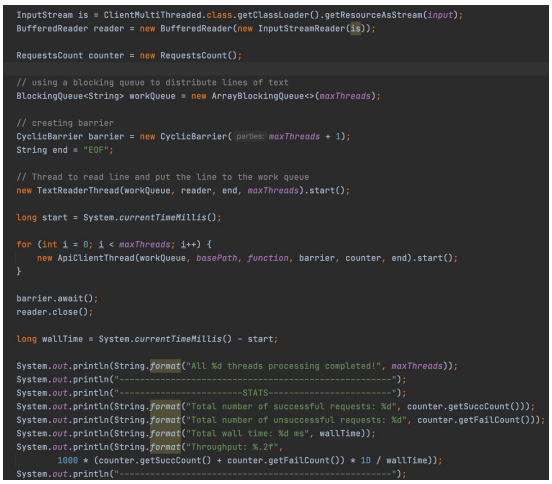
At first, it checks the arguments passed by the command line.

Then, it reads the specified text file using ***BufferedReader*** to save the time spent in I/O operation. And it also instantiates a **counter** for count the successful and unsuccessful requests.

Next, it creates a **cyclic barrier** and **blocking queue** serving for coordinating multiple threads and provides the requests-sending threads text line one by one.

In this implementation, I create a single thread for reading text line and put it into the work queue. I tested it and make sure the speed it produces the line is larger than the speed other threads consume line from the queue. Then I take timestamp and create ***maxThreads*** threads to send requests to the server simultaneously.

Finally, the main thread waits for other threads to complete and ready to print the results report.



```
InputStream is = ClientMultiThreaded.class.getClassLoader().getResourceAsStream(input);
BufferedReader reader = new BufferedReader(new InputStreamReader(is));

RequestsCount counter = new RequestsCount();

// using a blocking queue to distribute lines of text
BlockingQueue<String> workQueue = new ArrayBlockingQueue<>(maxThreads);

// creating barrier
CyclicBarrier barrier = new CyclicBarrier( parties: maxThreads + 1);
String end = "EOF";

// Thread to read line and put the line to the work queue
new TextReaderThread(workQueue, reader, end, maxThreads).start();

long start = System.currentTimeMillis();

for (int i = 0; i < maxThreads; i++) {
    new ApiClientThread(workQueue, basePath, function, barrier, counter, end).start();
}

barrier.await();
reader.close();

long wallTime = System.currentTimeMillis() - start;

System.out.println(String.format("All %d threads processing completed!", maxThreads));
System.out.println("-----------------------------------------------------");
System.out.println("----------------------STATS----------------------");
System.out.println(String.format("Total number of successful requests: %d", counter.getSuccCount()));
System.out.println(String.format("Total number of unsuccessful requests: %d", counter.getFailCount()));
System.out.println(String.format("Total wall time: %d ms", wallTime));
System.out.println(String.format("Throughput: %.2f",
        1000 * (counter.getSuccCount() + counter.getFailCount()) * 10 / wallTime));
System.out.println("-----------------------------------------------------");
```

1.2 ApiClientThread

This is the http requests handler class and extends ***Thread*** so that it will be created as a single thread to send requests. It constructor accepts **workQueue** to take line text, **basePath** to which the api client connects, **function** the post parameter, **barrier** the synchronized barrier, **counter** to count successful requests and unsuccessful requests, **end** to mark the end of sending requests.

This class will instantiate an **api client once** and use this client to send all requests. In its **run()** method, it will continuously attempt to take a line from the blocking queue in a loop and check if it reaches the end of the file. If yes, it then breaks to wait other threads complete. Otherwise, it will send the request to server and wait for response.

