DISTRIBUTED SYSTEMS

Load balancer to HTTP server

Presented By

Group 16

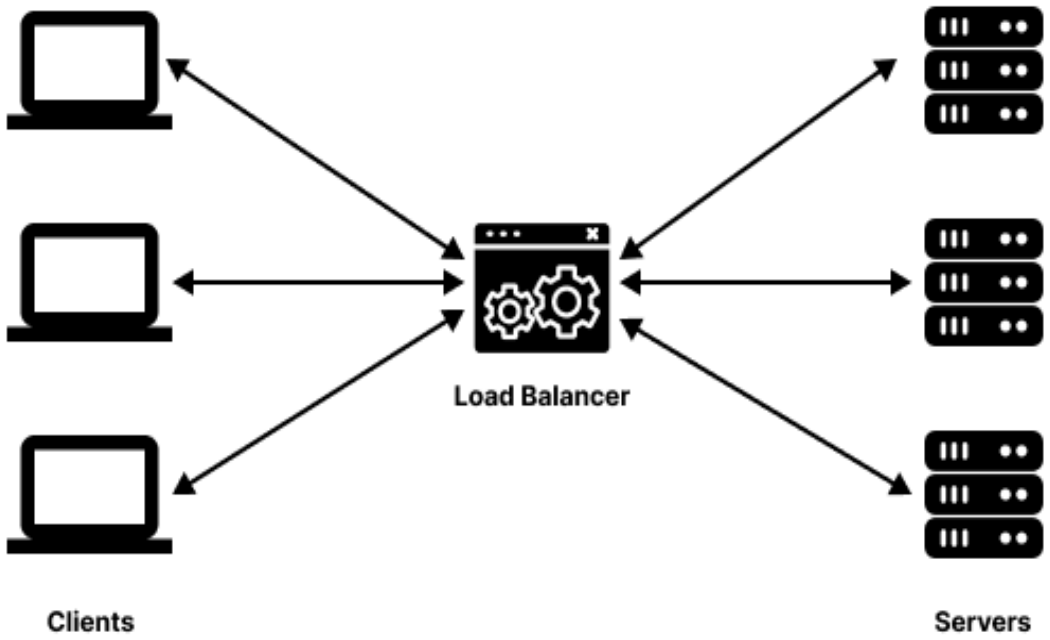
USTH

Overview

01. Problem
02. Design
03. Scenario
04. Result

DISTRIBUTED SYSTEMS

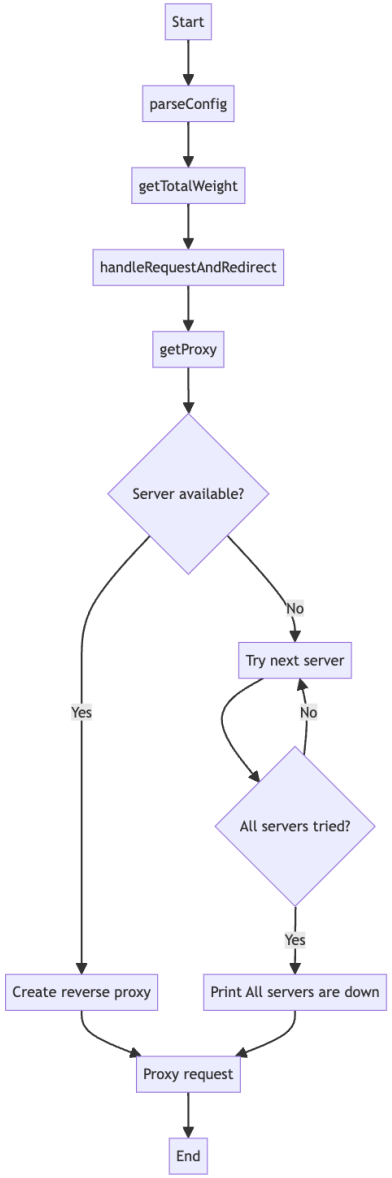
University of
Science and Technology
of Hanoi



Problem

Load balancer to HTTP server

The goal of this study is to evenly distribute incoming web traffic across multiple servers, preventing any single server from being overwhelmed. This ensures optimal performance, reliability, and availability of the web application.



