Project 2 — Load balancer

Overview

In this project, we will learn how to model a simulation of a large company load balancing web requests using a load balancer.

This project will require the ability to create multiple Load Balancers, and a real-world implementations of a queue for each. We also need a struct, a lower-level data structure that allows data to be contained in an object (requests).

Loadbalancers are available commercially, but in this project you will build one that has:

- Requests - generates random IP addresses, both in and out, and random times for each request to be processed. This time can be considered clock cycles, so each request will likely take a minimum of 3 cycles. You do not have to use a system clock for this, an iterator is fine.
    - IP in
    - IP out
    - Time (integer)
- Webserver(s)
    - Receives requests from the Load Balancer
    - Process the requests
    - Asks for another
- Load Balancer
    - Queue of requests
    - Keeps track of time.

Project Objectives:

You will be able to:

- **Show** your understanding of building and using a load balancer.
- **Develop a guide** for your loadbalancer that does not get overloaded or underutilized.

Instructions:

1. This is an individual project, you must turn in your own work.

2. Collaboration for ideas is acceptable, duplication of work is not.
3. Use C++ to write this program.
4. Your program will contain multiple .cpp files. The main program should set up the number of servers (user input), the time you want to run the load balancer (user input) and generate a full queue (usually servers * 5).
5. You will need a webserver class so you can create the number of webservers requested.
6. You will need a request class (or struct) that holds a request.
7. You will need a queue of type request. (called requestqueue?)
8. You will need a loadbalancer that manages the webservers and the request queue.
9. You will need to add new requests at random times to simulate new requests after the initial full queue you set up.
10. Exit when the simulation time (can be input) is complete. Call othe rfunction
11. Try different numbers as input to understand the capacity of your loadbalancer configurations.
12. Create logic that adds and removes webservers as the load increases and decreases.
13. All files must be commented with Doxygen using html output and loaded to a page on people.tamu.edu. Turn in this link.
14. Demo the loadbalancer in class.

Deliverables:

All deliverables must be at least a minimum viable solution to the problem assigned. No attempt at grading will be made for non substantial submissions.

1. Create documentation for each of your code files in HTML, hosted on people.tamu.edu
2. Create a log of 10 servers running for 10000 clock cycles
3. Turn in your git repository link.
4. Demonstration of usability.

Grading:

This project is worth 100 points.

Documentation 30%
Log and successful completion of load balancing 20%
Demonstration, code, and Git usage 50%