Research and Realization of Nginx-based Dynamic Feedback Load Balancing Algorithm

Zepeng Wen[1], Gongliang Li*[1], Guanghong Yang[1]
1.Institute of Computer Application, CAEP, Mianyang, China
wenzp@icaep.com, ligl@icaep.com,yanggh@icaep.com

***Abstract*—With the rapid development of internet technology, internet has brought great conveniences to the vast majority of users. However, the exploding of concurrent visits also raised huge challenges to the working capacity of network servers. Thus server cluster and load balancing technologies become a powerful approach to deal with the bottleneck of server terminals. The Nginx-based load balancing method, featured with its high cost performance and extensibility, has been widely applied. This paper raised a Nginx-based dynamic feedback load balancing algorithm. This loading algorithm module is realized by designing an algorithm on the basis of queuing theory and through Nginx plug-in mechanism. Then the paper established an experiment environment, and used Loadrunner software to conduct a comparison test on the performances of the Nginx built-in IP Hash algorithm & Weighted Round-Robin Scheduling and the dynamic feedback algorithm raised in this paper. The results showed that dynamic algorithm raised in this paper can better realize the load balancing.**

Keywords—Cluster;Nginx;queuing theory;Loadrunner; load balancing

I. INTRODUCTION

The rapid development of internet has significantly changed people's traditional lifestyles. Despite of the great conveniences that internet has brought to people, the exploding of concurrent visits due to the increase of internet users is also a huge challenge to the concurrent response ability of network servers. Meanwhile, larger number and more complicated Web transaction processing also raised higher requirement on the performances of servers. Currently, the most common solution is to use the multi-server policy based on load balancing concept. In this policy, multiple servers are used to form a cluster system and provide a unified web services for users. The number of servers can be adjusted according to the business demands.

In recent years, new technologies and schemes are coming up, either DNS-based load balancing schemes or reverse proxy load balancing schemes such as Apache-based, Nginx-based or Lighthhpd-based schemes. A large number of researchers have made great contributions to this field. Load balancing has a wide application, no matter in homogeneous cluster systems and heterogeneous cluster systems, or in distributed systems[1] and cloud systems.

Nginx is featured with low memory consumption, high concurrency, high extensibility, rich third-party modules and open source codes. Hence, this paper optimized Nginx and made some researches and improvements on the load balancing algorithm. Then, based on the study and realization principle of Nginx framework, the paper raised a dynamic feedback load balancing algorithm based on queuing theory.

II. RELATED TECHNOLOGIES

Load balancing algorithm includes static algorithm, dynamic algorithm and adaptive algorithm[2]. Static load balancing algorithm has been widely researched, which cannot obtain the accurate information of the global state at a given moment; while scheduling algorithm cannot make any change according to the global state. So the effect is rather limited; dynamic load balancing algorithm is to change the distribution tasks dynamically according to the load information of processing nodes to help cluster system reasonably solve the balancing between each node[3][4][5].

A. Static load balancing algorithm

Static scheduling algorithm realizes the distribution of customer's visit requests by using load balancer to execute the algorithm strategy according to the predesigned scheduling strategy. Static scheduling algorithm has no consideration on the load status and real-time connection of each node in the cluster system. It conducted load balancing scheduling only according to the preset strategy. Static scheduling algorithm is simple to realize and convenient to use. However, this algorithm has a poor concurrency. It is efficient only under certain circumstances. Round-Robin and Weighted Round-Robin Scheduling are typical examples of static algorithm.

B. Dynamic load balancing algorithm

Dynamic scheduling algorithm usually needs to obtain and analyze some real-time load information of the cluster system such as the load state and the number of connected requests of each server node, etc. Then, the algorithm will distribute the new requests to each server node in the cluster system according to the feedback of load information for processing. The purpose is to realize an overall dynamic load balance. In most cases, dynamic scheduling algorithm is better than static scheduling algorithm. However, dynamic scheduling algorithm requires extra system resources when obtaining, storing, analyzing and calculating the real-time information returned from each node. But it is acceptable when considering the overall performance promotion of the dynamic scheduling algorithm in the cluster system. Therefore, dynamic scheduling algorithm developed very well, and great achievements have been made on the research of this algorithm[6][7][8].

