Load Balancing for Content Distribution

-Project Report-

The Load Balancing Strategy

We adopted Round Robin as a load balancing method. Round Robin is based on first-come, first-served. According to the order, the load balancer continues to send requests to servers. The first client, for example, will connect to Server 1, the second client to Server 2, and so on. This distributes the server load equally, allowing it to manage huge traffic. Our designed strategy was a combination of the Round Robin algorithm and the Random class, this way, the Load Balancer will send requests to servers randomly limited by the range we defined for requests and servers. The reason we used the Round Robin algorithm is because its load balancing is easy to learn and apply and also because we have very limited knowledge on the other load balancing methods. Our algorithm should be able to handle numerous client requests at the same time. As a result, we employed threads so that each time a client request came in, a new thread could be allocated to handle it. The LoadBalancer class runs on a thread to allow several clients to access a web page at the same time.

High-Level Approach

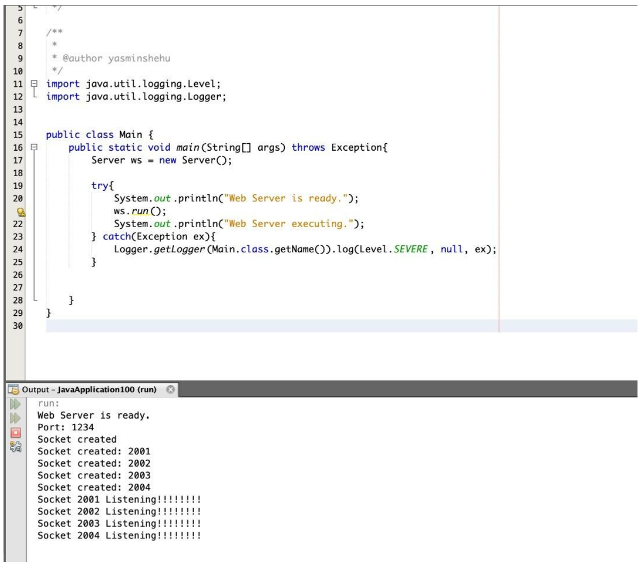
TCP was the protocol that we utilized. The HTTP protocol is used by the server to transfer the HTML file to the client. The HTTP program layer requests that the TCP layer establish the connection and transmit the file. In the WebServer class, we set up the connection.
Our algorithm should be able to handle numerous client requests at the same time. As a result, we employed threads so that each time a client request came in, a new thread could be allocated to handle it. The LoadBalancer class runs on a thread to allow several clients to access a web page at the same time.

Challenges We Have Faced:

At first, We were looking for a strategy to connect multiple clients to multiple servers without using threads, and since we only knew how to connect one client to one server, it was difficult to manage. After some research, we found some ways to implement that; the first was using a List of servers and the second using Threads on the client side.Another challenge was implementing the Load Balancer itself. We thought it had to be a Software Load balancer like Nginx or HAProxy until we managed to code one that runs on a thread.

Running The Project

- We run the Main.java file (It creates the server and opens a socket connection to the server. When the server is ready, using Port 1234, it creates 4 sockets and these sockets start to listen for connections)

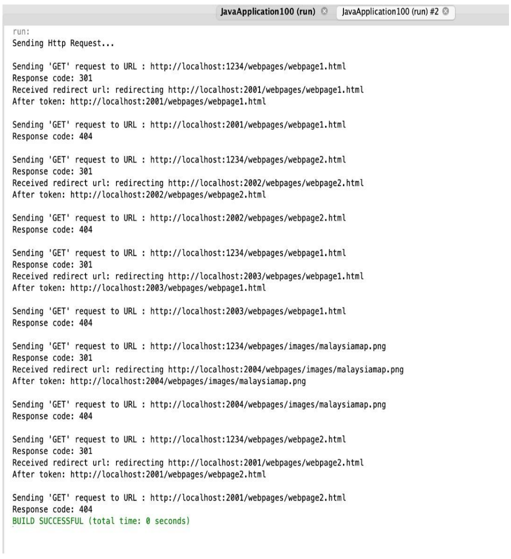


```java
 */

/**
 *
 * @author yasminshehu
 */
import java.util.logging.Level;
import java.util.logging.Logger;


public class Main {
    public static void main(String[] args) throws Exception{
        Server ws = new Server();

        try{
            System.out.println("Web Server is ready.");
            ws.run();
            System.out.println("Web Server executing.");
        } catch(Exception ex){
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }


    }
}
```

Output – JavaApplication100 (run)

