Performance Evaluation Report

Name: Dobre Gigi-Alexandru
Group: 341C5
Resolved tasks:I, II, III, IV
Considered Grade:(97/100)

I. Prerequisites.

The setup has been made following the instructions on A and B, using the command *sudo python3 topology.pyc gigi.dobre*.

II. Evaluation - System Limits Analysis

* How many requests can be handled by a single machine?

I have modified both test.py and client.py to obtain the number of requests. For test.py, I changed the function to only start the listeners on each server. For client.py, I changed the arguments that the python file is receiving, adding the number of req to be made to a specific server. After running "*c1 python3 client.py -p http "serverIP":"serverPort" 1000"*, I made the conclusion that each server can only handle 308 HTTP GET requests. Each one of them will block when trying to receive the response from the 309th request.

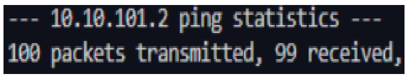
* What is the latency of each region?

To calculate the latency of each region, we will be sending pings from each machine to the router of their region.
h1 -> r1: 42ms, h2 -> r1: 36ms, Asia: 39ms (packet sent/response received) (19.5ms)
h3 -> r2: 52ms, h4 -> r2: 66ms, Emea: 59ms (packet sent/response received) (29.5ms)
h5 -> r3: 18ms, h6 -> r3: 26ms, US: 22ms (packet sent/response received) (11ms)

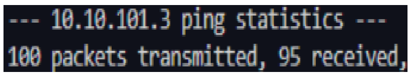
* What is the server path with the smallest response time? But the slowest?

To calculate the server paths with the smallest and highest response time, we need to also add to the latency of ping from c1 to each router from the region.
c1 -> r1: 12.2ms; c1 -> r2: 38.3ms; c1 -> r3: 48.2ms
c1 -> h1: 54.4ms; c1 -> h2: 48.4ms
c1 -> h3: 90.6ms; c1 -> h4: 104.6ms
c1 -> h5: 66.5ms; c1 -> h6: 74.5ms
Smallest response time: c1 -> h2. Highest response time: c1 -> h4

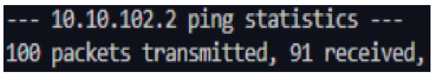
* What is the path that has the greatest loss percentage?



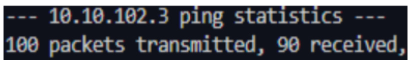
```
--- 10.10.101.2 ping statistics ---
100 packets transmitted, 99 received,
```



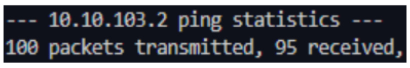
```
--- 10.10.101.3 ping statistics ---
100 packets transmitted, 95 received,
```



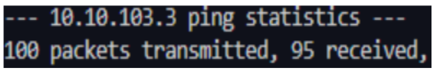
```
--- 10.10.102.2 ping statistics ---
100 packets transmitted, 91 received,
```



```
--- 10.10.102.3 ping statistics ---
100 packets transmitted, 90 received,
```



```
--- 10.10.103.2 ping statistics ---
100 packets transmitted, 95 received,
```



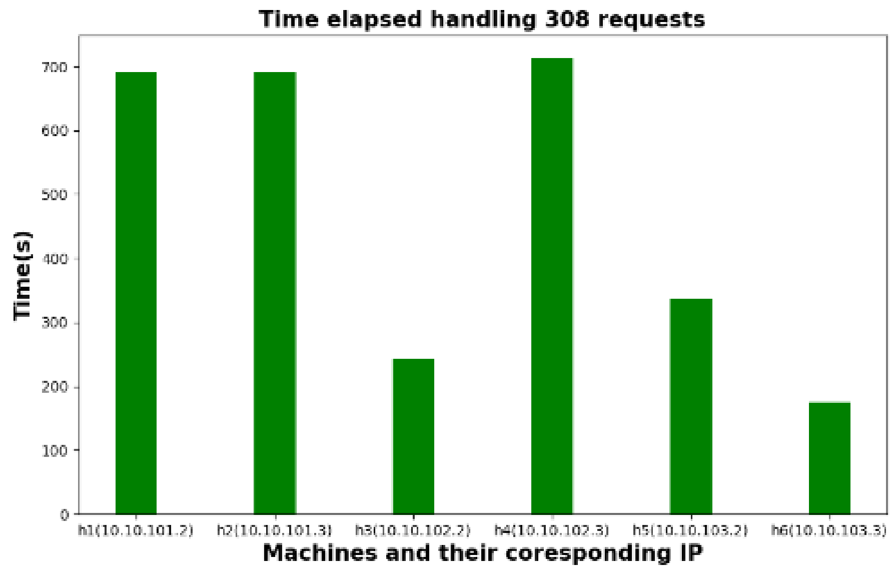
```
--- 10.10.103.3 ping statistics ---
100 packets transmitted, 95 received,
```

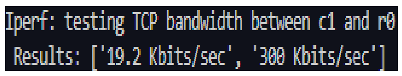
The path that has the greatest loss(pings from c1 to each one of the machines) is the one to the h4.