C. Adaptive load balancing algorithm

Adaptive load balancing algorithm is an intelligent algorithm, which can flexibly change the algorithm parameters and the adopted algorithm strategies to adjust the task scheduling behaviors of the load balancer. Hence, the algorithm has an excellent adaptability to the changes of cluster system status and external network environment. Normally, the adaptive balancing process will choose proper task scheduling strategies according to different cluster system status and network structures[9]. That is why adaptive load balancing algorithm is very complicated and involves a wide area of knowledge. Currently, the study on adaptive load balancing algorithm is still at the stage of development.

III. DESIGN AND REALIZATION OF DYNAMIC FEEDBACK LOAD BALANCING ALGORITHM

A. Modeling of dynamic feedback load balancing algorithm

According to the queuing theory, when customer's request reaches the cluster, the load balancer will distribute the request to the specified node according to a certain scheduling algorithm. At this moment, the M/M/1 model of the queuing theory can be used for load balancing, and following conclusion can be drawn: both the service request arrival rule and the time that the cluster node deals with the service request are described by probability distribution function.

The modeling of load balancing can be performed according to the queuing theory model. The assumed conditions of the model include:

1 : One cluster system consists of one cluster manager and two cluster nodes.

2 : Cluster's service request arrival rate complies with the Poisson distribution (parameter=λ).

3 : The request processing time of each cluster node complies with the negative exponential distribution (parameter=μ).

4 : The time of the cluster manager in load information collection and task scheduling for one time of one cluster node can be ignored.

5 : Cluster manager and cluster node use FIFO (first in first out) mode to deal with the arrived service requests.

B. Design of dynamic feedback load balancing algorithm

One important parameter of dynamic feedback is the load degree of each current node. The change of such load degree has a direct impact on the adjustment of server weight. Thus it is very important to the evaluation on the load status. The selection of load parameters has apparent impact on the whole scheduling. The indexes of server load parameters mainly include service rate (R1), CPU usage rate (R2), memory usage rate (R3), network usage rate (R4), and node response rate (R5), etc. The node response time refers to the time delay between the sending of request from the load balancer and response on the request of the node. The node response time can reflect the request waiting queue length and the processing time of the request on the server. A series of coefficients γi are used to represent the degree of importance of each parameter. Based on the above analysis, the resource availability of each server node can be expressed by a function of load parameter Rn, and the following formula can be obtained.

$$R_{all} = \prod_{i=1}^{5} R_i^{\gamma_i}, \sum_{i=1}^{5} \gamma_i = 1, \gamma_i \geq 0 \tag{1}$$

The whole cluster system is a queuing system. There are also task queues in cluster nodes. Hence, each individual cluster is also an independent queuing system. In this system, all the task requests are treated equally.

The whole cluster system is regarded as a queuing model. Each customer request is regarded as the customer, waiting for services. After forwarded by the load balancer, each cluster node will also queue up when receiving customers. Hence, a cluster node can be also taken as a queuing system.

The load balancer plays as the forwarder. Assume the arrival probability of customer's requests on the cluster load balancer to be ω, and the arrival probability of customer's requests on each cluster node to be ωi. This queuing system is a M/M/1 infinite source waiting system model, with the total queue length of Qmax, and waiting queue length of Q.

In this queuing model, window's customer reception ability amounts to the processing ability of the cluster node, which can be expressed by β (unit/second). In the cluster system, the higher the node load, the more the customer's requests, and the lower the processing ability. Similarly, the smaller the β. Therefore, the following comparison expression between the node load and β can be established:

$$\beta = \alpha \cdot R_{all} \tag{2}$$

Where, α refers to the number of processed customer requests by the cluster node device in each second.

