Checkpoint 2

Alex Costinescu and Suman Hosmane

I. Changes to Proposal

Due to time constraints, we have decided to remove the configurable GUI portion of our project to instead focus on the core functionality of the load balancer and deploying the balancer and servers to GCP.

II. New Timeline

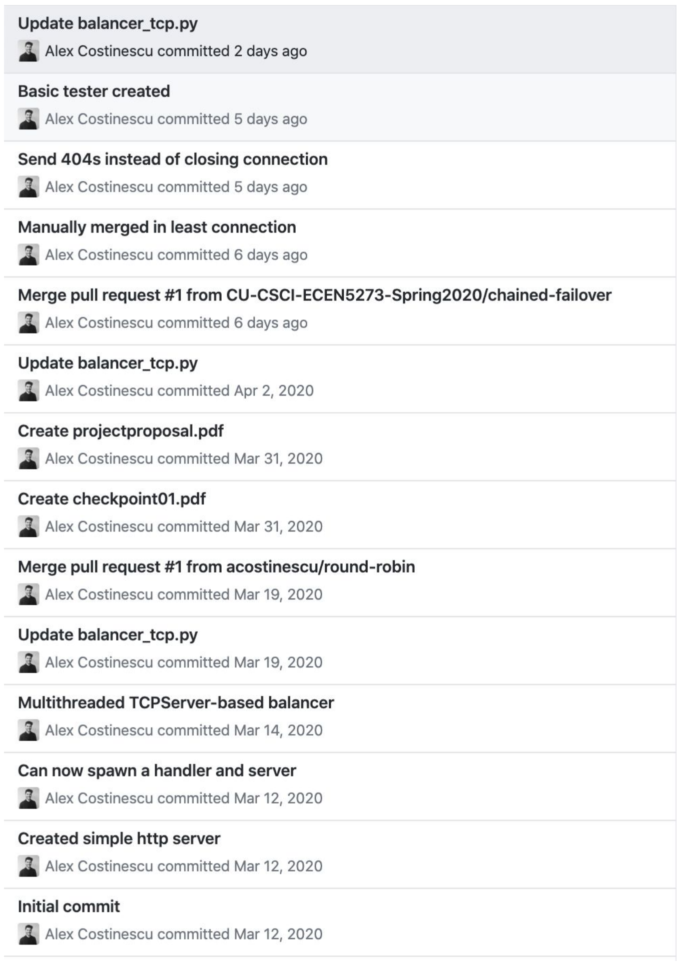
Status	Task	Completed By
Done	Build a basic handler which is able to forward requests and responses between a client and a server	Alex
Done	Add the ability to configure the load balancer with many servers using Round Robin	Alex/Suman
Done	Implement Least Connection	Suman
Done	Implement Chained Failover	Alex
N/A	Build a UI for load-balancing strategy tweaking	N/A
Done	Build a testing tool	Alex
Done	Develop script to randomize request serve times	Suman
April 20	Deploy on GCP	
April 28	Run tests and gather results, prepare results for presentation	
April 30	Write final report	

Table.1: Combined Timeline table

🟦 **Checkpoint 1** 🟧 **Checkpoint 2**

Commit Screenshots:

As you can see, we have been consistently working throughout the project. There are two sets of commits because we originally had a private repo which we transitioned to the repo that is now in the Github classroom.



Update balancer_tcp.py
Alex Costinescu committed 2 days ago

Basic tester created
Alex Costinescu committed 5 days ago

Send 404s instead of closing connection
Alex Costinescu committed 5 days ago

Manually merged in least connection
Alex Costinescu committed 6 days ago

Merge pull request #1 from CU-CSCI-ECEN5273-Spring2020/chained-failover
Alex Costinescu committed 6 days ago

Update balancer_tcp.py
Alex Costinescu committed Apr 2, 2020

Create projectproposal.pdf
Alex Costinescu committed Mar 31, 2020

Create checkpoint01.pdf
Alex Costinescu committed Mar 31, 2020

Merge pull request #1 from acostinescu/round-robin
Alex Costinescu committed Mar 19, 2020

