Improved dynamic load balancing algorithm based on Least-Connection Scheduling

Liangshuai Zhu[1], Jianming Cui[2], Gaofeng Xiong[1]
1.School of information science and engineering，Guilin University of Technology
2.School of continuing education，Guilin University of Technology;
Guilin，China
liangshuai_zhu@126.com, cjm@glut.edu.cn, 1414047483@qq.com

***Abstrac* — At present, the commonly used load balancing algorithms are mainly round_robin and least_conn, but they do not take into account the constant changes of server performance over time. The paper on the basis of the Least-Connection Scheduling, proposes a dynamic load balancing algorithm, collection server static load factor and dynamic load factor to calculate the weight of the closest server real-time performance parameters, and then introduced the response time and the number of connections as statistics class load factor, finally calculated the current server load. By testing the built-in round_robin and least_conn strategies of Nginx and the dynamic load balancing strategies proposed in this paper, it is found that the improved load balancing strategy makes more efficient use of resources in the server cluster and improves the throughput and processing capacity of the system.**

Keywords: Load balancing; Nginx; Least-Connection Scheduling

1. INTRODUCTION

Since the previous single server overload will cause network delay or even system crash, people have proposed server cluster technology [1]. However, the performance of a single server in a cluster is different. In order to use the server more efficiently, load balancing technology has been proposed.

Classical load balancing algorithms include round-robin algorithm, weighted round-robin algorithm, random allocation algorithm and dynamic feedback scheduling [2-4]. However, with the development of load balancing technology, scholars at home and abroad have proposed some new load balancing algorithms. As described in reference [5], a dynamic feedback load balancing algorithm based on load weight is proposed. The load weight of the node is defined by the ratio of node load to node performance, so that the task is assigned according to the weight of the load; Reference [6] proposed a consistent hash load balancing algorithm based on critical acceleration and decrement. This algorithm uses virtual nodes to perform real server node allocation and uses the accelerating decrement of critical factor mechanism to control the load of nodes, and as a result, excessive is avoided. In addition, some scholars apply genetic algorithm, particle swarm algorithm and ant colony algorithm in swarm intelligence technology to load balancing problems. For example, in the reference [7], the advantages of genetic algorithm and simulated annealing algorithm are considered for the load balancing of server clusters. A genetic simulated annealing algorithm is proposed, which enhances the mountain climbing ability of genetic algorithm and effectively solves the premature problem of genetic algorithm.

At present, most load balancing algorithms use static parameters such as CPU, memory capacity and dynamic parameters such as memory utilization as optimization targets. Server performance is not considered to change with time. In this paper, the load capacity and the load of the node are calculated according to the collected load information, and then the weight of the node will be calculated. Finally, the composite load will be calculated according to the response time and the number of connections of the node. The larger the weight of the node is, the smaller the composite load is, the greater the probability that it is assigned to the request. Therefore, according to the minimum value of the composite load and weight ratio of the nodes in the back-end cluster, a

node is selected to forward the request. If there are multiple nodes with the minimum value, the round_robin policy is adopted. Select a node from it.

2. IMPROVED DYNAMIC LOAD BALANCING ALGORITHM

2.1 Algorithm design ideas

Based on the least_conn algorithm, this paper proposes an improved dynamic load balancing algorithm called the enhancement_conn algorithm. In this algorithm, the weight of the server is determined by two parts, namely the server carrying capacity and the response of static parameters such as memory capacity. The server load condition of dynamic parameter response such as memory utilization is determined; but it is not enough to allocate the load according to the weight of the server. Based on the least_conn algorithm, this paper introduces the response time and calculates the composite load according to the response time and the number of connections. The ratio of the composite load to the weight finds the optimal node.

The algorithm flow chart is shown in fig 2.1:

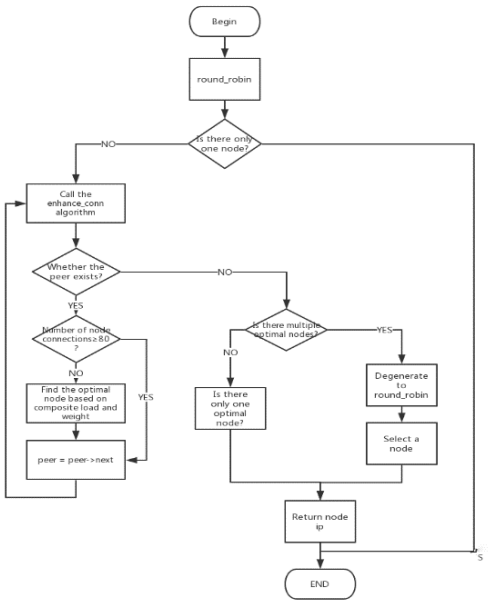

fig 2.1 Algorithm flow

2.2 Calculation of related indicators

Q_n represents the n_{th} node in the backend server cluster, and the cluster with i_{th} nodes can be represented by Q=(Q_1, Q_2, Q_3, ..., Q_i). The CPU frequency, memory capacity, and network bandwidth are represented by Cs, Ms, and Ns. CPU utilization, memory usage, and network bandwidth usage are represented by Cd, Md, and Nd, respectively. The response time and the number of connections are represented by R_{time} and N_{conn} .

2.2.1 Calculation of node carrying capacity

The static load factor reflects the server's carrying capacity, which is the performance of the server. Let $PE(Q_n)$ be the carrying capacity of the n_{th} node, then

$$Ac = \frac{\sum_{n=n}^{i} Cs}{i} \tag{2-1}$$

$$Am = \frac{\sum_{n=n}^{i} Ms}{i} \tag{2-2}$$

$$An = \frac{\sum_{n=n}^{i} Ns}{i} \tag{2-3}$$

$$PE(Q_n) = \alpha_c \frac{Cs}{Ac} + \alpha_m \frac{Ms}{Am} + \alpha_n \frac{Ns}{An} \tag{2-4}$$

Ac, Am, and An are the average CPU frequency, average memory capacity, and average network bandwidth of the back-end cluster, respectively, where $\alpha_c+\alpha_m+\alpha_n=1$, which is used to reflect the importance of each factor.

2.2.2 Calculation of node load conditions

The dynamic load factor reflects the load on the server. Let $LS(Q_n)$ be the load of the n_{th} node, then

$$LS(Q_n) = \alpha_c Cd + \alpha_m Md + \alpha_n Nd \tag{2-5}$$

When $LS(Q_n)$ is greater than a certain value, the load of node i can be considered too large, and the load should be stopped from being distributed to it.

2.2.3 Calculation of node weights

Let the weight of the node be $WE(Q_n)$. The larger the weight, the greater the probability of being assigned to the request. $WE(Q_n)$ is positively related to the carrying capacity $PE(Q_n)$ of the server. The larger the PE (Q_n), the stronger the performance of the server, and the more requests that can be processed. However, $PE(Q_n)$ is static. As the server runs, the remaining performance of the server will change continuously. $PE(Q_n)$

cannot correctly reflect the load change of the server. Therefore, it is necessary to introduce $LS(Q_n)$. $LS(Q_n)$ reflects the resource usage of the server itself to some extent, and indirectly reflects the load condition, which can make up for the shortage of $PE(Q_n)$. $WE(Q_n)$ is negatively related to $LS(Q_n)$. The larger $LS(Q_n)$, the larger the load on the server, and the lower the probability that it is assigned to the request. The calculation formula of $WE(Q_n)$ is as follows.

$$WE(Q_n) = \frac{PE(Q_n)}{LS(Q_n)} \quad (2\text{-}6)$$

2.2.4 Calculation of composite load

The response time will be affected by the server network status. Simply distributing the request according to the response time will cause the load to be unbalanced. The number of connections and the time taken are also related to the server performance. For a server with high performance, The resources used to process multiple requests may only be a small part of their own carrying capacity. If the request is distributed to other servers with few connections but the server performance is weak, it seems that its load is small, but actually Processing these requests has taken up most of its own resources, and there will be surface load balancing, which is actually not balanced.

Therefore, this paper proposes a composite load (CTL) in combination with response time and number of connections, and uses CTL (Q_n) to represent the composite load of node n.

$$CTL(Q_n) = \beta_t R_{time} + \beta_c N_{conn} \quad (2\text{-}7)$$

Among them, $\beta_t+\beta_c=1$ is used to reflect the importance of each factor. When the CTL (Q_n) is smaller, the probability of allocating a request to the node n is larger.