As the queuing system between the node and customer's requests is a M/M/1 model, so the number of service windows n = 1, and D = ρ. Assume the expected task queue Qq = Qmax–Q. Qq shows that the node device can still provide service from another aspect. The average arrival rate of the expected task is defined as ω (unit/second). Then according to the definition of queuing theory:

$$Q_q = \rho\left(1 + \frac{D}{1-\rho}\right) = \frac{\omega_i}{\beta - \omega_i} \tag{3}$$

According to Formula 2 and Formula 3:

$$\omega_i = \frac{Q_q}{1+Q_q} \cdot \alpha \cdot R_{all} \tag{4}$$

Thus the relationship between ωi and the node resource availability Rall can be obtained. When the node load increases, the average arrival rate of the expected task will decrease, which is in accordance with the load balancing theory. When the cluster node load is heavier, the server response rate should be reduced properly, and the distributed customer requests will be also reduced. This indirectly proved the feasibility of this method.

In the load balancer, all the received customer requests can be regarded as a queue. The distribution of requests to the cluster nodes are based on the expected arrival probability to the cluster nodes. The larger the expected arrival rate ωi, the larger the node resource availability Rall will be, and more requests will be distributed; the smaller the expected arrival rate ωi, the smaller the node resource availability Rall will be, and less requests will be distributed; assume the server node collection D= {d1, d2, …, dn}, then the distribution strategy of the cluster can be defined:

$$Q_{d_i} = Q_{all} \frac{\omega_i}{\sum_{j=1}^{n} \omega_j} \quad (5)$$

Where, Q_{di} represents the queue that the cluster load balancer distributed to d_i in the node device; Q_{all} represents all the customer request queues received by the cluster load balancer; n refers to the number of cluster nodes in the cluster system; ω_i refers to the arrival rate of the expected requests at the cluster node i.

IV. REALIZATION OF DYNAMIC FEEDBACK LOAD BALANCING ALGORITHM

Based on Nginx plug-in system, this paper made an extension on the upstream module to realize the real-time dynamic feedback load balancing algorithm. First, the entry function of this module is defined as "ngx_http_upstream_queue_adjustment". The realization of the function is as following.

```
static char *ngx_http_upstream_ queue_adjustment (ngx_conf_t
*cf,ngx_command_t *cmd, void *conf){
....
/ * Set up the initialization load  balancingmodule * /
uscf = ngx_http_conf_get_module_srv_conf(cf,
ngx_http_upstream_module);
uscf->peer.init_upstream = ngx_http_upstream_init_ queue_adjustment;
....
return NGX_CONF_OK;
}
```

The function obtained the back-end server information of the configuration file, and set the callback function "ngx_http_upstream_init_queue_adjustment" initialized by "peer.init_upstream". The function is realized as following.

```
static ngx_int_t ngx_http_upstream_init_ queue_adjustment (ngx_conf_t
*cf,ngx_http_upstream_srv_conf_t *us){
/ * Call the weighted polling policy initialization function * /
if (ngx_http_upstream_init_round_robin(cf, us) != NGX_OK) {
return NGX_ERROR;
}
/ * Reset the ngx_http_upstream_peer_t structure of the init callback
method  * /

us->peer.init = ngx_http_upstream_init_ queue_adjustment _peer;
return NGX_OK;
}
```

Weighted Round-Robin Algorithm is the basis of other Nginx load balancing algorithms. So during the initialization of the algorithm, the "ngx_http_upstream_init_round_robin()" function should be called to perform the initialization and revise the "peer.init" initialized callback function as "ngx_http_upstream_init_queue_adjustment_peer". At this time, the callback function will be called by Nginx when receiving customer's HTTP requests. The core code is as following:

```
static ngx_int_t ngx_http_upstream_init_ queue_adjustment
_peer(ngx_http_request_t *r,ngx_http_upstream_srv_conf_t *us){
....
if (ngx_http_upstream_init_round_robin_peer(r, us) != NGX_OK) {
return NGX_ERROR;
}
....
/ * Callback function settings * /
r->upstream->peer.get = ngx_http_upstream_get_ queue_adjustment
_peer;
r->upstream->peer.free = ngx_http_upstream_free_ queue_adjustment
_peer;
r->upstream->peer.tries = qap->peers->number;
....
return NGX_OK;
}
```

