Research on Nginx Dynamic Load Balancing Algorithm

E Qin[1], Yuanli Wang[1], LiPing Yuan*[2], Yi Zhong[1]

1. School of Information Engineering, Wuhan University of Technology, Wuhan 430070, China
2. School of Information Engineering, Wuhan HuaXia University of Technology, Wuhan 430223, China
*corresponding author's email: Whutylp@whut.edu.cn, 94524948@qq.com

***Abstract*—With the development of network technology, the Internet has brought great convenience to people's daily lives. However, due to the rapid increase in the number of people using the Internet and the explosion of data volume, the server needs to handle high concurrent access and large amounts of data, which also brings great challenges to the performance of the network server. Server clusters and load balancing technologies have emerged at the historic moment, and have optimized the server performance in terms of hardware and software. Nginx, as an open source, lightweight and high-performance server, has been widely used in recent years for its high performance and ease of expansion. By learning the source code of Nginx, each module, the processing flow of HTTP, and the load balancing algorithms in Nginx, it was found that the algorithms of Nginx cannot feedback the real-time performance of the server, and dynamically adjust the server weight to suit the server performance. Therefore, a dynamic load balancing algorithm based on Nginx is proposed, which collects server load information in real time and dynamically adjusts weights.**

Keywords—Nginx; load balancing; dynamic weight; high concurrency

I. INTRODUCTION

As one of the important technologies in the cluster system, load balancing is a research hotspot in recent years, and has made great progress. From the simple use of static load balancing strategy, it is now possible to dynamically adjust the strategy according to the real-time load situation of the server cluster. Load balancing is mainly optimized from two aspects of hardware and software. Generally, both are optimized at the same time to achieve better cost performance. Under the same hardware facilities, the efficiency of load balancing is mainly reflected in the quality of the load balancing algorithm.

In 2012, Sharda. Patil and Arpita Gopal from Malaysia proposed a load balancing algorithm for Linux cluster systems[1]. This algorithm uses a centralized method in load distribution and a distributed method in other cases. Then collect the load information periodically, including the current status, current load, process information and past process records of the server nodes. Each processor stores the current information of all processors, and each node sends its own load information at intervals. When any idle node receives a command packet, it collects information on the LAN (Local Area Network) and distributes the command packet to each load node for distributed processing. If there is no idle node, it is collected and distributed by a low-load node. Through this strategy, the system reliability is improved to a certain extent. Kunihiko Tounura and Naokazu Nemoto from Japan proposed a scalable server load balancing strategy in 2013[2], which better solved the problem of single point failure in the load balancer. In 2015, Yunlu Wang, Dushyantha A. Basnayaka, Harald Haas proposed a dynamic load balancing algorithm to find the optimum access point (AP) assignment to meet some average per-user data rate constraints while achieving the minimum outage performance[3]. Grosu, Chronopoulos and Ming Ying Leung combined game theory to propose a cooperative load balancing game strategy[4]. Fahimeh Ramezani, Jie Lu, Farookh Hussain developed a novel method to resolve VM overload and achieve system load balancing by assigning the arrival task to another similar VM in the cloud environment[5].

Wei Zhao, Lin Zhao proposed a load balancing/relay selection (LBRS) strategy for RA networks[6]. The proposed LBRS strategy makes the relay selection of RA networks under the assumption of imperfect channel state information a multi-level decision (MD) problem. Using a stochastic dynamic program, an optimization algorithm is studied to solve the proposed MD problem. Tao Hu, Peng Yi proposed a distributed decision mechanism based on switch migration in a multi-subdomain SDN network (DDM)[7]. By collecting network information, a distributed migration decision domain is constructed based on the controller load. Then select the migration switch according to the selection probability, and determine the target controller by integrating the three network costs, including data acquisition, switch migration, and controller status synchronization. Finally, set the migration countdown to achieve an orderly switch migration, which can achieve better performance controller load balancing.

At the same time, some enterprises have also proposed optimized cluster load schemes. Local Director load balancer of Cisco uses the least connections faster response algorithm to distribute requests from the network to the fastest back-end server node. IBM's Network Dispatcher load balancer can detect the number of backend server connections in real time. The order of task allocation is selected according to the number of connections. the load balancer of Intel uses a faster response algorithm.