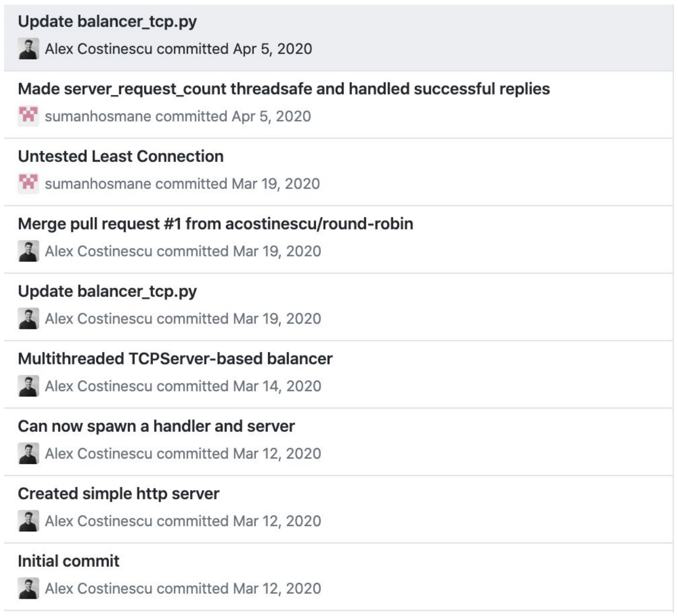
Update balancer_tcp.py
Alex Costinescu committed Mar 19, 2020

Multithreaded TCPServer-based balancer
Alex Costinescu committed Mar 14, 2020

Can now spawn a handler and server
Alex Costinescu committed Mar 12, 2020

Created simple http server
Alex Costinescu committed Mar 12, 2020

Initial commit
Alex Costinescu committed Mar 12, 2020


Update balancer_tcp.py
Alex Costinescu committed Apr 5, 2020

Made server_request_count threadsafe and handled successful replies
sumanhosmane committed Apr 5, 2020

Untested Least Connection
sumanhosmane committed Mar 19, 2020

Merge pull request #1 from acostinescu/round-robin
Alex Costinescu committed Mar 19, 2020

Update balancer_tcp.py
Alex Costinescu committed Mar 19, 2020

Multithreaded TCPServer-based balancer
Alex Costinescu committed Mar 14, 2020

Can now spawn a handler and server
Alex Costinescu committed Mar 12, 2020

Created simple http server
Alex Costinescu committed Mar 12, 2020

Initial commit
Alex Costinescu committed Mar 12, 2020

III. Evaluation

We will be evaluating each load balancing method using a custom tool we are working on building using Javascript. This tool will allow the user to configure the number of requests to send the server as well as the request frequency. It will then asynchronously send the requests to the server and await the responses for those requests.

Once the tool has received all of the responses from the server, it will show how long it took to complete all of the requests as well as how many requests were successful vs how many resulted in 404 errors (which our load balancer responds with when it is unable to select a suitable server).

Using this tool, we will test our load balancer with each balancing strategy as well as testing using just a single server to handle responses as a baseline. For each strategy, we will test a number of situations (many requests at once, many requests over time, few requests over time, etc) to determine how it performs compared to other strategies as well as the baseline.

For testing, the load balancer and servers will be run on GCP virtual machines networked using a VPC. This plan has not changed from our original plan for testing and running.

IV. Challenges

- Creating a GUI could not be fit into the available time frame and had to be moved out of the scope of this project. On the other hand, a GUI would not have increased our learning in terms of the networking application, it would have only enhanced the aesthetics of the tool and made operation of the load balancer easily customizable.

Checkpoint 1

Alex Costinescu and Suman Hosmane

I. Problem Description

As part of the final project required for the course, we decided to implement a load balancer setup. To give an introduction, a load balancer is one of the most basic components required to run an efficient and fail-safe customer facing real-time network architecture. A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load balancers are used to handle multiple simultaneous users and reliability of applications. They improve the overall performance of applications by decreasing the burden on servers associated with managing and maintaining application and network sessions, as well as by performing application-specific tasks.

In a real-world scenario, where millions of requests are generated every second, it is important that most services are running on multiple server instances distributed across geographical locations. This facilitates quicker response for each request while also making it easier to enable backup plans in cases of failure.

Load balancers can be implemented as an independent hardware or a software running on another switch/server instance.

There are multiple strategies for implementing a load balancer. Some of the common methods used in load balancers are,

- Round robin
- Weighted round robin
- Least connections
- Chained Failover
- Least response time
- Weighted Response Time

As part of our project, we will be implementing a basic network architecture on GCP (Google Cloud Platform) consisting of a single VM instance running the load balancer application, 3-4 VM instances serving the requests directed by the load balancer. We will implement 3 strategies and compare the results and turn around times with each of the modes to draw a better conclusion as to the best strategy for each situation.

