International Core Journal of Engineering Volume 7 Issue 8, 2021
ISSN: 2414-1895 DOI: 10.6919/ICJE.202108_7(8).0014

# Research on an Optimization Algorithm of Load Balancing based on Ribbon

Jia Liang

College of Electronic and Information, Southwest Minzu University Chengdu 610041, Sichuan, China.

## Abstract

**With the rapid development of science and technology information, a large number of data information and system visits increase sharply, which leads all systems to face the problem of how to deal with high concurrency. At this time, it is generally handled by building server clusters and adopting load balancing technology to realize scientific scheduling of server clusters. In this paper, the load balancing strategy under micro-service architecture is studied, the default RoundRibbon load balancing algorithm in the Ribbon component of spring boot is optimized, and a load monitor is added, which not only solves the problem of excessive pressure on a single server caused by polling strategy, but also effectively avoids the situation that the server is unavailable or even avalanche due to overload of the server, and improves the stability of the system.**

## Keywords

**Micro Service; Load Balancing; Ribbon.**

## 1. Introduction

With the development of Internet technology, a variety of e-commerce platforms, such as Taobao, Jingdong and other online shopping APPs, continue to increase, which also brings great convenience to our daily life. Home shopping has also become the mainstream way of contemporary life. With the gradual increase in the scale of network users, it is undoubtedly to the website and application system to bring a huge amount of network visits. However, the increase in the number of visits also brings a huge load pressure to the back-end server. In the face of such high concurrent access demand, a single server device is difficult to bear. At present, the common method is to build a server cluster to improve the system performance to deal with the huge traffic demand. Server cluster to multiple servers to form a cluster system to provide users with a unified network services, this will increase the system ability of concurrent processing and single machine malfunction system error redundancy ability, effectively avoid single server cluster system can't normal when running abnormal situation, implement system of high reliability and high availability of resources [1]. Cluster system, how to reasonable allocation of system resources and the load balancing cluster system is the most critical problems [2], load balancing technology as the key technology of server cluster, usually the negative allocation strategy will load balancing in distributed to the backend server, parallel processing and the backend server load request [3, 4], improve processing efficiency cluster system.

## 2. An introduction to load balancing based on microservices

### 2.1 Server-side load balancing and client-side load balancing

The general principle of server-side load balancing is that the user sends a request first, and then through the load balancing algorithm, selects one of the multiple servers for access, that is, the load balancing algorithm is distributed on the server side, such as nginx, hapoxyLVS, LVS, etc.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202108_7(8).0014

The general principle of client-side load balancing is that the client will have a list of server addresses. Before sending a request, the client will select a server through the load balancing algorithm and then visit it. This is client-side load balancing, that is, the load balancing algorithm is allocated on the client, such as the Ribbon load balancing used in this paper.

Traditional server load balance in the face of high concurrency scenarios exist defects, such as a double tenth, jingdong618 when the client request volume surge, many large vendors will increase the server system to meet the demand in advance, the proxy server load balancing method required at this time in the load balancer to the new server IP configuration, after the success of the change to restart operations such as, Then read the server IP from the configuration file and load balance. If the promotion ends, it is necessary to recycle the new servers at this time, that is, to remove a large number of IP addresses from the proxy load. For businesses such as Ali Jingdong, which has tens of thousands of servers, it takes a lot of labor to use the server load balancing. Therefore, the client load balancing module of micro-service architecture is adopted. Once the new server is opened, Eureka Discovery and Registry will find the service and automatically register it. The client will obtain the service address from the registry and distribute the server through the Ribbon Loadbalance module to reduce the manual consumption. In this paper, the default RoundribBonrelu load balancing algorithm in its module is optimized to achieve a better scheduling mode.

### 2.2 Principle of Ribbon load balancing technology

Micro-service based load balancing is implemented through Eureka, a Netflix-based tool for service registration and discovery, and the Ribbon. The Ribbon is a load balancer released by Netflix. It helps control the behavior of HTTP and TCP clients. After the Ribbon provides a service address, the Ribbon can automatically help service consumers request it based on a load balancer algorithm. Eureka is responsible for service discovery and Ribbon is responsible for service consumption. Service discovery and consumption are actually two activities, which are performed by different objects: service discovery is done by Eureka client, and service consumption is done by the Ribbon. The Ribbon is a client load balancer based on HTTP and TCP. When we use the Ribbon with Eureka, the Ribbon retrieves the list of servers from the Eureka registry and polls them to achieve the function of load balancer. Whether the server is online or not is left to Eureka to handle.

## 3. Optimization algorithm design

### 3.1 Design of data acquisition module

A large number of studies have shown that evaluating the load of server nodes usually uses CPU utilization and memory utilization. Therefore, this project plans to use CPU utilization rate and memory utilization rate as load evaluation indexes to collect node load information in a certain time cycle, which can not only reduce the communication cost when collecting load information, but also understand the node load situation in real time. Determining an effective collection cycle is key. Frequent collection of load information will incur additional resource consumption. If the time is set too long, the collected load information may become outdated and do not reflect the current load of the server. In this paper, the method of periodically collecting load information of the server is adopted to determine the optimal load information collection period through experiments. The module is designed in three steps: (1) Introducing the Metrics -Core package: Build tools are commonly used during Java application development. In the case of the Maven build tool, you need to introduce a metrics-core dependency in your project's POM file; (2) MetricRegistry instantiation: MetricRegistry. Class, as the core of Metrics, acts as the metric container and is mainly responsible for the creation, management and maintenance of Metrics. So first you instantiate such a container; (3) Instantiation of ConsolePorter: ConsolePorter.Class is one of the simplest forms of report output in Metrics. It functions like a class name and prints report information from the console. An instance of this class needs to be associated with a concrete metrics container to complete the report output of the container's registered metrics.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202108_7(8).0014

### 3.2 Optimized design module of RoundRibbon

For RoundRibbon relu, there is an obvious drawback, that is, under some special weights, the polling scheduling will generate an uneven instance sequence. Such uneven load may cause some instances to experience instantaneous high load, leading to the risk of system downtime. In order to solve this scheduling defect, the Ribbon default scheduling algorithm is optimized. In order to increase the reliability of the system, a monitor is added to monitor the load of the server cluster.