1.3 RequestsCount

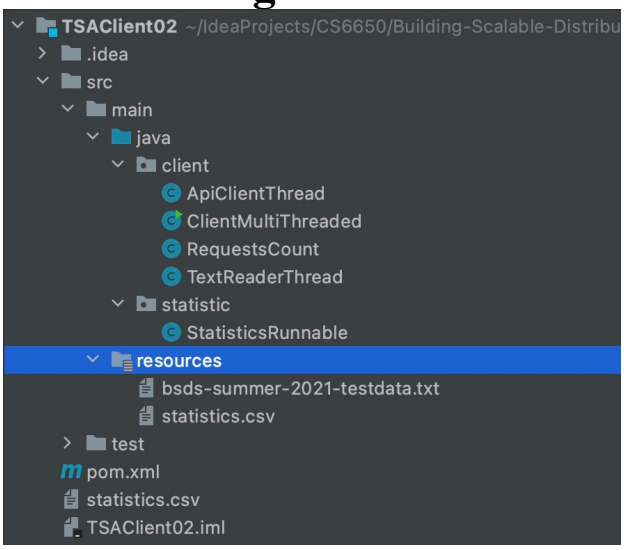
This is the class used for counting the requests number and it is **thread-safe**. It has two method: **markSucc()** and **markFail()** to increase successful requests number or unsuccessful requests number by one. And it provides getter to get these counts.

1.4 TextReaderThread

This class is responsible for read lines one by one from a **reader** (text) and put these line to a blocking queue to serve the requests-sending threads. Its constructor accepts work queue the blocking queue to put lines, reader the bufferedReader object to read lines, end the mark of **EOF**, maxThreads the number of the end mark.

In its **run()** method, it will continuously read lines from the reader until it reaches the end. Then it put the **end mark** in a number of threads number to the queue to inform other threads that there is no other lines to read.

2. Client2 Design



Unlike Client1, it has a separate package "**statistic**" to place code related to **writing statistics** into CSV file. The other classes have the similar code and functionality with clien1. I will introduce the modified part of client2 compared to client1 in the following paragraphs.

2.1 ClientMultiThreaded

```
// Thread pool to write statistics to csv file
BufferedWriter writer = new BufferedWriter(new FileWriter(csvPath));
ThreadPoolExecutor pool = new ThreadPoolExecutor(
        corePoolSize: 10,  maximumPoolSize: 10,  keepAliveTime: 50, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<>());
```

I create a **BufferedWriter** to write statistical lines to the specified csv file. And I also instantiate a thread pool with unbounded work queue and 10 core threads size for write lines to the writer after sending requests. I design like this so that my api client thread **won't need to spend time in writing csv file**. They just submit the runnable statistics task to the thread pool so that the writing to csv file won't affect the overall throughput of our program.

```
long wallTime = System.currentTimeMillis() - start;

// wait for writing csv
while (true) {
    if (pool.getActiveCount() == 0) {
        pool.shutdown();
        break;
    }
}

// analyze the statistics
writer.flush();
writer.close();
// analyze the statistics
String statistics = analyze();
```

And after sending all request, I need check if the thread pool has active count (is there still tasks to writing csv). If not, it will shut down the thread pool and close the writer. And do the analysis report.

Besides above, the other parts of this class remain the same.

2.2 ApiClientThread

Unlike client1, its constructor now accepts an additional **writer** and a **thread pool** to for writing csv file after sending requests.

```
    long latency = System.nanoTime() - start;
    pool.execute(new StatisticsRunnable(writer, new String[]{"" + start, "POST", "" + latency, "" + responseCode}));
```

2.3 StatisticasRunnable

This class represents a task to write line to a writer of a specified CSV file. Its constructor accepts a **writer** and the **data** in an array needed to be written. In the run() method, it creates a line from the array and write it to the writer. And we don't need to worry about synchronizing here because of the nature that writer is **thread safe**.

```
public class StatisticsRunnable implements Runnable {

    private final BufferedWriter writer;
    private final String[] lineData;

    public StatisticsRunnable(BufferedWriter writer, String[] lineData) {
        Objects.requireNonNull(writer);
        this.writer = writer;
        this.lineData = lineData;
    }

    @Override
    public void run() {
        try {
            writer.write( str: String.join( delimiter: ",", lineData) + "\n");
        } catch (IOException e) {
            System.err.println("fail to write statistical info to csv file");
        }
    }
}
```