2157-1481/20/$31.00 ©2020 IEEE
DOI 10.1109/ICMTMA50254.2020.00138

II. RELATED TECHNOLOGIES

Load balancing algorithms include static load balancing, dynamic load balancing, and adaptive load balancing. Static load balancing algorithms have been thoroughly studied at present, and their shortcomings are obvious, that is, they cannot comprehensively collect server node status information and perform load adjustment based on their status information. Later development of dynamic load balancing made up for this shortcoming.

Static load balancing algorithms are currently widely used in round robin, ratio, and priority algorithms, among which the most classic are round robin algorithms and weighted round robin algorithms. Weighted round robin algorithm analyzes server performance to allocate the weight of each server, and loops through each server sequentially. The server with the higher weight allocates more requests. Round robin algorithm assumes that all servers have the same performance, so all servers allocate the same number of requests when scheduling. The algorithm is simple and easy to implement, but when the server is down or overloaded, the round robin algorithm cannot be adjusted accordingly.

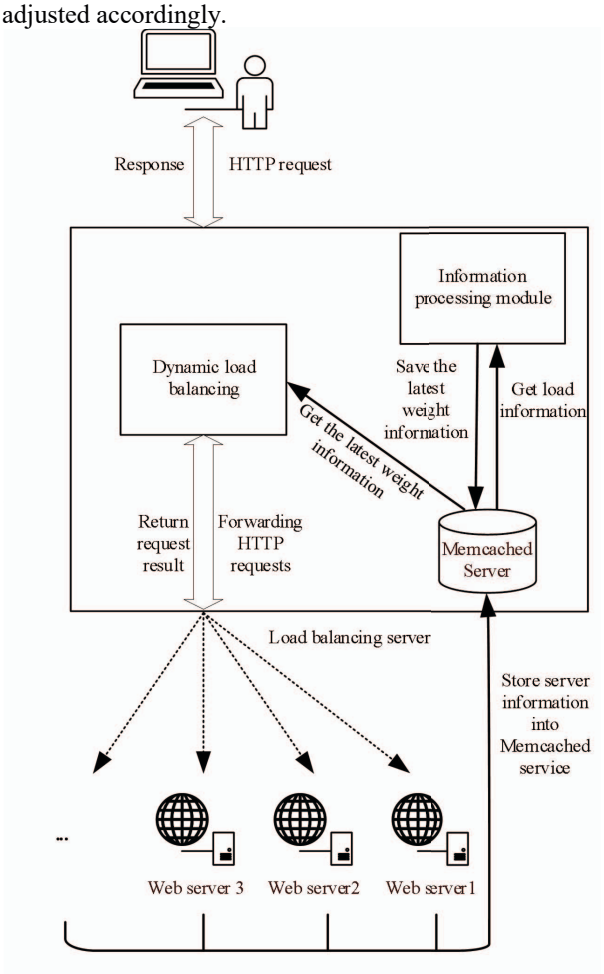

Figure 1. Overall structure diagram

The dynamic load balancing algorithm has the least connections, the fastest response, prediction, dynamic ratio-APM and dynamic server act. The dynamic load balancing algorithm needs to obtain server load information in real time or periodically[8], such as CPU utilization, network bandwidth, and number of connections. Then calculate the new weights of each server and distribute the requested tasks according to the new weights. In most cases, the dynamic load balancing algorithm performs better than the static load balancing algorithm. It has good interaction with the servers, and can adjust the server weight in real time, so that the servers can process access requests in a better state[9]. However, the dynamic load balancing algorithm will generate additional overhead and increase server load since server information is collected and processed in real time[9]. For this, a dynamic weighted round robin algorithm based on Nginx load balancing is proposed. On the basis of minimizing the burden of information collection on the server, improve the weighted round robin algorithm that comes with Nginx to achieve a better load effect.

III. ALGORITHM IMPROVEMENTS

Dynamic weighted round robin algorithm based on Nginx is mainly divided into three modules: server load information collection module, server information processing module, and Nginx dynamic weighted round robin module. The server load information collection module is responsible for collecting the load information of each web server and storing it in the Memcache service, including CPU utilization, memory utilization, network utilization, and disk IO utilization. The server information processing module obtains the load information of each server from the Memcache service, and determines whether the server is load balanced. If it is not balanced, it must recalculate the weight and store it in the Memcache service. The Nginx dynamic weighted round robin module obtains the latest server weight information from the Memcache service and updates the server weights in the weighted round robin to achieve dynamic weighting. The overall structure of the system is as follows.