Suppose there are three service instances, the service instances are A, B and C respectively, and the corresponding weight values are 5,1,1 respectively. The specific scheduling situation of RoundribBonrelu is as follows: Start a counter count, iterate through the list of services in the while loop, and before getting the list get a subscript from the increment And Get Modulo method, which is a self-growing number that is incremented by 1 and then modulated to the total number of services, so that the subscript does not cross the line. If the service is not fetched 10 times in a row, a warning will be generated that the service is not available after multiple attempts.

Suppose there are N instances S = {$S_1$, $S_2$... , $S_n$}, initial configuration weight is denoted as W, W = {$W_1$, $W_2$... $W_n$}, this is a static weight. In addition to having a configuration weight $W_i$ for each instance i, there is also a current valid weight, currentWeight, denoted as $CW_i$, which is a constantly changing weight, and $CW_i$ is initialized to 0. Indicates that currentPos represents the currently selected instance ID, initialized to -1; The configuration weights and values of all instances are total Weight; For instance i, the current load average is $U_i$, and an overload threshold of N is set. On a normal system, it is the number of CPUs that determines if the system is overloaded. Generally, the safety line for single-core machine load is 0.7, while the load for four-core machine used in this paper should be kept below 3(4*0.7 = 2.8).

## 4. Algorithm implementation and testing

### 4.1 Algorithm implementation

First, we need to create the server virtual list, wich called ServerIps class. We assume that the weights of servers A, B and C are divided into 3, 2 and 2. Then, we design the core algorithm, create a RoundRibbonV2 class, and define the get Server () method inside it to implement server allocation. The core code is shown in Figure 1.

```
//  currentWeight+=weight
for(Weight weight : weightMap.values()) {
    weight.setCurrentWeight(weight.getCurrentWeight() + weight.getWeight());
}

//max(currentWeight)
Weight maxCurrentWeight = null;
for(Weight weight : weightMap.values()) {
     if (maxCurrentWeight == null || weight.getCurrentWeight() > maxCurrentWeight.getCurrentWeight()) {
         maxCurrentWeight = weight;
     }
}

//max(currentWeight)-=sum(weight)
maxCurrentWeight.setCurrentWeight(maxCurrentWeight.getCurrentWeight()-totalWeight);
```

Figure 1. Optimization algorithm of the core code

### 4.2 Algorithm testing

After the algorithm runs the test, it calls the server A, B, C, A, B, C. The result of virtual server allocation by the improved algorithm is relatively uniform, and the user requests can be evenly distributed to the server without repeatedly visiting the same server consecutively, which improves the pressure on a single server caused by multiple visits to the same server in the polling strategy. And the waste of server resources caused by allocating more requests to a server while the rest of the

ISSN: 2414-1895 DOI: 10.6919/ICJE.202108_7(8).0014

servers are idle. The virtual server allocation of the original algorithm and the improved algorithm is shown in Figure 2.

It can be found that the access frequency of the polling strategy to server A is as high as 4 times, while the improved algorithm is reduced to 3 times. The access frequency of the polling strategy to server C is only once, and the improved algorithm is increased to 2 times. If C server performance is good, A server performance is poorer, consequences of using round-robin scheduling strategy is A high performance server idle, low performance of the server has been handle requests, if coupled with high concurrency scenario, most likely due to server overload lead to part of the service is not available and the server goes down. The improved algorithm avoids repeated access to one server during server allocation, reduces the load pressure on a single server, makes the server allocation more scientific and reasonable, and achieves better load balancing.

| Number of requests | Round Ribbon | Optimization algorithm |
|---|---|---|
| 1 | A | A |
| 2 | A | B |
| 3 | A | C |
| 4 | B | A |
| 5 | B | B |
| 6 | C | C |
| 7 | A | A |

Figure 2. Comparison between the original algorithm and the optimization algorithm

## 5. Conclusion

This paper proposes an improved load-balancing algorithm to deal with the load imbalance or server overload caused by the default RoundRibbon polling strategy of the Ribbon load-balancing component when dealing with high concurrency. In addition, the load monitor is added, which not only enables users to reasonably allocate requests under high concurrency, but also enables load monitoring. The reliability of the system is improved, and the feasibility of the improved scheme is verified through experiments, and the expected effect is achieved.

## References

[1] Ramana K. Ponnavaikko M. A Multi-Class Load Balancing Algorithm (MCLB) for Heterogeneous Web Cluster [J]. Studies in Informatics & Control, 2018, 27(4).

[2] Juan Liu. Research on application of cluster computer technology [J]. Communication world. 2018. (07): 41-42.

[3] Ze Miao.Nginx high performance Web server details [M]. Electronic Industry Press. 2013.10.

[4] Ramana, K. & Ponnavaikko, M. Web Cluster Load Balancing Techniques: A Survey [J]. International Joumal of Applied Engineering Research, 2016, 10(19). 39983-39998.