Dynamic Balance Strategy of High Concurrent Web Cluster Based on Docker Container

Weizheng Ren, Wenkai Chen and Yansong Cui

School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, China.
Email: 1101959390@qq.com

Abstract. With the rapid development of the Internet in today's world, there are more and more applications based on network in our daily life, and the changes put forward higher requirements to the web systems of concurrency, scalability and availability. Based on the research of the Docker container virtualization technology and the container management platform Kubernetes, by improving the load balance strategies of Round-Robin and Weighted Round-Robin, this paper proposes a high-concurrency and easy-extending high concurrency cluster dynamic balance strategy to support the web system, which can guarantee the web system of high concurrency and high availability performance, implement reasonable and flexible configuration of system resources and realize the functions of automatic management and performance monitor of the system.

1. Introduction

With the rapid increase of user and data, the web system generally supports high-concurrency performance of the system through server clusters and distributed methods and the external requests need to be balanced by the load balancer's balance strategy. With the development of container virtualization technology, operate system level virtualization technology has been rapidly developed and applied. By improving the load balance strategies of Round-Robin and Weighted Round-Robin and combining the characteristics of container virtualization service, this paper proposes a dynamic load balance design scheme to support high concurrent web cluster system based on Docker and Kubernetes.

2. Docker and Kubernetes

As shown in figure 1 and figure 2, Docker is an open source application container engine based on Go which can package and publish applications[1]. And Kubernetes is Google's open source container cluster management system which provides container application deployment, maintenance, and extension mechanisms[2].

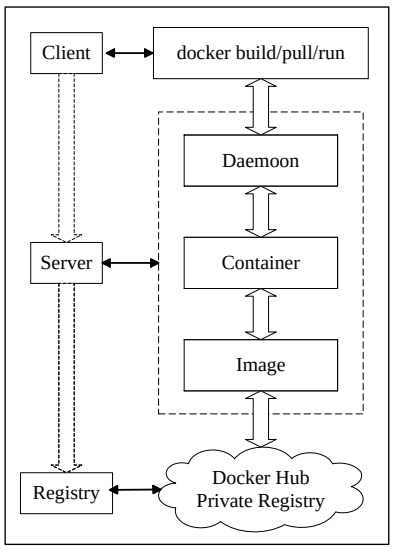

Figure 1. The Architecture of Docker

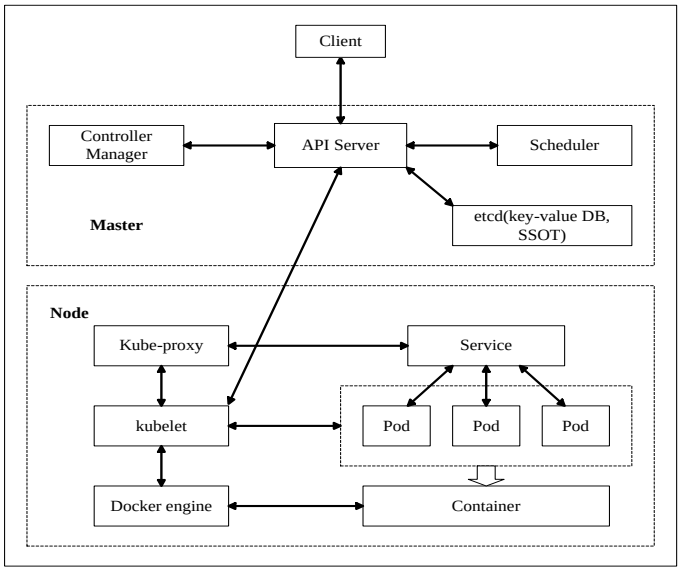

Figure 2. The Architecture of Kubernetes

3. The Challenge of Traditional Balance Strategy based on Nginx

As shown in figure 3, Nginx is generally selected as the load balancer to implement request forwarding and processing. The traditional balance strategy based on Nginx such as Round-Robin, Weighted Round-Robin do not consider the dynamic change of server performance during the system running process, and the number of backend server groups can't be adjusted according to the request amount because of no dynamic strategy based on different workloads and no state monitor for the backend server[3-5].

