Computing, Performance and Communication systems (2016) 1: 1-7
Clausius Scientific Press, Canada

The Improvement and Implementation of the High Concurrency Web Server Based on Nginx

Baiqi Wang[1, a], Jiayue Liu[2,b] and Zhiyi Fang [3,*]

[1] Qianjin Street 2699, Jilin University, Changchun, Jilin Province, China
[2] Qianjin Street 2699, Jilin University, Changchun, Jilin Province, China
[3] Qianjin Street 2699, Jilin University, Changchun, Jilin Province, China
[a]645101851@qq.com, [b]18657684223@163.com, [*]fangzy@jlu.edu.cn

Keywords: high concurrency, Nginx, Web server, load balancing

Abstract: According to the original Nginx load balancing strategy, this paper proposes a load balancing strategy that adjusts weight dynamically which is based on the server performance index. This strategy, which improves on existed weighted polling strategy, dynamically updates the node weights according to the CPU, memory, disk IO and network performance of the nodes. So that each node can be assigned to tasks that can fit its load capacity, which enhances the cluster performance and improves the efficiency. By using the siege performance testing tool, the performance of the original weighted polling strategy and the improved dynamic policy is tested. The experiment proves that the dynamic strategy proposed in this paper is better to implement the load balancing technology.

1. Design and Implementation of High Concurrent Web Server

1.1 Nginx load balancing policy

The current load balancing strategy mainly includes IP hash and weighted polling strategy. The limitations of traditional load balancing are follows:

Weighted Polling Load Balancing Strategy: Multiple requests from the same client may be assigned to different back-end servers for processing, which cannot meet the needs of the session-maintained application. During the cluster running, the policy cannot judge the situation of dynamic load of the server node. When the node load is uneven, some servers are prone to high load. IP Hashing Load Balancing Strategy: When the system receives many requests from a particular address at a time, the pressure on one server may be particularly high and other servers may be idle.

1.2 A Dynamic Load Balancing Strategy with Adjusted Weight

In view of characteristics and limitations of the strategies above, this paper presents a dynamic load balancing strategy with adjusted weight. The strategy is based on the original weighted polling strategy, and makes the following improvements: 1)According to the CPU, this strategy assign the weights dynamically to solve the problem of overload caused by unequal processing of tasks by back-end server nodes. 2)Set the threshold and load redundancy values for the back-end server node. When the node reaches the "modified threshold", it starts the weight modification strategy of the

system. When the node redundancy value is too low, the request task will not be allocated to the node, which solves the problem that the server cannot receive the node's feedback in time when it is saturated.

1.2.1 Definition of relevant parameters.

1) Static Default weight

Suppose that P is the performance index of the server nodes, S_i is the i-th server node. $i \in (1, \ldots, \ldots n)$. $C(S_i)$、$M(S_i)$、$D(S_i)$,$W(S_i)$respectively, represent the performance of CPU, memory, disk IO and network bandwidth of the server node. $P_c(Total)$、$P_m(Total)$、$P_d(Total)$、$P_w(Total)$ represent the sum of the performance of the cluster situation CPU, memory, disk and network IO bandwidth of all nodes. The following equations (1), (2), (3), (4) represent the sum of the performance of all nodes.

$$P_c(Total) = \sum_{i=1}^{n} C(S_i), \tag{1}$$

$$P_m(Total) = \sum_{i=1}^{n} M(S_i), \tag{2}$$

$$P_d(Total) = \sum_{i=1}^{n} D(S_i), \tag{3}$$

$$P_w(Total) = \sum_{i=1}^{n} W(S_i), \tag{4}$$

In order to obtain a more accurate ratio of the performance of the server node， In the consideration of server performance, we select node's performance of CPU, memory, disk IO and network bandwidth to calculate the proportion of performance of the node.

By dividing the performance in one aspect of each node by the sum of the performance in this aspect of all the nodes, multiplied by the proportion of performance of each aspect in the total performance, the real performance specific gravity of the node can be got.

$$W_p(S_i) = A * \left(K_c * \frac{P_c(S_i)}{P_c(Total)} + K_m * \frac{P_m(S_i)}{P_m(Total)} + K_d * \frac{P_d(S_i)}{P_d(Total)} + K_w * \frac{P_w(S_i)}{P_w(Total)}\right), \tag{5}$$

In the formula (5), $W_p(S_i)$ denotes the default weight, S_i denotes the i-th server node, i ∈ (1, ..., n), $P_c(S_i)$,$P_m(S_i)$,$P_d(S_i)$,$P_w(S_i)$ represent the performance of CPU, memory, disk IO and network bandwidth . As described above, $P_c(Total)$,$P_m(Total)$,$P_d(Total)$,$P_w(Total)$ represent the sum of all nodes' performance of CPU, memory, disk IO and network bandwidth in the cluster. K_c,K_m,K_d,K_w denote the specific gravity of CPU, memory, disk IO and the network bandwidth in the node, and $K_c + K_m + K_d + K_w = 1$. A is an adjustment constant for making $W_p(S_i)$ an integer.

