## Team 7: Dennis (Xu) He, Yanchi Li

## Project: Super Load Balancer

### Introduction

The inspiration behind our load balancer project is that modern technology companies that handle multi-million requests per minute use load balancer to achieve their customer's requests at scale. Amazon, for example, offers a service called "Elastic Load Balancing" through AWS that works with other AWS services such as EC2, Lambda, SNS, SQS, and much more. We can see how important and critical the load balancer is through the AWS service since it is embedded in the whole ecosystem deeply and is compatible with most of their other services.

### Project Goals

Before starting the project, we had several goals for this project. Below are the goals we wanted to achieve and labeled them as regular goals and stretch goals:

- Implement load balancer with weighted round robin algorithm that dispatches client requests to different servers (regular).
- Have multiple engineer thread on each server to service requests (stretch)
- Having chain replication and recover feature that works with load balancing (stretch)

### What we achieved and wasn't able to achieve

We were able to implement the weighted round robin algorithm that works with the load balancer by distributing workloads to different server nodes. Below is how the process work:

- Client sends robot requests to the load balancer
- Load balancer distributes to workload to different server nodes
- Server nodes process the requests
- Server nodes send back completed requests to load balancer
- Load balancer sends completed requests to the client

Each server node has multiple engineer threads to work on requests.

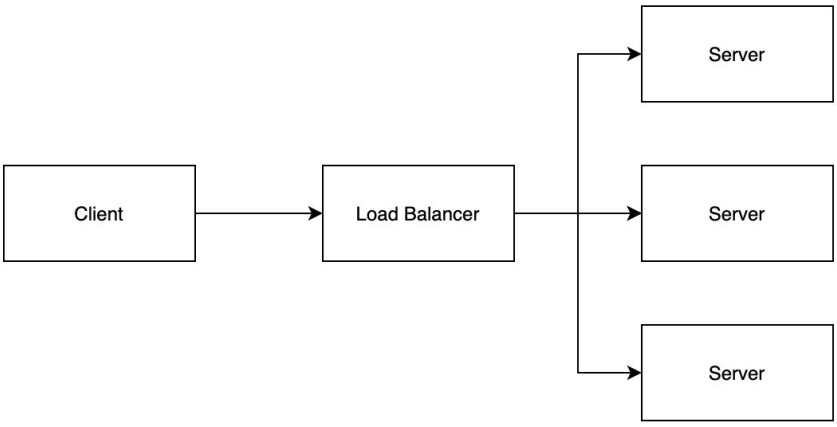
We were unable to achieve our stretch goal of having chain replication and recovery features that work with the load balancer.

### Design and Implementation

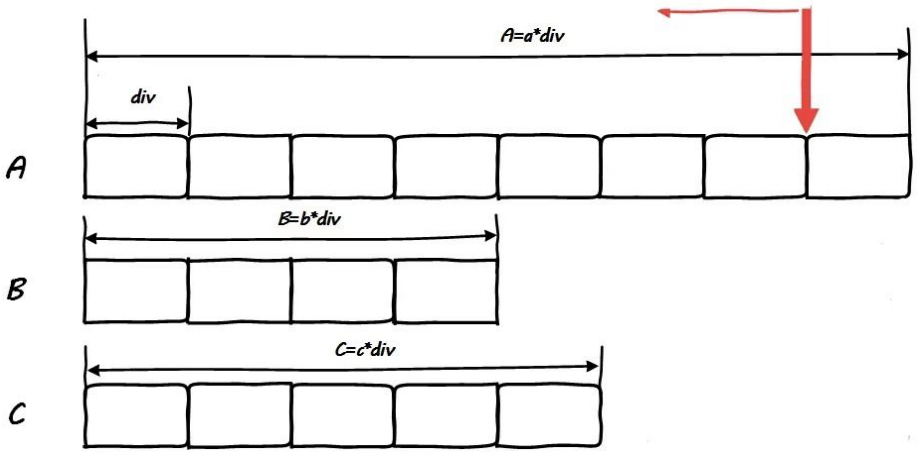
Assumptions:

- The load balancer and server nodes will not fail
- The capability of each nodes can be reflected by weights

Our load balancer design is pretty straight forward; the client will communicate with the load balancer by sending various robot requests to the load balancer. Then the load balancer will dispatch client requests to multiple servers based on the weighted round robin algorithms. The servers will then process the workload received from the load balancer and send back completed requests. And lastly, the load balancer will send the client the completed requests that it received from the server nodes.

