Dynamic Load-balance Scheduling Approach in Linux Based on Real-time Modifying LVS Weight

YihuaLIAO[1, a],Min LIN[2]

[1]College of Computer Science and Technology, Wuhan University of Science and Technology, Hubei Province 430065, China

[2]Wuhan Engineering Institute, Wuhan City, Hubei Province, China

[a]ycina@foxmail.com

Keywords: Linux Virtual Server; Load Balance; Cluster; WLC

Abstract. Linux Virtual Server (LVS) cluster technology is widely applied at high-performance web service. In LVS, Weighted Least-Connection (WLC) scheduling algorithm is usually adopted due to its good effect. However, WLC scheduling algorithm can’t always make real server (RS) run efficiently. In order to improve RS usage and make RS in health load, we proposed a dynamic load balance scheduling approach (DLSA) which dynamically modifies the weight of RS by the feedback information from RS. Experimental results show that DLSA can make full use of RS in LVS.

Introduction

Load balancing[1,2,3,4] technology based on existing network structure provides a cheap and effective method to extend the server bandwidth, increase the server throughput, strengthen the network data processing capability, and improve the flexibility and usability of the network. Cluster technology is a computer network platform. Cluster technology can be used to connect the scattered resources to complete the task that the original single node cannot complete. Linux Virtual Server (LVS) was founded by Zhang Wensong according to the cluster technology. LVS cluster uses IP load balancing technology and content request distribution technology. The scheduler, which has high throughput, transfers requests evenly to different server and automatically shields errors. The structure of LVS cluster is transparent to the client, moreover the client and server programs need not be modified. Load balancing equipment is a performance optimization device, rather than a basic network device. LVS provides a series of algorithms to solve different situations.

Least-Connection Scheduling (LC) assigns new connection requests to the servers with the smallest number of connections. LC is a dynamic scheduling algorithm that estimates the load balancing of the server through the current number of active connections in the server. When the system is implemented, the scheduler needs to record the number of connections that each server has established. When a request is scheduled to a server, the number of connections is incremented by 1. When the connection is aborted or timed out, the number of connections is decremented by 1. If the server weight is 0, it indicates that the server is not available and cannot be scheduled.

Weighted Least-Connection Scheduling (WLC) is an improved algorithm that based on LC. In this algorithm, the corresponding weight is used to represent the processing ability of each server. The greater the weight is, the stronger the processing power is. And the default weight is 1.

The weight of each server of WLC is determined in advance by the administrator, who knows the system characteristics.In [7], the author has mentioned the limitation of WLC, the weight in the

default WLC is a static value and the setting of weight is unscientific and unreasonable. In [5], the author shows the impact of weights on the performance of server load balancing systems, and badly chosen weight may led to unpredictable substantive worse results. So a reasonable and scientific weight is very important. The author of [6] has mentioned that how to set a reasonable weight, which is based on the performance of the server hardware and the performance of the server network. In this paper we focus on the factors of server hardware, the factors of server network will not discuss, but we will make it in the same network to make it less impact to the weight.
The rest of this paper is organized as follows. Section 2 starts with a brief review of LVS system structure and load balance methods. DLSA is proposed in Section 3, followed by performance evaluation on DLSA in Section 4.Conclusions and remarks on possible further work are given finally in Section 5.

LVS System Structure and Load Balance Methods

LVS is a software implementation of the cluster technology, and the architecture is formed by the three layers shown as Figure 1.

1) Load Balancer/ Director: it is the external front-end machine of server cluster and sends the client's requests to a set of servers to be executed. And client thinks that the service is provided by the IP address which can be called a virtual IP address.
2) Server Pool/ Real Server: it is a group of servers which truly execute the client's requests. The servers receive requests from the load balancer, and the executive services generally consist of WEB, MAIL, FTP, DNS and so on.
3) Shared Storage: it provides a shared storage area for RS, so it is easy to make the server pool own the same content and provide the same service.

