Homework 4a Report

Random Group:

Yinuo Feng, Nuanxin Jin, boyuan sun, Zihe Chen, Chenyu Jiang

1. GitHub link

Github link: https://github.khoury.northeastern.edu/nightfriday/HW4_Random

2. Client Description

Dependencies

The project uses the following dependencies:

- Apache HttpClient 5: For making HTTP requests.

- Apache Spark: For data processing and analysis.

- Java Standard Library: For core functionalities such as file handling, concurrency, and I/O operations.

Class Structure

1. Main:

- Main takes four parameters the threadGroupSize, numGroup, delay as in second, and URI of the server

- Initializes HTTP client. HTTP client uses closeableHttpClient that has retry strategy according to description. The HTTP client also use PoolingHttpClientConnectionManager to manage connections

- Initialize Spark as a table.

- Start and manage the execution of multiple threads for sending HTTP requests, using countDownLatch to wait all threads to complete.

- Processes latency data using Spark.

2. ClientPost:

- Implements `Runnable`.
- Construct Multipart form data using builder
- Sends multipart POST requests to upload files and associated data.
- Retrieve and parse response data
- Records latency and status code of each request, append them to a list as Row object.

4. ClientGet:

- Implements `Runnable`.
- Sends GET requests to retrieve album information.
- Retrieve and parse response data.
- Records latency and status code of each request.

General Flow of the Program

1. Initialization:

- The `Main` class initializes the one HTTP client with a connection manager and retry strategy.
- A Spark session is created for data processing.
- Main will instantiate one File, one ClientGet, one ClientPost, one ArrayList (for data recoding) object

2. Execution:

- The program creates and starts multiple threads using lambda function to call ClientGet.run() and ClientPost.run() in a loop to send HTTP requests.
- In the .run() methods, when response is received, the method will convert latency data to Row object and append to a list

3. Data Collection and Analysis:

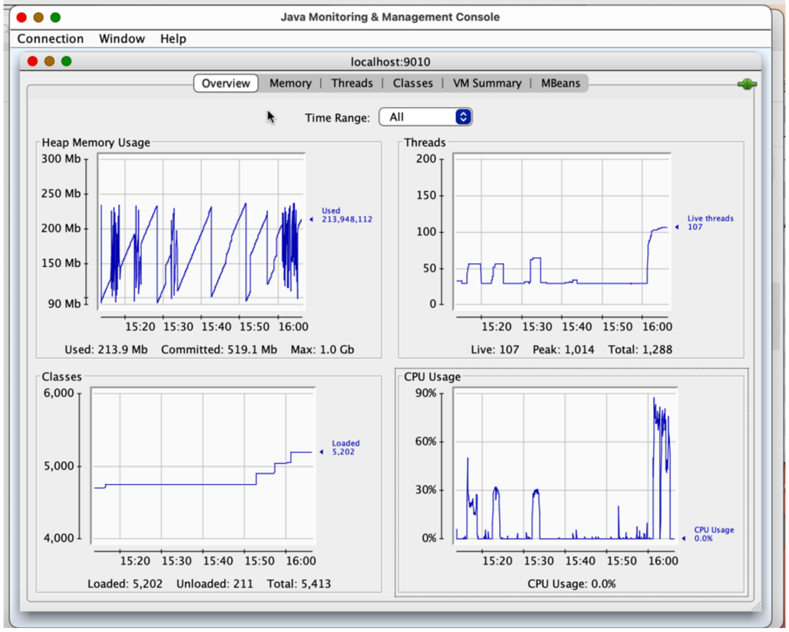
- After all requests are completed, the data is converted into a Spark DataFrame.

- Mean, min, max, percentiles are calculated for GET and POST requests.
- The results are printed to the console and saved to a CSV file.

3. Client Part 1:

3.1 Java Servlet with Tomcat:

threadGroupSize = 10, numThreadGroups = 10, delay = 2 (seconds)