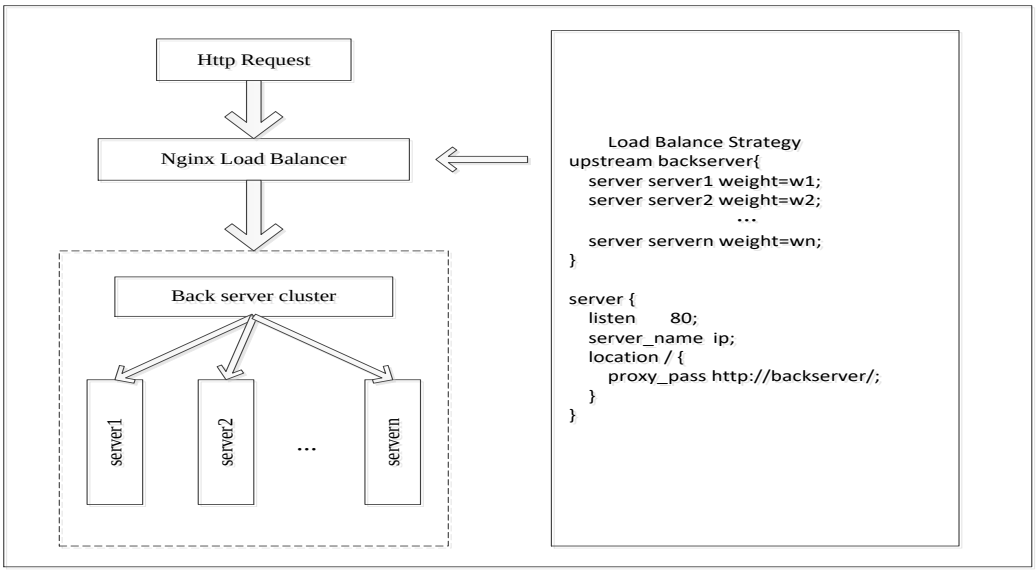

Figure 3. The Architecture of Nginx load balance

4. The Design and Implement of Dynamic Balance Strategy

4.1. System Architecture

As shown in figure 4, the system mainly consists of a Master node and multiple Node nodes based on the Kubernetes container management platform. The dynamic balance control module dynamically adjusts the cluster number and the weight of the cluster server group according to the performance

data according to the dynamic balance strategy, thereby achieving reasonable forward of high concurrent requests and improving the overall throughput of the system[6-8].