3. Client1 Results

- 2 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 2
All 2 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 11151 ms
Throughput: 890.50
```

- 4 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 4
All 4 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 6113 ms
Throughput: 1624.41
--------------------------------------------------
```

- 8 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 8
All 8 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 3893 ms
Throughput: 2550.73
--------------------------------------------------
```

- 16 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 16
All 16 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 2921 ms
Throughput: 3399.52
--------------------------------------------------
```

The throughput reaches maximum in 16 threads and then start to decrease due to the context switching and contention between multiple threads

- 32 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 32
All 32 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 3304 ms
Throughput: 3005.45
--------------------------------------------------
```

- 64 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 64
All 64 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 3661 ms
Throughput: 2712.37
--------------------------------------------------
```

- 128 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 128
All 128 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 3713 ms
Throughput: 2674.39
--------------------------------------------------
```

- 256 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 256
All 256 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 3695 ms
Throughput: 2687.42
--------------------------------------------------
```

- 2 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 2
All 2 threads processing completed!
--------------------------------------------------
------------------------STATS-----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 2.2368 ms
Median response time : 2.0995 ms
99th Percentile response time : 4.1210 ms
Max response time : 266.8765 ms
```

- 4 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 4
All 4 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 2.3573 ms
Median response time : 2.0962 ms
99th Percentile response time : 5.9225 ms
Max response time : 276.7612 ms
Total wall time: 6060 ms
Throughput: 1638.61
--------------------------------------------------
```

- 8 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 8
All 8 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 2.9106 ms
Median response time : 2.2785 ms
99th Percentile response time : 7.4550 ms
Max response time : 270.7904 ms
Total wall time: 3830 ms
Throughput: 2592.69
--------------------------------------------------
```

- 16 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 16
All 16 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 4.2743 ms
Median response time : 2.8653 ms
99th Percentile response time : 13.7921 ms
Max response time : 284.0846 ms
Total wall time: 2886 ms
Throughput: 3440.75
--------------------------------------------------
```

- 32 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 32
All 32 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 9.6917 ms
Median response time : 8.3350 ms
99th Percentile response time : 31.9219 ms
Max response time : 302.9346 ms
Total wall time: 3395 ms
Throughput: 2924.89
--------------------------------------------------
```

- 64 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 64
All 64 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 20.4453 ms
Median response time : 18.3184 ms
99th Percentile response time : 95.4754 ms
Max response time : 350.7344 ms
Total wall time: 3601 ms
```

- 128 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 128
All 128 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 35.2699 ms
Median response time : 30.4079 ms
99th Percentile response time : 210.9081 ms
Max response time : 499.3412 ms
Total wall time: 3752 ms
Throughput: 2646.59
--------------------------------------------------
```

- 256 threads

