Network Load Balancer

For this project, I made a network load balancer using different network loading algorithm. The supported network loading algorithm are: Retry Rule, Round Robin Rule, Random Rule, Weighted Response Time Rule, and Best Available Rule.

The repository contains the three modules. They are consumer module, provider module, and server module. Server module is used to start the Eureka server using Spring Boot. The consumer module contains the various mapping that contains the different network loading algorithm and different routing url logic. The provider module contains the service that is used by the consumer module.

When we send the http request to GET information from the server, the server will handle the destination and which service will handle the request. The logic is completely determined by the network loading balancer algorithm.

How to run the project

Download the project

1. `git clone https://github.com/cydy8001/Network-Load-Balancer.git`
2. `cd spring-cloud-parent`

Basic setup

Java version is JDK1.8 and Maven version is 3.6.3 Make sure the 3 modules are using the same JDK 1.8. Otherwise, there will be conflict.

Run Maven

Maven contains many dependencies. So, it is important to download all dependencies required. If you are using Intellij, you can right click the 4 pom.xml files and load the dependencies. In your Run Edit configuration: Allow multiple instances in Provider.

Config

You can modify the application.yml file under the resources folder.

```
server:
  port: 8000

eureka:
  instance:
    hostname: localhost
    prefer-ip-address: true
    ip-address: 127.0.0.1
    instance-id: ${eureka.instance.ip-address}:${spring.application.name}:${server.port}
    lease-renewal-interval-in-seconds: 3
    lease-expiration-duration-in-seconds: 9
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
spring:
  application:
    name: eureka-provider
```

All of the consumer ,provider and server have the application.yml file. So, you can modify all of them but I suggest keeping them default.

Implementation

After doing some research on how to build this project, I learned that the network load balancer is mostly used in many companies in backend. Due to the large amount of traffic, it is very important to split the pressure from one server to multiple servers. By this way, the server will not be overloaded.

Test

1. Round Robin Rule
    - GET: localhost:9000/oder/goods2/1
    - title: Iphone13pro: 8002
    - title: Iphone13pro: 8001
    - title: Iphone13pro: 8000
    - The port numbers are 8002, 8001, and 8000 will follow the Round Robin Rule
2. Random Rule
    - GET: localhost:9000/oder/goods2/1
    - title: Iphone13pro: 8002
    - title: Iphone13pro: 8002
    - title: Iphone13pro: 8000
    - The port numbers will be completely random

Members and Contributions

My partner dropped the course, I am the only member of the team

Yihang Cheng

- Designed the architecture of the project
- Brainstormed the technology we want to use, SpringCloud, Eureka, and Ribbon
- Implemented the project and make sure the structure is similar in the industry
- DAO, Service, Controller, and Service Implementation for provider and consumer
- Implemented the different network loading rule
- Implemented the different routing url logic
- Completed the README.md file

Analysis

In the end, I learned a lot from this project. SpringCloud and SpringBoot is widely used in the industry. Network Load Balancing is also a very important concept in the industry as well. I learned how to create MVC architecture with Load Balancing algorithm. With my implementation of the Ribbon load balancer, I am able to handle the traffic coming from client and split the traffic to different servers. I also learned how to use Eureka server to register the service and deal with those traffic.

Limitation of the project

- No real database
    - The response it returned is hard coded. It is not from the database. But if we have a real database setup, it will make the project more realistic. Also, how the server handles the traffic will be same but it will consider the time it takes to query the database. It also depends on the database design.
- The protecting mechanism needs to be improved
    - I am observing that if the clients maliciously send very large amount of GET requests at the same time, the server will be overloaded, If the number of providers is not enough, and the load balancing algorithm is not handling it well. Some of the responses will be delayed. So, I added the expiration mechanism in the yml file. But I think in the actual coding, it also needs to be improved to reflect this protection mechanism.
- The project is not deployed in the cloud
    - So far all changes are in local. When the project is on the cloud, it will be more realistic. How to proejct handles the traffic could be different, because there will be more traffic coming from different places.

Conclusion

Throughout the course, I learned so many topics. Different layers, network protocols and the related algorithms. I know there are so many projects that I can do which can make me apply the concepts in the class to the real world project. I found that network load balancing is the most appealing to me. It is also widely used in the industry so I think it is a very good practise for me to implement this project. During the course of the project, the hardest thing for me is how should I

design the architecture. I went through many documents and am aware that not only the network load balancer is needed, but also the design of the architecture should be similar in the industry. I also realized that springcloud is well-suited for this project because the library it has and the popularity in the industry are very high. By considering all of the factors, java along with spring framework is my choice.