Design Document

Sooyoung Kim
cruzID: skim208

CSE130, Spring 2020

1 Goal

The goal of the program “loadbalancer” is to make a connection between a client/clients and a server/servers with tracking the performance of each server. The loadbalancer prioritizes the server that has fewer entries than other servers have to forward requests. We can set the number of parallel connections and number of requests (for getting a health check in every this number of requests) bypassing the arguments "-N n"(n is a number of threads) and "-R r"(r is a number of requests) respectively.

2 Assumptions

I am assuming that loadbalancer program would throw appropriate status codes for each different case. I must assume that the load balancer would throw 500 Internal Server error if none of the servers are up and wait until at least one of them becomes alive. Also, it would throw 500 error if there were any errors occurred while connecting the sockets. Other possibles for getting different status codes are identical to Assignment 2. Similar to Assignment 2, I must assume that there might be more requests than the number of threads that handle tasks. Therefore, I need to queue up the requests for later threads who are done with their jobs. Also, I assume that I have to create another different thread for the health check in order to signal the GET healthcheck request independently from the worker threads. I must assume that I have to update the entries and errors for each server after receiving the healthcheck data.

3 Design

The general approach that I’m taking is to utilize the starter code. I assume that the starter code already has some functions for binding the client socket with the server that is up. First I would use getopt() to see arguments that are passed in. If there is -N following with a number, I would set that to the number of worker threads. If there is -R, I would set that to the number of requests. The arguments that are left will be in the order of one

client port number and server port numbers for the rest. Then, I would calculate the number of servers, create server arrays, and initialize the information of each index of server arrays such as errors, entries, port number, and liveness. I would then create worker threads and health check thread. Check if the server/servers are up and if all of them are down, I would send 500 errors. I would also keep track of the number of requests and if it's divisible by r which I got from the arguments, signal the request for a healthcheck. If it gets signaled, I would send GET/ healthcheck request in order to get errors and entries. I would go through the for loop to iterate the server arrays and update the entries and errors for each server. If the response is not 200 OK, I would set the server to down status. I would set the timer for 20 seconds so that it would update healthcheck every 20 seconds. I would also make the load balancer wait for 5 second before sending the error response. Worker threads will just forward the request from the client to the server and forward the response from the server to the client. Using the if statement, I would choose the port number that has the least amount of entries to forward requests.

4 Pseudocode

This is the pseudocode for the load balancer program

```
Struct server{
        Port
        Entries
        Errors
        Updown
}
 Queue{

}
Starter Code{
        Client_connect
        Server_listen
        Bridge_loop
        Bridge_connections
}
Void handle Health(){
```

```
        Lock
        Signal
        Time = 20
        If (server up) {
                GET healthcheck
                Wait 5sec
                Receive data
                Update server.entries
                Update server.errors
        else(server down){
                Update server.updown = down
        unlock
}

Void handle task (){
        Lock
        while
                Worker threads wait
        Workerthread got the request!
        dequeue
        Unlock

        bridge_loop(send sockets for accepting and connecting)
}


main(argument count, argument array)
        N = 4, R = 5
        getopt(N, R)
                if(N)
                        N = numThread
                        if(n < 1) exit
                if(R)
                        R = numRequest

                if(optind)
```

```
        Listenport = argv[optind]
        Start of connectports = argv[optind +1]
        Number of servers = argc - (optind +1)
        Initialize the connect ports {
            for(i < numservers){
                Server[i].port = argv[optind = i];
                Server[i].error = 0;
                Server[i].entries = 0;
                Server[i].updown = down;
            }
        }
        if(port == “”) no port
            Send 500 errors
        if(port != number && port == 0) exit

server_listen(listenport);
Create queue
Create n threads(in handle task)
Create health thread
Loop until EOF
    accept(client request)
    if(fails)
        Exit
    (if sofar request == numRequest){
        Sofar = 0
        Lock
        Signal healthcheck
        Unlock
    }
    sofar++
    Find port that has least entries{
        Connecting  = client_connect(port with least entries)
    }

    lock
    enqueue(client request)
```

```
        Signal worker threads(with accepting and connecting socket)
        unlock
}
```