PAPER • OPEN ACCESS

An Integrated Load-balancing Scheduling Algorithm for Nginx-Based Web Application Clusters

To cite this article: Ruoyu Li *et al* 2018 *J. Phys.: Conf. Ser.* **1060** 012078

View the article online for updates and enhancements.

An Integrated Load-balancing Scheduling Algorithm for Nginx-Based Web Application Clusters

Ruoyu Li[1], Yunchun Li[1] and Wei Li[1]

[1]School of Computer Science and Engineering, BeiHang University, Beijing, China

*Corresponding author e-mail: {liruoyu, lych, liw}@buaa.edu.cn

Abstract. In recent years, the LXC (Linux Container)-based Docker technology has attracted great attention due to its lightweight and convenient features. Because of the performance differences among the hosts and the network complexity, it's necessary to improve the resource utilization of WEB application clusters. This paper analyzes the WEB application deployment architecture based on container technology, calculates the host load metrics combined the host's performance indicates and status of the operating containers, including CPU utilization, memory utilization, network utilization, and the proportion of unused memory that has been allocated for the containers, and proposes a Dynamic Weighted Least-Connection Algorithm(DWLC). The experiment shows that, the DWLC algorithm helps the WEB application response 52.6% and 46.4% faster than the ordinary Round-Robin and Least-Connection algorithm.

1. Introduction

As the most popular container technology in recent years, the LXC (Linux Container)-based Docker technology attracted great attention due to its lightweight and convenient features. Docker uses Linux Cgroup for resource allocation and ensures the independence and security of each container resource through Linux Namespace. Docker can run an application on any docker-installed server by packaging its dependent binaries, libraries, configuration files and any other things that the application needed into a lightweight, portable, self-contained container. Compared with virtual machine, Docker can provision performance and user-space isolation with extremely low virtualization overheads [1]. Although Docker uses the host's hardware resources in a shared manner, it can create thousands of containers and improve the overall utilization of system resources.

Since "Build once, run anywhere" is the most appropriate statement about Docker, the development and deployment of applications increasingly tends to depend on Docker containers with its strong scalability and portability. As the vast majority of WEB applications deployed on the cloud platform [2], more and more service providers use the Docker container-based clustering approach for deployment. At the entrance of the cluster, there will always be a load balancer, such as nginx, haproxy, F5, etc. The load balancer minimizes the probability that any particular server will be overwhelmed, optimizes the bandwidth available to each server, facilitates traffic prioritization, provides end-to-end application monitoring, helps protect against malicious activity, and "evenly" distributes the received requests to back-end servers.

However, in the face of container-based WEB clusters, these load balancers can only statically set the load balancing algorithm based on the initial performance of the host, instead of dynamically adjusting based on the real-time monitored host state. Therefore, how to properly monitor the status of the host and the containers on it and form a dynamic adaptive load balancing algorithm, so as to truly achieve load balancing of web requests at the entrance of the cluster, it has great significance in improving system utilization to the current increasing container-based Web application cluster.

The remainder of this paper is organized as follows. We discuss the related work of load balance algorithm in section II. In section III, we design the structure of an integrated load-balancing scheduling algorithm for nginx-based Web application cluster. We introduce the selection of the performance parameters and the algorithm in section IV. Section V shows the experimental results and we conclude our work in the last section.

2. Related Works

Under the comprehensive consideration of the memory usage of the server, the CPU usage of the server and the current network bandwidth, Luo Shuhua et. al. calculated weight of each server, proposed a dynamic weight least-load balancing algorithm [3]. This algorithm realized the dynamic adjustment of server weight, resulting in more stable network operation and better load distribution.

Xu Zongyu proposed an optimized Route Robin load balancing strategy based on WEB server cluster [4]. By calculating and comparing the load values of various servers, this strategy sends incoming WEB requests to the next server node, which is not the most heavily loaded. All this implements a dynamic optimization of the Route Robin algorithm and WEB request allocation, and achieves an even distribution of WEB requests.

Considering features of Docker, various applications' requirements, and available resources in cloud date centers, Guan Xinjie et. al. proposed a communication-efficient resource allocation algorithm and designed a framework for Docker container-based application oriented dynamic resource allocation, to minimize the application deployment cost [5].