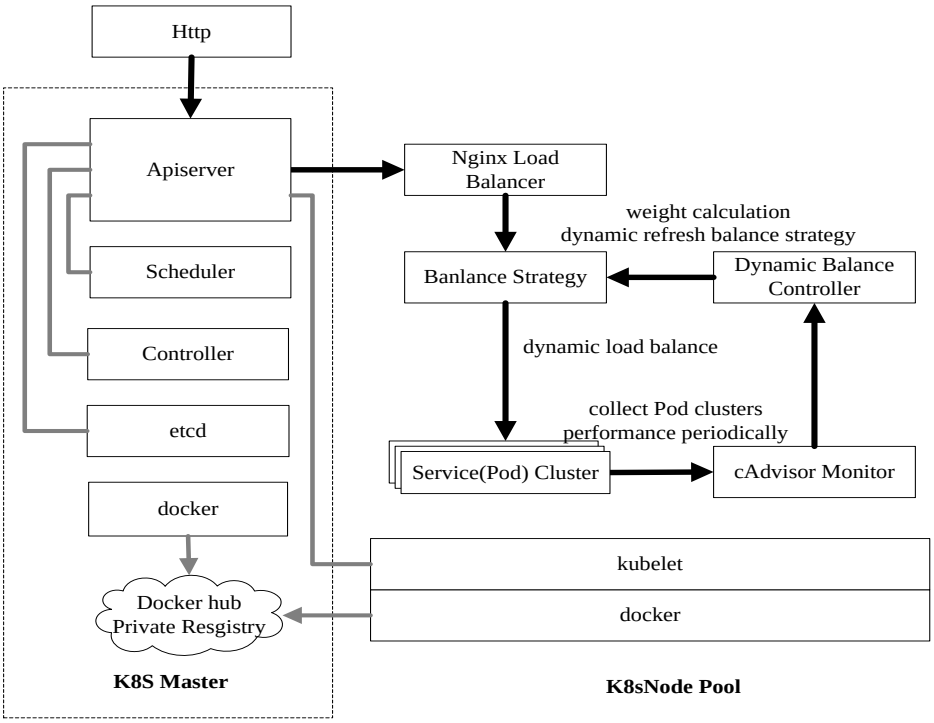

Figure 4. The Architecture of Dynamic load balance system

4.2. Design and Implement

Firstly, the strategy defines the performance quotas of the quantized Pod service and the weight of relative performance quotas. And then calculates real-time performance weight ratio of the cluster by the weight summation algorith. Finally, adjusts the number of clusters and the cluster service weight dynamically according to the dynamic balance strategy [9-10].

4.2.1. Definition and Calculation of Pod Performance Quotas. The matrix is defined:

$$\mathrm{K} = [\mathrm{Kcpu}, \mathrm{Kmem}, \mathrm{Knet}, \mathrm{Kio}, \mathrm{Kcon}]$$

The K is defined to describe the performance quota of Pod, Kcpu represents available CPU cores, Kmem represents available memory, Knet represents network transmission rate, Kio represents the percentage of I/O idle time in one second and Kcon represents available TCP connections. And all the quotas can be obtained and calculated by cAdvisor API or linux command.

4.2.2. Relative Weight Definition of Pod Performance Quotas. The matrix is defined:

$$\mathrm{P} = [\mathrm{Pcpu}, \mathrm{Pmem}, \mathrm{Pnet}, \mathrm{Pio}, \mathrm{Pcon}]$$

The P represents the proportion of CPU, memory, network, IO and connection number in the overall performance of the Pod service, which can be adapted to different workloads by changing its weight ratio:

(1)CPU intensive: A large amount of computational CPU resources are required, such as high-definition decoding of video resources. At this time, the proportion of CPU and memory should be increased appropriately, for example $P = [2, 2, 1, 1, 1]$.

(2)IO intensive: A large number of IO resources are required such as network data transfer and database operations. At this time, the proportion of network and IO should be increased appropriately, for example $P = [1, 1, 2, 2, 1]$.

(3)Response time intensive: Most online websites focus on the user experience and require the system to respond quickly. At this point, every factor should be considered to ensure that the most requests can be forwarded to the server with better average system performance, for example: $P = [1, 1, 1, 1, 1]$.

4.2.3. Normalization Process and Calculation of Overall Weight Ratio. The data needs to be normalized to eliminate different dimensions impact on the overall weight ratio. Assume that the number of service Pod clusters is N which are sequentially numbered i=1, 2...n, and then use the monitor module to periodically acquire the Pod cluster server Podi in a certain performance quota (CPU, memory, IO, network rate, and available connections) Xi (i = 1, 2, 3… n). After normalization, the normalization value Ki of the number i Pod cluster node under this quota can be calculated:

$$K_i = \frac{X_i}{\sum X_i} (i = 1,2 \dots n)$$

The Wi which represents the overall weight ratio of the number i Pod cluster node Podi is defined:

$$W_i = K_i * P^T = [Kcpu_i, Kmem_i, Kio_i, Knet_i, Kcon_i] * [Pcpu, Pmem, Pio, Pnet, Pcon]^T$$

And W which represents the overall weight ratio of the Pod cluster can be calculated by performing the above calculation for each Pod:

$$W = K * P^T = [K_1, K_2, \dots K_n] * P^T$$

4.2.4. Flow Chart of Dynamic Balance Strategy. The Pod cluster node service is deployed by the container. The number of clusters can be dynamically adjusted by the load of the system according to the CPU and memory utilization in the cluster.