2) Dynamic utilization

In the formulas (6) and (7), $U(S_i)$ denotes the utilization rate of node resource, $W_L(S_i)$ denotes the consumed weight, and S_i denotes the i-th server node, i ∈ (1, ..., n). $U_c(S_i)$,$U_m(S_i)$,$U_d(S_i)$,$U_w(S_i)$represent the real-time utilization of CPU, memory, disk IO and network bandwidth respectively . According to the proportion of each part of the coefficient of the node, the resource consumption can be calculated. Resource consumption multiplied by the original performance weight, the weight of the node's current consumption can be got.

$$U(S_i) = K_c * U_c(S_i) + K_m * U_m(S_i) + K_d * U_d(S_i) + K_w * U_w(S_i), \tag{6}$$

$$W_L(S_i) = U(S_i) * W_P(S_i), \quad (7)$$

1.2.2 Optimization thought.

When a client sends a task request to the server, the Nginx load balancing node allocates tasks to nodes according to the load information of received service sub-nodes. Among them, the allocation of tasks is determined by weight settings. Based on the principle of high efficiency and high concurrency, the author puts forward a optimization thought on the weight setting process.

1) Sets the threshold for modifying weight

When adjusting the weight of the load balancing strategy dynamically, frequent changes in weight will not bring performance improvements, but will cause the system jitter. Therefore, by calculating the standard deviation of the node resource usage, it is judged whether the node load is balanced. If the standard deviation is higher than the preset threshold, the weight modification process is initiated.

As shown in the formula (8), the standard deviation S_u of each node resource utilization rate, the resource utilization rate $U(S_i)$ of each node, and the resource usage average $\bar{U}$ of each node are set. The threshold value T_1 is set, and when the value of S_u is larger than T_1, the weight modification flow is started.

$$S_u = \sqrt{\frac{\sum(U(S_i)-\bar{U})^2}{(n-1)}}, \quad (8)$$

2) Introduction of redundant parameters

In the load regulation period, the load redundancy parameter $R(S_i)$ is introduced to evaluate the load capacity of each server node.

$$R(S_i) = W_P(S_i)/W_L(S_i), \quad (9)$$

In the above formula, $W_L(S_i)$ is the load weight calculated in the previous period. When the client request in the previous period is allocated to the node, that is, $W_L(S_i)$ is not 0, the redundancy value $R(S_i)$ is calculated by the formula (9). The load balancer will evaluate each server node in subsequent periods based on this redundancy value. When $W_L(S_i)$ is 0, that is, the service node in the previous period has not requested records, the default redundancy value is the maximum.

When the load on the server node is overloaded, the load redundancy value will continue to decrease and approach to 1. In order to predict the load capacity of each sub-server node effectively and ensure the working efficiency of the whole service cluster, the author sets the boundary value R_{min} as the lowest value of redundancy. When the node redundancy is greater than R_{min}, it indicates that the node still has the ability to process the request, the sub-service node can accept the task request. When the load redundancy value is less than R_{min}, the load is too high, so in the current service period, the node is no longer assigned tasks.

1.2.3 Calculation of weight.

The size of the resource utilization is inversely proportional to the processing power of the back-end server, that is, the larger the resource utilization is, the worse the processing capability of the node is. Based on the above analysis, a variable σ is introduced to represent the ratio of current node resource usage.

As shown in formula (10), the node resource utilization ratio σ is obtained by dividing the

resource utilization $U(S_i)$ of the node by the average of the resource utilization of all the nodes in the cluster. Normally, the proportion of node resource utilization should be average, which is 1, and the ratio of node resource utilization deviation from 1 is larger, which indicates that the load condition of nodes in the cluster is more unstable. As shown in equation (11), when σ <1, the node resource utilization is low, that is, the node task is not saturated. So increase the original weight in the next round of request, so that the node is assigned more tasks. And when σ> 1, the situation is just the opposite, the node task allocation has been overloaded, so reduce the weight. Since the original weights are assigned according to the system performance, the original weight is modified according to the principle of slow-add and fast-decrease. That is, when the weight is increased, a slow addition of less than 1 is performed; when the weight is reduced, a fast subtraction of more than 1 is performed.

