Fuzzy Logic Load-Balancing Strategy Based on Software-Defined Networking

Guoyan Li(✉), Tianying Gao, Zhigang Zhang, and Yadong Chen

School of Computer and Information Engineering, Tianjin Chengjian University, Tianjin 300384, China
{ligy,gty,zzg,cyd}@tcu.edu.cn

Abstract. Traditional load balancing hardware is expensive and lacks scalability and flexibility. We propose a load balancing strategy based on fuzzy logic (LBSFL), which exploits the control and forwarding separation architecture characteristics of software-defined networking (SDN). First, the fuzzy membership function that affects the performance parameters of the server load is analyzed. Based on this, the load state of the virtual server is evaluated through fuzzy logic. Then the centralized control capability of SDN's controllers for the whole network is utilized to monitor virtual server information in real time and to schedule virtual server tasks. Individual servers can be hibernated or restarted, to save power or to increase performance as necessary. Finally, the dynamic balance between the overall load, performance and energy consumption is realized. Simulation experiments showed that the proposed strategy improves overall performance of the network, especially when dealing with communication-intensive tasks and using a high-latency network.

Keywords: SDN · OpenFlow protocol · Load balancing · Fuzzy logic

1 Introduction

In recent years, with the development of internet, e-commerce and big data technology, the scale, flow and user base of the internet has exploded. To meet the needs of network users, many internet service providers use load-balancing technology to provide high-quality and reliable service through the rational use of resources. However, traditional load-balancing devices are expensive and lack adequate scalability and flexibility.

Software-defined networking (SDN) is a clean slate project by a Stanford University study group [1] that proposes a new network architecture paradigm. Its core technology is the OpenFlow network protocol, which creates an interface between the device control plane and the data plane [2–7]. The resulting platform provides flexible network traffic control, innovation and application of the core network. In an SDN network, each switch has a flow meter, which is primarily a collection of process data streams for all actions, such as looking up and forward. The main flow table contains headers, counters and actions, three fields in which the actions field is represented, forwarding rules, and flow meters, which are updated intermittently. Because the SDN controller determines the

C. Li and S. Mao (Eds.): WiCON 2017, LNICST 230, pp. 471–482, 2018.
https://doi.org/10.1007/978-3-319-90802-1_42

traffic forwarding rules, the load balancing algorithm is in the controller, meaning that load balancing takes place at each link.

Several scholars have studied load balancing technologies based on SDN. In [8], the authors proposed an SDN-based publish/subscribe system that constructed and fine-tuned topic-connected overlays to disseminate events efficiently and non- redundantly, based on a global topology overview. Handigol [9] proposed a web traffic model founded on SDN. Based on the Openflow environment, Kaur [10] achieved network load balancing using a polling algorithm. Similarly, Zhang [11] determined the minimum number of network connections using a load balancing polling algorithm under an SDN framework. However, although [10, 11] applied traditional load balancing algorithms to the SDN architecture, they could not effectively reduce the server response time.

Shang [12] overcame this response time drawback by incorporating a middlebox, based on SDN architecture, to achieve load balancing by collecting server information. While this scheme effectively reduced server response time, it increased the complexity of the server architecture. In [6], a load balancing algorithm was proposed, based on server response times by using the advantage of SDN flexibility. Its lack of reliability depended on only server response times.

Fuzzy logic, where fuzzy sets are expressed with mathematical formulas, can solve many complicated problems which are not accurately represented by mathematical models [13]. In this task-scheduling model of SDN, the load status of each node is nonlinear and unpredictable. Given the technical limitations associated with collecting node information, extra time is required to obtain and report information. The information stored in this middleware can represent only past node load information, rather than the current load situation, because the system has an inherent delay. Considering the accuracy of the virtual server, the load state of the node is evaluated and the estimated quantity is more effectively expressed in fuzzy terms.