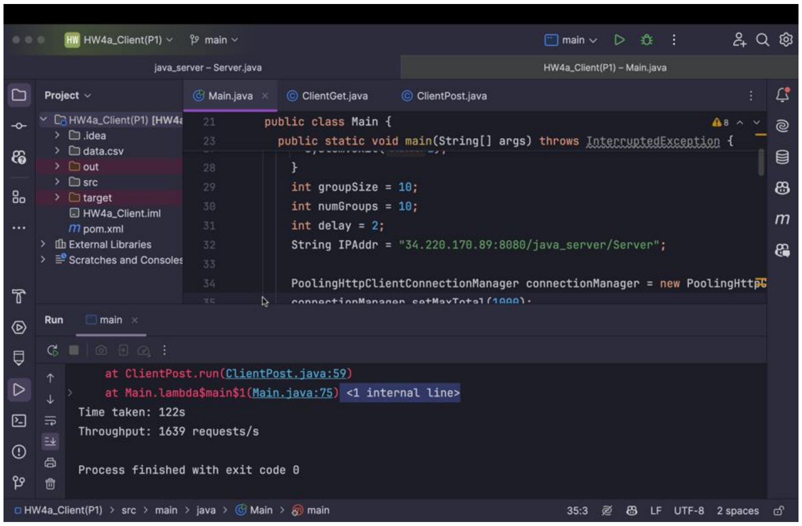


Used: 213.9 Mb Committed: 519.1 Mb Max: 1.0 Gb

Live: 107 Peak: 1,014 Total: 1,288

Loaded: 5,202 Unloaded: 211 Total: 5,413

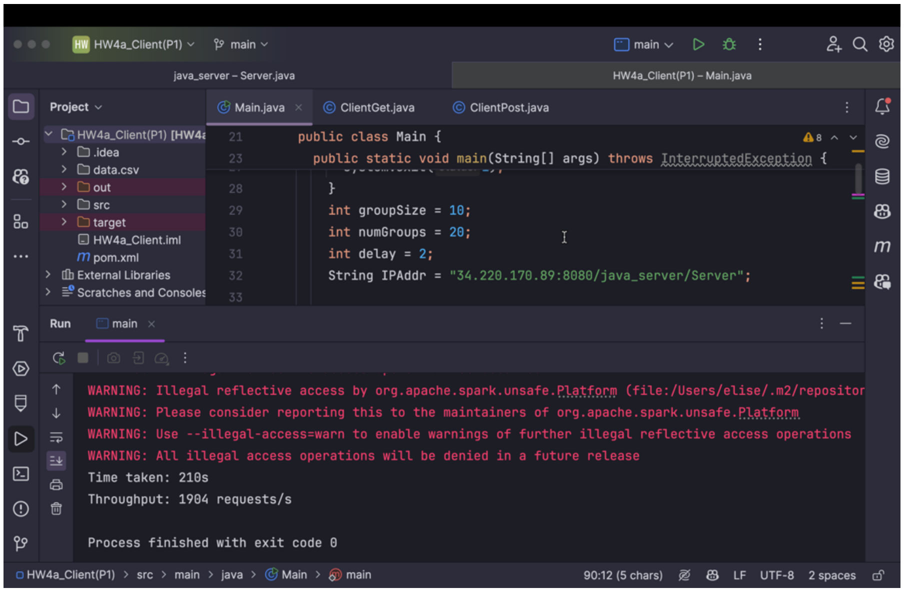
CPU Usage: 0.0%



```java
public class Main {
  public static void main(String[] args) throws InterruptedException {
    }
    int groupSize = 10;
    int numGroups = 10;
    int delay = 2;
    String IPAddr = "34.220.170.89:8080/java_server/Server";

    PoolingHttpClientConnectionManager connectionManager = new PoolingHttp
    connectionManager.setMaxTotal(1000);
```

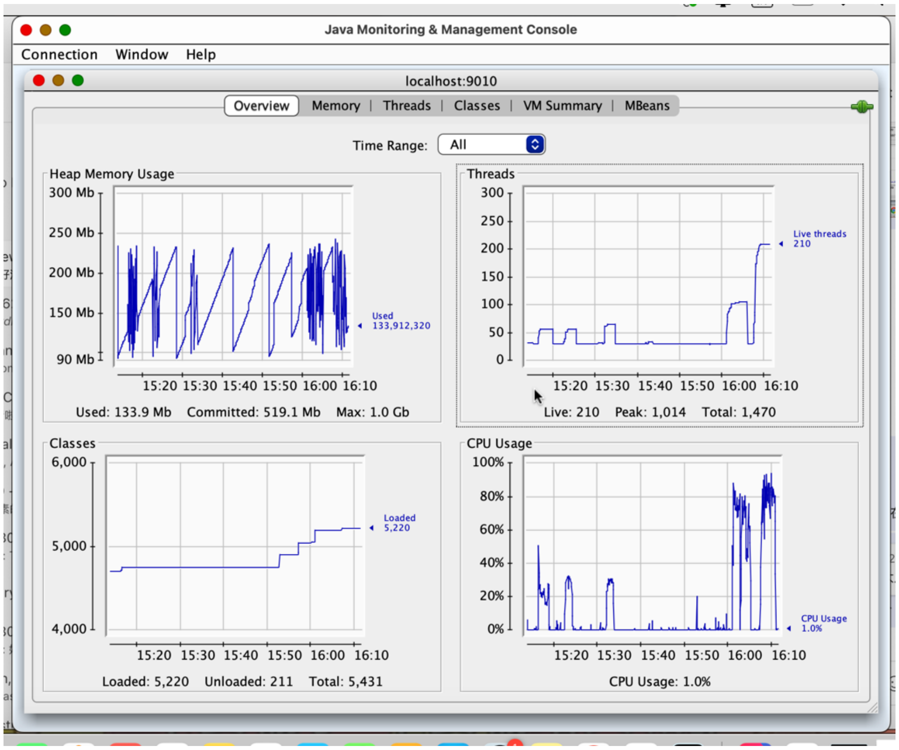
```
        at ClientPost.run(ClientPost.java:59)
        at Main.lambda$main$1(Main.java:75) <1 internal line>
Time taken: 122s
Throughput: 1639 requests/s

Process finished with exit code 0
```

threadGroupSize = 10, numThreadGroups = 20, delay = 2 (seconds)



```
public class Main {
  public static void main(String[] args) throws InterruptedException {
    }
    int groupSize = 10;
    int numGroups = 20;
    int delay = 2;
    String IPAddr = "34.220.170.89:8080/java_server/Server";
```

```
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/Users/elise/.m2/repositor
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Time taken: 210s
Throughput: 1904 requests/s

Process finished with exit code 0
```



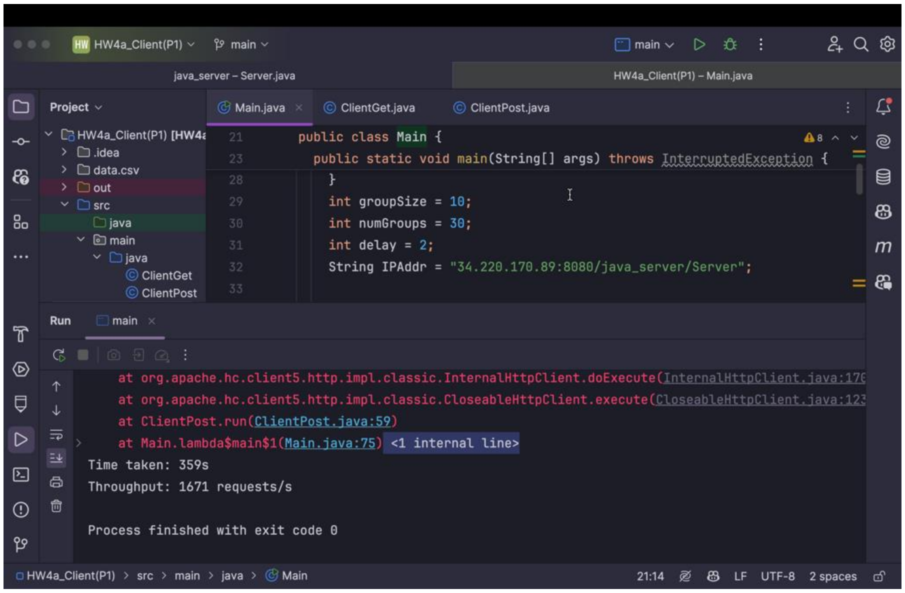
Used: 133.9 Mb Committed: 519.1 Mb Max: 1.0 Gb