II. High-Level Architecture

As part of the implementation, we shall follow the most common architecture in the industry, where the load balancer receives the requests from end-users or clients and based on the load balancing strategy or mode defined: redirects the requests to the appropriate server. A pictorial representation of the architecture can be seen in Figure.??.

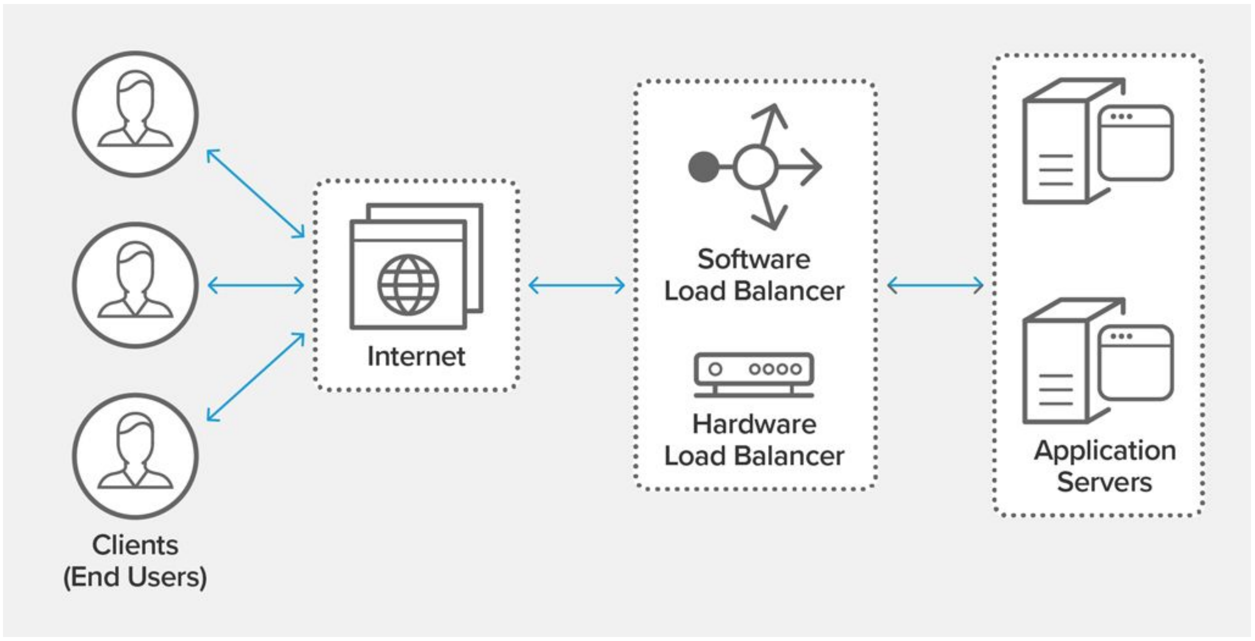


Figure 1. High-Level architecture diagram of the setup.

Client requests come in with the external world IP address. DNS (Domain Name Server) resolves the request to an appropriate network which is handled by the load balancer (assuming there are no DMZ servers filtering out malicious requests). The application on the load balancer is pre-aware of the addresses for the internal servers. The load balancer adds the destination as one of the servers on the internal network, based on the various modes that are configured in the application. Our implementation involves 3 modes of operation as discussed below,

Round-Robin mode:

Requests are directed from the load balancer in a sequential fashion. The first request goes to the first server on its routing table, second to the second server and so on.

When the last server has been sent a request, the routing is reset to the first server and the routine restarts. Irrespective of the number of requests a server is already serving, requests are sent in a sequential fashion.

Least Connection mode:

In this mode, load balancing is achieved by being aware of the number of requests already being served by a particular server. New requests are directed to a server with the least number of requests running presently. This balances out the requests per server, since each request does not necessarily take the same execution time.

Chained Failover mode:

In this method, a predetermined order of servers is configured in a chain. All requests are sent to the first server in the chain by default. If it can't accept any more requests or if the server is unreachable the next server in the chain is sent all requests, then the third server and so on.

Once a request is processed, a direct reply from the server to the client can cause an IP address mismatch at the client firewall. To circumvent this scenario, we will be replying back to the client via the load balancer. The load balancer adds the appropriate client and server IP addresses and forwards the processed request to the appropriate requesting client.

In our setup, the load balancer will be a more powerful VM instance than the server considering that the requests are not too intensive but can arise plenty in number. On the Google Cloud Platform, we will be hosting 3-4 virtual machines with a single core CPU, 512MB memory as servers and a 2 core CPU, 1GB memory machine for the load balancer.