```
[ec2-user@ip-172-31-5-218 assignment01]$ java -jar client02-jar-with-dependencies.jar bsds-summer-2021-testdata.txt 256
All 256 threads processing completed!
--------------------------------------------------
-----------------------STATS----------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Mean response time : 74.4866 ms
Median response time : 62.0125 ms
99th Percentile response time : 442.3282 ms
Max response time : 784.8303 ms
Total wall time: 3618 ms
Throughput: 2744.61
--------------------------------------------------
```

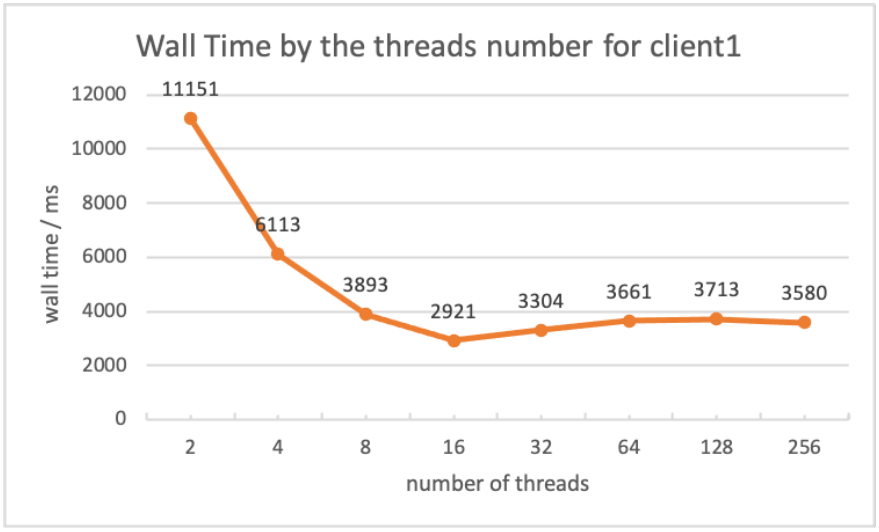