Live: 210 Peak: 1,014 Total: 1,470

Loaded: 5,220 Unloaded: 211 Total: 5,431

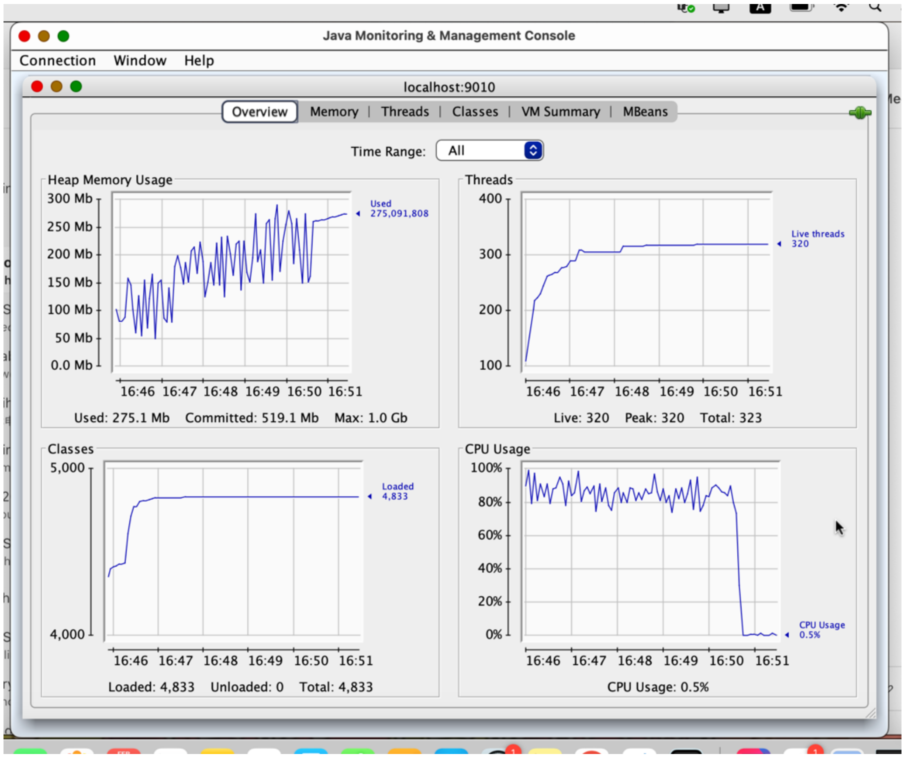
CPU Usage: 1.0%

threadGroupSize = 10, numThreadGroups = 30, delay = 2 (seconds)



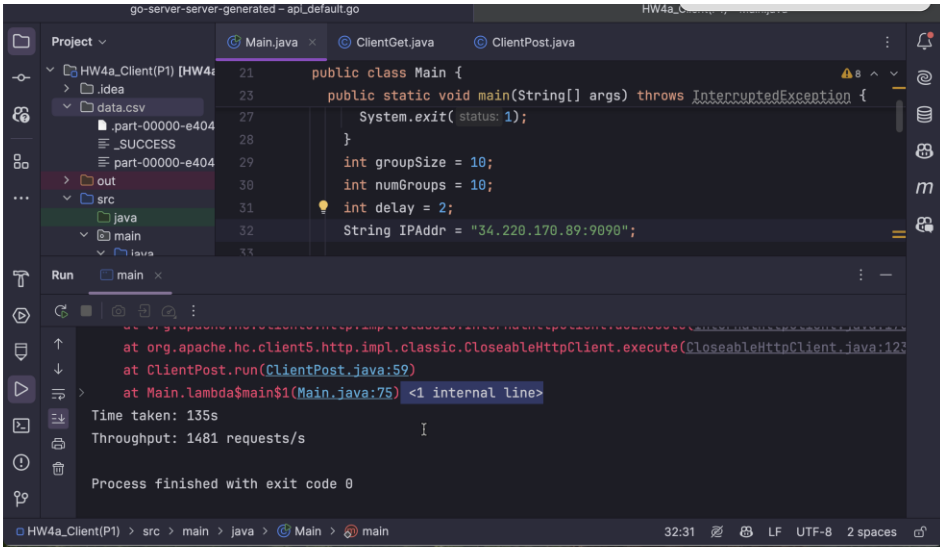
```
public class Main {
    public static void main(String[] args) throws InterruptedException {
        }
        int groupSize = 10;
        int numGroups = 30;
        int delay = 2;
        String IPAddr = "34.220.170.89:8080/java_server/Server";
```

```
        at org.apache.hc.client5.http.impl.classic.InternalHttpClient.doExecute(InternalHttpClient.java:17
        at org.apache.hc.client5.http.impl.classic.CloseableHttpClient.execute(CloseableHttpClient.java:12
        at ClientPost.run(ClientPost.java:59)
        at Main.lambda$main$1(Main.java:75) <1 internal line>
Time taken: 359s
Throughput: 1671 requests/s

Process finished with exit code 0
```