III. Dataset/Tools

While industry tools do exist for testing HTTP loads/load balancing, they are expensive and unnecessarily complex. To test our implementations against each other and against a baseline, unbalanced stack we will build a simple tool in javascript that can be configured to send bursts of requests.

The load balancer itself will be written in Python using libraries for websockets, threading, HTTP request forming, etc. The servers will also be written using Python and will provide basic functionality for testing. The load balancer and servers will all be hosted on GCP for testing.

IV. Challenges

There are several challenges that we will need to solve in implementing our load balancer. The first and biggest challenge is implementing it in a way that is transparent to the client. Since a client expects the same port/IP address combination to respond to its request as the port/IP that it sent the request to, we will need to be careful about how to implement our balancer to preserve and/or spoof that combination. To address this, we have the option of either implementing the balancer as either an inline or a proxy load balancer. In other words, we need to select if the load processor will also be responsible for handing back the responses to the client, or if servers will send responses directly with a spoofed IP/port.

Another challenge that we will need to address is how the load balancer will obtain information about the servers. Certain load balancing strategies, such as least connection, require the load balancer to be aware of server loads/availability. We will need to build an efficient way for the load balancer to maintain/obtain information about the servers.

A third challenge is that the load balancer will need to be able to handle many requests at the same time (and potentially also responding to those requests given an inline balancer). Given this, we will be looking into implementing multithreading into the load balancer so it can handle many requests asynchronously.

V. Timeline

Done	Build a basic handler which is able to forward requests and responses between a client and a server
Done	Add the ability to configure the load balancer with many servers using Round Robin
Done	Implement Least Connection

April 16	Implement Chained Failover
April 16	Deploy on GCP
April 16	Build a UI for changing which strategy to use
April 20	Build a testing tool
April 28	Run tests and gather results, prepare results for presentation
April 30	Write final report

VI. Concerns

- Problem with direct interaction of server and client during reply (due to firewall). Though replying to the client directly from the server is an easier and time saving approach, firewall security does not allow us to take this approach to increase security of traffic entering the client machine.

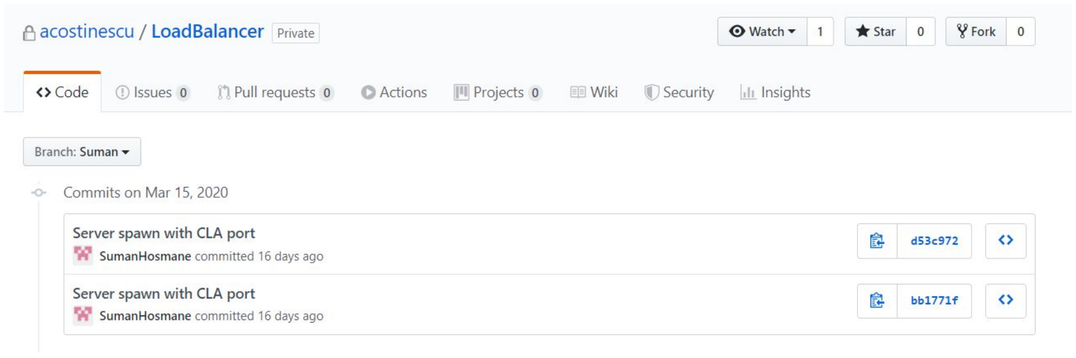
- Least Connection and Chained Failover mode load balancing requires a reply from the server about the number of requests it is serving.
  Since these 2 modes require information about the server status, even before the request is forwarded to the servers, a preliminary information is required by the load balancer to make a decision as to where a particular request needs to be redirected to.

VII. Changes Made to Project Proposal

Initially the architecture involved a direct reply from the server to the client eliminating a need to pass the reply through the load balancer. However this required either using virtual IPs or opening special aspects in the client's firewall. To make it simpler, the current architecture involves replying to client through the load balancer.