Unfortunately, for a Docker container-based WEB application cluster, how to effectively monitor the load status of the host, distribute WEB requests to the back-end service container in an even and balanced manner, thereby balancing the load of the entire system at the entrance of the cluster, there is no better way to deal with it. So we propose a Dynamic Weighted Least-Connection (DWLC) load balancing algorithm based on Nginx.

3. System Structure and Modelling

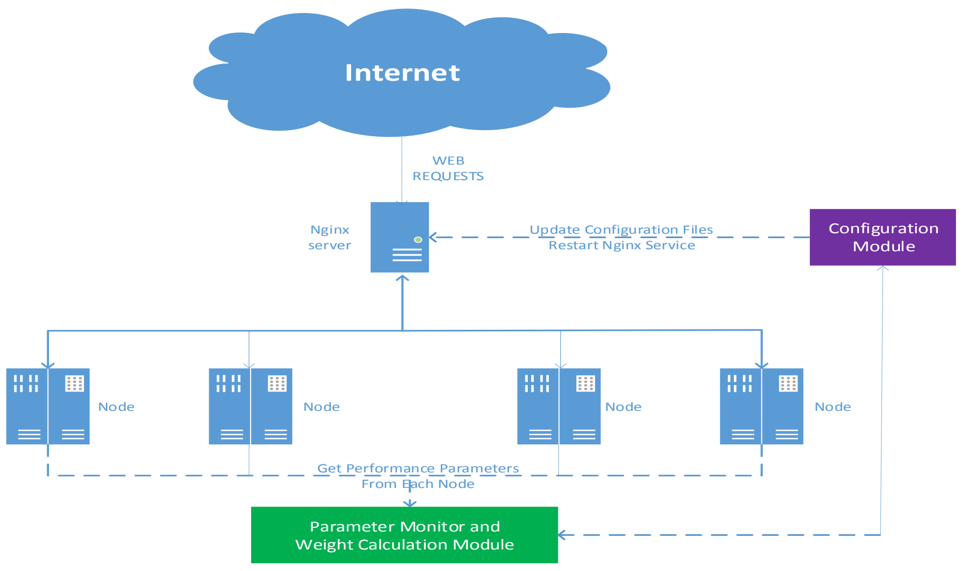
Internet
WEB REQUESTS
Nginx server
Update Configuration Files
Restart Nginx Service
Configuration Module
Node
Node
Node
Node
Get Performance Parameters From Each Node
Parameter Monitor and Weight Calculation Module

Figure 1. System Structure

Nginx is a software-based load balancing server with eight different load balancing algorithms. According to different requirements, Nginx can reverse proxy multiple web applications at the same time by setting different configuration files. However, the original Least-Connection or Weighted Least-Connection algorithm in Nginx is static and cannot fully reflect the node's overall processing capability during system operation. So we propose a Dynamic Weighted Least-Connection (DWLC) Algorithm combined the host's performance metrics with status of the operating containers.

As shown in figure 1, we establish the parameter acquisition and weight calculation module, as well as the nginx configuration module. The system monitors the performance parameters of each node in real time and calculates the weight of the node according to a certain algorithm. Then the nginx configuration module decides whether to update the weight of the node or restart the nginx service.

4. Performance Parameters and Algorithm

4.1. Selection of Performance Parameters

In a hybrid cloud environment, the load status of the whole docker container-based WEB cluster can not only be calculated by the host's performance indicates, but also the status of the operating containers. So we choose four performance parameters-CPU utilization ratio, Memory utilization ratio, Network utilization ratio and the proportion of used memory that has been allocated for the containers to calculate the load of the node.

We assume that the WEB cluster has n nodes. For each node i ($i \in n$), we get the four performance parameters every t seconds.