3.2 Go server:

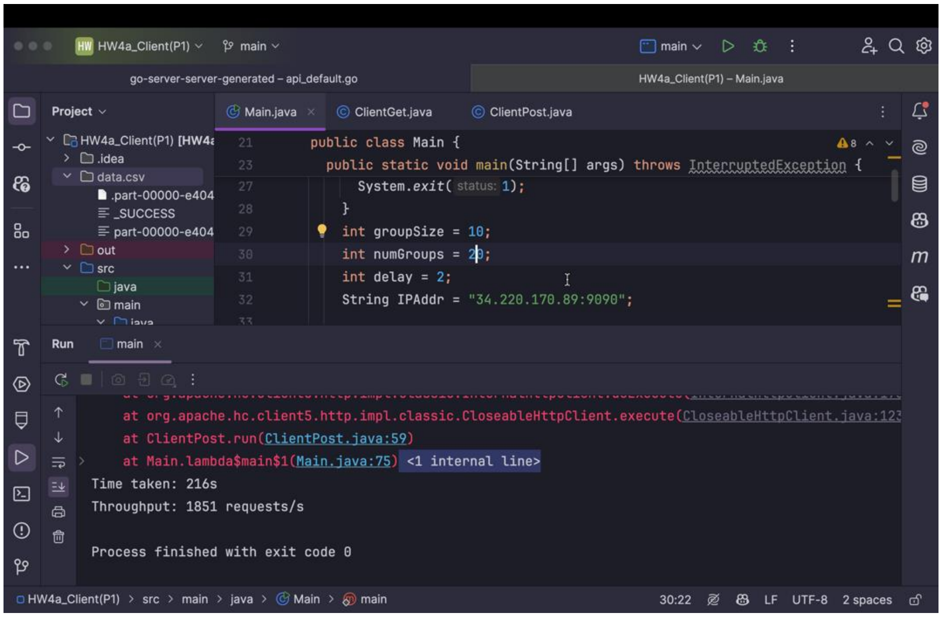
threadGroupSize = 10, numThreadGroups = 10, delay = 2 (seconds)



```
    public class Main {
      public static void main(String[] args) throws InterruptedException {
          System.exit( status: 1);
        }
        int groupSize = 10;
        int numGroups = 10;
        int delay = 2;
        String IPAddr = "34.220.170.89:9090";
```

```
    at org.apache.hc.client5.http.impl.classic.CloseableHttpClient.execute(CloseableHttpClient.java:123
    at ClientPost.run(ClientPost.java:59)
    at Main.lambda$main$1(Main.java:75) <1 internal line>
Time taken: 135s
Throughput: 1481 requests/s

Process finished with exit code 0
```

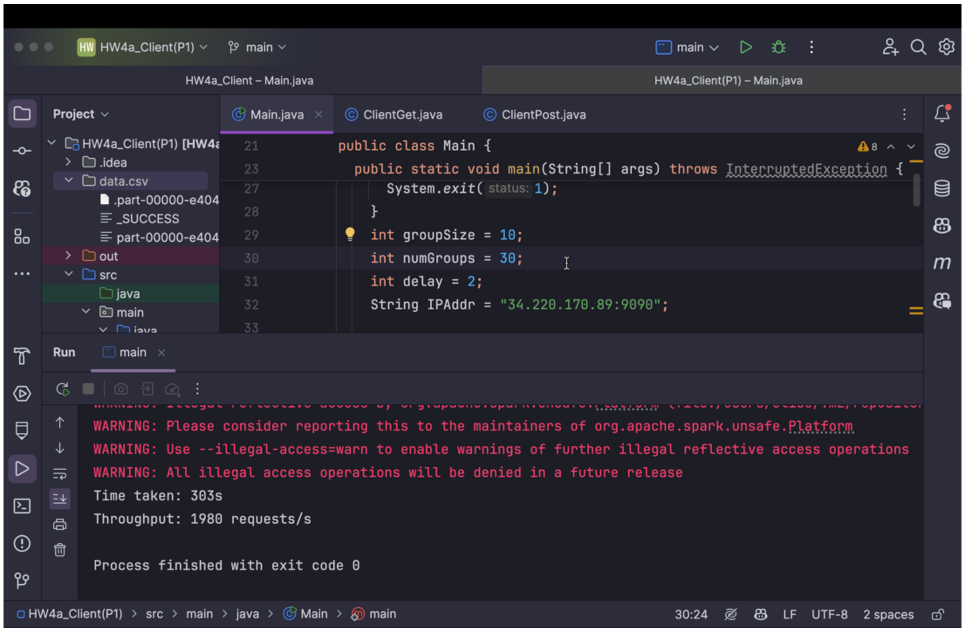
threadGroupSize = 10, numThreadGroups = 20, delay = 2 (seconds)



```
    public class Main {
      public static void main(String[] args) throws InterruptedException {
          System.exit( status: 1);
        }
        int groupSize = 10;
        int numGroups = 20;
        int delay = 2;
        String IPAddr = "34.220.170.89:9090";
```