A. Data normalization

In this experiment, a total of 3 web servers and 4 performance indicators were collected: CPU, disk IO, network bandwidth, and memory. Because the measurement units of the indicators are not consistent, the data must be normalized first. It uses Min-max normalization to map the result value to [0,1].

Let n indicators be $X_1,X_2,...,X_n$, k web servers, and the standardized values be $Y_1,Y_2...,Y_n$, where $X_i = \{x_{i1},...,x_{ik}\}$ is the value of index i on k servers. The normalized calculation formula is as follows:

$$Y_{ij} = \frac{X_{ij} - min(X_i)}{max(X_i) - min(X_i)} \quad (1)$$

Y_{ij} is the index i of the server j.

B. Determine the weight factor

The server *j*'s proportion of the index *i* in this index is p_{ij}.

$$p_{ij} = \frac{Y_{ij}}{\sum_{j=1}^{k} Y_{ij}} \quad (2)$$

Let the information entropy of each index be E_1,E_2,...,E_n.

$$E_n = -\ln(k)^{-1} \sum_{j=1}^{k} p_{ij} ln(p_{ij}) \quad (3)$$

The weight of each index is W_{ij}.

$$W_{ij} = \frac{1-E_n}{n-\sum E_n} \quad (4)$$

In this paper, the index data obtained during the experiment is used as the experimental data, normalized by the formula (1), and then the weight coefficients W_{cpu}, W_{net}, W_{io}, W_{mem} of the four indexes are calculated using the entropy weight method.

C. Update weights

Let C_j、N_j、D_j、M_j be the server *j*'s remaining CPU performance, network bandwidth remaining performance, disk remaining performance, and memory remaining performance. T_C、T_N、T_D、T_M are the remaining CPU performance, network bandwidth performance, disk performance, and memory performance of all server node. For any server node, its initial weight is W_{S_j}.

$$W_{S_j} = 1000 * \left(W_{cpu}\frac{C_j}{T_C} + W_{net}\frac{N_j}{T_N} + W_{io}\frac{D_j}{T_D} + W_{mem}\frac{D_j}{T_M}\right) \quad (5)$$

When the server cluster processes a large number of HTTP requests, some nodes will be under high load and some nodes will be under low load because of the amount of allocated requests and the amount of data that needs to be processed in the request, which makes the performance of the server cluster cannot be fully utilized. It is necessary to monitor the load of the server nodes in real time and adjust the node weights. However, computing node weights frequently consumes too much computer resources, so the following formula is used to calculate the load of the current server node.

$$U_j = W_{cpu} * C_j + W_{net} * N_j + W_{io} * D_j + W_{mem} * M_j \quad (6)$$

U_j is the overall remaining performance of the current node. We set two thresholds H and L. When $U_j \leq H$, it means that the current node *j* is in a high load state. When server *m* is in a high load state and $U_j \geq L$ of server *j*. It is determined that the current and node *j* are in a low load state.

$$F_j = \frac{U_j}{\sum_{m=1}^{k} U_m} \quad (7)$$

F_j is the remaining performance ratio of the current node among all nodes of each server. The server node weights can be adjusted as follows.

$$W_j = F_j * W_{S_j} \quad (8)$$

The situation of load imbalanced nodes is improved through weight adjustment. However, the weighted imbalance is generally caused by high concurrent access in a short time. Therefore, when there is no high load node, the server node weight is gradually adjusted to the initial weight.

IV. RESULTS ANALYSIS

This experiment mainly tests the system through siege. The improved load balancing algorithm in this paper is compared with weighted round robin algorithm of Nginx and the third-party fair algorithm in terms of actual highest concurrent number, average response time, and throughput in different concurrent situations.