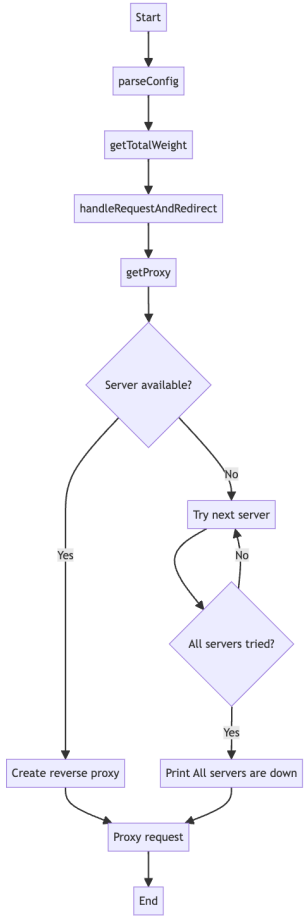
Design

Configuration Parser:

The load balancer supports reading server configurations from config.yaml.

If the YAML file is not found or has an error, the load balancer uses a default set of servers defined in the code.

The configuration file specifies an array of server configurations, each containing an address, port, and weight.



Design

Server Representation:

Each backend server is represented by a *Server* struct, which contains the server's URL and a weight value. The servers slice holds all the configured backend servers.

Round-Robin with Weighted Distribution:

The load balancer uses a round-robin algorithm with weighted distribution to select the next backend server for each incoming request.

Design

Health checking

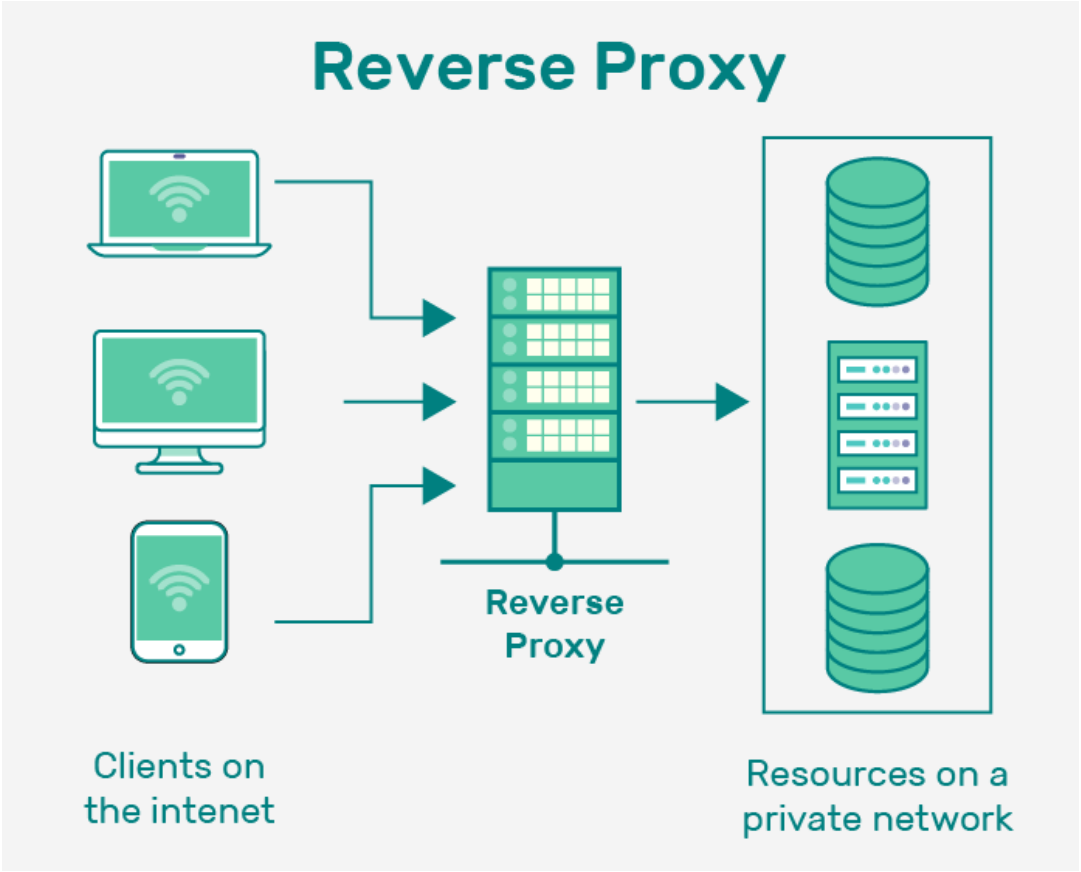
The load balancer checks the server's health by sending a GET request to the /health endpoint, if:

- Sever available creates a reverse proxy
- Sever not available: try next server
- All sever are down

Request handling

The load balancer listens on port 8080 for incoming requests.

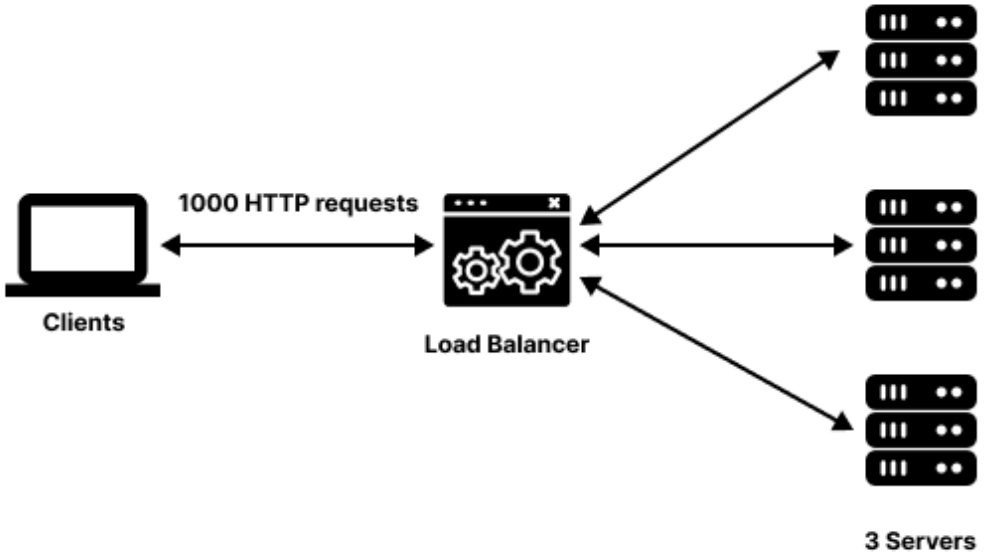
The **handleRequestAndRedirect** function retrieves a reverse proxy using the **getProxy** function.



Design

Reverse Proxy

- The load balancer uses the httputil.ReverseProxy to forward request to BE server.
- The ReverseProxy automatically handles tasks and sending respond to the client.



Testing Scenario

We have a load balancer distributing 1000 requests from a client to three servers(containers).

