Pâslă Codruța-Ștefania 342 C5b

All parts of the assignment were completed, except the bonus.

I think my grade will be 100p/ 110p.

Documentation

Implementation – This homework was made in a virtual machine with 4 cpus

I implemented 5 methods for load balancing policies**. The first method** is named “random_lb”, this generates for every machine a random number r between 1 and the number of requests. If the machine is the last, then all remaining requests are sent to it. **The next method** is called "round_robin_lb". Here an equal number of requests are sent to each machine and if the number of requests is not divided exactly by the number of machines, then the requests are distributed in a uniform manner to each machine. **The third method** “second_random_lb” is similar to the first one, except that here was tried to avoid the case when a machine can receive all the requests. The requests are divided by the number of machines and the resulted number is stored. The algorithm for the first machine will generate a random number between 1 and that stored number, the next machine will receive a random number between 1 and the stored number plus the difference remained from the previous step, and so on. If at the finish there are remaining requests, they are sent to the last machine. **The method** “cpu_work_lb” gets the number of cpus and divides the total number of requests at the number of cpus. For every cpu is created a thread and the requests are sent to “/work. **In the last method**, after an evaluation, it was observed that the ratio at the level of response time between regions was 2:2:1 (asia, emea, us), so for every machine was allocated a cost of 2 or 1. Thus, the requests are sent depending on the cost of every machine.

Evaluation

A)1) How many requests can be handled by a single machine?

For this evaluation I used SIEGE tool. I run this command which hits the server with 150 concurrent requests for every machine “siege 'http://0.0.0.0:5000/work/machine_name/machine_id' -c150 -t300s” until some requests were rejected. Example:

```
(tema_ep) codruta@codruta:~/Desktop/tema_ep/load_balancer$ siege 'http://0.0.0.
0:5000/work/asia/0' -c150 -t300s
** SIEGE 4.0.4
** Preparing 150 concurrent users for battle.
The server is now under siege...^C
Lifting the server siege...
Transactions:                   3198 hits
Availability:                  98.61 %
Elapsed time:                  88.19 secs
Data transferred:               0.26 MB
Response time:                  3.91 secs
Transaction rate:              36.26 trans/sec
Throughput:                     0.00 MB/sec
Concurrency:                  141.83
Successful transactions:        3198
Failed transactions:              45
Longest transaction:            8.44
Shortest transaction:           0.77
```

The results were: 3198 maximum requests for asia 0, 3185 for asia 1, 3213 for emea 0, 2049 for us 0, 2383 for us 1.

2) What is the latency of each region?

I calculated latency as the difference between response_time and work_time. So, the latency for region ASIA was approximately 600ms, for EMEA 320ms and for US 430ms.

3) What is the computation time for a work request?

I considered the computation time as work_time from the JSON returned for every request, The results were approximately of 20ms for every region.

4) What is the response time of a worker when there is no load?

For this task, I used the browser for taking the response_time of every machine and the results were approximately: 600ms for asia 0, 600ms for asia 1, 300ms for emea, 500ms for us 0, and 450ms for us 1.

5) What is the latency introduced by the ***forwarding unit***?

For this task I used SIEGE. I accessed the app directly using the link provided by Heroku (ex: https://codrutapasla-worker-asia-1.herokuapp.com/work) and the response time was smaller than accessing the app through the router. Using the output from SIEGE the results were: asia 0=0.62-0.28s=0.34s, asia 1=0.61-0.28s=0.33s, emea 0=0.40-0.28s=0.12s, us 0=0.55-0.30s=0.25s, us 1= 0.54-0.29s=0.25s.

6) How many requests must be given in order for the ***forwarding unit*** to become the bottleneck of the system? How would you solve this issue?

After a test running with SIEGE tool, was observed that the forwarding unit became the bottleneck of the system at approximately 3200 requests. I think this problem is because of internal limitation of the router. For solving this issue we could prioritize the requests. Another approach could be to increase the capabilities of the routing processor.