VIII. Student Check-Ins

These commits were to spawn server instances with a custom port using command line parameters.


acostinescu / LoadBalancer Private

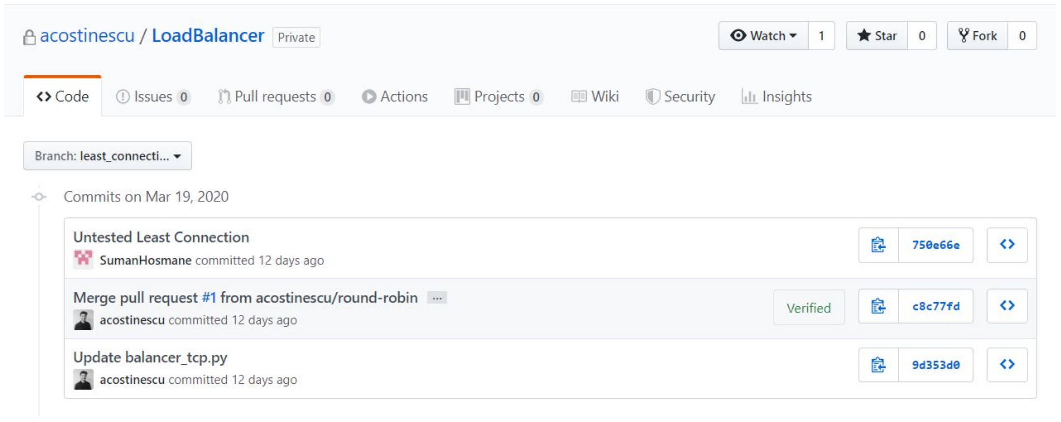
Branch: Suman

Commits on Mar 15, 2020

Server spawn with CLA port
SumanHosmane committed 16 days ago — d53c972

Server spawn with CLA port
SumanHosmane committed 16 days ago — bb1771f

Implemented a preliminary working of the least connection load balancing mode without re-entracy check.


acostinescu / LoadBalancer Private

Branch: least_connecti...

Commits on Mar 19, 2020

Untested Least Connection
SumanHosmane committed 12 days ago — 750e66e

Merge pull request #1 from acostinescu/round-robin
acostinescu committed 12 days ago — Verified — c8c77fd

Update balancer_tcp.py
acostinescu committed 12 days ago — 9d353d0

IX. Evaluation

We are planning to use the GCP for setting up the virtual infrastructure. Since there is a $300 credit we are planning to use the same for creating low powered VMs. The basic outline of the testing is to create a barrage of test requests, varying from a simple html page requisition to running scripts that take 4-8seconds for execution.

The modes of load balancing will be changed and the turn-around time for the same number of requests in each mode will be compared to obtain an idea of the most suited mode for our application. In the process we will also be analyzing the strengths and weaknesses of each mode.

Original Project Proposal

Alex Costinescu and Suman Hosmane

Project Objectives:

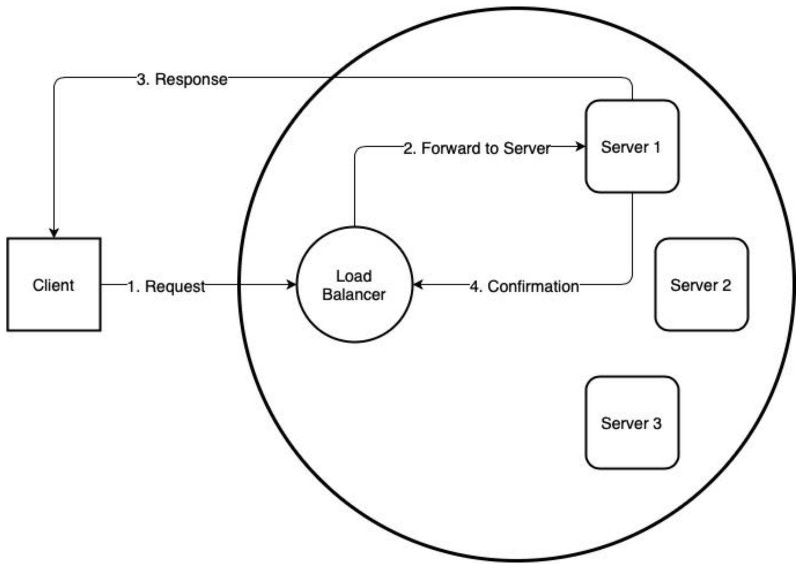
- Implement a configurable load balancer with a basic infrastructure
- ~~Add GUI for load balancer management~~
- Deploy infrastructure on Google Cloud Platform VMs

Initial Goals:

- Support for 3 load balancing strategies:
    - Round Robin
    - Least Connection
    - Chained Failover
- Allowing weighing of servers for certain strategies
- GUI allows for switching between strategies on the fly
- Deploy on GCP
- Performance analysis of each strategy

Stretch Goals:

- Add more load balancing strategies with increased complexities
- Servers report current load to balancer


3. Response
2. Forward to Server
Server 1
Client
1. Request
Load Balancer
4. Confirmation
Server 2
Server 3