The CPU utilization ratio ($CPU\_Ratio_i(t)$) is the ratio of the sum of the cpu time in user mode and system kernel mode to the sum of the cpu time in user mode, kernel mode, and idle mode. It can be calculated using the following formula:

$$\mathrm{CPU}_{\mathrm{Ratio}_\mathrm{i}(\mathrm{t})} = \frac{CPU_{usr(t)} + CPU_{kernel(t)}}{CPU_{usr(t)} + CPU_{kernel(t)} + CPU_{free(t)}} \tag{1}$$

The Memory utilization ratio ($MEM\_Ratio_i(t)$) is the non-idle ratio of current host memory. It can be calculated using the following formula:

$$\mathrm{MEM}_{\mathrm{Ratio}_\mathrm{i}(\mathrm{t})} = 1 - \frac{MEM_{free(t)}}{MEM_{total}} \tag{2}$$

The Network utilization ratio ($NET\_Ratio_i(t)$) is the ratio of the sum of the bytes received and sent by the node within the time interval t to the network bandwidth(NET_band). When a node join the cluster, first of all, we will use iperf (iperf is a network performance testing tool that can measure the maximum bandwidth between the client and the server) to calculate the network bandwidth (NET_band) between the node and the load balancing server. And then, we calculate the sum of the bytes received and sent by the node within the time interval t. So the network utilization ratio can be calculated using the following formula:

$$\mathrm{NET}_{\mathrm{Ratio}_\mathrm{i}(\mathrm{t})} = \frac{NET_{sent(t)} + NET_{rev(t)}}{NET_{band}} \tag{3}$$

The proportion of used memory that has been allocated for the containers (Di(t)). We can use the Docker native command "docker stats" to read the real-time memory information of each container on the node, and then $D_i(t)$ is the ratio of the sum of MEM USAGEs to the sum of MEM LIMITs:

$$\mathrm{D_i(t)} = \frac{\Sigma mem_{usage}}{\Sigma mem_{limit}} \tag{4}$$

4.2. Algorithm

When a node joins the cluster, we will arrange the node an initial weight 100, and then we execute the weight calculation every t seconds.

First of all, we calculate the real-time load $L_i(t)$ of node i every t seconds using the following formula:

$$\mathrm{L_i(t)} = a_1 * \mathrm{CPU}_{\mathrm{Ratio_i(t)}} + a_2 * \mathrm{MEM}_{\mathrm{Ratio_i(t)}} + a_3 * \mathrm{NET}_{\mathrm{Ratio_i(t)}} + a_4 * \mathrm{D_i(t)} \tag{5}$$

Where $L_i(0)=0$, and $a_1+a_2+a_3+a_4=1$, $0<a_i<1$. Normally, a_i can be set different value due to different kinds of application. Here, We set $a_1 = 0.35, a_2 = 0.35, a_3 = 0.2, a_4 = 0.1$.

Then the current weight $W_i(t)$ of node i is:

$$\mathrm{W_i(t)} = 100 - \mathrm{Li(t)} \tag{6}$$

In order to avoid the node's weight fluctuation caused by small weight changes, we establish a load variation threshold A. We get the weight of the node i W_i from the nginx configuration file, and then calculate the weight variation λ within the time interval t:

$$\lambda = \mathrm{W_i(t)} - \mathrm{W_i} \tag{7}$$

If the absolute value of λ is smaller than A, the weight of the node remains, else the weight of the node changes as formula below:

$$W'_i = \begin{cases} W_i(t) & |\lambda| \geq A \\ W_i & |\lambda| < A \end{cases} \tag{8}$$

Here, W_i is the weight of node i in the nginx configuration file, λ is the weight variation, $W_i(t)$ is the current weight of node i and W'_i is the new weight of the node i.

After all this calculation finished, if W'_i is not equal to W_i, we update the nginx configuration file and reload the nginx service.

5. Experiments

Apache Bench (ab) is a single-threaded command line tool for measuring the performance of HTTP web servers. It can simulate multiple users creating a large number of concurrent access threads to stress a URL address [6].