Each server has a different weight assigned to it, indicating its capacity to handle requests. The load balancer logs each request, recording the client IP address and the server it was directed to

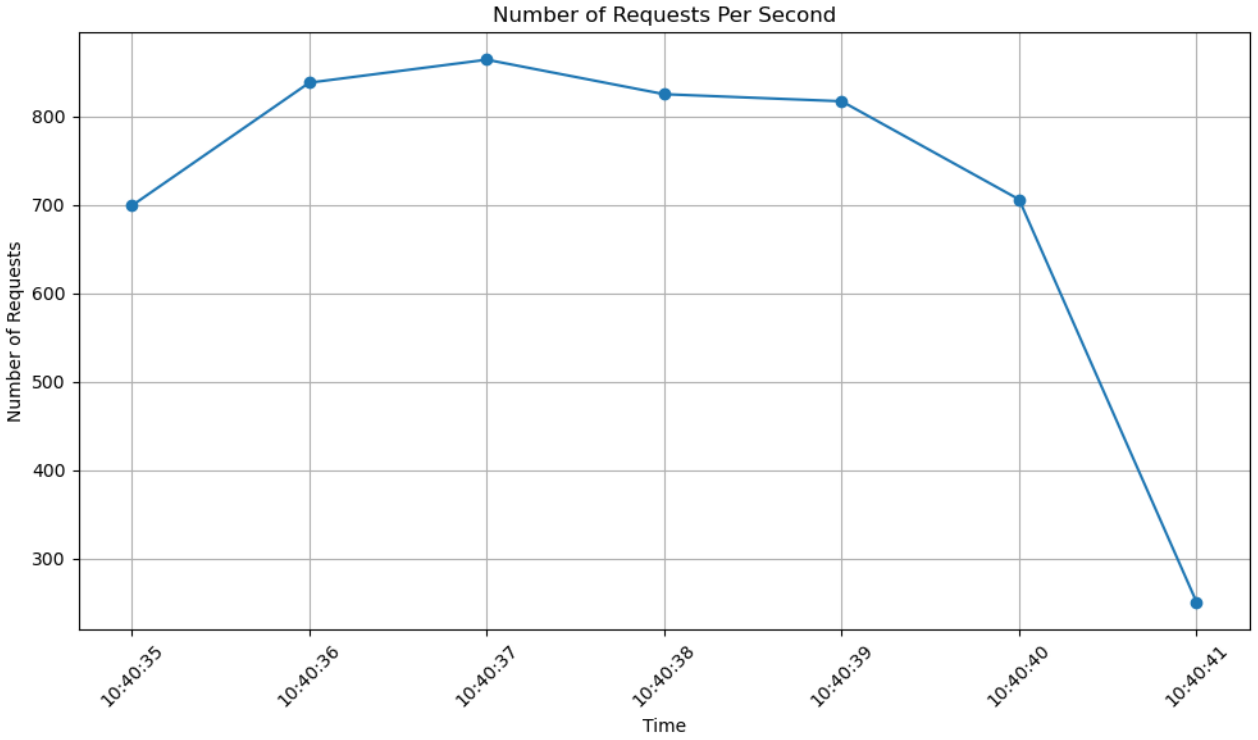
```
Analyze for all 3 servers up:
CLIENT 0: 172.20.0.1
172.20.0.3: 334 requests
172.20.0.2: 500 requests
172.20.0.4: 166 requests
====================

Analyze for all 2 servers up:
CLIENT 0: 172.20.0.1
172.20.0.2: 499 requests
172.20.0.3: 501 requests
====================

Analyze for all 1 servers up:
CLIENT 0: 172.20.0.1
172.20.0.3: 1000 requests
```

Distributing Result

Load balancer Distribution and Failover



Performance Result

Result around 800rps

Conclusion

- The load balancer distributes incoming HTTP requests to multiple servers based on the round-robin algorithm.
- The health check feature is implemented to check the status of the servers.
- The result shows that the load balancer distributes the requests based on the weight of the servers.
- Performance testing showed the load balancer could handle around 800 requests per second from a single client under the testing conditions.

Overall, this solution effectively distributes HTTP traffic based on weighted capacity and health status

THANK YOU

USTH VIETNAM FRANCE UNIVERSITY
University of Science and Technology of Hanoi