Testing:

For this assignment, testing was done throughout the assignment on my own vm and using the CI testing. I started off with just the forwarding of requests. Then I started working on the parsing of the header which I reimplemented since it wasn't working correctly in assignment 2. Then I implemented the Serverlist and Cache data structures and all the functions for those data structures, where I tested each individual function if it worked correctly. After this I implemented all the other features needed for http proxy except the health checks. All these tests were done on one thread, but being able to handle multiple servers. Then I scaled all my code so it could handle concurrency and also implement the health check thread that checks health. After this I created my own test script, where I tested the load balancing and caching. When all of this worked, I got Jess's test script from the discord server and tested it, this is where I found a couple bugs but was able to iron them out. After completing my own test and jess's test I felt confident I had a solid version of the httpproxy.

Questions

For this assignment, your proxy distributed load based on 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?

For this assignment, we didn't use performance attributes from the machine running the servers because all the servers are on the same machine, so there wouldn't be much of a difference between choosing what server to choose or not. When using performance attributes from the machine running the server, in our case it is all being run on the same machine. So when measuring performance of the machine running the servers, all the servers would have the same performance since all the servers are running on the same machine. This would cause the http proxy to just choose a random number and not actually load balance the requests made to the proxy.

Repeat the same experiment, but turn off one of the servers. Is there any difference in performance? What do you observe?

After doing this experiment, I noticed a better performance when having two servers open. When only running one server with the proxy, I noticed that there was a delay of a couple seconds when only using one server. So the load balancing definitely works.

Using one of the files that you created for the previous question, start the provided server with only one thread, then do the following:
– Start your proxy without caching.
– Request the file ten times. How long does it take?
– Now stop your proxy and start again, this time with caching configured so that it can store the selected
file.
– Request the same file ten times again. How long does it take now?

First when running the proxy with caching disabled, the time to get the files was 10 seconds. Then with proxy enabled and the right size set, the time to get the file significantly decreased to 4 seconds. Which is exactly what caching is supposed to do, allowing for an increased performance.

What did you learn about system design from this class? In particular, describe how each of the basic techniques (abstraction, layering, hierarchy, and modularity) helped you by simplifying your design, making it more efficient, or making it easier to design.

What I learned about this class, that when using techniques like abstraction, layering, hierarchy, and modularity, is that you are able to exponentially reduce the complexity of the system. When designing something like http proxy, I was able to use modularity to break down the design into different parts allowing me to work piece by piece, making the whole design more clear and understanding what I was supposed to do where. Abstraction allowed me to make sure that the things I was implementing actually worked, for example when removing a file out of the cache, if that function works I can expect consistency when using that function. Also abstraction makes it easier to do testing on certain implementations since I could test that function in a separate environment and see if the output is always consistent. Hierarchy allowed me to combine subsystems into one coherent system, like the data structs for the list of servers and cache datastructs could be combined into the shared data struct which could eventually combine the threads, allowing for a system where threads can all work together with the same data. To combine the abstractions, I created and build on top of each other, allowing for a connected system, creating a coherent system.