

Client Description

This program is for CS6650 assignment 1. It utilizes HttpClient to build a multi-thread client to make POST requests to a specified URL, and test the performance of the server.

The repository url: <https://github.com/tjuqxb/6650-assignment-1>

Server url: http://34.216.68.235/LAB2_2_war

How it works

The program entry point Class is ClientService. The typical way of running the program (both part 1 and part 2) is specifying the parameters in parameters.xml file (another way is using command line). The user needs to specify:

- arg0: maximum number of threads to run (numThreads - max 1024)
- arg1: number of skier to generate lift rides for (numSkiers - max 100000), This is effectively the skier's ID (skierID)
- arg2: number of ski lifts (numLifts - range 5-60, default 40)
- arg3: mean numbers of ski lifts each skier rides each day (numRuns - default 10, max 20)
- arg4: IP address and port number of the server

Part 1 client would only produce output window results. Part 2 client would produce output window results, a CSV file and a time graph. Users can use class ImageCreator to create other graphs based on output window results.

Main classes

ClientService: This is the entry point Class for the program. It would accept user specified parameters, generate multiple SingleClientThread threads, and use a thread pool and multiple CountdownLatches to control the whole process of the 3 phases. After execution of all the generated threads, it would gather information (the thread results are returned using Future) and generate the final results.

SingleClientThread: This is the class of a single thread client using HttpClient to make multiple requests to specified URL (or generated URLs). It contains one instance of URLBuilder, which is responsible for generating dynamic POST URLs, and one instance of RandInfoCreator, which is responsible for generating random information for a POST. It would record the number of successful and failed requests, the latency of each record, the maximum latency and the minimum latencies and the start and end time (implemented in part 2).

URLBuilder: This class can build URL string based on input parameters.

RandInfoCreator: This class can generate random information and create a SkierPost object based on the information.

ImageCreator: This is a utility class using JFreeChart to produce images.

Utils: This class contains static common utility functions.

Parameters: The wrapper class for mapping XML parameters.

ThreadInfo: The wrapper class for storing thread returned information, including the number of successful and failed requests, the latency of each record, the maximum latency and the minimum latencies and the start and end time (implemented in part 2).

SkierPost: This class can be mapped to a standard POST JSON object specified by the skier API.

Packages

domain: This package is used to store classes which can be mapped to standard POST JSON objects specified by the API. It currently contains one class: SkierPost.

Calculation of Throughput

Using single thread to test, the average latency is about 28 ms and the throughput is about $(1000/28 = 35.7/s)$. Also, according to Jmeter test, the maximum throughput of the naive server is about 2500/s to 3000/s.

Assuming the number of max threads is N. The total number of requests is M.

There are total 3 phases, phase 1 would use $0.25N$ threads to send $2/9$ M requests, phase 2 would use $0.6N$ threads to send $2/3$ M requests and phase 3 would use $0.1N$ threads to send $1/9$ M requests.

According to test, when 20% of total number of threads of one phase have finished, more than 95% of the total number of the requests of that phase also have been sent. (It is reasonable as all the threads of one phase would have the same priorities and at any point they would almost have sent the same number of requests). So, it is reasonable to calculate the wall time of the 3 phases and add them up for prediction.

There are total 180000 requests. Use average latency = 28 ms to estimate.

Using Little's law, we can get a function: $N = \text{Throughput} * W$ (NumberOfThreads = Throughput * Latency).

$\text{Throughput} = \text{NumberOfThreads} / \text{Latency}$

For 32 threads:

$\text{phase1-throughput} = 32 * 0.25 * 1000 / 28 = 285$

$\text{phase2-throughput} = 32 * 1000 / 28 = 1143$

$\text{phase3-throughput} = 32 * 0.1 * 1000 / 28 = 114$

$\text{walltime} = 40000 / 285 + 120000 / 1143 + 20000 / 114 = 420.8s$

$\text{throughput} = 180000/420.8 = 428$

For 64 threads:

$\text{phase1-throughput} = 64 * 0.25 * 1000 / 28 = 570$

$\text{phase2-throughput} = 64 * 1000 / 28 = 2286$

$\text{phase3-throughput} = 64 * 0.1 * 1000 / 28 = 228$

$\text{walltime} = 40000 / 570 + 120000 / 2286 + 20000 / 228 = 210.4s$

$\text{throughput} = 180000/210.4 = 856$

For 128 threads:

phase1-throughput = $128 * 0.25 * 1000 / 28 = 1140$

phase2-throughput = $\min\{128 * 1000 / 28, 2500\} = \max\{4572, 2500\} = 2500$

phase3-throughput = $128 * 0.1 * 1000 / 28 = 456$

walltime = $40000 / 1140 + 120000 / 2500 + 20000 / 456 = 126.9s$

throughput = $180000 / 126.9 = 1418$

For 256 threads:

phase1-throughput = $256 * 0.25 * 1000 / 28 = 2280$

phase2-throughput = $\min\{256 * 1000 / 28, 2500\} = \max\{9144, 2500\} = 2500$

phase3-throughput = $256 * 0.1 * 1000 / 28 = 912$

walltime = $40000 / 2280 + 120000 / 2500 + 20000 / 912 = 87.5s$

throughput = $180000 / 87.5 = 2057$

Output and Charts

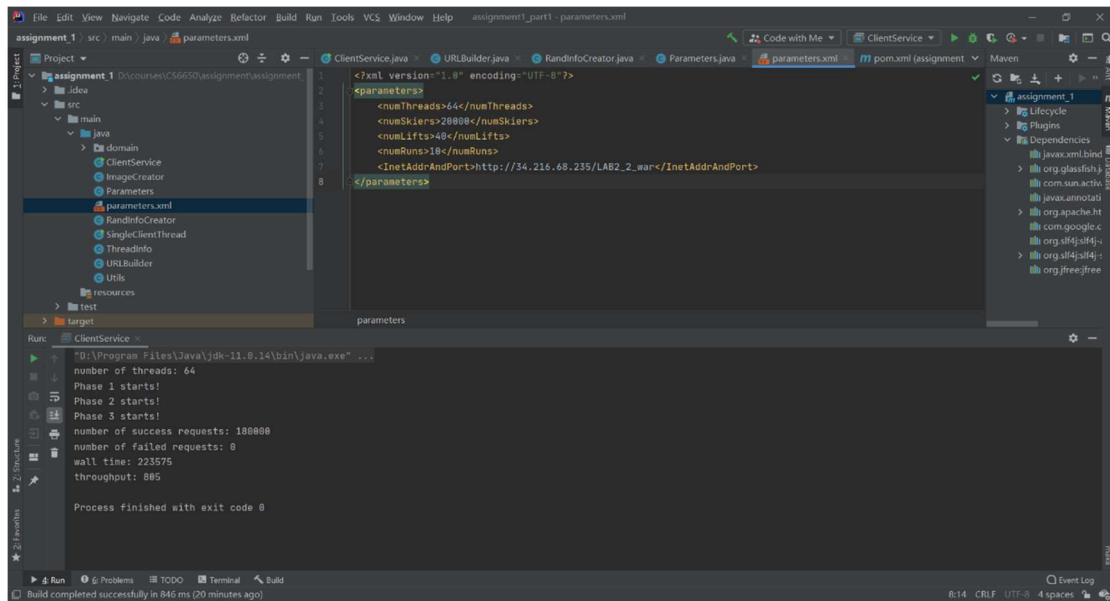
Part1:

```
assignment_1 - ClientService.java
Project: assignment_1
src
  main
    java
      ClientService.java
      ThreadInfo.java
      URIBuilder.java
      RandInfoCreator.java
      Parameters.java
      parameters.xml
      pom.xml (assignment_1)
  test
    java
      ClientPost.java
      Parameters.java
      RandInfoCreator.java
      SingleClientThread.java
      URIBuilder.java
      Utils
  resources
  target
  pom.xml

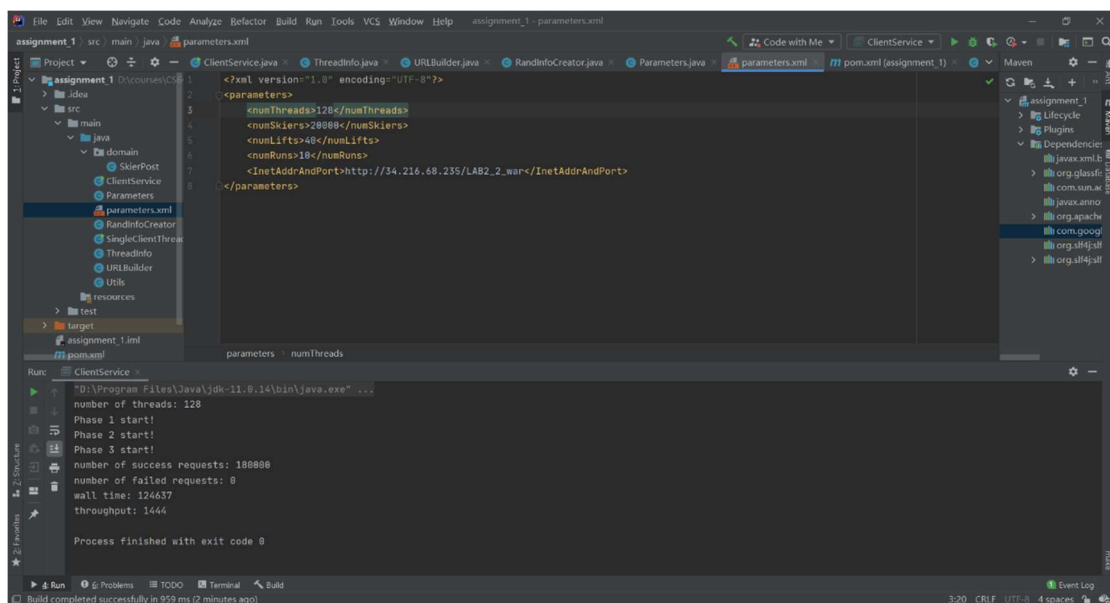
Run: ClientService
"D:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...
number of threads: 32
Phase 1 start!
Phase 2 start!
Phase 3 start!
number of success requests: 189000
number of failed requests: 0
wall time: 431425
throughput: 417

Process finished with exit code 0
```

