Report for extra

Socket Programming with a Load Balance

Taka Nakamura, Alexander West, and Dakota Wilkerson

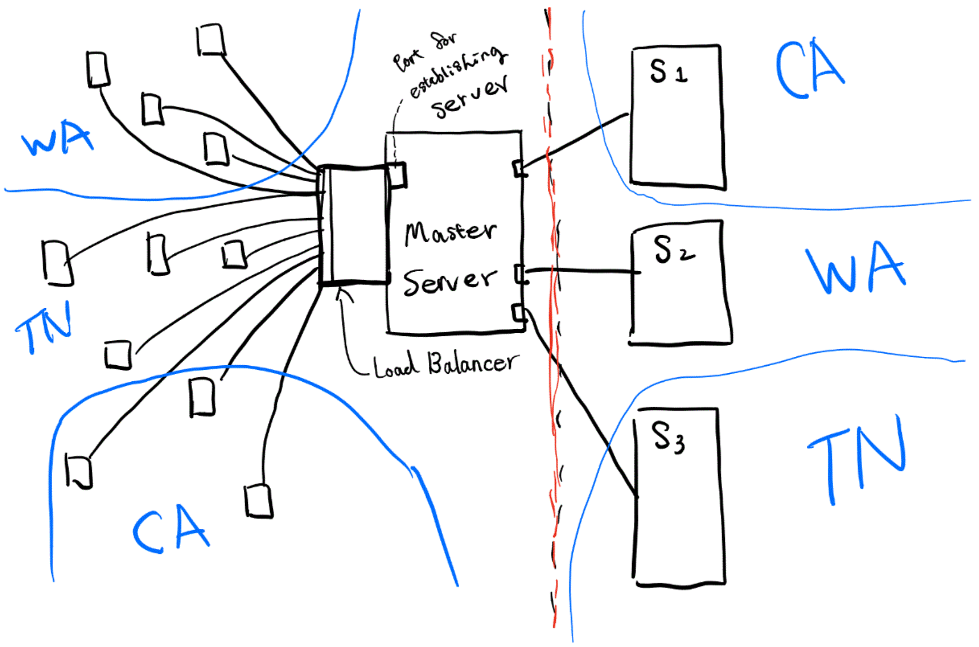
Implementation of a load balance

We implement a load balancing system on our reservation system with TCP connection. That improves a speed of a communication between clients and servers by allowing client to connect nearest server from its location. With this implementation we can enhance some algorithm to reduce traffic.

Design/Concepts

In the diagram below, on the left side small boxes are clients maximum n number of clients can connect to Master server placed in the middle in the figure at one time. On the right side, servers are located: also, multiple servers can connect at the same time. Each server has each being threaded flight/car/room server inside.

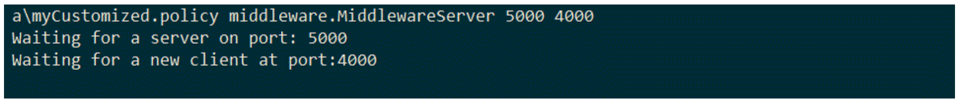
Master server where we implemented our load balancing router. Inside the master server, there is a router table that stores each port number and location of server is located. Our master server can work as a load balancing router.



How it works

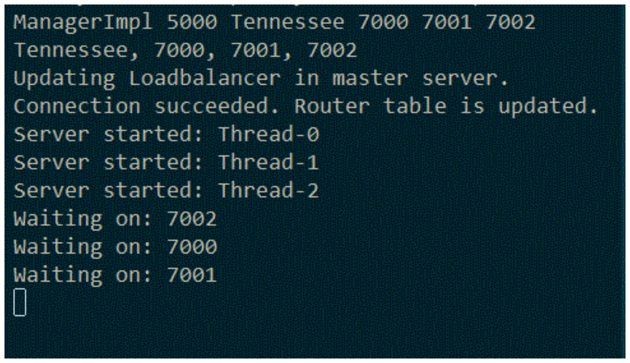
We will explain this step by step.