Upon the receipt of customer's requests, nginx will call the function for initialization. The "ngx_http_upstream_init_round_robin_peer()" Weighted Round-Robin Strategy is called. After multiple selective failures of the algorithm, it will degrade into Weighted Round-Robin Algorithm. According to the code, the callback function assigned by "peer.get" pointer is set as "ngx_http_upstream_get_queue_adjustment_peer". This function is the core part to realize the algorithm of this paper. The main effect of the function is to choose optimal server from the cluster to process user's HTTP requests. The function can be realized as following.

```
static ngx_int_t ngx_http_upstream_get_ queue_adjustment
_peer(ngx_peer_connection_t *pc, void  *data){
....
/ * Get the best peer * /
bestpeer = get_best_peer(peers);
....
app_server->sockaddr = bestpeer->sockaddr;
app_server->socklen = bestpeer->socklen;
app_server->name = &bestpeer->name;
....
}
```

This function is mainly used to seek the server bestpeer which has best processing ability. Then it will assign the related information of bestpeer to the sockaddr, socklen and name of app_server. Based on these parameters, "upstream" will distribute the HTTP requests to the server "app_server->name" for processing.

V. PERFORMANCE TEST AND RESULT ANALYSIS

In this paper, tests on Nginx built-in IP Hash algorithm, Weighted Round-Robin Algorithm, and the load balancing algorithm raised in this paper will be performed. The average response time, click rate, throughput, and number of failure events of each algorithm will be summarized.

The tests are carried out by one load balancing server and three application servers (consist of two servers and one desktop computer). In order to simulate the real cluster environment, the three application servers are set with different software and hardware configurations (See Table 1). LoadRunner testing tool is used to simulate user's high concurrency requests. High-pressure tests on the above four load balancing algorithms are performed respectively.

TABLE I. SOFTWARE AND HARDWARE CONFIGURATION OF EXPERIMENT ENVIRONMENT

Environment	**Load Balancing Server**	**App Server A**	**App Server A**	**App Server A**
OS	*Oracle Linux 6.2 64-bit*	*Windows 7 64-bit*	*Windows XP 32-bit*	*Windows server 2008 64-bit*
CPU	*Intel Xeon E7 4820 x2*	*Intel Core i5-2400*	*Pentium E5200*	*Intel Xeon E7 4820 x2*
Memory	*DDR3 16GB*	*DDR3 4GB*	*DDR3 4GB*	*DDR3 8GB*
Web Server	*Nginx 1.8.0*	*Jboss 4.3.2 GA*	*Jboss 4.3.2 GA*	*Jboss 4.3.2 GA*

A. Load test based on IP Hash algorithm

The LoadRunner set the number of concurrent users as 500. The test results are:

Passed Transactions : 47679; Failed Transactions: 7704, Errors: 40343 (all of them have the error of "Failed to connect to server: [10061] Connection refused"); user visits failure rate: 50.19%. Test result for response time and throughput are shown in the figures 1 and 2.