In the present paper, we propose a load balancing strategy based on fuzzy logic (LBSFL). Initially, we analyze the correlation among several parameters that impact load balancing and obtain the load of multiple virtual servers through a fuzzy logic algorithm. We then examine the virtual servers' load status in real time, select the lightest virtual server to handle the request, and, if necessary, set the sleep/restart policy of server. Finally, to verify the correctness and effectiveness of this load balancing algorithm, we constructed an SDN simulation platform. The experimental results show that the proposed strategy is stable and highly effective, resulting in faster and more consistent system response times.

2 System Architecture

The controller, or network operating system, is the heart of an SDN and is responsible for controlling and managing all of the OpenFlow switches [14–16]. We deployed the OpenDayLight SDN controller in our scheme. With the control plane and the data plane being separated in the OpenFlow environment, software configurations are customized through the controller to achieve effective load balancing. OpenFlow switches provide a unified interface and data forwarding function to the controller, so the controller unifies

control of the flow table of OpenFlow. The controller periodically obtains the running state of the virtual server and utilizes the load balancing algorithm to calculate the desired state of the server. To improve overall system performance, the load balancing algorithm migrates tasks from overloaded virtual servers to lightly-loaded virtual servers. If necessary, this strategy sleeps or restarts the virtual server to achieve load balance. We show the proposed system architecture in Fig. 1.

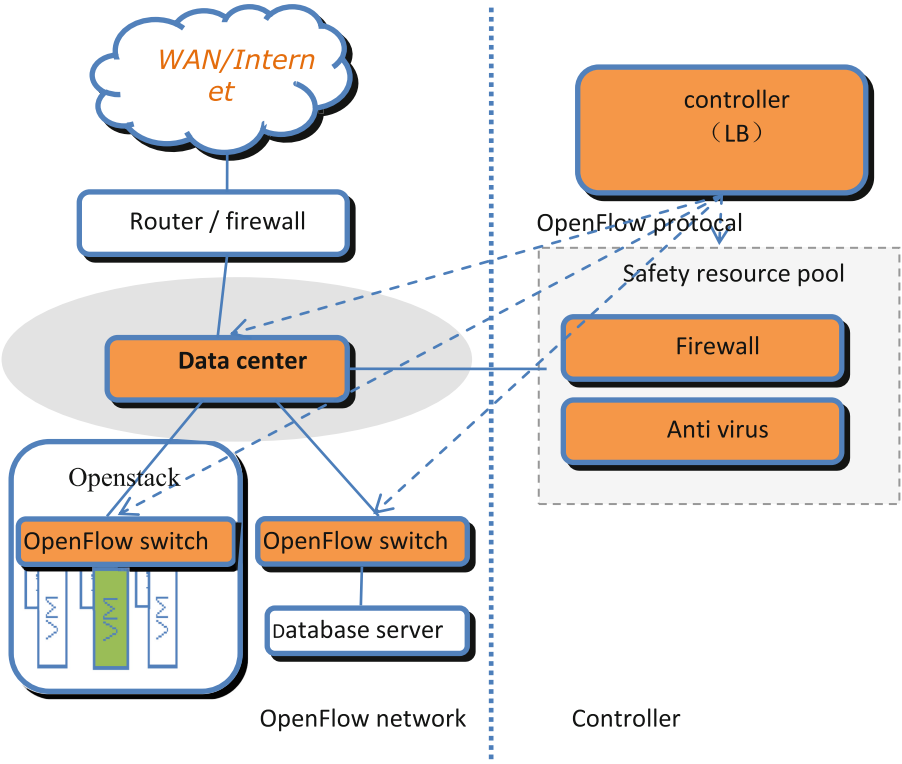

Fig. 1. System architecture

3 Load Balancing Algorithm Based on Fuzzy Logic

The current load condition of the virtual server can be calculated by using parameters such as CPU utilization and IO utilization. However, the load represented by these parameters is a fuzzy concept, meaning that there are no accurate mathematical models or control rules. When monitoring load condition, we considered the advantage of describing emergent problems and uncertainty problems in fuzzy mathematics, and introduced fuzzy-logic theory to solve load balance problems. The fuzzy logic system can be divided into three parts: fuzziness, fuzzy reasoning and solution.