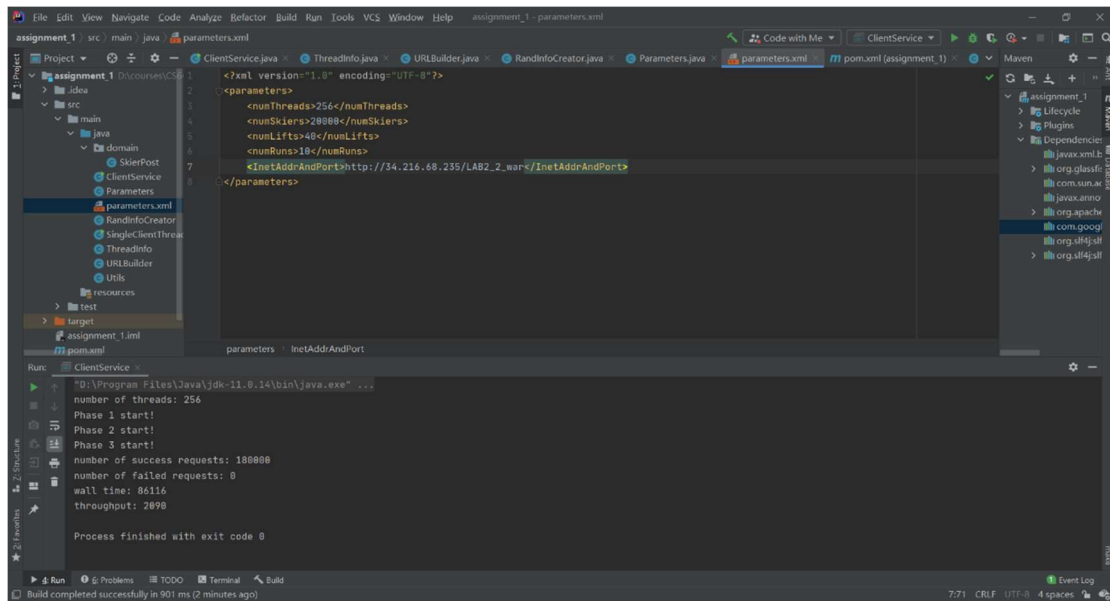
32 Threads



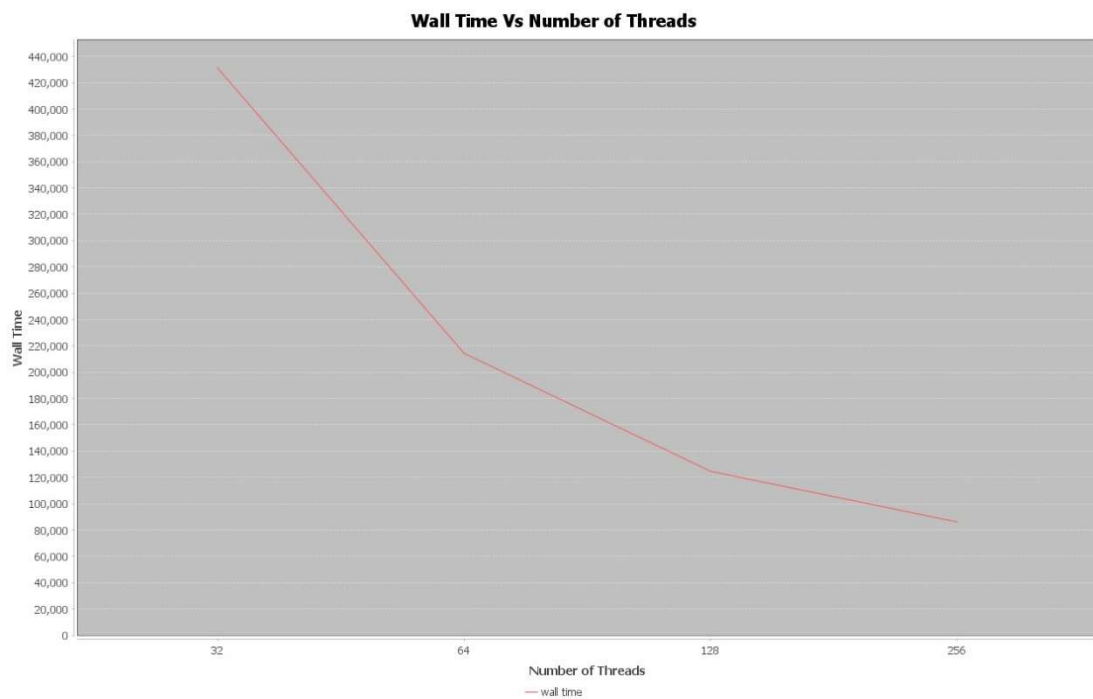
64 Threads



128 Threads



256 Threads



Walltime VS Number of Threads

Part2:

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help assignment1_part2 - ClientService.java
assignment1_part2 | src | main | java | ClientService | processRecordsAndWriteFile
Project | ClientService.java | ThreadInfo.java | Utils.java | Parameters.java | URLBuilder.java | ImageCreator.java | SingleClientThread.java | Maven
assignment1_part2 | D:\courses\CS669 |
src |
main |
java |
domain |
ClientService |
ImageCreator |
Parameters |
Run: ClientService
"D:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...
number of threads: 32
Phase 1 starts!
Phase 2 starts!
Phase 3 starts!
number of success requests: 100000
number of failed requests: 0
wall time: 417226
throughput: 431
mean response time: 28.8 milliseconds
median response time: 26.0 milliseconds
P99 response time: 98 milliseconds
min response time: 10 milliseconds
max response time: 3501 milliseconds
Process finished with exit code 0
Build completed successfully in 2 s 222 ms (11 minutes ago)
```

32 Threads

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help assignment1_part2 - ClientService.java
assignment1_part2 | src | main | java | ClientService | processRecordsAndWriteFile
Project | ClientService.java | ThreadInfo.java | Utils.java | Parameters.java | URLBuilder.java | ImageCreator.java | SingleClientThread.java | Maven
assignment1_part2 | D:\courses\CS669 |
src |
main |
java |
domain |
ClientService |
ImageCreator |
Parameters |
parameters.xml |
RandomInfoCreator |
SingleClientThread |
ThreadInfo |
URLBuilder |
Utils |
resources |
test |
Run: ClientService
"D:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...
number of threads: 64
Phase 1 starts!
Phase 2 starts!
Phase 3 starts!
number of success requests: 100000
number of failed requests: 0
wall time: 227867
throughput: 789
mean response time: 32.2 milliseconds
median response time: 28.0 milliseconds
P99 response time: 82 milliseconds
min response time: 11 milliseconds
max response time: 4901 milliseconds
Process finished with exit code 0
All files are up-to-date (43 minutes ago)
```