The throughput for client also reaches maximum in 16 threads and then start to decrease due to the context switching and contention between multiple threads

Wall time difference:

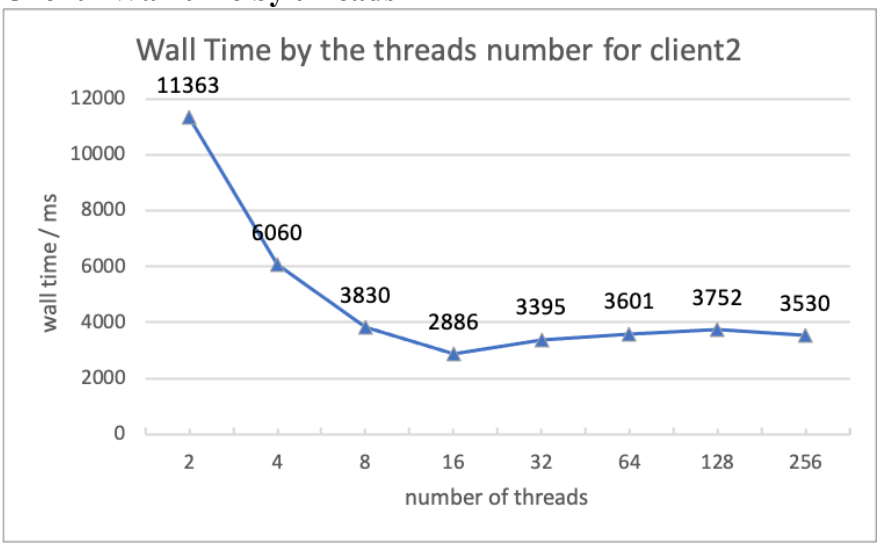
The difference between two clients is within 5%, but it is fluctuated.

Wall Time (ms) Difference Between Clients					
threads	clien01	client02	diff	throughput	mean latency (ms)
2	11151	11363	1.87%	873.89	2.2368
4	6113	6060	-0.87%	1638.61	2.3573
8	3893	3830	-1.64%	2592.69	2.9106
16	2921	2886	-1.21%	3440.75	4.2743