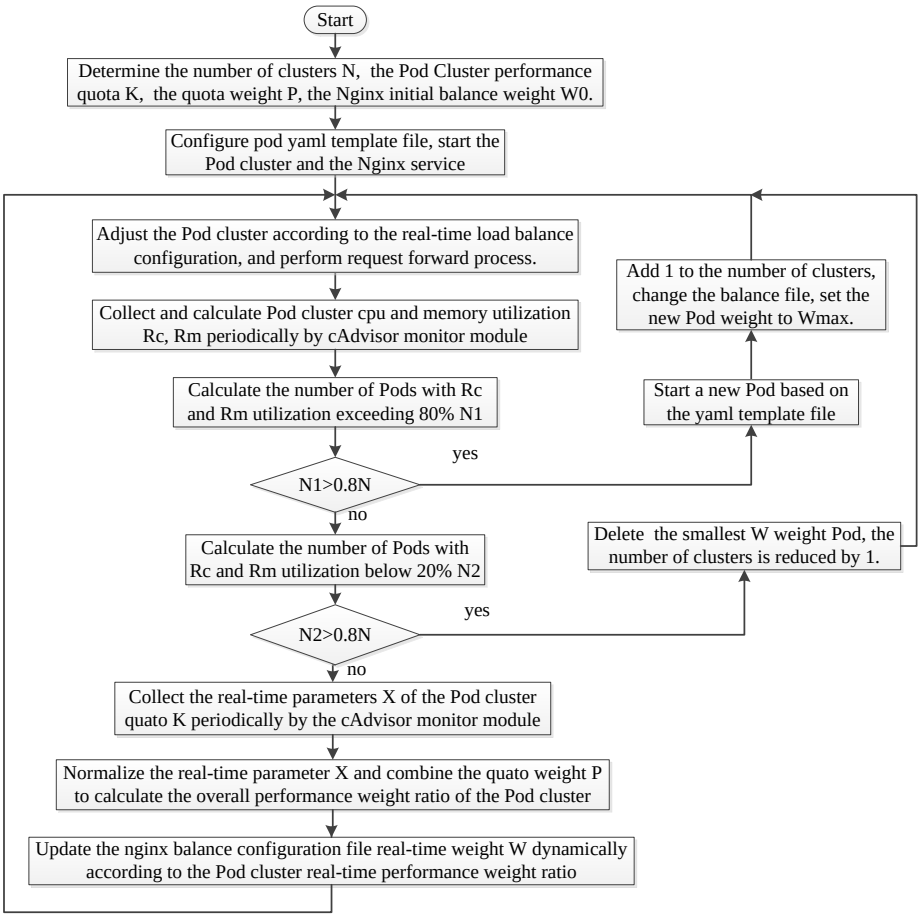

Figure 5. Flow Chart of Dynamic Balance Strategy

As shown in figure 5, the dynamic balance strategy starts with the Pod cluster, determines the quantization performance quotas and corresponding weights, sets the initial balance weights W0, and collects the quotas of the Pod cluster by the cycle time T. According to the periodic performance quota real-time data X, the data under different performance indicators is normalized to K, and the real-time overall performance of the Pod service cluster W is calculated according to the K and the corresponding weight P. And then the balance configuration file is dynamically refreshed according to the real-time performance weight ratio of the cluster W, thereby implementing dynamic processing and forwarding of the request.

5. Experiment and Result Analysis

5.1. Setup of Experimental Environment

The physical conditions of the experimental physical machine are as shown in table 1.

Table 1. Hardware Environment

Node	CPU		memory	bandwidth
Master	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	2core	8GB	1000Mb/s
Node	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz	2core	4GB	1000Mb/s

Set the performance parameters of the monitor module collect the periodic time T to 10s, let $P = [1,1,1,1,1]$. Use http_load to simulate 1000 access, gradually increase the number of access requests from 1000 to 10000, and experiment the changes in throughput and response time of the system under different strategies.

5.2. Experiment Result and Analysis

The result of the change trend graph of the throughout and average response time is as shown in figure 6 and figure 7.