Table 1. the software and hardware of each server

Role	NUM of CPU	MEM	Bandwidth(Iperf)	Operation System	Main Software
Worker	8	4GB	942 Mbits/sec	Ubuntu14.04 LTS	DOCKER1.13
Worker	8	4GB	941 Mbits/sec	Ubuntu14.04 LTS	DOCKER1.13
Worker	4	8GB	23.0 Gbits/sec	Ubuntu14.04 LTS	DOCKER1.13
Worker	4	8GB	18.5 Gbits/sec	Ubuntu14.04 LTS	DOCKER1.13
Worker	4	8GB	21.7 Gbits/sec	Ubuntu14.04 LTS	DOCKER1.13
Nginx Server	4	4GB		Ubuntu14.04 LTS	Nginx1.4.6

We use 1 nginx server and 5 worker servers to conduct the experiment. The software and hardware configuration of the servers are shown in table 1. We deploy the SNS Web application UCenter Home in docker, using the nginx container and the php container. Then we use iperf to measure the bandwidth between the nginx server and each worker server. We use the ab command to simulate 100 users for 10,000 concurrency every time to see the response speed of the web application cluster. In all figures, the abscissa is the proportion of all requests completed, and the ordinate is the response speed (in milliseconds).

First of all, to confirm the threshold of the weight changes, we test the value of the threshold from 1 to 6. As shown in Figure 2, the response time is almost no change when the value of the threshold changes. The experiment result shows that the algorithm is insensitive to the threshold. However, Figure 3 shows that the response speed of the web cluster is relatively faster when the threshold is 5 than others when 95% requests are finished. So we choose 5 as the threshold of the weight change.

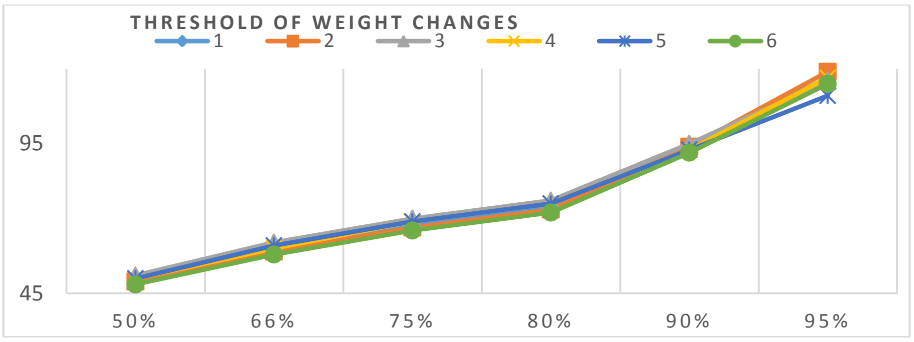

Figure 2. threshold experiment

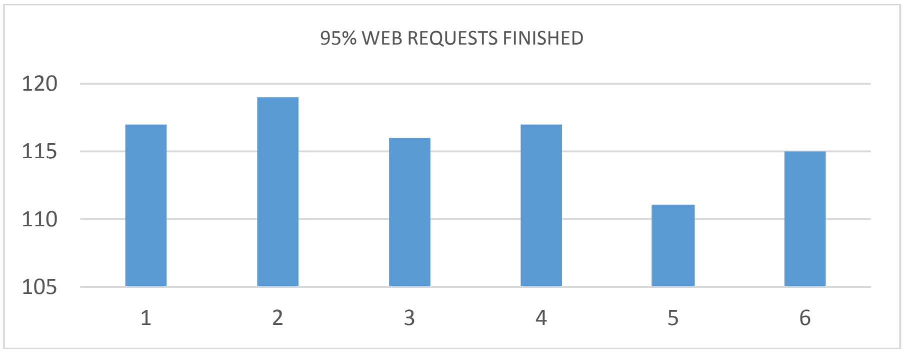

Figure 3. Response speed when 95% finish

Then in order to avoid taking up too much system resources or feeding back the load of the node not in time because of bad time interval choice, we test the time interval t from 1s to 5s. As shown in Figure 4, the response speed of the cluster is the fastest when the time interval is 3s. So we set the time interval 3 seconds.

At last, we set the threshold of weight changes 5 and the time interval 3s, and compare the response speed of Round Robin algorithm, Least-Connection algorithm and our DWLC algorithm. As shown in Figure 5, the DWLC algorithm has obvious advantages and can improve the Web cluster response speed 52.6% and 46.4% faster than the ordinary Round Robin and Least-Connection algorithm when 95% requests are finished.