32	3304	3395	2.68%	2924.89	9.6917
64	3661	3601	-1.67%	2757.57	20.4435
128	3713	3752	1.04%	2646.59	32.2699
256	3695	3618	-2.13%	2744.61	74.4866

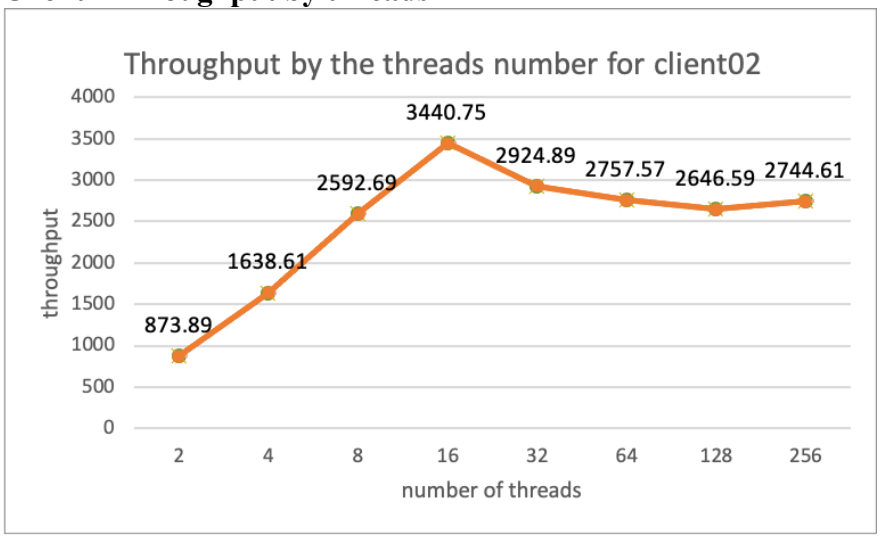
Client1 Wall time by threads



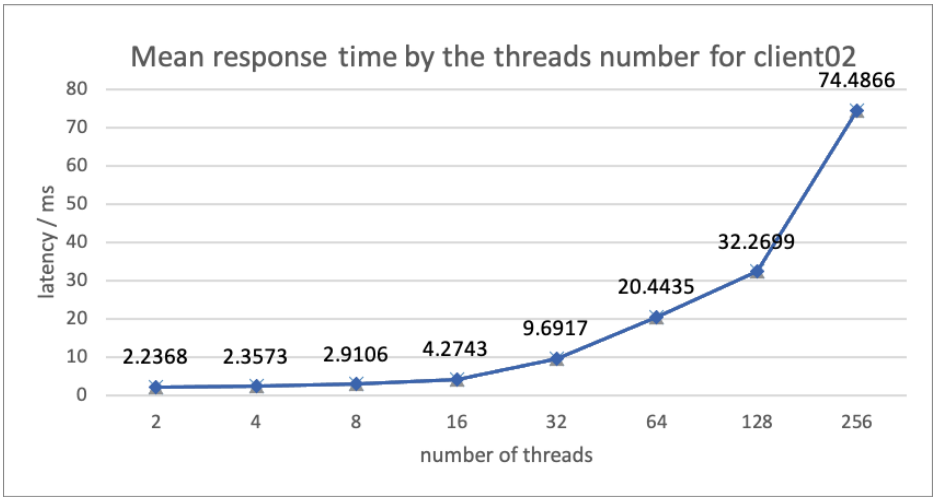
Client2 Wall time by threads



Client2 Throughput by threads



Client2 Mean response time by threads

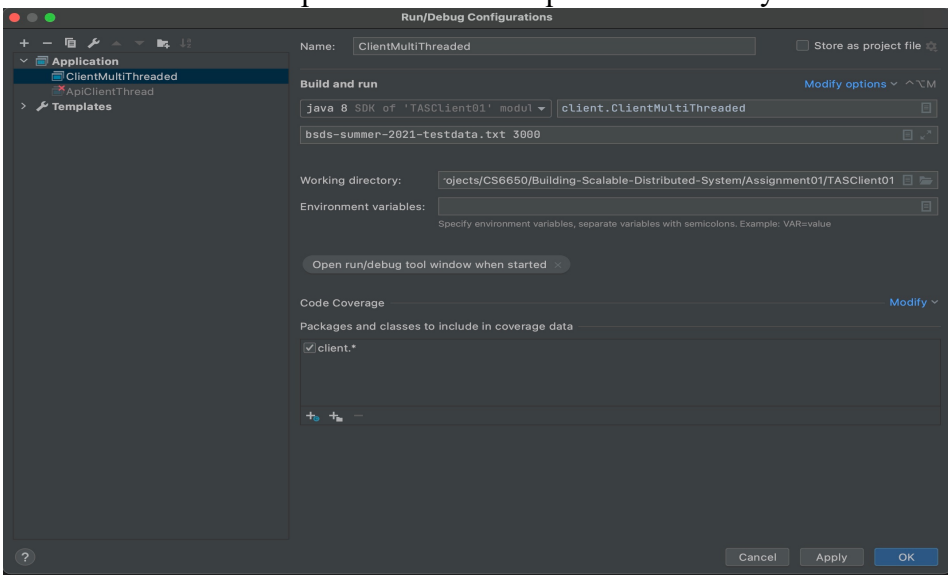


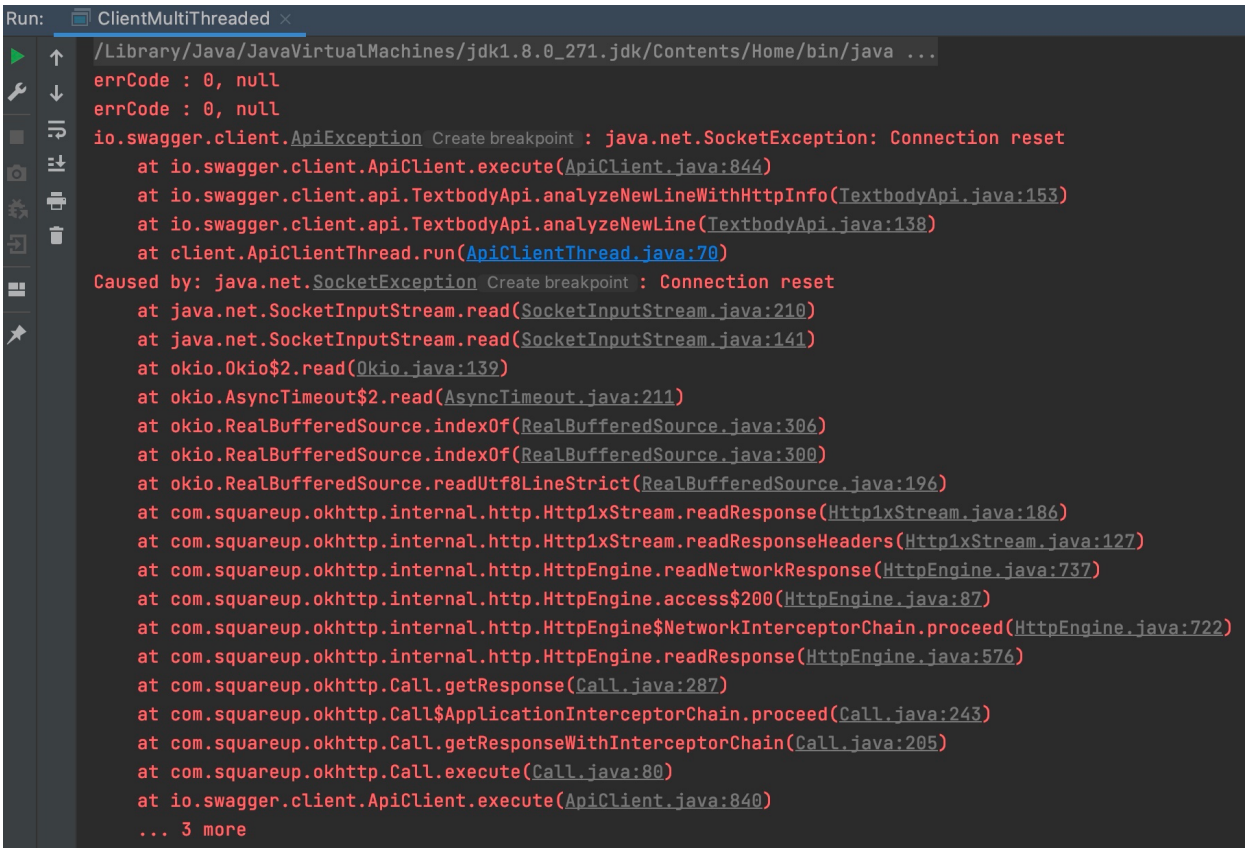
5. Break things

Try 2000 threads, nothing happens except the decreased throughput due to intensive context switching between threads.