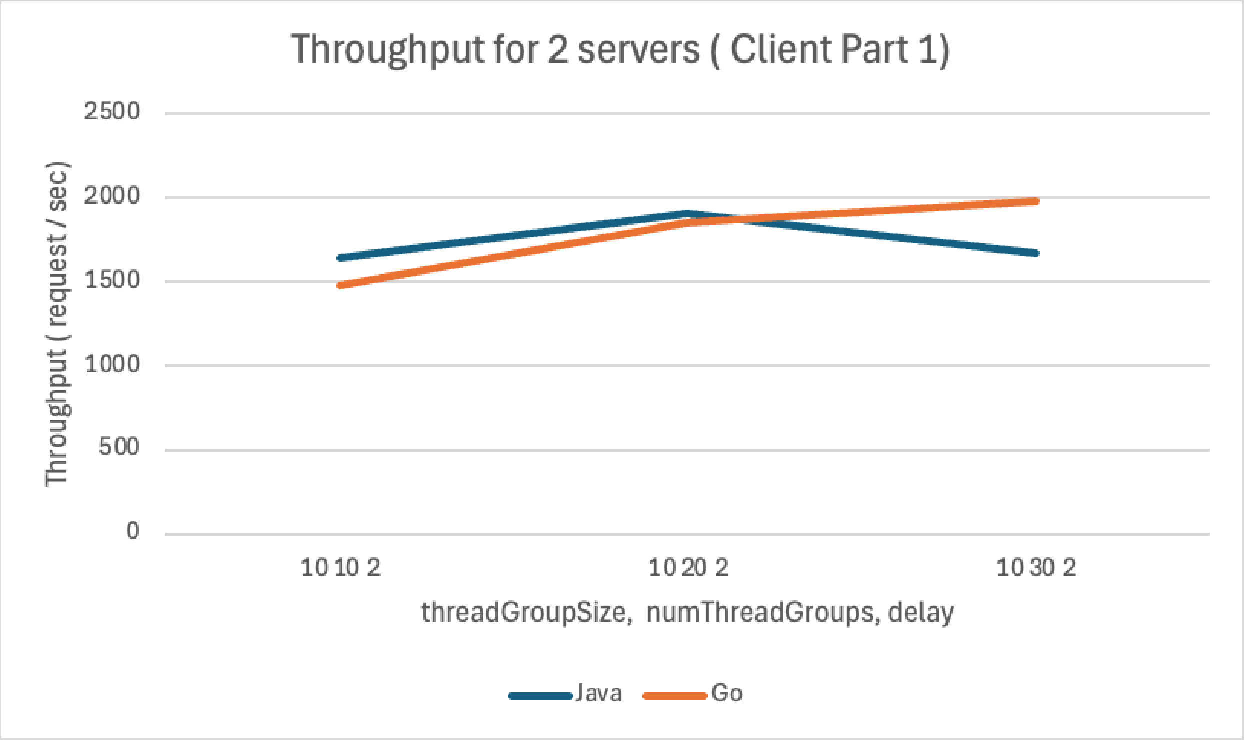
```
    at org.apache.hc.client5.http.impl.classic.CloseableHttpClient.execute(CloseableHttpClient.java:123
    at ClientPost.run(ClientPost.java:59)
    at Main.lambda$main$1(Main.java:75) <1 internal line>
Time taken: 216s
Throughput: 1851 requests/s

Process finished with exit code 0
```

threadGroupSize = 10, numThreadGroups = 30, delay = 2 (seconds)



```
public class Main {
  public static void main(String[] args) throws InterruptedException {
      System.exit( status: 1);
    }
    int groupSize = 10;
    int numGroups = 30;
    int delay = 2;
    String IPAddr = "34.220.170.89:9090";
```

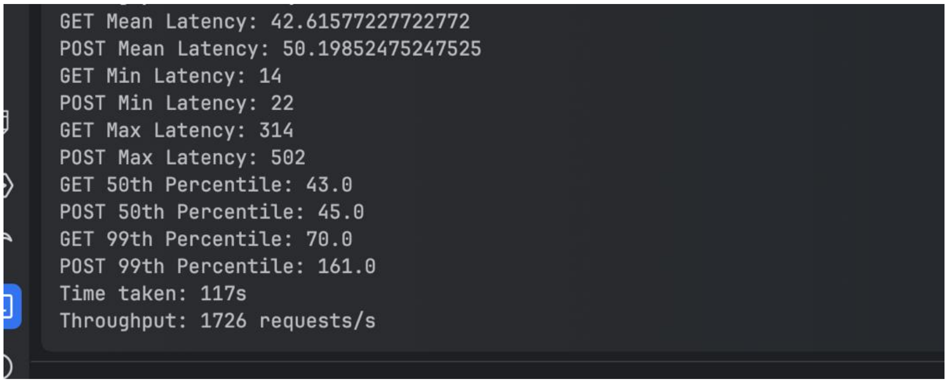
```
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Time taken: 303s
Throughput: 1980 requests/s

Process finished with exit code 0
```



4. Client Part 2:

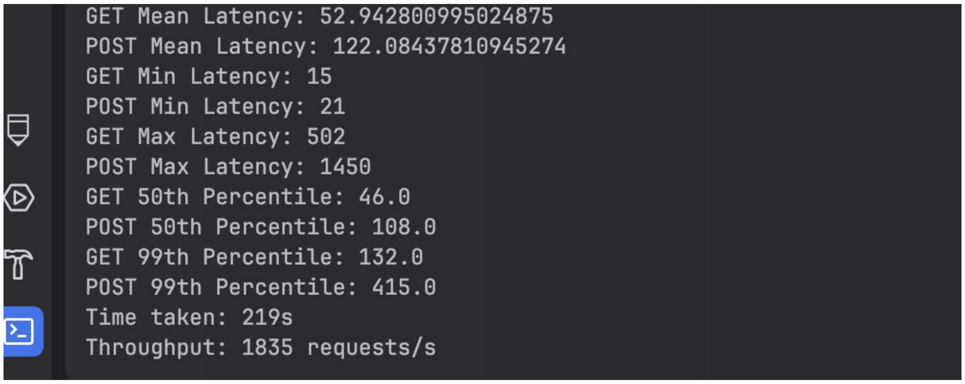
4.1 Java Servlet with Tomcat

threadGroupSize = 10, numThreadGroups = 10, delay = 2 (seconds)