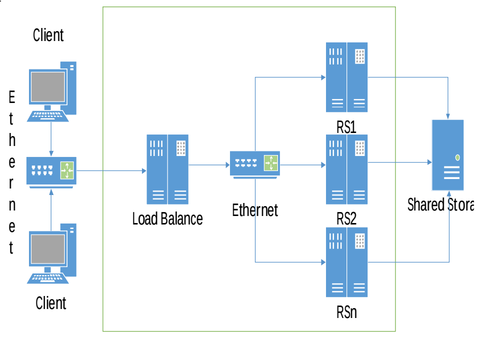

Figure 1 LVS System Structure

The main features are listed as follows:

1) **Work in the network layer:** it can achieve high-performance, highly available server cluster technology;
2) **Cheap:** it can integrate many low performance servers to form a super server;
3) **Manageability:** configuration is very simple, and there are a variety of load balance methods;
4) **Reliability:**even one server cannot work in the cluster server, it does not affect the overall effect;
5) **Scalability.**

There are many ways to implement load balancing techniques, such as: based on DNS domain name rotation analysis method, based on client scheduling access method, based on application layer system load scheduling method and based on IP address scheduling method. In the above methods, the most efficient method is the IP load balancing technology. LVS IP load balancing technology is achieved through the IPVS [8] module. IPVS is the core software of LVS cluster system. It is installed on Director Server and applies a virtual IP address which is usually called VIP on Director

Server. Users must access the service through the virtual IP address. When users access server through VIP, a service node is selected from RS lists by Director to response user's request.
There are three kinds of IPVSload balancing methods in LVS, namely the network address translation (VS/NAT), IP tunnel (VS/TUN) and direct routing (VS/DR). In this paper, direct routing technology is chosen.
The VS/DR method is implemented by rewriting the MAC address of the request message. The load balancer and the RS must be physically connected through an uninterrupted LAN with a network card. The RS that are bound to the virtual server are configured on the non-ARP network devices, and the VIP addresses of the load balances are visible, and the VIPs of the RS are invisible. The address of the RS can be either an internal address or a real address shown as Figure 2.

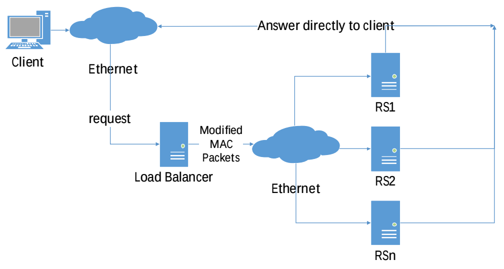

Figure 2 VS / DR architecture

DLSA: Dynamic Load-balance Scheduling Approach

We have mentioned that there are shortages in LVS load balance algorithms, namely, the weight of RS can only be set in advance by the administrator, and cannot be dynamically modified in the running process. So DLSA[9] is proposed as follows.
Due to the limitations of WLC, this paper feeds back the load information of RS to LB and updates the weight value w_i of RS in real time, so that LB can achieve the best load balance effect.

A. Definitions

Definition 1 A RSs_i contains the following four attributes:
$cpu(s_i), memory(s_i), disk(s_i), response(s_i)$ CPU usage, memory usage, disk usage and response time of s_i respectively. And:

$$0<cpu(s_i), memory(s_i), disk(s_i)<1$$

Definition 2 Set *c, m, d, r* as the weight factor of CPU, memory, disk, response, and:

$$c + m + d + r = 1 \quad (c, m, d, c \in [0,1))$$

The weight factor represents the weight relation of the parameter. The larger the value, the stronger RS load capacity is. The parameter need not be considered if its value is 0.
Definition 3 The comprehensive load of RS $s_i$is$load(s_i)$, and *i* represents the *i*thserver.

$$load(s_i) = c \times cpu(s_i) + m \times memory(s_i) + d \times disk(s_i) + r \times response(s_i)$$

Definition 4 Set the percentage of load balancer as$RS(s_i)$, then:

$$RS(s_i)=c \times cpu(s_i) + m \times memory(s_i)$$

B. DLSA Procedure

In this paper, DLSA based on WLC algorithm is put forward. The main processing flow is shown in Figure 3.
DLSA dynamically collects RS information, calculates integrated server node, and modifies the RS weight by the comprehensive RS information. DLSA is a supplement to WLC, so that LVS cluster load scheduling can be more reasonable, and the server can be fully utilized.

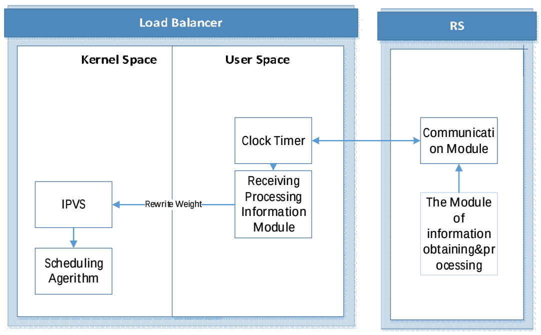

Figure 3 DLSA procedure in LVS

The new algorithm can be divided into three modules according to the process.