```
All 2000 threads processing completed!
-------------------------------------------------------
------------------------STATS--------------------------
Total number of successful requests: 9930
Total number of unsuccessful requests: 0
Total wall time: 5551 ms
Throughput: 1788.87
```

Try 3000 threads, it fails in one-third of the run. It return code 0 with a null body. The Connection Reset Exception means the request is refused by the server.

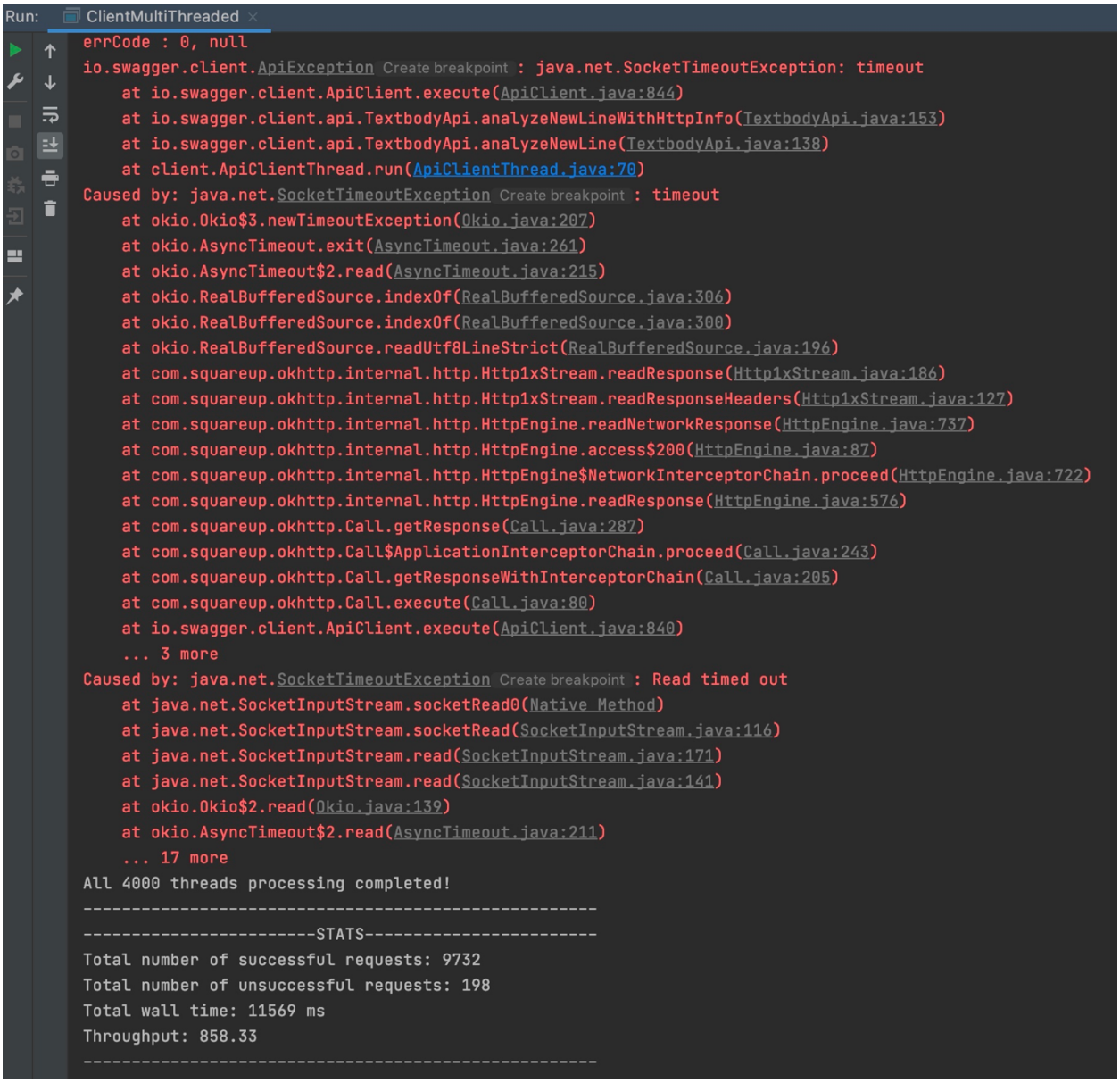




And there are 35 unsuccessful requests.