* What is the latency introduced by the first router in our path?

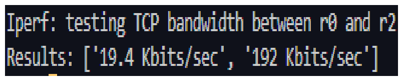
The latency introduced by the first router in our path was calculated from 10 pings as the mean of them. The latency is 0.04ms (packet sent/response received) (0.02ms).

* Is there any bottleneck in the topology? How would you solve this issue?

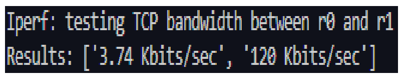




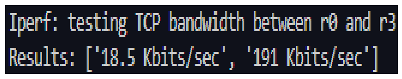
```
Iperf: testing TCP bandwidth between c1 and r0
 Results: ['19.2 Kbits/sec', '300 Kbits/sec']
```



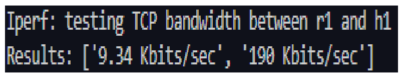
```
Iperf: testing TCP bandwidth between r0 and r2
Results: ['19.4 Kbits/sec', '192 Kbits/sec']
```



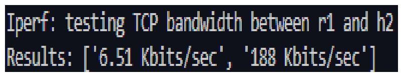
```
Iperf: testing TCP bandwidth between r0 and r1
Results: ['3.74 Kbits/sec', '120 Kbits/sec']
```



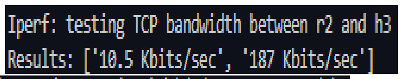
```
Iperf: testing TCP bandwidth between r0 and r3
Results: ['18.5 Kbits/sec', '191 Kbits/sec']
```



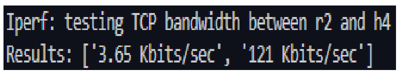
```
Iperf: testing TCP bandwidth between r1 and h1
Results: ['9.34 Kbits/sec', '190 Kbits/sec']
```



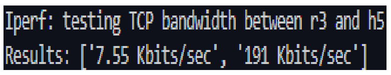
```
Iperf: testing TCP bandwidth between r1 and h2
Results: ['6.51 Kbits/sec', '188 Kbits/sec']
```



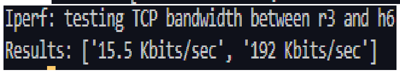
```
Iperf: testing TCP bandwidth between r2 and h3
Results: ['10.5 Kbits/sec', '187 Kbits/sec']
```



```
Iperf: testing TCP bandwidth between r2 and h4
Results: ['3.65 Kbits/sec', '121 Kbits/sec']
```



```
Iperf: testing TCP bandwidth between r3 and h5
Results: ['7.55 Kbits/sec', '191 Kbits/sec']
```



```
Iperf: testing TCP bandwidth between r3 and h6
Results: ['15.5 Kbits/sec', '192 Kbits/sec']
```

As we can see from both the bar chart and the output of the “iperf” command where the bandwidth between the components is returned, the transfer of data is being slowed between r0 & r1, making it a bottleneck for its region, while for h4, it seems that the problem is between r2 and the machine, so probably between the switch and the machine since from r2 to h3 there is no problem. A way to solve this would be to verify the “cables" and change the faulty ones. A second way to solve this would be to upgrade the hardware. Lastly, we can add some load balancing policies to route the requests to different machines rather than one.

* What is your estimation regarding the latency introduced?

Since there won’t be a new router/worker, the latency introduced would probably only appear because of the load balancing policies, finding out where to send the request.

* What downsides do you see in the current architecture design?

Since some of the paths have a little bandwidth compared to the others, and because there are no load balancing policies, sending all the requests to only one machine might be problematic considering the bottlenecks.

III. Implementation

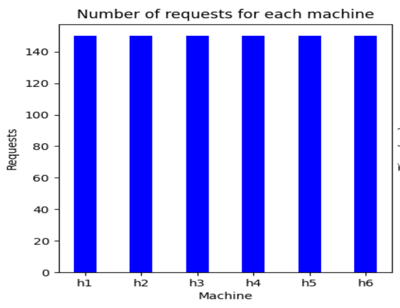
Each test was done with 900 requests so we can see the differences in the policies.