64 Threads

```
private static void createTimeGraphFromMap(HashMap<Long, List<Long>> timeRecords) throws IOException {  
    int[] sumRecords = new int[(endTime - startTime)/1000 + 1];  
    int[] cntRecords = new int[(endTime - startTime)/1000 + 1];  
    for (Long k: timeRecords.keySet()) {  
        List<Long> list = timeRecords.get(k);  
        long sum = 0;  
        for (int i = 0; i < list.size(); i++) {  
            sum += list.get(i)[0];  
        }  
        sumRecords[(int)((k - startTime)/1000)] += sum;  
        cntRecords[(int)((k - startTime)/1000)] += list.size();  
    }  
}
```

Run: ClientService

```
"D:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...  
number of threads: 128  
Phase 1 starts!  
Phase 2 starts!  
Phase 3 starts!  
number of success requests: 100000  
number of failed requests: 0  
wall time: 124055  
throughput: 1458  
mean response time: 39.2 milliseconds  
median response time: 32.0 milliseconds  
P99 response time: 94 milliseconds  
min response time: 10 milliseconds  
max response time: 5919 milliseconds  
Process finished with exit code 0
```

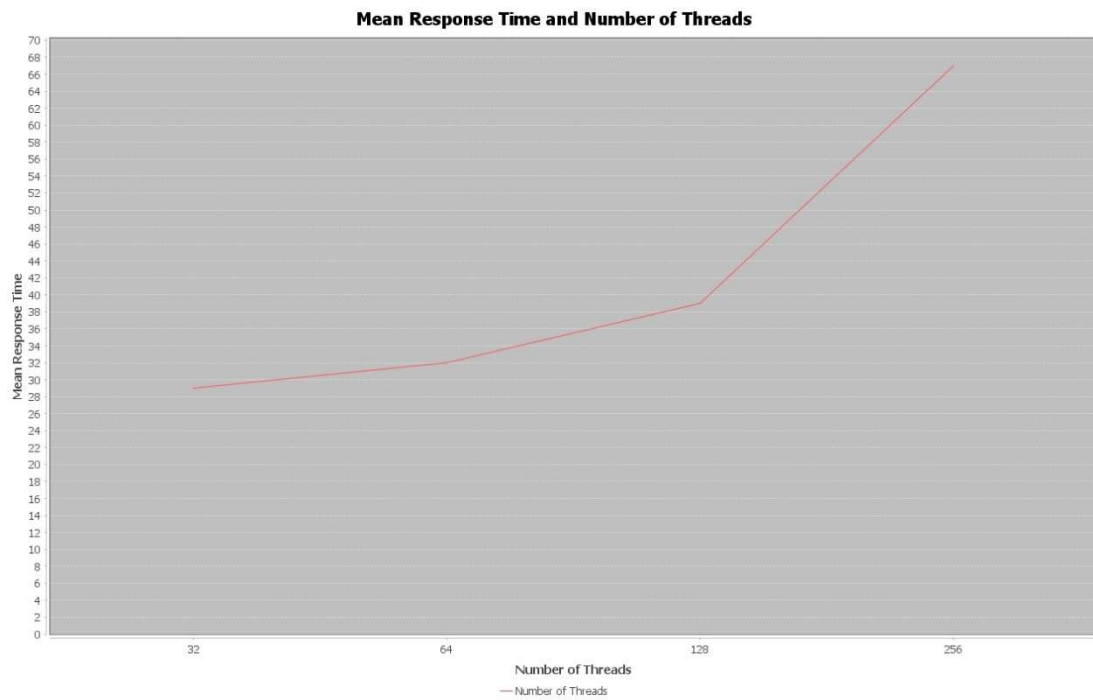
128 Threads

```
for (int i = 0; i < sumRecords.length; i++) {  
    if (cntRecords[i] != 0) {  
        sumRecords[i] = sumRecords[i] / cntRecords[i];  
    }  
}  
ImageCreator.createTimeGraph(sumRecords, idName: String.valueOf(numThreads) + " Threads");  
  
private static boolean validateCommands(Integer[] argsRec) {  
    if (argsRec[0] == null || argsRec[0] < 1 || argsRec[0] > 1024) return false;  
    if (argsRec[1] == null || argsRec[1] < 1 || argsRec[1] > 100000) return false;  
}
```

