Vinson Chen

Assignment 3: Load Balancer server Writeup

Small tests (Unit-Testing)

Most of the functions created in this project were tested using unit testing. Each function will be isolated from the other functions. They will be given a set of predefined inputs, and then having the outputs compared to the correct outputs in the program. These tests are mostly done in C. Each function will be tested with at least 3 different predefined inputs to determine whether the function meets its minimum functionality.

One of the unit-tests was to check for the functionality of the Queue. A few functionality tests include whether or not the enqueue function would append the function to the back of the queue. Another test includes whether or not the dequeue function would delete from the front. For this functionality test, valgrind is used to determine whether or not there was any un-freed node since dequeue should delete and free the node.

Another unit-test includes whether or not the feed forwarding system worked. Instead of passing the a client request, the unit-test gave it predefined message structures to process the request. A few functionality tests include curling a PUT request, curling a GET request, curling the HEAD request, and mix and matching between the requests.

Large tests (Whole system testing)

Once the smaller Unit-Testings were passed and the program was complete, the program underwent a set of bash scripts and manual curl tests to determine the functionality of the program. A few of the whole-program tests ran was: testing to see whether a PUT/GET/HEAD request would properly execute, testing to see if multiple PUT/GET/HEAD requests would run in parallel, testing the health check to see if the function returns the correct values and testing the correctness of the logging portion.

For tests to determine the correctness of the multi-threading, a bash script was created. The script will do both sequential PUT/GET/HEAD as well as parallel PUT/GET/HEAD. The script will also send an enormously large parallel request, exceeding the maximum number of threads.

There is another script that tests the health check. It will test a correct GET request with logging enabled to see the response, a correct PUT/HEAD request with healthcheck as the server's file name, as well as a GET/PUT/HEAD request when logging is disabled.

Clark has supplied a bash script used to test concurrent GET requests. Since there can be potential heisenbugs, Clark's script was placed inside a for loop of 100 iterations to decrease the chance that there are heisenbugs.

To check the correctness of the logging portion, manual tests were done. This is in parallel to the scripts created for the multi-threaded servers. A few of the script's tests would be executed, and then a manual test to see if the log file is written in the correct format.

Writeup Questions

- For this assignment, 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?

  Performance attributes are harder to isolate and test to see whether or the server loads correctly. This will cause the answer to be non-deterministic, meaning that we won't quite know what the answer to the program will be.

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

  One improvement I can see is the load balancer can detect if the client is asking for a health check. Without processing the client's request, there is no way to see if it's a health check or not. By knowing that it is a health check before the program sends to the server, the load balancer can return a 403 error code. The cost for these improvements will be minimal. This is because the method and the file name will be in the front of the buffer. By doing a `sscanf()` and two small char containers for the method and a bit of the file name (just big enough to hold healthcheck + 1 more character). Then it will be a string comparison between what was scanned and the requirements for the health check.

- 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?

  When I repeated the experiment with one of the httpservers as nc, the health check took a while to process. This is because the health check thread waits for the nc to respond, which it never will. The health check thread will then determine nc as dead, and process it if it wasn't alive. When the connections are accepted, the connections would never be routed to nc because the load balancer thinks it's dead. Thus the time for the get connections were roughly the same for the load balancer and the load balancer with a nc.