$$\sigma = \frac{U(S_i)}{\frac{1}{n}\sum_1^n U(S_i)}, \tag{10}$$

$$W(S_i) = \begin{cases} W_p(S_i) + \sigma, & (\sigma < 1) \\ W_p(S_i) - \sigma, & (\sigma > 1) \end{cases}, \tag{11}$$

1.3 Realization of Load Balancing Strategy with Dynamic Adjustment Weight.

1.3.1 Algorithm processing flow

The load balancer collects the resource usage of each node periodically and solves the standard deviation S_u. It is judged whether the standard deviation S_u of the current resource usage is greater than T_1, and if it is greater than T_1 (Pre-set thresholds), it indicates that the distribution of the load is not uniform and needs to be adjusted. Otherwise it will not be adjusted. When the weights need to be adjusted, the load balancer reevaluates the load capacity of each node according to the dynamic policy. If the redundancy value of the current node's processing capacity is less than R_{min}, the task is not allocated to the node in the next round. So the effective_weight of the load capacity is assigned to zero. Otherwise the new weight is calculated. After the evaluation of the load capacity of each node, the new weights of each node are updated.

1.3.2 Algorithmic Scheduling Implementation

1) Global Preparations

When Nginx starts, it will parse the configuration file, the configuration file will be converted to Nginx corresponding variable value.

2) Node scheduling

After Nginx is started, it initializes the configured policy. When a request arrives at Nginx, the load balancer will select the backend node through this policy. This load balancing strategy which is based on the original weighted polling strategy adjusts weights dynamically. That is, during the operation of the weighted polling policy, the load balancing weight is updated according to the dynamic running state of the server node to prevent the node from overloading. Its scheduling process is the same as weighted polling. That is, firstly, initialize the list of load balancing server, when receiving the client request, nginx will choose the most appropriate node to provide services for the client. The size of the current weight is the basis for judging whether the node is suitable or not.

3) Update of weights

In this paper, the part of updated weight is achieved by ngx_lua. When loads the configuration

file, nginx process runs the specified lua script to register lua global variables. Upon receiving a client request, ngx_lua will evoke a coroutine to process the request. This article achieves the assessment of the load information and update operations by using dynamic_update_weight.lua script. Load equalizer reads the memcached cache to obtain the resource usage information $U(S_i)$ of each node, and calculates the standard deviation to judge whether the weight needs to be updated. When the weight update operation is needed, each node is traversed in turn to obtain the corresponding weight. Finally, the weights are updated.

2. Performance Analysis of High Concurrent Web Server

Through the test cases, we test the performance of the proposed dynamic strategy and the original weighted polling strategy and compare the results. Simulate the number of concurrent connections using the test tool siege, and collect response times, actual maximum concurrency, and throughput for two load policies when the number of concurrent connections is between 100 and 1000.

2.1 Response time

It can be seen from Figure 1 that when the number of concurrent connections is increasing, the response time of the original weighted polling strategy and the dynamic load balancing strategy are increasing. The response time of the two strategies is basically the same before the number of concurrent connections increases to 600, but when the number of concurrent connections increases again, the response time of the dynamic strategy is slightly reduced compared with the original one.

