Load-Balancer

Compiling

loadbalancer compiles on Ubuntu VM with no errors or warnings.

Testing

For some of the tests below I had *loadbalancer* print out the status of all the servers whenever it is calling the health-check method. This way I could see which servers were up/down, and the stats reported for each server.

Test-1 (notice clients): No servers running. See that loadbalancer returns 500-error messages to a sequence of client requests and also to several client requests arriving asynchronously.

Test-2 (healthchecks):

- Run with a single server (from assignment 2). Print from the code the results of the health-checks, and see they are as expected (number of requests increases each time).
- Abort the server and see that loadbalancer notices it is down.
- Run the server again and see that the loadbalancer notices it is up.

Test-3 (healthcheck): Repeat Test-2 with several servers.

Test-4 (forwarding messages): Run with a single server and sequential client requests. See the response arrives properly at the client terminal.

Test-5 (selecting server and hashmap):

- Run with a single server and send multiple requests its way (bash file with several curl commands). Have loadbalancer print to stdout the server stats, as in test 2. Print also the hashmap table. Check pairs are added and removed properly as requests arrive and responses are sent. (I am now aware I could have done without a hashmap, but initially I thought I might need it. Since it works well, I left it as is.)
- Add a second server. Verify that the requests are directed to the new one, which has less total-requests, by noticing the stats of the first server increase slowly (when healthcheck requests are sent), while the stats of the second server increase faster, until they reach a similar number of total requests.
- Abort the first server and rerun it, so its number of requests is reset. See that the first server is preferred.

Test-6 (stress test):

- Use a bash file with requests sent in the background (using '&'), inside a loop (so that many requests would arrive at the loadbalancer).

- Run loadbalancer with 5 servers. Have a couple running initially.
- Run the bash file.
- Add some servers, remove some servers.
- See the 'correct' servers are selected (based on the print out from the healthcheck step). See also that the requests receive the expected response (and not err-500).
- Check that when servers are down or when the load is very high, clients receive err-500 responses.

Question of the Day

1. Your load balancer distributed load based on the number of requests the servers had already serviced, and how many failed. A more realistic implementation would consider performance attributes from the machine running the server. Why was this not used for this assignment?

   Possibly because all of the servers run on the same machine.

2. This load balancer does no processing of the client request. What improvements could you achieve by removing that restriction? What would be the cost of those improvements?

   If we process the client request, we can have an estimate on the time it should take to respond to it. This way, we will be less likely to assume a server is down, when it is actually writing a 10T bytes file. There are several downsides to that approach. One is that parsing of the request takes time. Another issue is that we require the loadbalancer to be familiar with all the possible request formats and stay up to date when these change.

3. Using your httpserver from Assignment 2, do the following:
   - Place eight different large files in a single directory. The files should be around 400 MiB long.
   - Start two instances of your httpserver in that directory with four worker threads for each.
   - Start your loadbalancer with the port numbers of the two running servers and maximum of eight connections.
   - Start eight separate instances of the client at the same time, one GETting each of the files and measure (using time(1)) how long it takes to get the files. The best way to do this is to write a simple shell script (command file) that starts eight copies of the client program in the background, by using & at the end.

   Four requests timed-out after about 5 seconds (my max-timeout value) and received a 500 error message. The other four requests were all answered after a little over 2 minutes. In the meantime, one of the servers was completely busy handling the four massive requests and was not able to answer any other requests, including the

healthcheck requests that were regularly sent to it and accumulating in its queue. While that one server was toiling, the other server was completely idle except for its regular short healthcheck queries.

Why and how did that happen? All eight requests arrived at about the same time, so there was no time to get a healthcheck response in between the eight queries and as far as the loadbalancer was concerned, all were assigned to the same "best" server. Four of the eight were waiting in the selected server's queue for some information. Since sending the content of the 400M files took more than the timeout bound I set, the loadbalancer sent the 500 error-messages.

Note: even if I added a more clever tie-breaker mechanism, this could easily have happened. If one server has a better health "score" (perhaps because the other handled more queries), it would be selected by the assignment's specifications with no need for a tie-breaker.)

4. Repeat the same experiment, but substitute one of the instances of httpserver for nc, which will not respond to requests but will accept connections. Is there any difference in performance? What do you observe?

   I might have misunderstood how to perform this experiment. I used one server (5678) and also ran `nc -l 1234`. However, the loadbalancer quickly marked the "1234 server" as being down, because it does not respond to the healthcheck requests properly. Next, the nc-binary ends up being in my fd_set of readable ids, but when loadbalancer tries to receive bytes from it (recv()), it gets -1 bytes:

   ```
   received -1 bytes from 6
   recv: Connection reset by peer
   server on port 1234, health 0/0, and is down
   received 38 bytes from 7
   server on port 5678, health 0/115, and is up
   cannot connect to port 1234
   ```

   So, the results might not be exactly the same, but also not any better. When the loadbalancer chooses the actual server, the results should be the same as far as the clients are concerned. When the nc-binary is deemed the better-server (ignoring the fact that it would be rejected as unhealthy), time-outs would be noticed as well, because no responses are sent. No query would get a response.