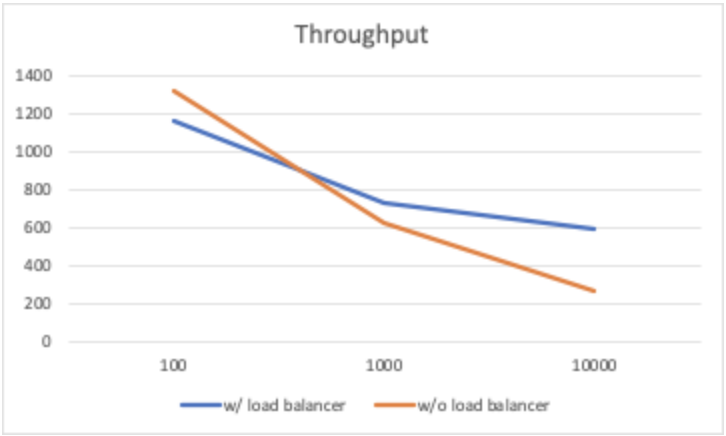



The basic idea of the weighted round robin algorithm is to assign different weights to all server nodes. These weights could be reflecting the capability of each node or the network condition between load balancer node and server node. In our project, we simply assigned weights randomly for simplicity. Once we have all those weights for each server node, we are able to split the weights into small divisions. The size of each division is the greatest common divisor of all weight values. When the load balancer receives a request, the load balancer first searches for the largest weight among all nodes. Then the load balancer distributes the request to it and deducts the weight of this target node by exactly one division size. So the intuition here is to always distribute the new request to the server with the largest weights and deduct its weight until it is not the top-weight node.


A=a*div
div
A
B=b*div
B
C=c*div
C

## Evaluation


Latency
1400
1200
1000
800
600
400
200
0
100
1000
10000
w/ load balancer
w/o load balancer


Throughput
1400
1200
1000
800
600
400
200
0
100
1000
10000
w/ load balancer
w/o load balancer

- Performance with and without the load balancer. For example, we tested the load balancer with 3 server nodes. For testing without load balancer, we can just test load balancer with just 1 server node (I believe this will mimic the performance without a load balancer since there is just 1 server). We tested this with different work loads (eg. 100, 1000, 10000 requests).
- We also tested the load balancer with 3 server nodes with different workloads (eg. 100, 1000, 10000 requests).
- The two charts above display the performance of executing certain workloads with and without the load balancer.
- The results are what we expected; the system with the load balancer displayed increased latency because of multi-server communication. However, we can see that the throughput is much better with the load balancer as workload increases because multiple servers are able to service client requests.

**Learning Outcomes**

We learned a lot while doing this project especially how a load balancer fits into a system in modern technology services. Below are some learning points for this project:

- How a load balancer helps to distribute requests among servers
- How to implement a weighted round robin algorithm that dispatches requests to different servers based on weights
- How to allow the load balancer to communicate with both client and server and relay information between them
- How load balancer can improve system overall performance by dispatching requests to different servers

**Instructions to Setup Test Environment**

1. Run make command in the root directory of our project to build server, client and load balancer programs
2. Start up the load balancer executable by passing in [number of servers(up to N), port number, ip_address server1, ip_address server2, ip_address server3, ip_address serverN]
3. Start up the server main executable by passing in [port number, serverId(int that represents this server)]
4. Start up the client main executable by passing in [ip_address of load balancer, port number, number of customers, number of orders, request type 1]

Please set all the port numbers to 12345 because we use a default value of 12345 in our load balancer program. We've run tests with 1 load balancer and 3 servers and we assign the number of orders to 100, 1000 and 10000. We've also tested the performance when there is

exactly 1 server node in the system. The number of orders are the same as the tests for 3 servers.

**Contributions**

Dennis and Yanchi both focused on achieving the goals set out for this project. Dennis worked on the server side implementation, and Yanchi worked on the load balancer implementation. We then connected the load balancer with the server to process workloads from the clients. Overall both of the members really enjoyed working on this project.