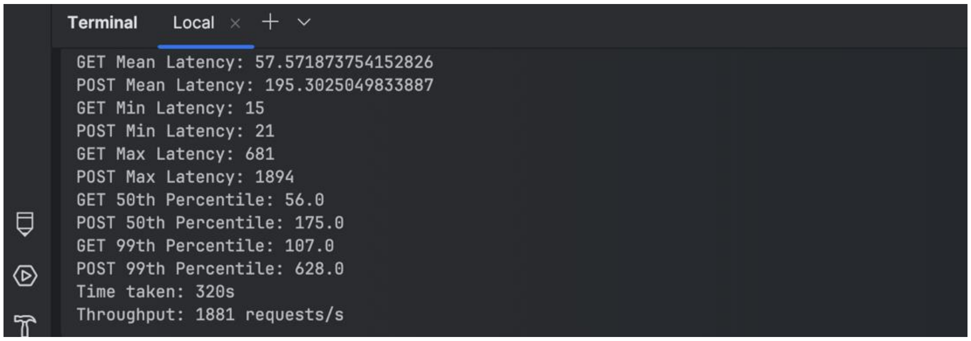
```
GET Mean Latency: 42.61577227722772
POST Mean Latency: 50.19852475247525
GET Min Latency: 14
POST Min Latency: 22
GET Max Latency: 314
POST Max Latency: 502
GET 50th Percentile: 43.0
POST 50th Percentile: 45.0
GET 99th Percentile: 70.0
POST 99th Percentile: 161.0
Time taken: 117s
Throughput: 1726 requests/s
```

threadGroupSize = 10, numThreadGroups = 20, delay = 2 (seconds)



```
GET Mean Latency: 52.942800995024875
POST Mean Latency: 122.08437810945274
GET Min Latency: 15
POST Min Latency: 21
GET Max Latency: 502
POST Max Latency: 1450
GET 50th Percentile: 46.0
POST 50th Percentile: 108.0
GET 99th Percentile: 132.0
POST 99th Percentile: 415.0
Time taken: 219s
Throughput: 1835 requests/s
```

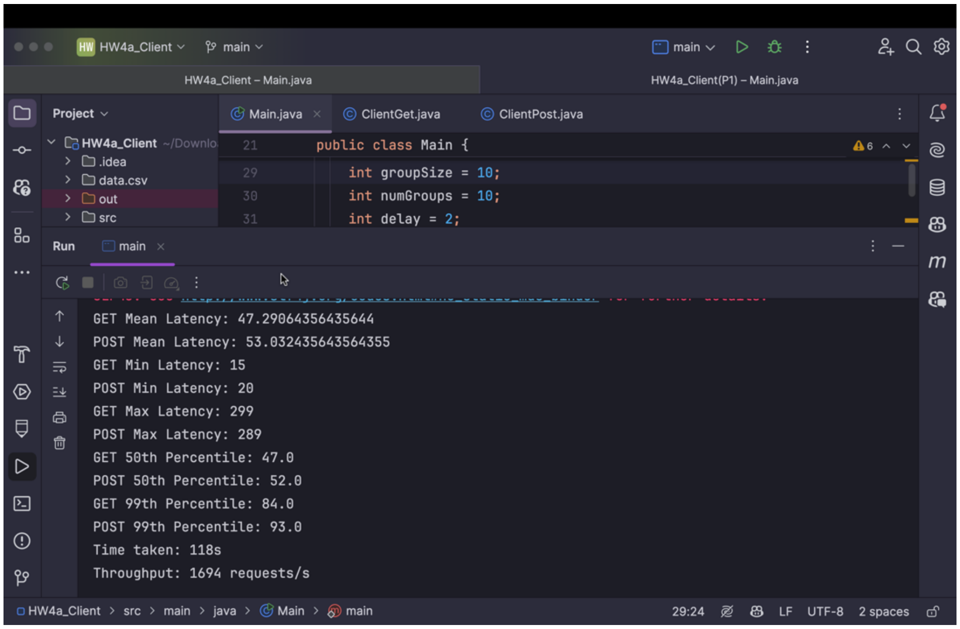
threadGroupSize = 10, numThreadGroups = 30, delay = 2 (seconds)



```
GET Mean Latency: 57.571873754152826
POST Mean Latency: 195.3025049833887
GET Min Latency: 15
POST Min Latency: 21
GET Max Latency: 681
POST Max Latency: 1894
GET 50th Percentile: 56.0
POST 50th Percentile: 175.0
GET 99th Percentile: 107.0
POST 99th Percentile: 628.0
Time taken: 320s
Throughput: 1881 requests/s
```