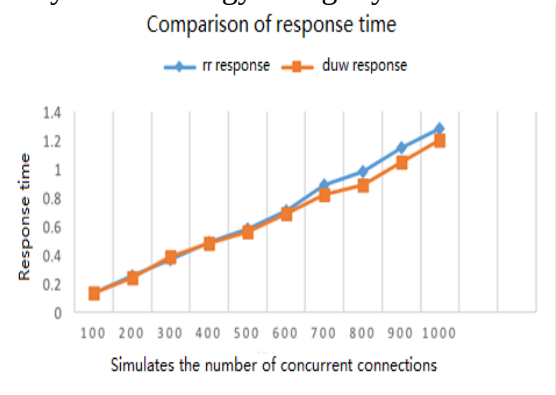

Figure 1.Figure response time comparison

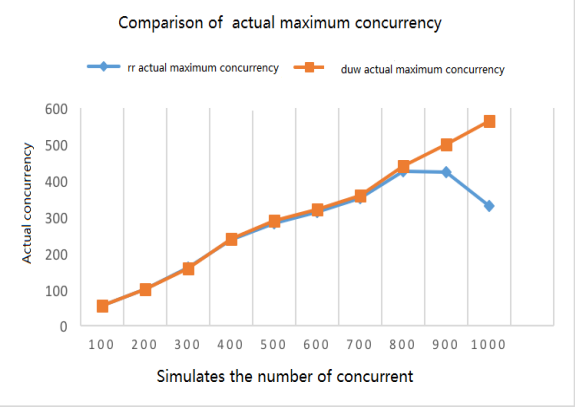

Figure 2.Figure the actual maximum number of concurrent comparison

2.2 The actual maximum number of concurrent

It can be seen from Figure 2 that when the number of concurrent connections is higher than 800, the actual maximum concurrency of the static strategy begins to decrease, and the improved dynamic strategy also presents an increasing trend, which shows that compared with the static strategy, dynamic policy can take on higher concurrent requests.

2.3 Throughput

As shown in Figure 3, after the number of concurrent connections is greater than 500, the throughput of the original weighted polling strategy appears to be unstable or even declining. The improved dynamic strategy will not decrease until the number of concurrent connections is higher than 800. With the increase in the number of concurrent connections, the throughput of the improved dynamic strategy is always better than the static strategy.

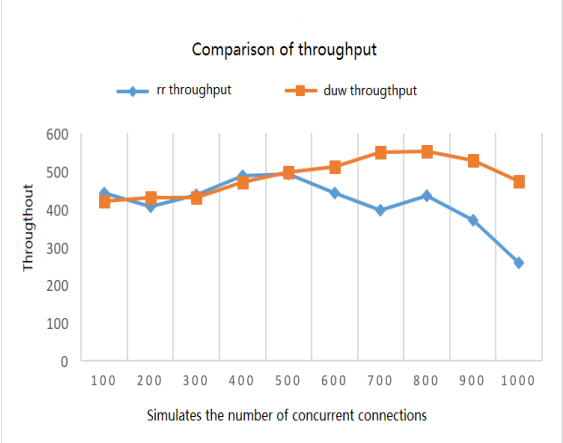

Figure 3. Figure throughput comparison

Compared with the original static weighted polling strategy, the dynamic load balancing strategy in this paper improves the throughput and the maximum concurrency of the system, reduces the response time of the system, and enhances the overall performance of the system.

3. Summary and Prospect

Based on the load balancing server Nginx, this paper proposes a dynamic load balancing strategy with weight adjustment, and improves the original weighted polling strategy. Compared with the original load balancing strategy, this dynamic load balancing strategy proposed in this paper has the following characteristics: 1) The node performance evaluation is more comprehensive. The performance weight of the node is calculated by collecting the performance of the back-end server's node CPU, memory, disk IO and network bandwidth. Compared with the empirical assignment, this method of calculation is more comprehensive and effective. 2) Set the weight modification threshold. In the dynamic adjustment of the weight of the load balancing strategy, frequent changes in weight will not bring performance improvements, but will cause the system jitter. Therefore, it is judged whether the node load is balanced by calculating the standard deviation of the node resource usage. If the standard deviation is higher than the preset threshold, the weight modification process is initiated. 3) Set the redundancy value. In order to predict the node load more accurately during the period and prevent node from overloading, the redundancy performance of the cycle is calculated according to the load of the previous cycle. When the redundancy value is too low, the

node is assigned fewer tasks. 4) Dynamically modify the weight. When judging the need to modify the weights, the system will be based on the node's resource utilization to calculate an increment. When the node is heavily loaded, the weight value subtracts the incremental value as the new weight, and vice versa.

In the follow-up study, for the load balancer, we will further study the design of distributed and combined manner, so that load balancing technology to better improve and improve.

References

[1] M, Burrows. The Chubby lock service for loosely-coupled distributed systems[R]. Seattle:Osdi Proceedings of Symposium on Operating Systems Design & Implementation, 2010. 335-350.

[2] H, Singh, S, Kumar. Dispatcher Based Dynamic Load Balancing on Web Server System[J]. International Journal of Grid & Distributed Computing, 2011, 4(1): 89-106.

[3] NK, Chawla. A New Fuzzy Algorithm for Dynamic Load Balancing In Distributed Environment[J]. International Journal of Computer Science and Information Security, 2009, 6(1): 3-7.

[4] M, Mehta, D, Jinwala. A Hybrid Dynamic Load Balancing Algorithm for Distributed Systems[J]. Journal of Computers, 2014, 9(8): 1825-1833.

[5] S, Dhakal, MM, Hayat, JE, Pezoa, C, Yang, DA, Bader. Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach[J]. IEEE Transactions on Parallel & Distributed Systems, 2007, 18(4): 485-497.