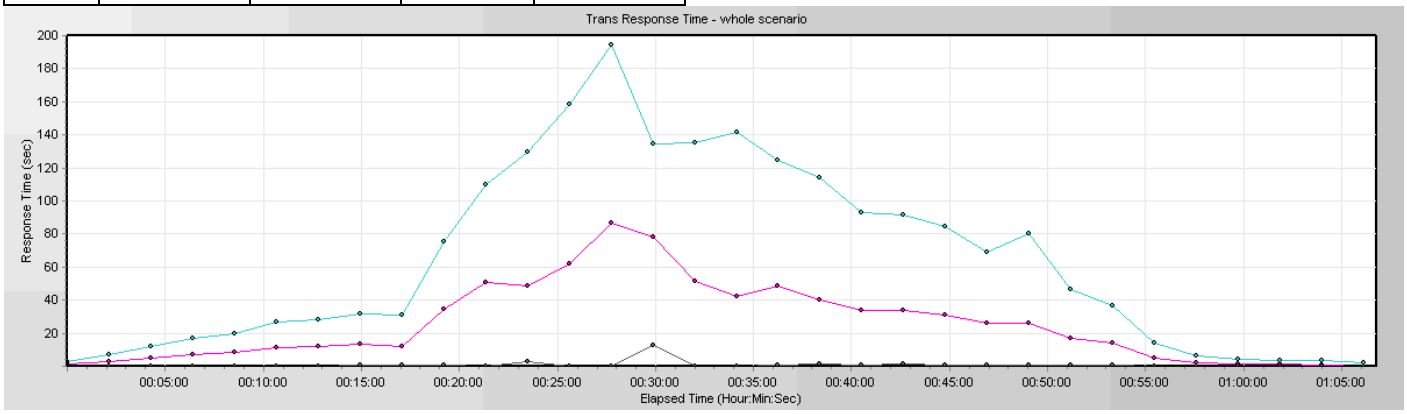

Fig. 1. Trans Response Time based on IP Hash algorithm.

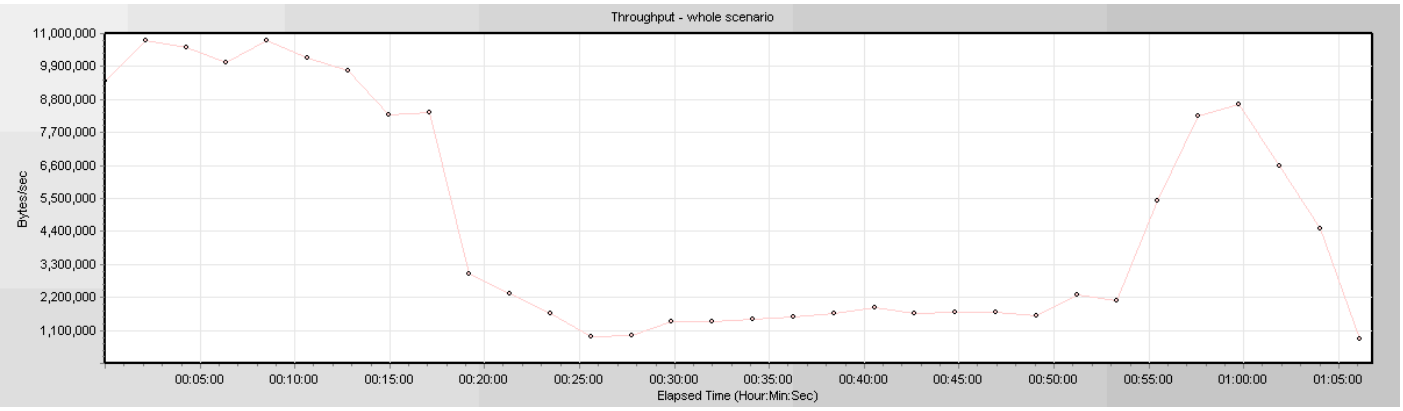

Fig. 2. Throughput based on IP Hash algorithm.

B. Load test based on Weighted Round-Robin Algorithm

The LoadRunner set the number of concurrent users as 500, and set the weights of A, B, C application servers as 2, 1 and 4 respectively according to our estimation. The main results were as follows:

Passed Transactions: 60972; Failed Transactions: 2119; Errors: 2220 (all of them have the error of "Step download timeout (120 seconds) has expired"); user visits failure rate: 3.36%. Test result for response time and throughput are shown in the figures 4 and 3.

