Natalie Petrosian
CSE 130 Principles of Computer Systems Design
Prof. Peter Alvaro

Assignment 3: Writeup

For this load balancer assignment, I implemented whole-system testing in which I ran my code in a variety of particular scenarios.

The following is a subset of tests that I tested on my program:

GET a small file
GET a large file
PUT a small file
PUT a large file
Process a HEAD request
Handle concurrent GET/HEAD/PUT (difference resource)
Handle concurrent GET/HEAD (same resource)
Handle a 400 error
Handle a 404 error
Handle concurrent 400 and 404 requests
Forward a GET request to the least loaded server (2 servers have the same load)
Forward a PUT request to the least loaded server (2 servers have the same load)
Forward traffic to the least loaded server (each server has a different load)
Forward traffic to the least loaded server (many servers have the same load)
Handle concurrent connections from multiple clients
Detect a single server going down
Detect multiple servers going down
Detect if all servers are down initially (500 error)
Detect if all servers are non-responsive (500 error)
Detect a server going down and coming back up
Detect invalid health check responses (status code != 200)

Question: 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?

Answer: It is true that in a more realistic implementation, the state of the hardware such as CPU usage, available memory, number of running processes, etc. will be scrutinized. Since this class hosts roughly 350 students with a variety of different personal hardware and running virtual environments, it would be difficult to expect a uniform platform to run tests and assess loads. Using HTTP servers from the previous assignment, is a practical workaround to this problem.

Question: 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?

Answer: If the load balancer intercepted the requests, it could possibly avert passing erroneous requests to the servers. However, intercepting every message passing through the load balancer would slow down the pipeline. Essentially, it becomes a choice between a slower pipeline and passing the burden of error checking to the servers. Since we communicate with many servers in parallel, it may be preferable to let the servers manage the error checking themselves.

Scenario:

- Place eight different large files in a single directory. The files should be around 400 MiB long.
- Start two instances of your http server in that directory with four worker threads for each.
- Start your load balancer 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.
- Repeat the same experiment, but substitute one of the instances of http server for nc, which will not respond to requests but will accept connections. Is there any difference in performance? What do you observe?

Answer: The second connection with the nc is slower by about a factor of 2. The reason for this is the thread that talks to the nc needs to timeout every time because nc is not responding. This shifts the burden of completing the requests to the httpserver. In contrast, when we are running two httpservers, they each take half of the total burden and run them in parallel, which is much faster.