```
All 3000 threads processing completed!
-------------------------------------------------------
------------------------STATS--------------------------
Total number of successful requests: 9895
Total number of unsuccessful requests: 35
Total wall time: 11478 ms
Throughput: 865.13
```

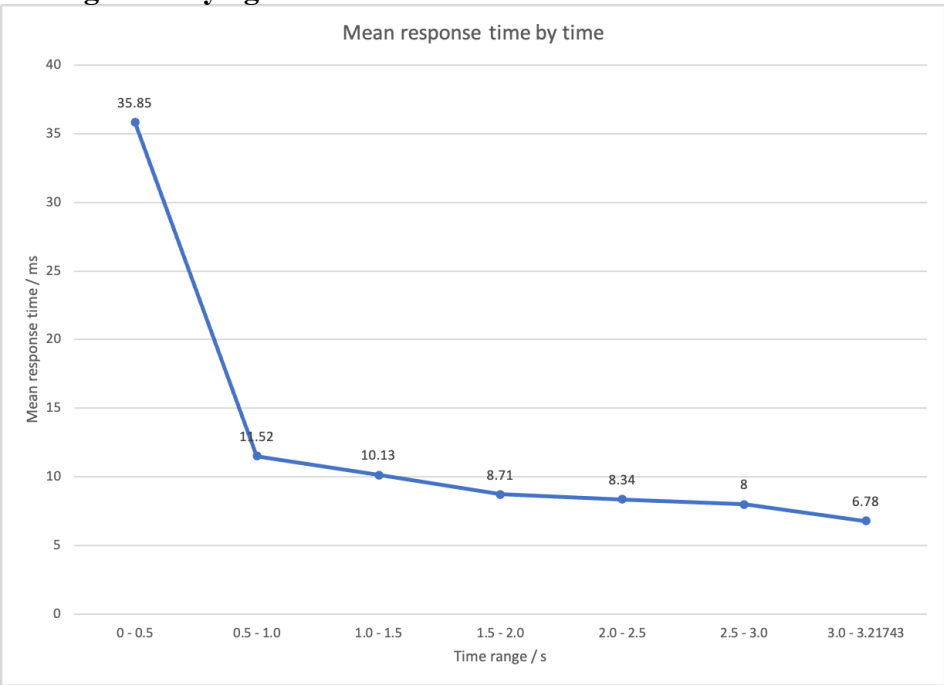
Try 4000 threads, the unsuccessful requests increase.
And many connections reset error.
Therefore, my server can support about less than 3000 client threads.



6. Charting

I experiment with 32 threads on client2 and generate two charts: one for average latency against time and the other for throughput against time. These two plots are genearted based on the data collected for every request in the **statistics.csv** file. The time interval is 0.5s.

Average latency against time



Average throughput against time