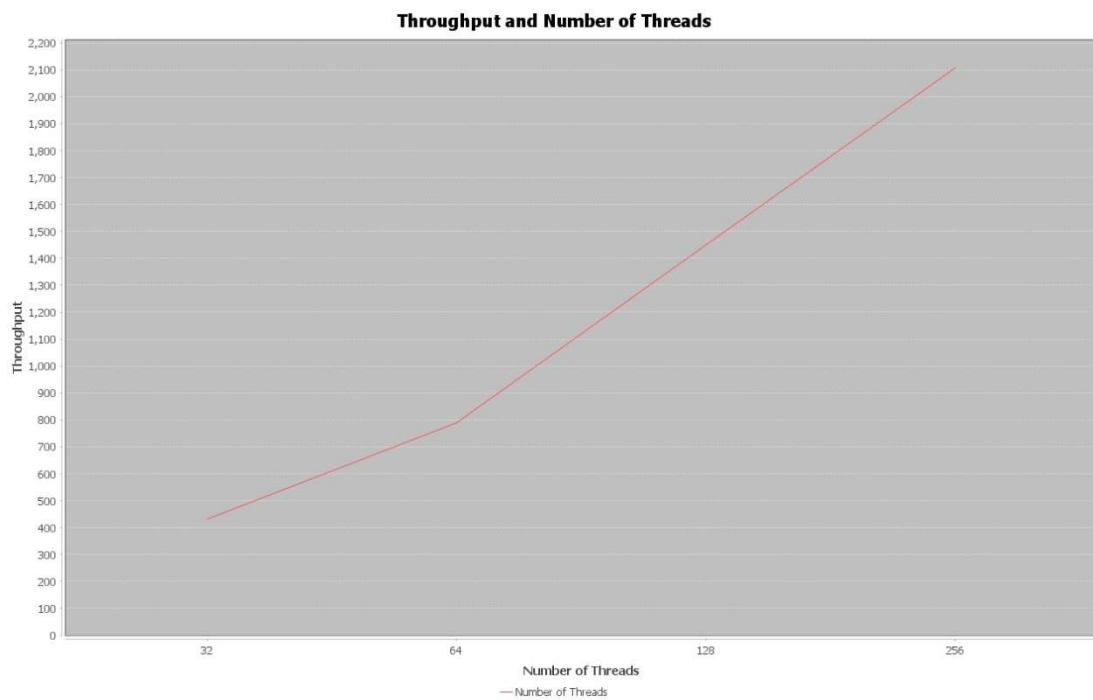
Run: ClientService

```
"D:\Program Files\Java\jdk-11.0.14\bin\java.exe" ...  
number of threads: 256  
Phase 1 starts!  
Phase 2 starts!  
Phase 3 starts!  
number of success requests: 100000  
number of failed requests: 0  
wall time: 85386  
throughput: 2108  
mean response time: 66.7 milliseconds  
median response time: 47.0 milliseconds  
P99 response time: 145 milliseconds  
min response time: 11 milliseconds  
max response time: 7698 milliseconds  
Process finished with exit code 0
```