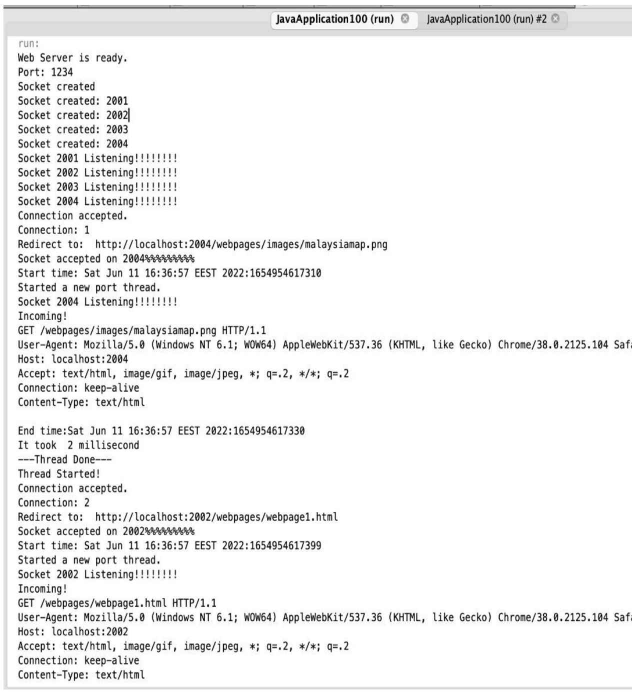
```
run:
Web Server is ready.
Port: 1234
Socket created
Socket created: 2001
Socket created: 2002
Socket created: 2003
Socket created: 2004
Socket 2001 Listening!!!!!!!!
Socket 2002 Listening!!!!!!!!
Socket 2003 Listening!!!!!!!!
Socket 2004 Listening!!!!!!!!
```

- The next step is to generate the Clients and to do this, we run the Client.java class which uses the ClientThread.java class ro create multiple client threads. For this test run, we will generate 5 clients. We are also sending the HTTP 'Get' Request.



JavaApplication100 (run) | JavaApplication100 (run) #2

```
run:
Sending Http Request...

Sending 'GET' request to URL : http://localhost:1234/webpages/webpage1.html
Response code: 301
Received redirect url: redirecting http://localhost:2001/webpages/webpage1.html
After token: http://localhost:2001/webpages/webpage1.html

Sending 'GET' request to URL : http://localhost:2001/webpages/webpage1.html
Response code: 404

Sending 'GET' request to URL : http://localhost:1234/webpages/webpage2.html
Response code: 301
Received redirect url: redirecting http://localhost:2002/webpages/webpage2.html
After token: http://localhost:2002/webpages/webpage2.html

Sending 'GET' request to URL : http://localhost:2002/webpages/webpage2.html
Response code: 404

Sending 'GET' request to URL : http://localhost:1234/webpages/webpage1.html
Response code: 301
Received redirect url: redirecting http://localhost:2003/webpages/webpage1.html
After token: http://localhost:2003/webpages/webpage1.html

Sending 'GET' request to URL : http://localhost:2003/webpages/webpage1.html
Response code: 404

Sending 'GET' request to URL : http://localhost:1234/webpages/images/malaysiamap.png
Response code: 301
Received redirect url: redirecting http://localhost:2004/webpages/images/malaysiamap.png
After token: http://localhost:2004/webpages/images/malaysiamap.png

Sending 'GET' request to URL : http://localhost:2004/webpages/images/malaysiamap.png
Response code: 404

Sending 'GET' request to URL : http://localhost:1234/webpages/webpage2.html
Response code: 301
Received redirect url: redirecting http://localhost:2001/webpages/webpage2.html
After token: http://localhost:2001/webpages/webpage2.html

Sending 'GET' request to URL : http://localhost:2001/webpages/webpage2.html
Response code: 404
BUILD SUCCESSFUL (total time: 0 seconds)
```



JavaApplication100 (run) | JavaApplication100 (run) #2

```
run:
Web Server is ready.
Port: 1234
Socket created
Socket created: 2001
Socket created: 2002
Socket created: 2003
Socket created: 2004
Socket 2001 Listening!!!!!!!!
Socket 2002 Listening!!!!!!!!
Socket 2003 Listening!!!!!!!!
Socket 2004 Listening!!!!!!!!
Connection accepted.
Connection: 1
Redirect to:  http://localhost:2004/webpages/images/malaysiamap.png
Socket accepted on 2004%%%%%%%%%%
Start time: Sat Jun 11 16:36:57 EEST 2022:1654954617310
Started a new port thread.
Socket 2004 Listening!!!!!!!!
Incoming!
GET /webpages/images/malaysiamap.png HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.104 Saf
Host: localhost:2004
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Type: text/html

End time:Sat Jun 11 16:36:57 EEST 2022:1654954617330
It took  2 millisecond
---Thread Done---
Thread Started!
Connection accepted.
Connection: 2
Redirect to:  http://localhost:2002/webpages/webpage1.html
Socket accepted on 2002%%%%%%%%%%
Start time: Sat Jun 11 16:36:57 EEST 2022:1654954617399
Started a new port thread.
Socket 2002 Listening!!!!!!!!
Incoming!
GET /webpages/webpage1.html HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.104 Saf
Host: localhost:2002
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Type: text/html
```

- Since we used a combination of Round Robin and Random, it redirects the servers in a random order as you can see in the screenshots above. It starts with socket 2004 instead of the traditional Round Robin way which is supposed to start from 2001 and so on. We're establishing a connection, creating a thread, and then redirecting to the client.

(Group20)
Balkisu Shehu Danmusa 218SE2421
Mohanad Altarah 219SE2035