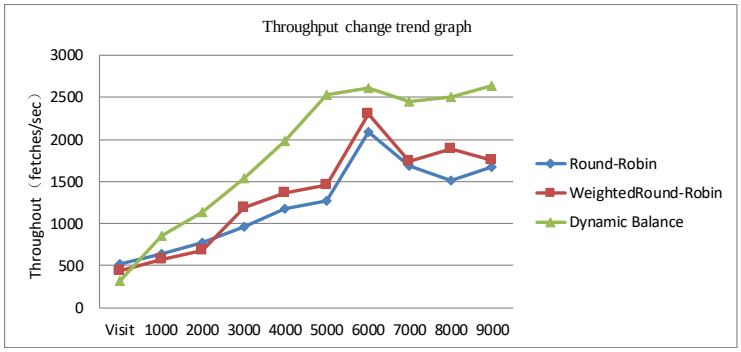

Figure 6. Throughput change trend graph

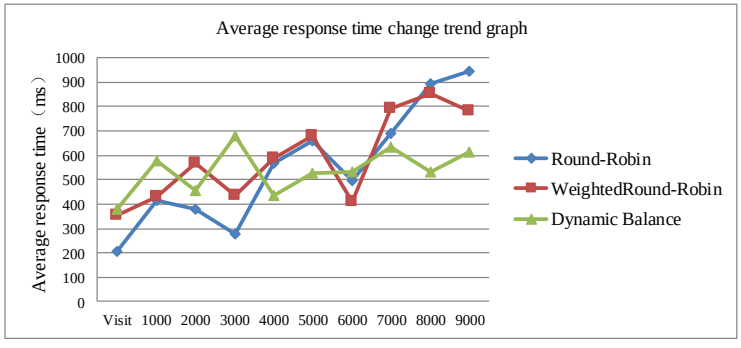

Figure 7. Average response time change trend graph

From the result we can known that when the number of access is small the traditional strategy performance are better because the traditional strategy need less calculation and can process the requests quickly. As the number of concurrent access increasing, the server cluster state changes dramatically and the traditional strategy can not maintain the stability of the throughput and response time and the throughput decreases and the response time increases. However the dynamic strategy can maintain the system throughput and response time within a reasonable range of variation. The dynamic balance strategy implements the performance monitor of the cluster by quantifying the cluster performance parameters, dynamically adjusts the number of clusters according to the resource utilization of the cluster, and adjusts the weight of the back-end services according to the overall real-time performance ratio of the cluster to implement dynamic forwarding and processing of requests which effectively improve the system's throughput and response time and other performance in high concurrency.

6. Conclusion

Compared with the traditional Round-Robin and WeightedRound-Robin strategy, the Dynamic Balance strategy can implement the monitor of the performance and the number of the service cluster and adjust the cluster real-time balance weight according to the running state of the service cluster which can improve the concurrent performance of the system throughput and response time in a certain business scenario.

7. References

[1] Siwei Liu, Qiang Li, Bin Li. Research on Container Isolation Based on Docker Technology[J]. *Software*, 2015, **36(04)**: 110-113
[2] Xugang Yin. Research and implementation of Docker-based PaaS platform technology [D]. Beijing University of Posts and Telecommunications, 2016
[3] Wei Wang. Research and Design of Dynamic Cluster Strategy Based on Web Application [D]. Suzhou University, 2014
[4] Lianlian Wang. Research and Optimization of High Performance Web Application System Architecture [D]. Beijing University of Posts and Telecommunications, 2016
[5] Yusen Zhang, Tao Chen, Kang Li. A comparative study on the principle and strategy of Nginx high concurrent load balancing [J]. *Industrial Control Computer*, 2018, **31 (01)**: 85-86+89
[6] Shengnan Liu, Shilin Wang. Improvement of Web Service Dynamic Load Balancing Strategy in Virtual Environment[J].*Computer Engineering & Science*, 2015, **37(09)**: 1607-1613
[7] Liyao Li, Shaoka Zhao, Dongsen Lin, Cong Xu, Jiahai Yang. Dynamic load balancing mechanism of virtual machine cluster system under cloud environment[J].*Computer Applications*, 2014, **34(11)**:3082-3085+3090
[8] Pengfei Yang. Research and implementation of resource dynamic scheduling based on Kubernetes[D]. Zhejiang University, 2017
[9] Changjun Xu, Tao Lin. Optimization of load balancing method based on Nginx [J].*Journal of Hebei University of Technology*, 2016, **45(06)**:48-52
[10] Mayuri A. Mehta, Devesh C. Jinwala. A Hybrid Dynamic Load Balancing Algorithm for Distributed System [J]. *Journal of Computers*, 2014, **9(8)**