7) What is your estimation regarding the latency introduced by Heroku?

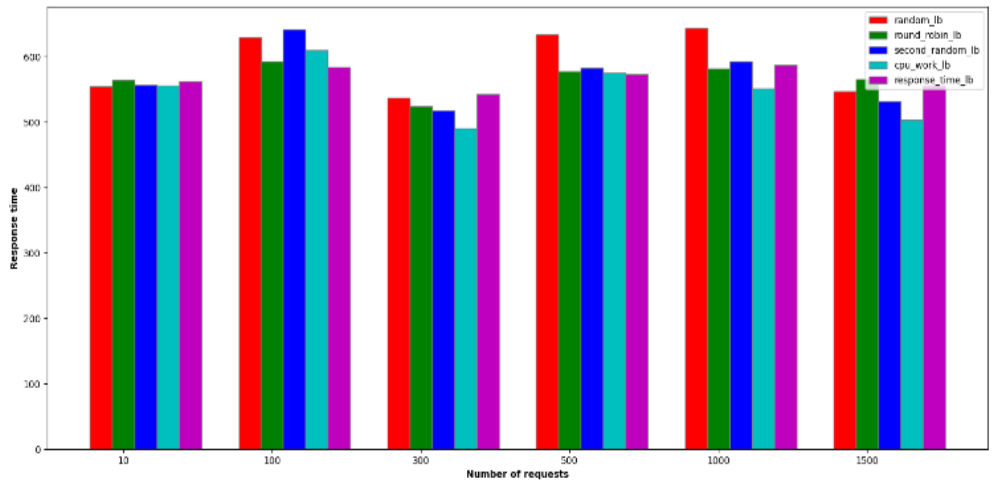
The work time from Heroku was approximately 20ms and response time through router was calculated at 5) and was 280ms. Thus, the latency is 280 – 20 = 260ms.

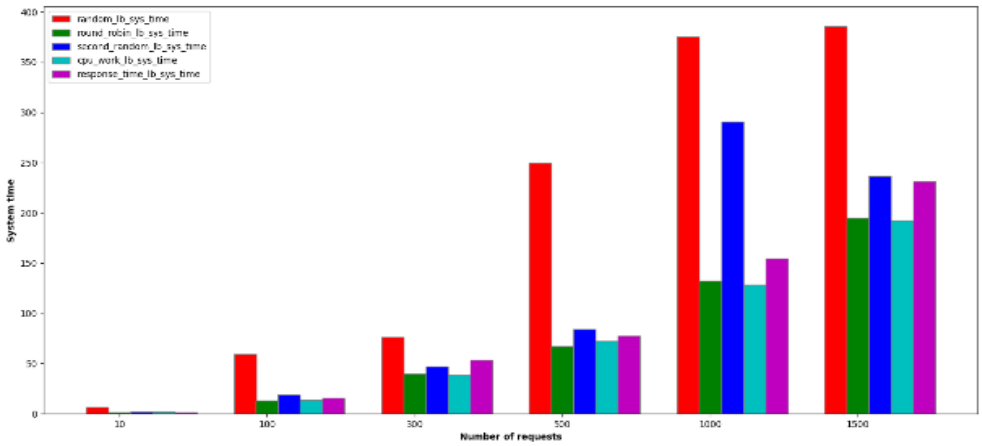
8) What downsides do you see in the current architecture design?

In the given architecture at a great number of requests, the servers get overloaded, forming traffic congestion. Another problem is that there is no redundancy, if the router is down the entire architecture will suffer because there is no other machine for backup.

B) For this task, I made graphs for response time and system time. The codes for these are in “response_time.py” and “sys_time.py”. For requests I used the following values:

10, 100, 300, 500, 1000, 1500.





The response time for implemented policies was relatively the same for the same number of requests. A difference was observed in the system time graph. The random policy was the slowest. As increasing the number of requests, the system was processing the requests in a slower way and the time was going up.