4.2 Go server

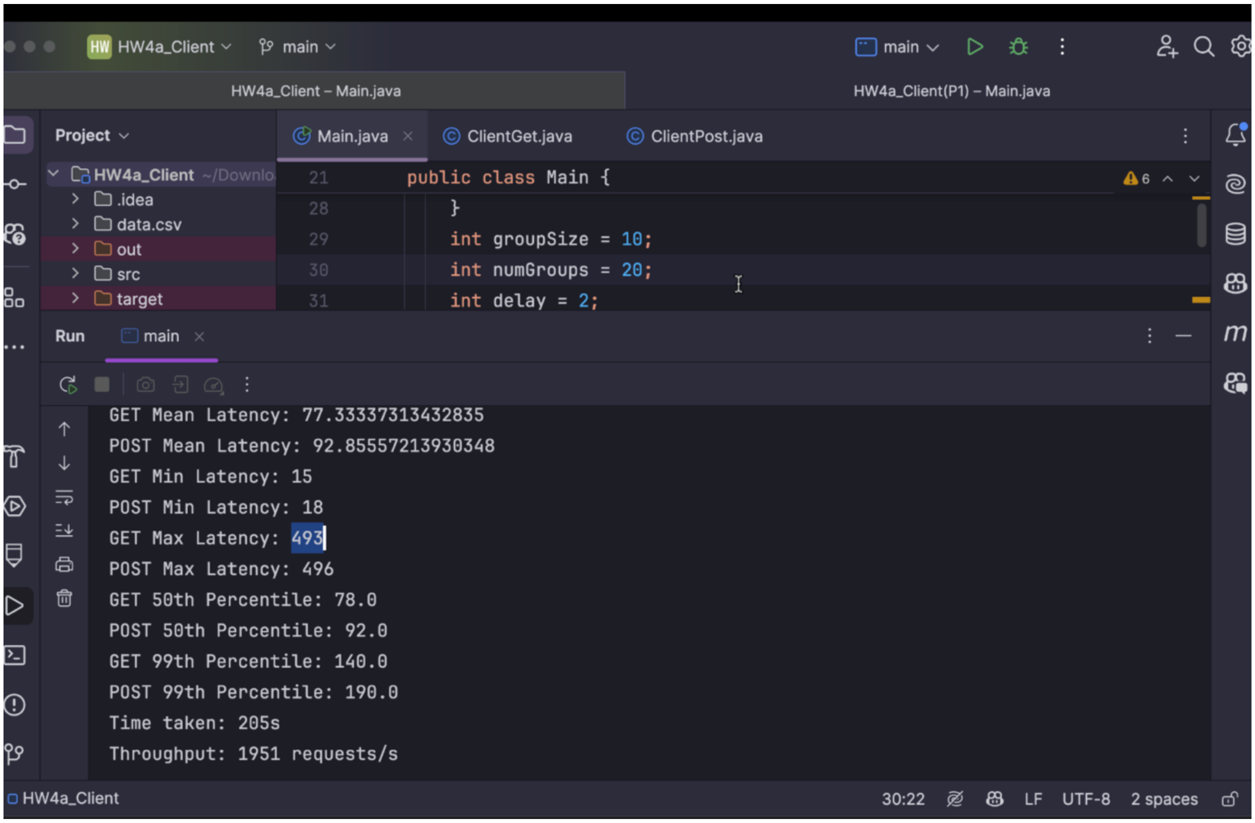
threadGroupSize = 10, numThreadGroups = 10, delay = 2 (seconds)



```java
public class Main {
    int groupSize = 10;
    int numGroups = 10;
    int delay = 2;
```

```
GET Mean Latency: 47.29064356435644
POST Mean Latency: 53.032435643564355
GET Min Latency: 15
POST Min Latency: 20
GET Max Latency: 299
POST Max Latency: 289
GET 50th Percentile: 47.0
POST 50th Percentile: 52.0
GET 99th Percentile: 84.0
POST 99th Percentile: 93.0
Time taken: 118s
Throughput: 1694 requests/s
```

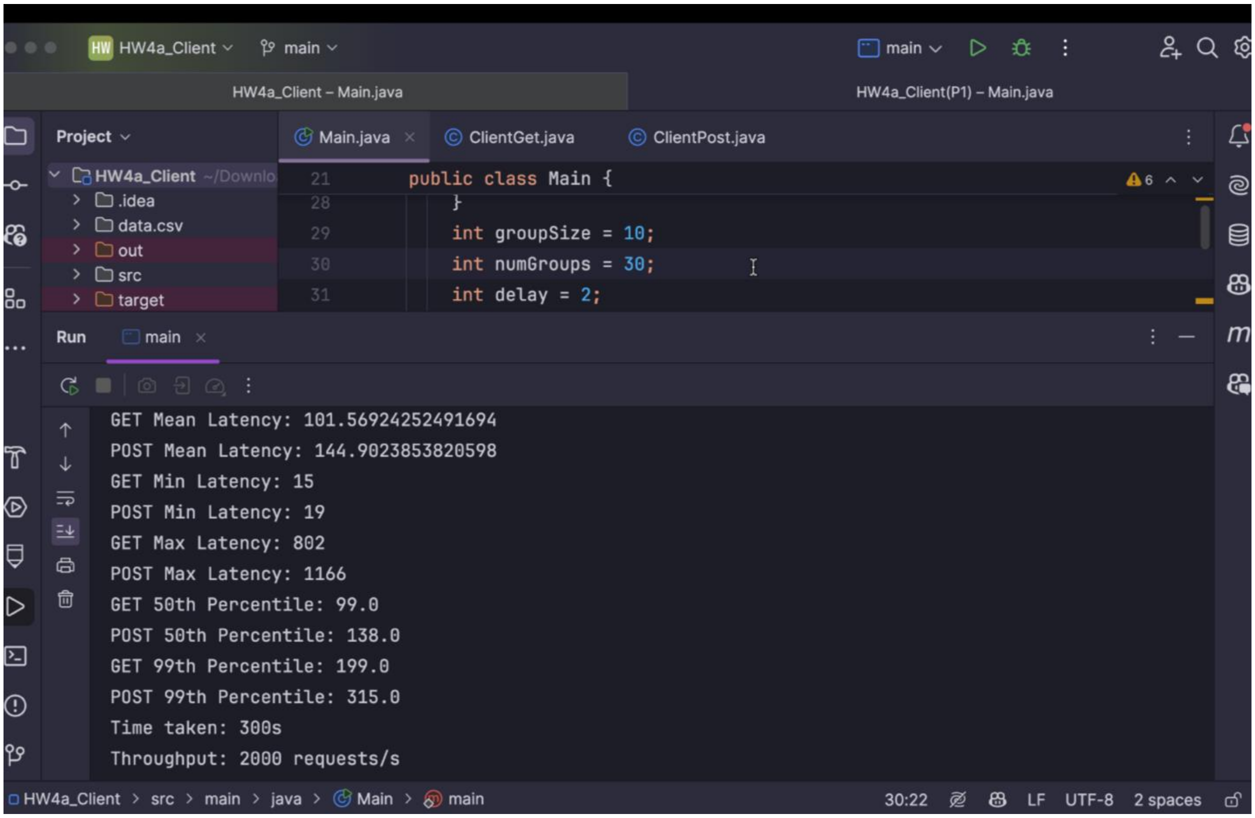
threadGroupSize = 10, numThreadGroups = 20, delay = 2 (seconds)



```java
public class Main {
    }
    int groupSize = 10;
    int numGroups = 20;
    int delay = 2;
```