1. Master server is established first. Ports are established.



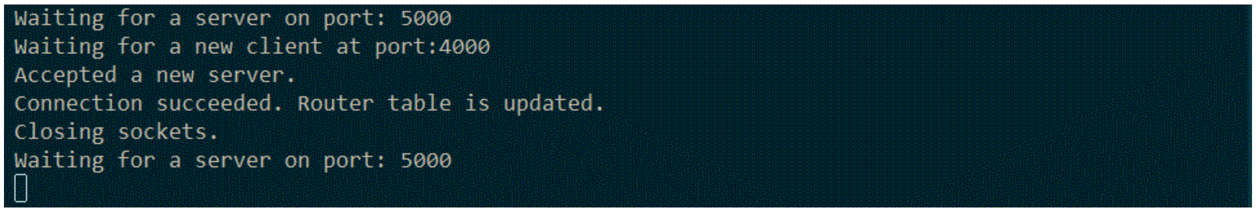
```
a\myCustomized.policy middleware.MiddlewareServer 5000 4000
Waiting for a server on port: 5000
Waiting for a new client at port:4000
```

2. Establish one of the servers in some location in this case it is located in Tennessee. When it is established, a request is sent to the Master server containing each port number and its location. Flight server has a port on 7000, Car server -> 7001, Room server ->7002.



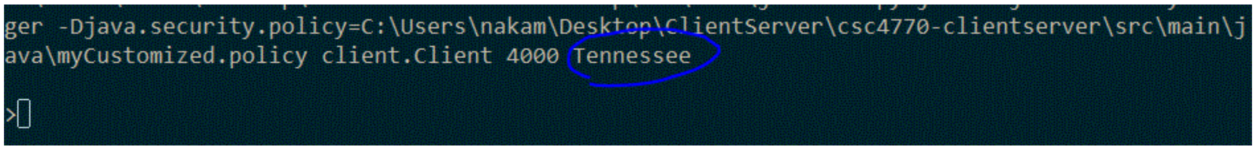
```
ManagerImpl 5000 Tennessee 7000 7001 7002
Tennessee, 7000, 7001, 7002
Updating Loadbalancer in master server.
Connection succeeded. Router table is updated.
Server started: Thread-0
Server started: Thread-1
Server started: Thread-2
Waiting on: 7002
Waiting on: 7000
Waiting on: 7001
```

3. Master accepted a request from the server and it stores the location (Key) and each port number (Vector<Integer>) in the router table (Hashtable). Next step is running a client.



```
Waiting for a server on port: 5000
Waiting for a new client at port:4000
Accepted a new server.
Connection succeeded. Router table is updated.
Closing sockets.
Waiting for a server on port: 5000
```

4. Run a client with extra argument that is a location of where the client is located.

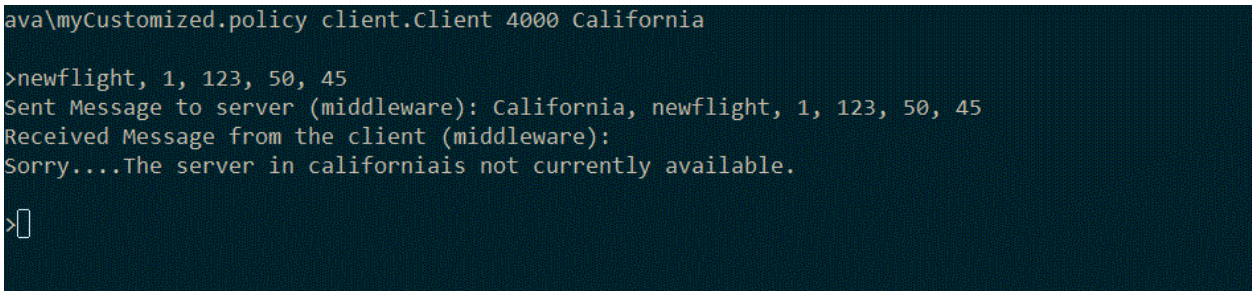


```
ger -Djava.security.policy=C:\Users\nakam\Desktop\ClientServer\csc4770-clientserver\src\main\j
ava\myCustomized.policy client.Client 4000 Tennessee

>
```

This location information will be added to at the very first value of our command protocol that will be sent to the master server through the master server.