2.2.5 Calculating the optimal node

Through the foregoing analysis, it can be known that the probability that the node Q_n is assigned to the request is negatively correlated with its composite load CTL (Q_n), and is positively related to its weight $WE(Q_n)$, and the node with the smallest value of $CTL(Q_n)/WE(Q_n)$ is selected. The optimal node is as follows.

$$\frac{CTL(Q_m)}{WE(Q_m)} = \min\left\{\frac{CTL(Q_n)}{WE(Q_n)}\right\} \quad (2\text{-}8)$$

3. TEST

In this thesis, the load balancing capability of Nginx proxy server is tested by Apache BenchMark. The load balancing ability of the built-in round_robin algorithm and the least_conn algorithm is tested. Finally, the enhancement_conn algorithm proposed in this paper is tested.

The indicators for evaluating the performance of the algorithm are mainly the average response time of the request sent by the client when the same number of requests are issued per connection under the high concurrent connection, and the number of requests that the Web cluster fails to process successfully.

3.1 Test Results

The test results for the round_robin, least_conn, and enhancement_conn algorithms are as follows.

(1) round_robin algorithm

The test result of round_robin is shown in fig 3.1.

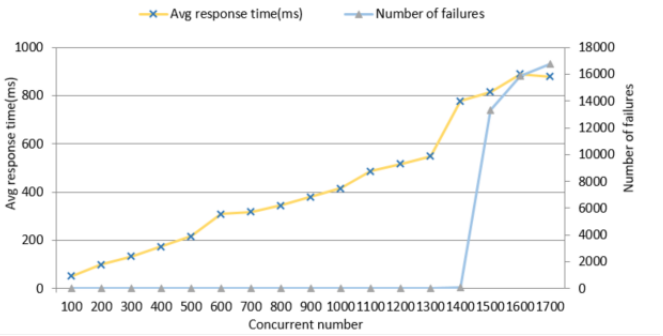

fig 3.1 round_robin algorithm test result graph

As can be seen from Fig 3.1, as the number of concurrent connections increases, the average response time also increases gradually. When the number of concurrent connections increases from 1300 to 1400, the average response time increases the most. After that, a large number of requests are lost.

(2) least_conn algorithm

The test result graph of least_conn is shown in fig 3.2.

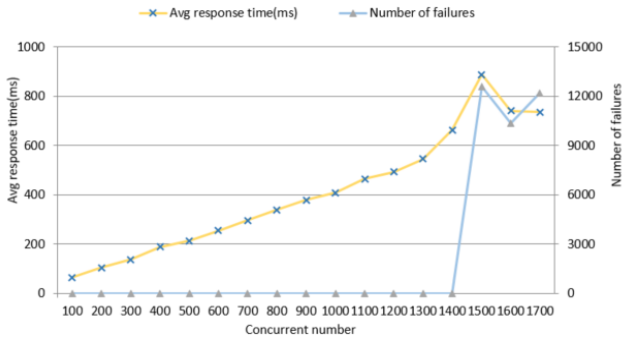

fig 3.2 least_conn algorithm test result diagram

As can be seen from fig 3.2, the average response time increases slowly as the number of concurrent connections increases. When the number of concurrent connections grows to 1300, the average response time increases as the number of concurrent increases increases. When the number of concurrent connections increased from 1400 to 1500, the average response time increased the most, and a large number of requests for loss began to appear.

(3) enhancement_conn algorithm

The test result graph of enhance_conn is shown in fig 3.3.

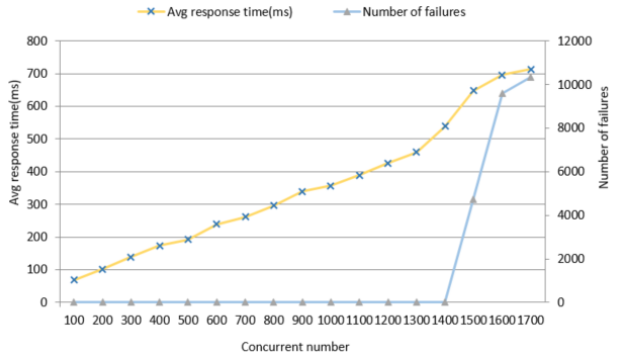

fig 3.3 enhance_conn algorithm test results

It can be seen from fig 3.3 that the average response time increases gently with the increase of the number of concurrent connections. When the number of concurrent connections increases from 1300 to 1500, the average response time increases slowly, and the request loss begins gradually.

3.2 Result analysis

The comparison of the number of failed requests and the average response time of the round_robin, least_conn, and enhancement_conn algorithms are shown in fig 3.4 and fig 3.5.