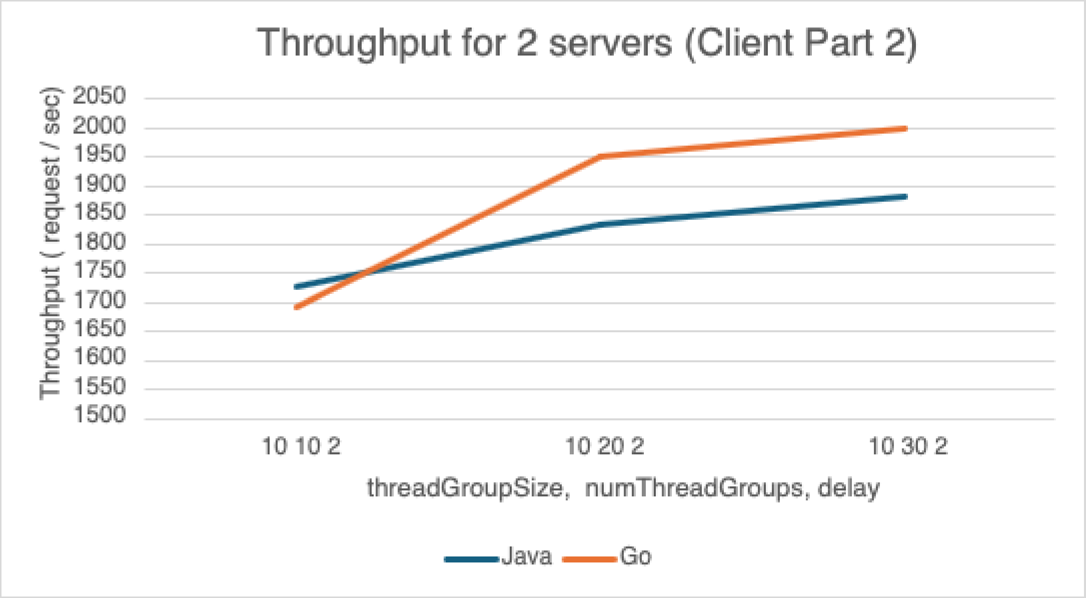
```
GET Mean Latency: 77.33337313432835
POST Mean Latency: 92.85557213930348
GET Min Latency: 15
POST Min Latency: 18
GET Max Latency: 493
POST Max Latency: 496
GET 50th Percentile: 78.0
POST 50th Percentile: 92.0
GET 99th Percentile: 140.0
POST 99th Percentile: 190.0
Time taken: 205s
Throughput: 1951 requests/s
```

threadGroupSize = 10, numThreadGroups = 30, delay = 2 (seconds)



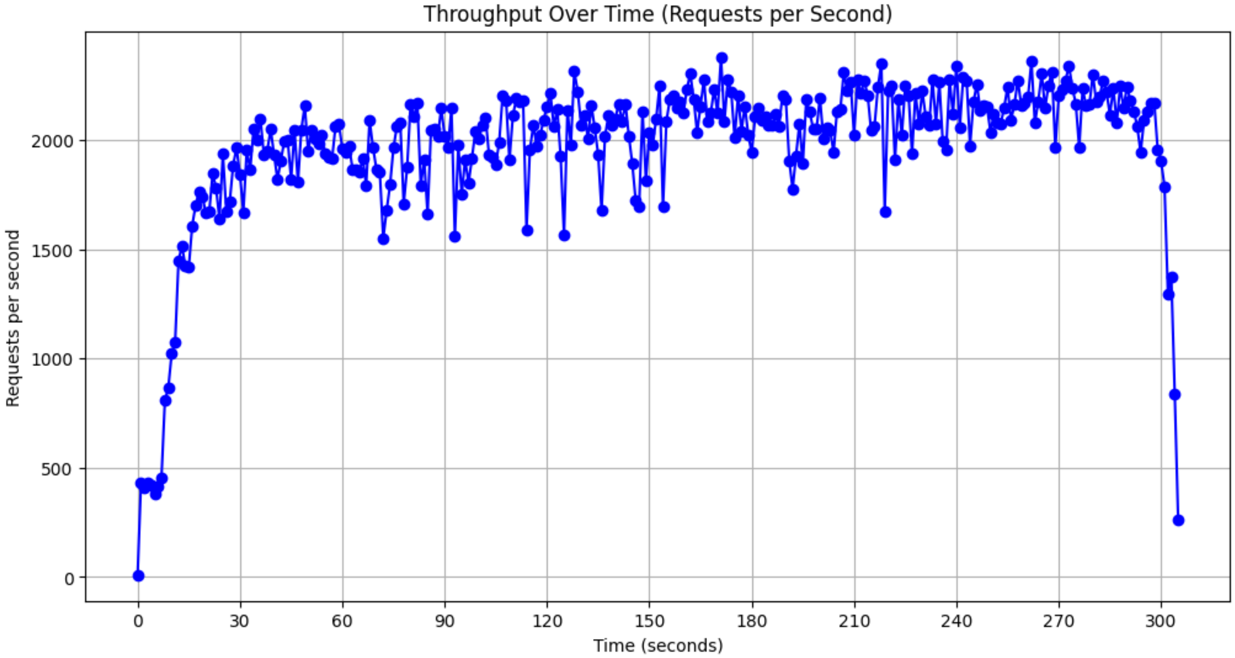
```java
public class Main {
    }
    int groupSize = 10;
    int numGroups = 30;
    int delay = 2;
```

```
GET Mean Latency: 101.56924252491694
POST Mean Latency: 144.9023853820598
GET Min Latency: 15
POST Min Latency: 19
GET Max Latency: 802
POST Max Latency: 1166
GET 50th Percentile: 99.0
POST 50th Percentile: 138.0
GET 99th Percentile: 199.0
POST 99th Percentile: 315.0
Time taken: 300s
Throughput: 2000 requests/s
```



5. Plot

Test with Go: threadGroupSize = 10, numThreadGroups = 30, delay = 2


Throughput Over Time (Requests per Second)
Requests per second
Time (seconds)