1) RS information acquisition;
2) BL receives and processes the information sent from RS;
3) BL rewrites new weights;

RS can use Windows or Linux, and different OS applies different methods to gain RS information. And Linux is selected in this paper. In Linux, CPU usage is divided into user mode and idle state, and we can use the pseudo-file system /proc/stat which is provided by Linux to calculate the CPU usage. The parameters of the system are user, nice, system and idle.Corresponding parameter can be got in pseudo file /proc/meminfo, such as total physical memory, physical memory, shared memory, and buffering. And pseudo file /proc/mtab can read all the information in the file system, and then count the total size of all file systems and the total size of the available disk.

Performance Evaluation

In this section, we mainly introduce the test tools and the implementation of DLSA. And then we analyze the experimental results [10, 11, 12].

A. Test Tools

In order to test the performance of DLSA, we implement DLSA as section *B*. The test devices are shown in Table 1, and the network topology is shown in figure 4. To execute stress test, we use Apache JMeter which is based on Java, and requests the JDK1.7 or above version.

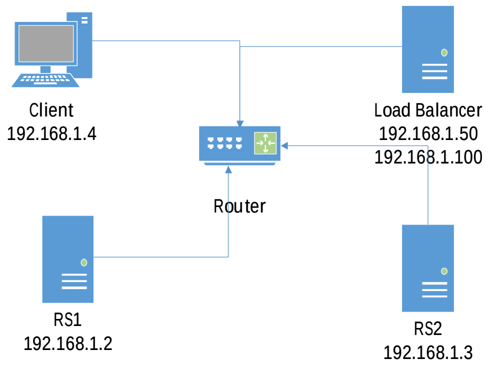

Figure. 4 network topology

B. DLSA implementation

Aim at the shortages of the WLC in the LVS algorithm, we propose some new ideas to solve the shortages.

1) Clock timer sends a signal to the RS per 5-10 seconds, and requires RS to return load(s_i).
2) After getting the feedback signal sent by the LB, RSobtains CPU, memory and disk information by different pseudo file systems, gains the CPU usage*cpu(s_i)*, memory usage *memory(s_i)*, disk usage *disk(s_i)*, and sets the weight factor *(c, m, d, r)*to (0.5,0.4,0.1,0) by those information.

3) We set two counts of *c_count*and*m_count*,similar to the C++ object count, respectively, and these two counts are used to record $cpu(s_i)$ and $\mathrm{memory}(s_i)$ range. When $cpu(s_i), memery(s_i) \in [0.3, 0.7]$,*c_count* and*m_count*,are set as 0. If $\mathrm{cpu}(s_i), memery(s_i) < 0.3$, then the two counts minus 1. If $\mathrm{cpu}(s_i), memery(s_i) > 0.7$, then the two counts add 1. When*c_count*or *m_count* is -5, which means the RS is idle state, so the weight needs to be set greater and negative $\mathrm{load}(s_i)$ is returned; if *c_count* or *m_count* is 5(5 is regarded as boundary in this paper), which means it is in high load state, its weight needs go down, so positive $\mathrm{load}(s_i)$ is returned; while *c_count* and *m_count* are 0, which means it is in health state; while *c_count* and *m_count* are not in the above conditions, which means there is fluctuating, so $\mathrm{load}(s_i)$ return 1. *c_count*and *m_count* are designed to prevent the network fluctuation, to prevent frequently changing weight. Modifying the weight will increase LB burden, according to *c_count* and *m_count* we don't have to modify weight all the time.
4) According to getsockopt() to obtain the kernel state of the RSw_i, through the return value of $\mathrm{load}(s_i)$, if 1, then $w_i$will not modify, otherwise $new\_w_i = w_i - w_i \times load(s_i)$
The new weight is rewritten to the kernel IPVS scheduling by the function setsockopt().
5) LB selects the most suitable RS by WLC.

C. Analysis

Test procedure:

1) The initial weight of RS1 and RS2 is set to 6 and 10 respectively;
2) Apache JMeter is run on the client. The test is divided into 3 groups and the concurrent connections is 100, 400 and 700 respectively;
3) Each group uses WLC and DLSA individually to test for 10 minutes, and each group is measured 5 times to take average value;
4) Monitoring CPU usage and memory usage of RS; then we can get figure 5.