256 Threads

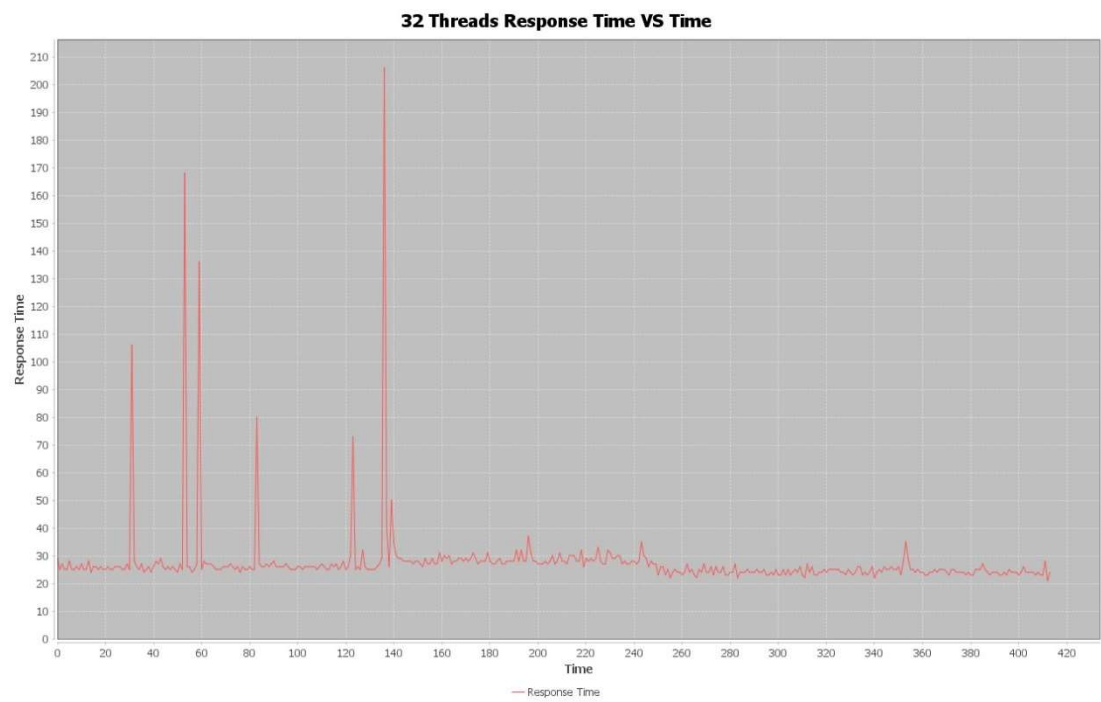


Mean Response Time VS Number of Threads

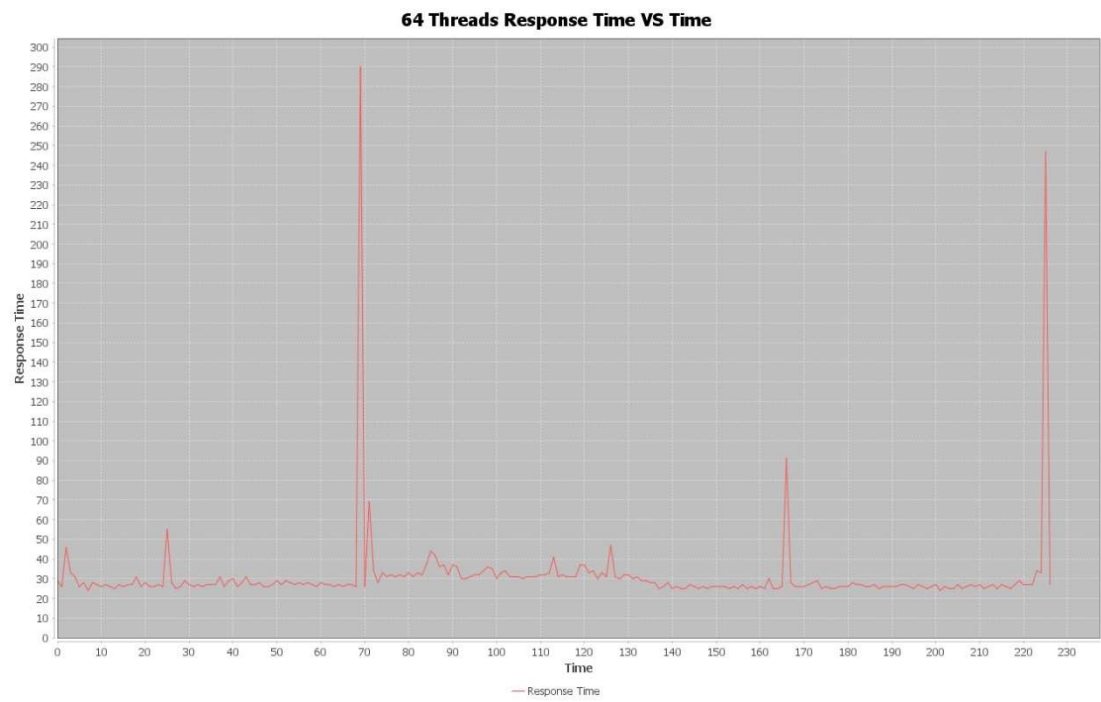


Throughput VS Number of Threads

Part 2 Time Charts:

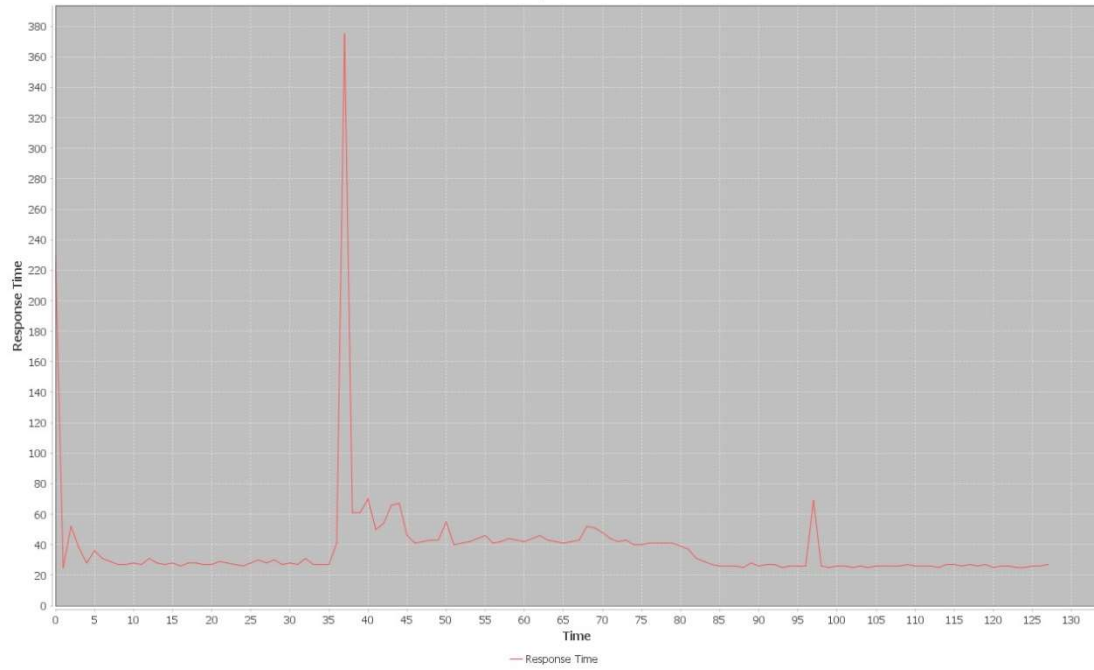


32 Threads



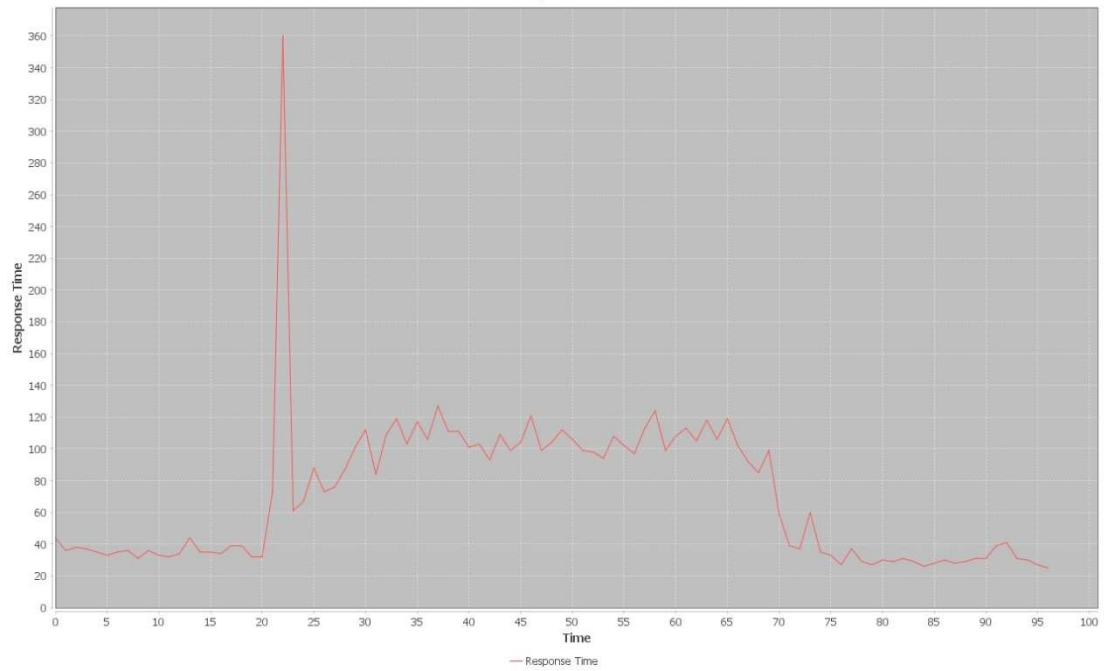
64 Threads

128 Threads Response Time VS Time



128 Threads

256 Threads Response Time VS Time



256 Threads