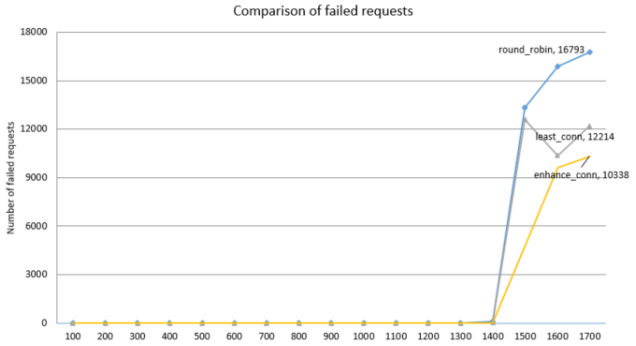

fig 3.4 Comparison of failed requests

As can be seen from fig 3.4, when the number of concurrent calls grows to 1400, the three algorithms begin to show more obvious request failures, in which the number of failed requests of the round_robin and least_conn algorithms increases rapidly, and the growth rate of enhancement_conn is Compared with the other two algorithms, when the same number of requests are issued under the same number of concurrent requests, the number of requests that can be successfully processed by enhancement_conn is improved compared with the other two algorithms, indicating that the proposed algorithm is better than Nginx's built-in round_robin and Least_conn algorithm.

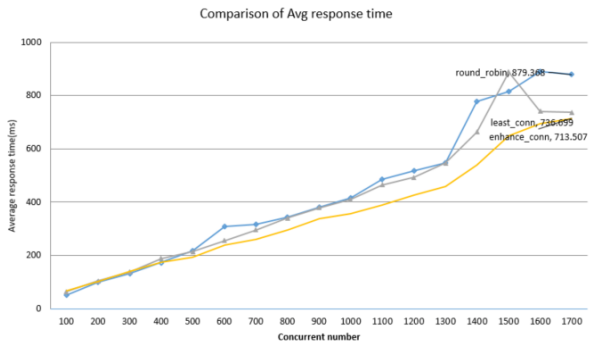

fig 3.5 Comparison of average response time

As can be seen from fig 3.5, the average response time of the three algorithms increases gradually as the number of concurrent connections increases to 1300. In the process of gradually increasing the number of concurrent connections, the average response time of the enhanc_conn algorithm is always lower than other. Two algorithms show that the proposed algorithm is better.

By analyzing the experimental results, we can conclude that the improved load balancing strategy proposed in this paper is better than Nginx's original built-in strategy. To a certain extent, it improves the performance of the back-end cluster and achieves better load balancing.

4. CONCLUSION

For Nginx's built-in load balancing algorithm, there is a problem of unbalanced load distribution. Based on least_conn, an improved load balancing algorithm enhance_conn algorithm is proposed. The weight WE of the node is calculated according to the static load factor and the dynamic load factor, and is calculated according to the statistical load factor. The composite load CTL of the node selects the node with the smallest CTL and WE ratio. If there are multiple minimum nodes, the round_robin algorithm is adopted for these minimum nodes, and one node is selected. Under the same number of concurrent connections, the average response time and the number of request failures of the user request are counted. Comparing the test results of the three algorithms, the enhance_conn algorithm proposed in this paper is better than Nginx's built-in two algorithms in terms of user request response time and actual number of requests processed.

REFERENCES

[1] WangLiping. Research and Improvement of the Load Blalancing Technology based on Nginx Server Cluster[D].SHANDONG University,2015.

[2] Dias D M, Kish W, Mukherjee R, et al. A scalable and highly available web server[C]// IEEE International Computer Conference. IEEE Computer Society, 1996:85.

[3] Cardellini V, Colajanni M, Yu P S. Dynamic Load Balancing on Web-Server Systems[J]. Internet Computing IEEE, 1999, 3(3):28-39.

[4] Casalicchio E, Tucci S. Static and Dynamic Scheduling Algorithms for Scalable Web Server Farm[C]// Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on. IEEE, 2001:369-376.

[5] Zhang Yufang,Wei Qinlei,Zhao Ying. Load balancing algorithm based on load weights[J]. Application Research of Computers, 2012, 29(12):4711-4713. (in Chinese)

[6] Deng Zhenrong,Tang Xingxing,Huang Wenming,Li Yinwei. A load balancing scheduling algorithm for web server cluster [J]. Computer Applications and Software. 2013(10):53-56. (in Chinese)

[7] Rao Lei,Tang Xiaochun,HouZengjiang. Research on Load Balancing Strategy for Server Cluster[J]. Computer and Modernization, 2013(1):29-32. (in Chinese)