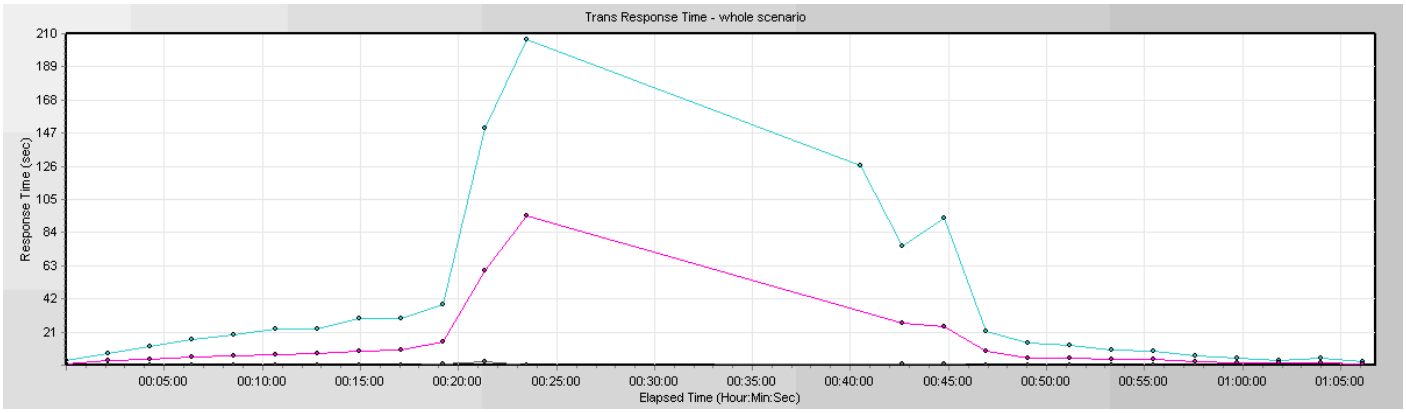

Fig. 3. Trans Response Time based on Weighted Round-Robin Algorithm.

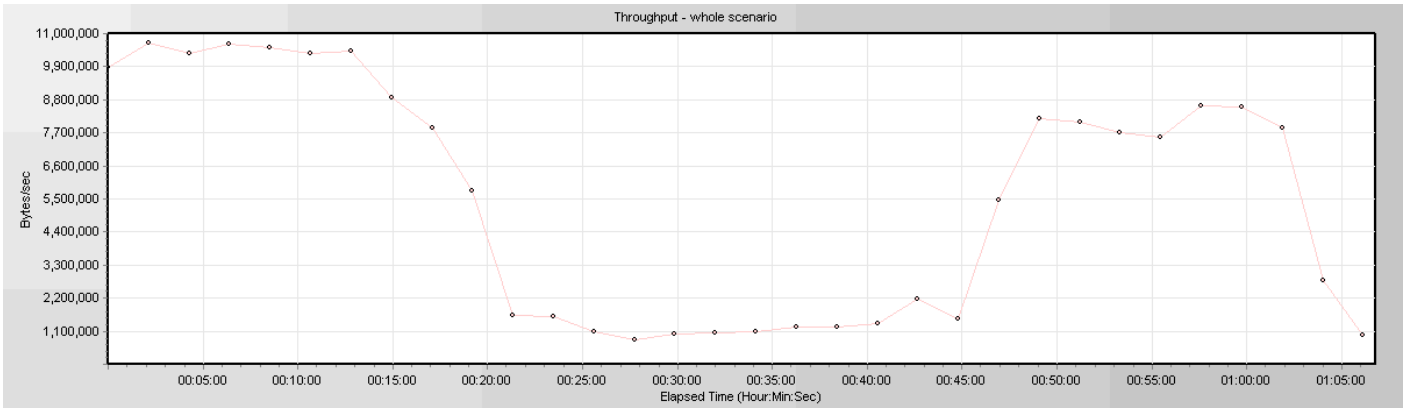

Fig. 4. Throughput based on Weighted Round-Robin Algorithm

C. Load test based on dynamic feedback algorithm

The LoadRunner set the number of concurrent users as 500. The main results were as follows:

Passed Transactions: 79833; Failed Transactions: 540; Errors: 602 (319 have the error of "Failed to connect to server: [10060] Connection timed out", and 283 have the error of "Server has shut down the connection prematurely"); user visits failure rate: 0.67%. Test result for response time and throughput are shown in the figures 5 and 6.