Table 1hardware configuration

	OS	CPU	RAM	IP
Director	OracleLinux 7.2	Intel-i5-2450M@2.5 GHz	4GB	192.168.1.50 VIP: 192.168.1.100
RS1	Ubuntu 16.0	Intel-P8700@2.53	4GB	192.168.1.2
RS2	Ubuntu 16.0	AMD@2.5GHz	2GB	192.168.1.3
Client	Windows10	Intel 53210@2.5GHz	4GB	192.168.1.4

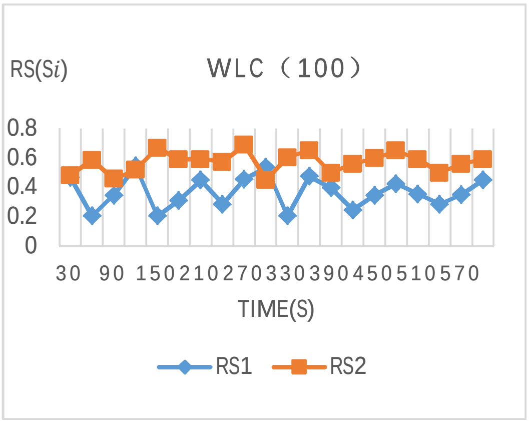

Figure a

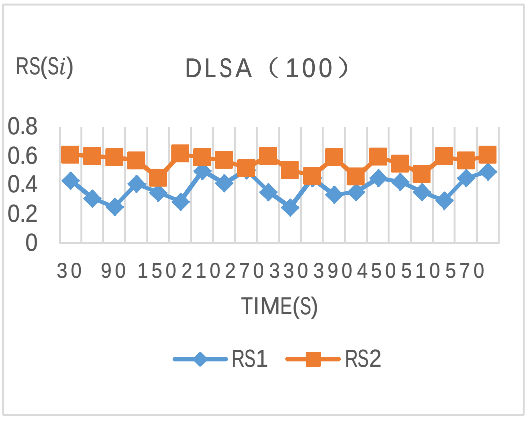

Figure b

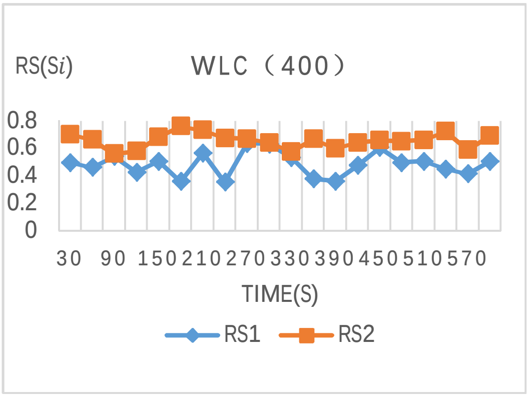

Figure c

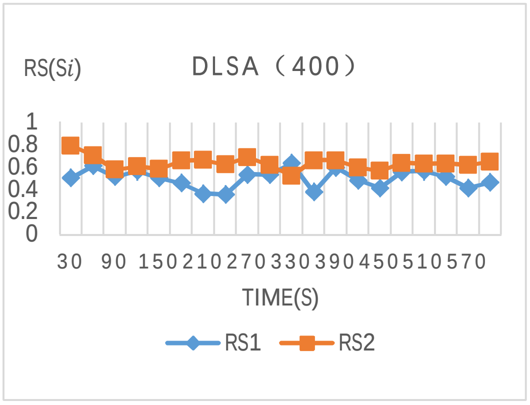

Figure d

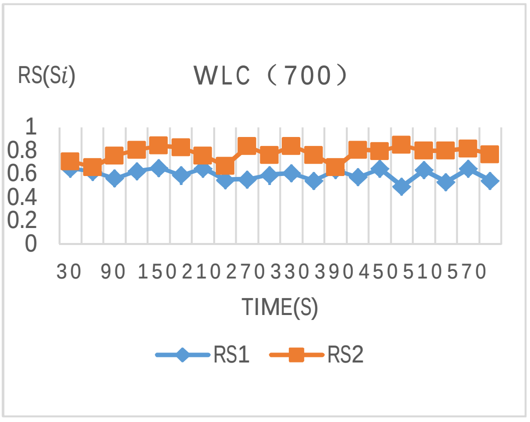

Figure e

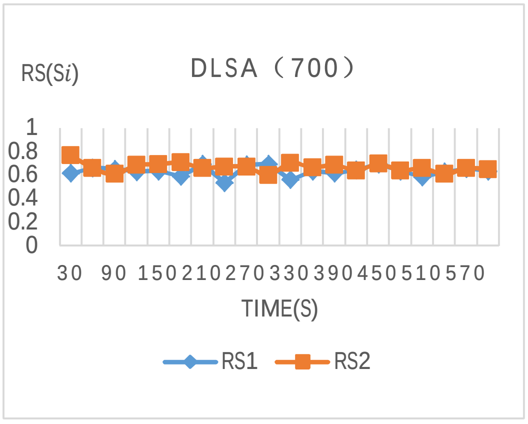

Figure f

Figure 5 OLD and NEW Algorithm

(a and b, c and d, e and f are in the same concurrent connections, but in different algorithm)