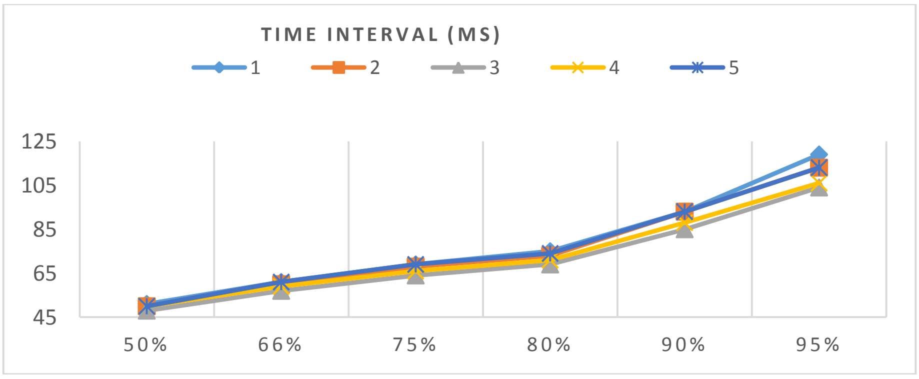

Figure 4. Time interval experiment

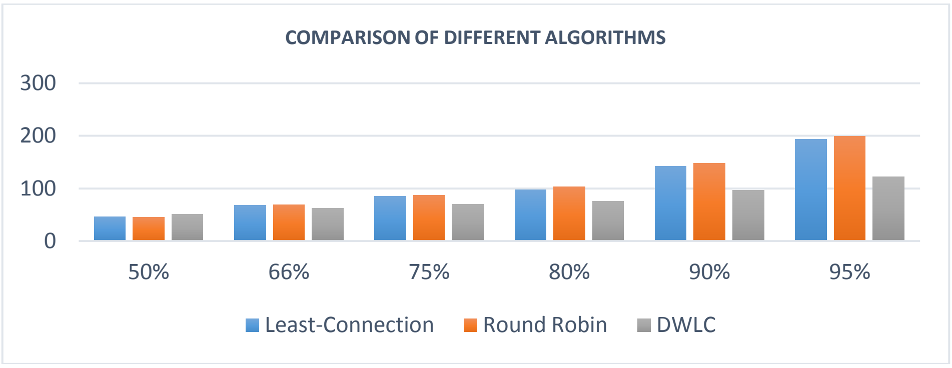

Figure 5. Comparison of different algorithms

6. Conclusion

In this paper, we analyse the WEB application deployment architecture based on docker container, calculate the host load metrics combined the host's performance indicates and status of the operating containers, and propose a Dynamic Weighted Least-Connection algorithm. The experiments show this algorithm can efficiently improve the response speed of the WEB cluster compared to RR algorithm and Least-Connection algorithm.

Acknowledgment

This work is partly supported by the National High-tech R&D Program ("863" Program) of China (No.2015AA01A202) and research project of Beijing Municipal Education Commission.

References

[1] J. García-Galán, P Trinidad , Rana OF, A. Ruiz-Cortés, "Automated configuration support for infrastructure migration to the cloud," Future Generation Computer Systems pp 200-212, Feb-2016 VILAPLANA,JORDI.SLA-aware load balancing in a web-based cloud system over OpenStack[J].Service-Oriented Computing-ICSOC 2013Workshops,2014(9):281-293.
[2] Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015, March). An updated performance comparison of virtual machines and linux containers. In Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On (pp. 171-172). IEEE. VILAPLANA,JORDI.SLA-aware load balancing in a web-based cloud system over OpenStack[J].Service-Oriented Computing-ICSOC 2013Workshops,2014(9):281-293.
[3] Ying Wu, Shuhua Luo,Qing Li.An Adaptive Weighted Least-Load Balancing Algorithm Based on Server Cluster[C].Intelligent Human-Machine Systems and Cybernetics(IHMSC), 2013,pp,224-227.
[4] Xu Zongyu, Wang Xingxuan.A Modified Round-robin Load-balancing Algorithm for Cluster-based Web Servers[C].Control Conference(CCC),2014,pp,3580-3584.
[5] Guan Xinjie, Wan Xili.Application Oriented Dynamic Resource Allocation for Data Centers Using Docker Containers. 23 December 2016, IEEE Communications Letters, pp. 504-507.
[6] Ab-Apache HTTP server benchmarking tool.http://httpd.apache.org/docs/2.4/programs/ab.html