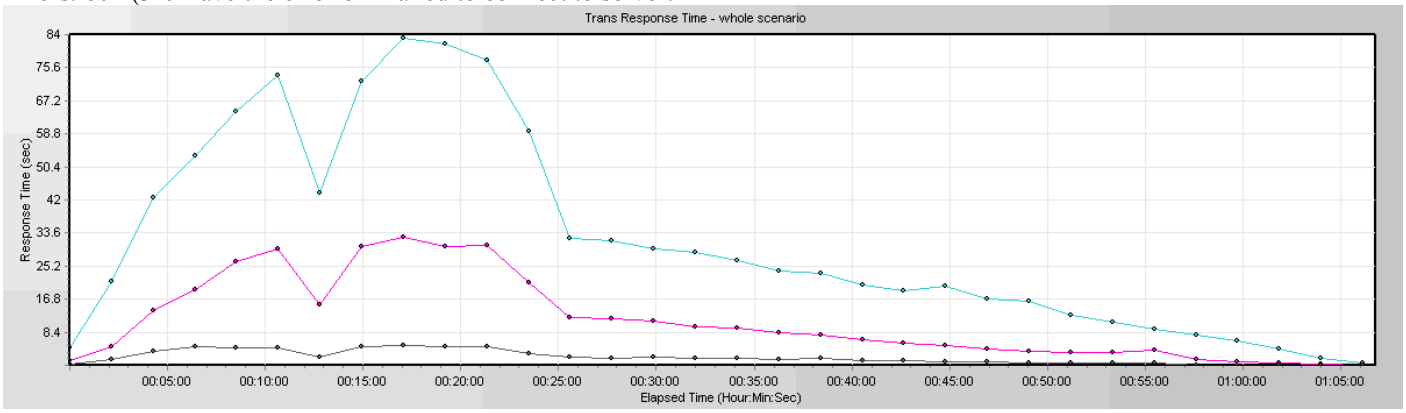

Fig. 5. Trans Response Time based on dynamic feedback algorithm

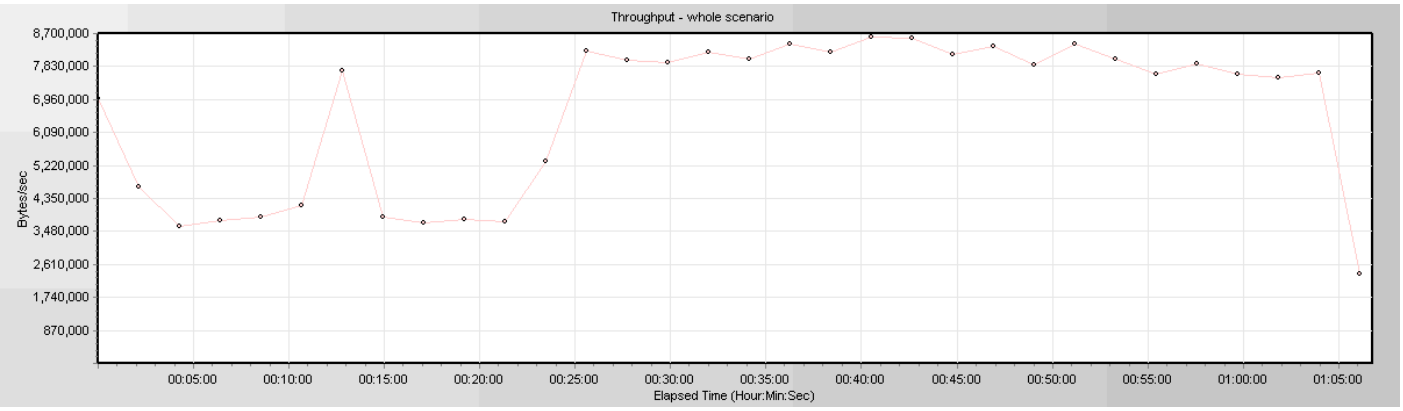

Fig. 6. Throughput based on dynamic feedback algorithm

D. Analysis of test results

TABLE II. COMPARISON OF TEST RESULTS BY DIFFERENT ALGORITHMS

	IP_Hash	***Weighted Round-Robin***	***Dynamic Feedback***
Passed Transactions	*47679*	*60972*	***79833***
Failed Transactions	*7704*	*2119*	***540***
Error	*40343*	*2220*	***602***
Failed rate	*50.19%*	*3.36%*	***0.67%***
Max Response Time	*298*	*209*	***84***
Max Clicks per second	*1456*	*1475*	***1210***
Max Tthroughput	*8000*	***10900***	*8700*

According to Table 2 : IP Hash takes no account of the performances of servers. It distributes the visit requests equally to the back-end servers, which will lead to poor performances and high occurrence of DoS (denial of service) of the servers; while Weighted Round-Robin Algorithm cannot adjust the visit requests dynamically, causing transient pressure on the high-performance servers and DoS; the dynamic feedback algorithm raised in this paper dynamically adjusts the user distribution proportion of three servers with different performances according to the real-time load status of the back-end server. This algorithm can achieve a lowest user request failure rate.

VI. CONCLUSION

This paper mainly introduced some relevant technologies on server cluster load balancing. Based on the existing research findings of load balancing algorithm, the paper raised a dynamic feedback load balancing algorithm. This algorithm uses queuing theory to predict the load status of the application server and dynamically adjusts its load strategies. It is realized through Nginx plug-in mechanism and verified by establishing an experiment testing environment. Under the experiment environment, a comparison test