Compare figure a to figure b, the $RS(s_i)$ fluctuations were roughly the same, becauseLDSA's $s_i$weightdidn't changewhile RS(s_i) ∈ (0.3,0.7), and in this range it can be regarded as a health load.However, with the increase of the number of concurrent connections, $RS(s_i)$was over 0.7, shown in figure c, RS2 was waving between 0.6 and 0.8. In figure d, RS2 $RS(s_i)$ decreased significantly, because RS2's weight was reduced, so that the burden of RS2 was reduced; when the connections were 700, as figure e and figure f shown, figure e used WLC only and figure f used DLSA , in figure e, RS2's $RS(s_i)$was floating in 0.8, which was in over load, RS1's $RS(s_i)$was in health load, as the figure f shown RS1's $RS(s_i)$ and RS2's $RS(s_i)$ were almost the same in health load, compare RS2's $RS(s_i)$in figure e andfigure f, after using DLSA RS2's $RS(s_i)$was in health load. Through the comparison and analysis of the experimental data, with the increase of concurrent connections, the dynamic algorithm can adjust the weight of s_i to balance the load of RS, and make full use of the different load capacity of RS.

CONCLUSIONS

In this paper, we proposed a dynamic load balancing scheduling algorithm. When the usage of CPU and memory are in a certain range, which is $RS(s_i)$ in a certain range, and the RS is in health load, the proposed method doesn't modify the weight of RS and the burden of load balancing is reduced; By using count, the proposed method can solve a moment of rapid jitter situation and better reflect the real load of RS;Instead of modifying the weight of all servers, the proposed method only modifies the weight of specific RS. Experimental results showed that the dynamic algorithm can make better use of different performance of RS and the RS can work better under the condition of healthy load.
In our future work, we will investigate the situation that all RS are in overload conditions and make further improvement.

REFERENCES

[1] Zhang Wensong. Linux Server Cluster System. (A) LVS Project Introduction.

[2] Guo Z, Su M, Xu Y, et al. Improving the performance of load balancing in software-defined networks through load variance-based synchronization[J]. Computer Networks, 2014, 68(11):95-109.

[3] Lin P, Bi J, Hu H. ASIC:an architecture for scalable intra-domain control in OpenFlow[C]. International Conference on Future Internet Technologies. 2012:21-26.

[4] Wang R, Butnariu D, Rexford J. OpenFlow-based server load balancing gone wild[C]. Usenix Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. 2011:12-12.

[5] Zinke J, Schnor B. The impact of weights on the performance of Server Load Balancing systems[C] International Symposium on PERFORMANCE Evaluation of Computer and Telecommunication Systems. 2013:30-37.

[6] Liu Y, Fang Y K. Optimizing WLC scheduling algorithm of LVS[C] International Conference on Computer Application and System Modeling. IEEE, 2010:V6-585 - V6-588.

[7] Lin X, Du Z, Yang J. The simple optimization of WLC algorithm based on LVS cluster system[C] IEEE, International Conference on Cloud Computing and Intelligent Systems. IEEE, 2013.

[8] Shan R. Constructing the Expandability Web Service Based Linux[C] Seventh International Conference on Computational Intelligence and Security. IEEE, 2011:1451-1455.

[9] Rekaby A, Mohamed, Rizkaa A. The Improvement of Dynamic Load Balancing Strategy in Grid Computing[J]. 2012.

[10] Moniruzzaman A B M, Waliullah M, Rahman M S. A High Availability Clusters Model Combined with Load Balancing and Shared Storage Technologies for Web Servers[J]. Computer Science, 2014, 8.

[11] Wu K, Wang X, Chen H, et al. Improvement on LVS based IP network connection status synchronization[C]. IEEE International Conference on Software Engineering and Service Science. IEEE, 2015:746-749.

[12] Sharma S, Singh S, Sharma M. Performance analysis of load balancing algorithms[J].

Proceedings of World Academy of Science Engineering & Technolog, 2011:408 - 412.

[13] Chandra P K, Sahoo B. Performance Analysis of Load Balancing Algorithms for cluster of Video on Demand Servers[C]. Advance Computing Conference, 2009. IACC 2009. IEEE International. IEEE Xplore, 2009:408-412.

[14] Mesbahi M R, Hashemi M, Rahmani A M. Performance evaluation and analysis of load balancing algorithms in cloud computing environments[C]. Second International Conference on Web Research. IEEE, 2016:145-151.