Surya Rani

Design Doc asg3

Whats the goal?

The program **loadbalancer** will act as the middleman between the client and server. The load balancer will listen on a single port and will take in multiple client requests. It will then take these requests and send it to a prioritized server that it is connected to. We prioritize a server by requesting healtchecks from it and deciding which server to choose from based on who has the least amount of requests and errors. We repeat this process **every 5** seconds to check the health checks of the server to know if they are functioning or not. If a server does not send a response we assume that it is down and will not prioritize it until it comes back online and is able to send a response in a later health check.

How to implement

Structs

- ServerInfo - holds each individual server info such as fd, alive, errors, totalRequests
- Servers - holds all the servers in an array, int numServers

Globals

- Global queue to put client requests on
- Internal error message 500 to send when necessary
- Mutex to be used for threads
- The priority server index initially set to 0

Main

- Parse through program arguments similar to last asgn
- Check that if -R or -N is included to get the numbers
- Add all server ports to an array
- Connect to all server ports in loop
- Create thread to handle healthchecks periodically which calls **timedHealth(void* obj)**
- In infinite loop accept client connections and add them to queue
- Call **bridge_loop()** to be able to send data back and forth through the two file descriptors

clientConnect()

- Take in port number and connect to the server

serverListen()

- Take port number and listen for incoming client connections only on this port number

bridgeLoop()

- Take two file descriptors and continually call **bridgeConnection()** until one of them closes the socket
- This will send data to and from the fd's

serverInit()

- Loop through each server and intialize all variables that were set in the struct

checkHealth()

- This will loop through each server and first check if the server is alive
- Then it will call **parseHealth(int i)** to parse through the health check for each server
- Check that the server is alive again

parseHealth(int i)

- It will take the index given in the argument and call **getHealth(server[i].fd)**
- Gets message from getHealth and parses through it
- Checks for 200 code and gets the error count and total request amount
- If not in correct format then mark that server is down and set errors and total to -1
- Store this to **server[i].errors** and **server[i].total**

getHealth(int fd)

- Constructs healthcheck get message and sends to fd given
- Receives information or checks to see if their is no response
- If no response mark that the server is not alive
- Return received message back

prioritize()

- Loop through each server and compare totals
- Whichever has lowest total return that index
- If multiple servers have same total then tiebreak with success rate and return index

timedHealth(void* obj)

- This is the function called by our thread in main
- Call mutexLock
- Create timing struct to be able to measure time and if it is 5 seconds later
- My design decision is to do a healthCheck every 5 seconds
- Use **pthread_cond_timedwait(&cond,&mut,timeStruct)**
- call checkHealth() and set priority = prioritize()
- Call mutexUnlock
- Reconnect to all the servers because they will have been closed by the server to be able to send requests in bridge_loop