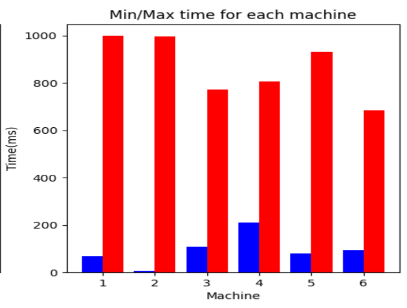
Usage: *sudo python3 topology.pyc gigi.dobre -t* (to start the listeners on each server)

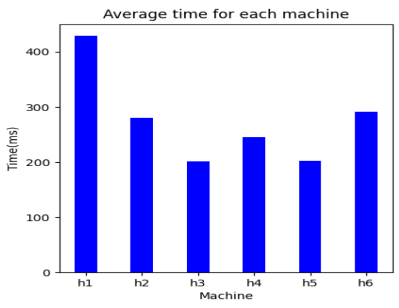
c1 python3 client.py -p (http/https) (1/2/3) numberOfRequests

1. Round-robin load balancing policy

For a round-robin load balancing policy, we send in order a request to each machine and repeat the process n/6 times until it finishes the number of requests we provide (n/6 times if the n is multiple of 6, otherwise, it will stop accordingly).

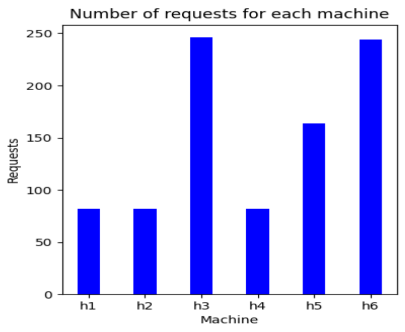


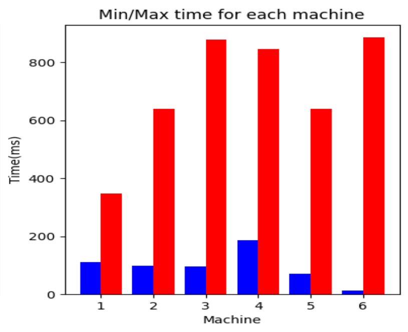


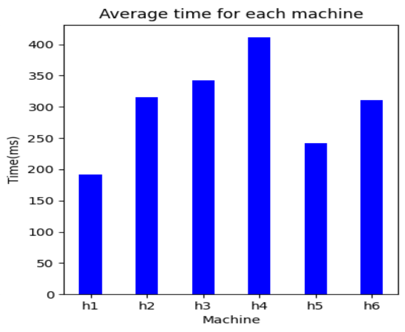


2. Weighted round-robin load balancing policy

For a weighted round-robin load balancing policy, we send in order requests to each machine based on their weight. The weights I chose were mainly based on previous results and they were -> h1:1; h2:1; h3:3, h4:1, h4:2, h3:3.

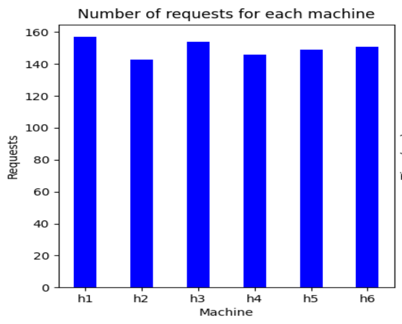


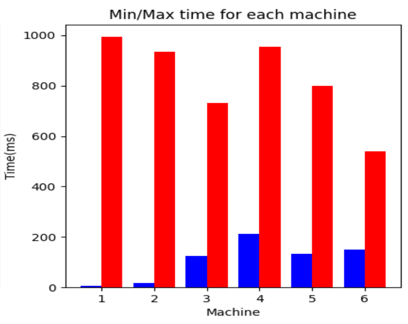


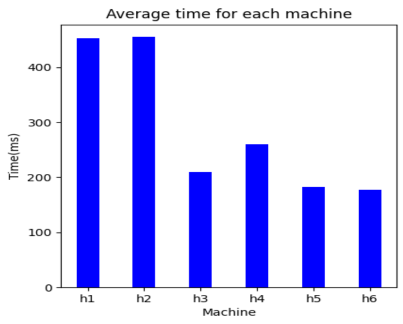


3. Random region round-robin load balancing policy

For a random region round-robin load balancing policy, we send in order a request to each region where we send the request to a random machine specific to that region, repeating the process n/3 times (n/3 times if the n is multiple of 3, otherwise, it will stop accordingly).







As you can see, the third one is showing the same result on where the bottleneck of the machine is, but on lesser requests per machine. Overall, making the average of average time for each policy, we can observe that the first one is the one that is the best to handle the requests for these servers in our situation where we have 2 machines/region. (275, 302, 289).