is performed on the dynamic feedback algorithm, the Nginx built-in Weighted Round-Robin Algorithm and IP Hash algorithm. The test results showed that the dynamic feedback algorithm is of high efficiency and feasibility, which can improve the performance of the server cluster to some extent. Next, more load parameters correspondent to the application servers will be added to further improve the algorithm and promote its loading ability.

REFERENCES

[1] M. A. Mehta,D. C. Jinwala.A Hybrid Dynamic Load Balancing Algorithm for Distributed Systems[J]. JOURNAL OF COMPUTERS,2014,9(8):1825-1833

[2] Onur Destanoglu,F. Erdogan Sevilgen. Hydrodynamic based hybrid dynamic load balancing [C]. IEEE Internet Computing, 2008.

[3] Wenmin Miao,Dongni Li,Wei Zhang. A Load-Balancing Dynamic Scheduling Algorithm under Machine Failure Conditions[C]. 2010 International Conference on Intelligent Computation Technology and Automation, 2010:144-147.

[4] B. Yagoubi and Y. Slimani, Task Load Balancing Strategy for Grid Computing,Journal of Computer Science 3 (3): 186-194, 2007.

[5] YU Li,RUAN Wen-tao. Research and Implementation of Load Balance Technology[J];Computer Technology and Development;2007-08

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

[7] Dan Wang;Xiaolong Qian;Xiaozheng Jin. Dynamical evolution of weighted scale-free network models [C]. 2012 24th Chinese Control and Decision Conference, 2012: 479-482P.

[8] Md. S. Q. Zulkar Nine;Md. Abul Kalam Azad;Saad Abdullah;Rashedur M Rahman. Fuzzy logic based dynamic load balancing in virtualized data centers [C]. 2013 IEEE International Conference on Fuzzy Systems, 2012.

[9] Cao Bin,Liu Xie,Sun Qi.Dynamically adaptive load balancing strategy under the software defined network structure[J]. Journal of Chongqing University of Posts and Telecommunications(Natural Science Edition), 2015, 4.