Table 1 Nginx load balancing server configuration

Name	Description
CPU	Type: i7-6700; quad core
Memory	DDR3 4.0G
Disk	20G
Nginx	1.14.0
OS	Centos 7 64bit

Table 2 Web server configuration

Name	Description		
	Server 1	Server 2	Server 3
CPU	dual core	dual core	quad core
	Type: i7-6700		
Memory	DDR3; 1G	DDR3; 2G	DDR3; 4G
Disk	20G		
Nginx	1.14.0		
Memcache	1.5.7		
OS	Centos 7 64bit		

Table 3 Client server configuration

Name	Description
CPU	Type: i7-6700; single core
Memory	DDR3 4.0G
Disk	20G
siege	4.0.4
OS	Centos 7 64bit

Tables 1 to 3 show the experimental test environments. The test cluster consists of a client server, a server as a load balancing server, and three web servers. The client server simulates the client request, and the web server processes the user request. The details of the load balancing server are shown in Table I. The details of the client server are shown in Table II. Table III are the details of the web servers. In the experiment, the number of concurrency starts from 200 and ends in 2000 with the growth rate of 200. Each time runs 10 times, and the average of the result is taken as the final result. Among them, the weighted round robin algorithm of Nginx is abbreviated as wrr, the third-party strategy fair is abbreviated as fair, and the improved dynamic weighted round robin algorithm in this article is abbreviated as dynlb.

A. Actual concurrent connections

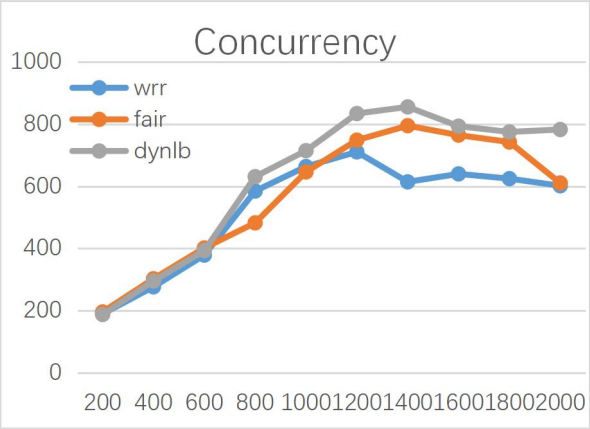

Figure 2. Comparison of actual concurrent connections

From the line chart of the number of concurrent connections in Figure 2, it can be seen that when the number of concurrent requests of the three algorithms does not reach 800, the actual number of concurrent connections is not much different, indicating that the three algorithms have the same ability to withstand the pressure in this range. When the number of concurrent connections reached 1200, the algorithm performance basically reached the upper limit, and the actual number of concurrent connections rose to the highest level. When the number of requests continued to rise, the actual number of concurrent connections didn't increase, or even slightly decreased, but it can be seen from the line chart the actual number of concurrent connections in the later period of the dynlb algorithm is slightly higher than the other two algorithms. It can be seen that the improved dynlb algorithm performs better than the fair and Nginx wrr algorithms on concurrent connections.

B. Response time

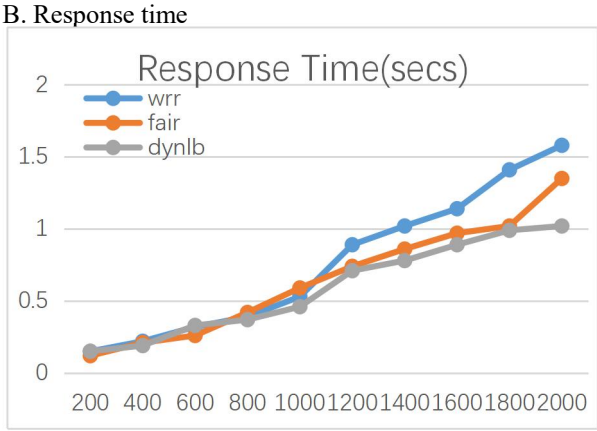

Figure 3. Response time

Figure 3 shows the comparison of response time of three load balancing algorithms under different concurrency conditions. The response time of the three algorithms gradually increases with the increase in the amount of concurrency. When the amount of concurrency is less than 800, the difference in the response time of the three algorithms is not obvious. However, when the concurrency reaches more than 1000, the response time of Nginx's weighted round robin algorithm increases faster than the other two algorithms. The response time of the fair algorithm and the dynlb algorithm is not much different, but the dynlb algorithm is generally lower than the fair algorithm. It can be seen from the line chart trend that the dynlb algorithm used in this paper is better than the fair and Nginx wrr algorithm.