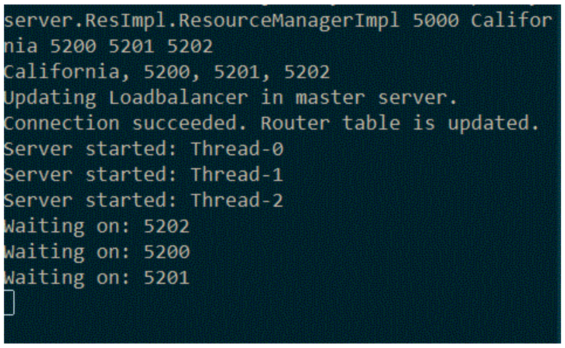
5. When master server receives a request/command from a client, it looks up the router table and get port number that is corresponding to its location. Then, it will run the command. Each time the command is executed, the corresponding socket is created. Once the communication is done with a server then the socket is immediately closed.

6. If a client in CA is established, but server is not available in CA, then it will show when a command is typed.



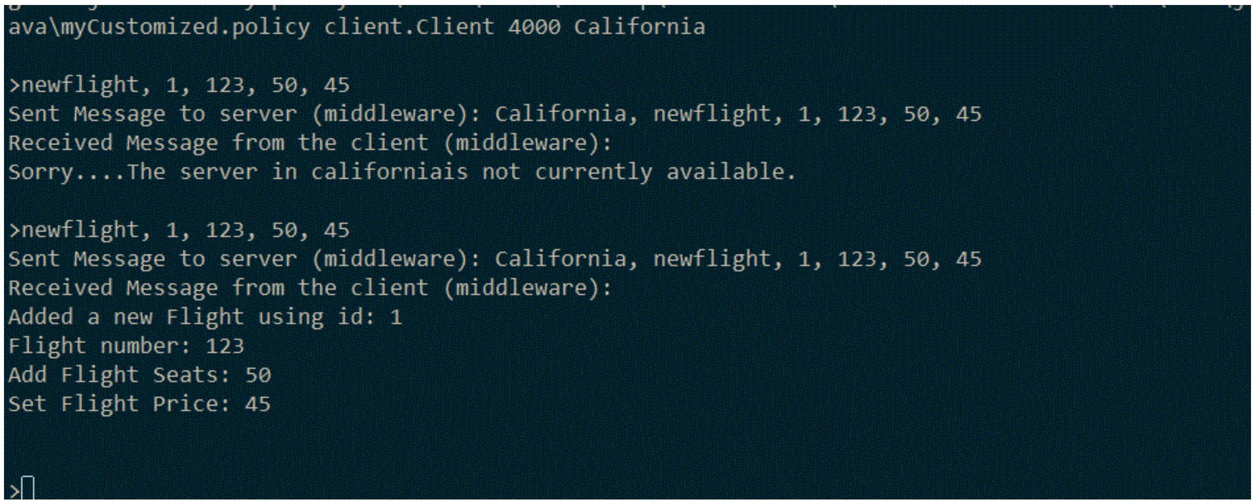
```
ava\myCustomized.policy client.Client 4000 California

>newflight, 1, 123, 50, 45
Sent Message to server (middleware): California, newflight, 1, 123, 50, 45
Received Message from the client (middleware):
Sorry....The server in californiais not currently available.

>
```

7. Since establish a thread in the master server, we can anytime add a new server, so the new server in California is added providing different port numbers, which action again sends a request to master server to update the router table and establish the ports.



```
server.ResImpl.ResourceManagerImpl 5000 Califor
nia 5200 5201 5202
California, 5200, 5201, 5202
Updating Loadbalancer in master server.
Connection succeeded. Router table is updated.
Server started: Thread-0
Server started: Thread-1
Server started: Thread-2
Waiting on: 5202
Waiting on: 5200
Waiting on: 5201
```

8. Now Client in ca can send a message to the server in CA!


```
ava\myCustomized.policy client.Client 4000 California

>newflight, 1, 123, 50, 45
Sent Message to server (middleware): California, newflight, 1, 123, 50, 45
Received Message from the client (middleware):
Sorry....The server in californiais not currently available.

>newflight, 1, 123, 50, 45
Sent Message to server (middleware): California, newflight, 1, 123, 50, 45
Received Message from the client (middleware):
Added a new Flight using id: 1
Flight number: 123
Add Flight Seats: 50
Set Flight Price: 45


>
```

For future work

We need a concurrency in each server. Meaning, in our case, each server communicates only inside their server, but not with another in different location. Also, we need to implement some better router system that if it is the scenario where server in CA was down when a client in CA sends a request, the master server/load balancing router should route the request to a second nearest server from the client.