C. Throughput

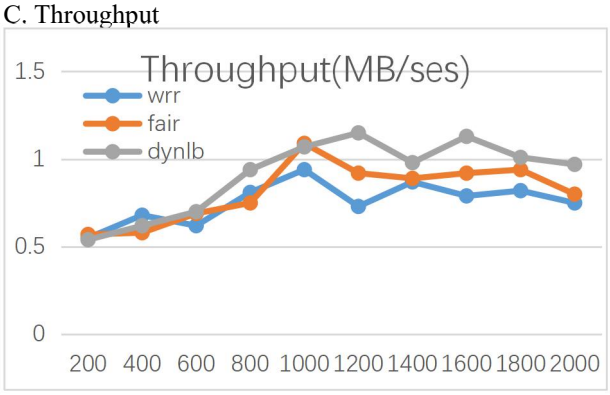

Figure 4. Throughput

Through the comparison of the throughput line chart in Figure 4, it can be seen that before the concurrent volume reaches 1000, the throughput trends of the three algorithms have gradually increased. When the amount of concurrency reaches more than 1000, the throughput shows a decreasing trend. Due to hardware or software performance limitations of the server system, the throughput of the three algorithms has decreased. In the line chart, when the concurrency is higher than 600, the throughput of dynlb is significantly higher than the other two algorithms. When the concurrency is gradually increasing, dynlb The performance of the algorithm in throughput is also better than the other two algorithms.

V. CONCLUSIONS

This paper proposes an improved dynamic weighted round robin algorithm based on Nginx weighted round robin. It determines the weights based on the current load of the server nodes. Using siege for testing, this algorithm is compared with third-party strategies fair and Nginx weighting round robin. The comparison in the above line chart shows that the dynamic weighted round robin algorithm proposed in this paper is superior to the other two algorithms in response time, actual number of concurrent connections and throughput.

ACKNOWLEDGEMENTS

This work is supported by NSFC (Grant No. 51978550): Life Extension Control of Fatigue Vibration of Orthotropic Steel Bridge Deck Based on Magnetorheological Elastomer and Macrofiber Composite, time: 2020.1-20202.

REFERENCES

[1] Mrs.Sharada Patil and Dr Arpita Gopal. "Need of New Load Balancing Algorithms for Linux Clustered System, " in *International Conference on Computational Techniques and Artificial Intelligence (ICCATAI)*,2012, pp.104-108.

[2] K. Toumura and N. Nemoto, "A Scalable Server Load Balancing Method Using IP Address Stealing," in *Computer Software & Applications Conference Workshops IEEE*, 2013, pp. 599-603.

[3] Y. Wang, D. A. Basnayaka and H. Haas, "Dynamic load balancing for hybrid Li-Fi and RF indoor networks," in *IEEE International Conference on Communication Workshop IEEE*, 2015.

[4] D. Grosu, A. T. Chronopoulos and Y. L. Ming, "Load balancing in distributed systems: An approach using cooperative games," in *Parallel & Distributed Processing Symposium IEEE*, 2016.

[5] Ramezani, Fahimeh, J. Lu, and F. Hussain, "Task Based System Load Balancing Approach in Cloud Environments," in *International Conference on Seventh International Conference on Intelligent System & Knowledge Engineering*, 2014.

[6] Wei Zhao, Lin Zhao, Weidong Wu, Sigen Chen, Shaohui Sun, and Yong Cao, "Loading-balance relay-selective strategy based on stochastic dynamic program." *Tsinghua Science & Technology*, 2018, pp. 493-500.

[7] Tao Hu, Peng Yi, Jianhui Zhang, and Julong Lan, "A distributed decision mechanism for controller load balancing based on switch migration in SDN." *China Communications*, 2018, pp.129-142.

[8] Nine, Md. S. Q. Zulkar, et al. "Fuzzy logic based dynamic load balancing in virtualized data centers," in *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on* IEEE, 2013.

[9] Jie Chang, Wen'an Zhou, Junde Song, and Zhiqi Lin, "Scheduling Algorithm of Load Balancing Based on Dynamic Policies, " *2010 Sixth International Conference on Networking and Services,* 2010, pp. 363-367.

[10] Dan Wang, Xiaolong Qian, and Xiaozheng Jin, "Dynamical evolution of weighted scale-free network models, " *2012 24th Chinese Control and Decision Conference*, 2012, pp. 479-482.