3.1 Fuzziness

Fuzziness scale-transforms the input values to scope of the domain expressed by the fuzzy set. In addition, it determines the corresponding fuzzy rank sequence for each type of input value, e.g. {high, mid, low}, and determines a membership degree between the input parameters and fuzzy grade. Therefore, we need to select the membership function to map the membership relation between the variables and the sequence.

Many factors can be used to characterize the load condition of the server in the virtual server cluster environment, such as the frequency and utilization of the CPU, size and utilization of memory, response times of each server and the number of the current implementation of the process. The network request call and the returned result must be transmitted through the network, taking additional time, CPU and memory resources to complete. Thus, CPU utilization, memory utilization and I/O utilization are chosen as the load parameters for performance evaluation of server nodes. OpenFlow switches regularly submit virtual server load status to the controller. Load index value based on a given threshold is classified into three categories and allocated a value between 0 and 1. Three fuzzy sets are used to describe the load index value, and the fuzzy membership functions for each parameter are defined below in the following sections.

3.1.1 CPU Utilization

In the present study, we use the current server's CPU (C) utilization as the domain. We define fuzzy memberships $\mu_h(C)$, $\mu_m(C)$ and $\mu_l(C)$ as parts of the fuzzy subset of the current server CPU load, indicating membership in "highly-loaded," "moderately-loaded" and "lightly-loaded," respectively. Thus, $\mu_h(C)$, $\mu_m(C)$ and $\mu_l(C)$ are computed according to the following:

$$\mu_l(C) = \begin{cases} 1 & C \le 25\% \\ 1.5 - 2 * C & 25\% < C \le 75\% \\ 0 & C > 75\% \end{cases} \tag{1}$$

$$\mu_m(C) = \begin{cases} 0 & C \le 25\% \\ 4 * C - 1 & 25\% < C \le 50\% \\ 3 - 4 * C & 50\% < C \le 75\% \\ 0 & C > 75\% \end{cases} \tag{2}$$

$$\mu_h(C) = \begin{cases} 0 & C \le 25\% \\ 2 * C - 0.5 & 25\% < C \le 75\% \\ 1 & C > 75\% \end{cases} \tag{3}$$

3.1.2 Memory Utilization

We use the current server's memory (M) utilization as the domain. We define fuzzy memberships $\mu_h(M)$, $\mu_m(M)$ and $\mu_l(M)$ as parts of the fuzzy subset of the current server

memory load, indicating membership in "highly-loaded," "moderately-loaded" and "lightly-loaded," respectively. Thus, are computed according to the following: $\mu_h(M)$, $\mu_m(M)$ and $\mu_l(M)$ are computed according to the following:

$$\mu_l(M) = \begin{cases} 1 & M \leq 25\% \\ 1.5 - 2 * M & 25\% < M \leq 75\% \\ 0 & M > 75\% \end{cases} \tag{4}$$

$$\mu_m(M) = \begin{cases} 0 & M \leq 25\% \\ 4 * M - 1 & 25\% < M \leq 50\% \\ 3 - 4 * M & 50\% < M \leq 75\% \\ 0 & M > 75\% \end{cases} \tag{5}$$

$$\mu_h(M) = \begin{cases} 0 & M \leq 25\% \\ 2 * M - 0.5 & 25\% < M \leq 75\% \\ 1 & M > 75\% \end{cases} \tag{6}$$

3.1.3 I/O Utilization

We used the current server's I/O (IO) utilization as the domain. We define fuzzy memberships $\mu_h(IO)$, $\mu_m(IO)$ and $\mu_l(IO)$ as parts of the fuzzy subset of the current server I/O load, indicating membership in "highly-loaded," "moderately-loaded" and "lightly-loaded," respectively. Thus, $\mu_h(IO)$, $\mu_m(IO)$ and $\mu_l(IO)$ are computed according to the following:

$$\mu_l(IO) = \begin{cases} 1 & IO \leq 30\% \\ 1.75 - 2.5 * M & 30\% < IO \leq 70\% \\ 0 & IO > 70\% \end{cases} \tag{7}$$

$$\mu_m(IO) = \begin{cases} 0 & IO \leq 30\% \\ 5 * IO - 1.5 & 30\% < IO \leq 50\% \\ 3.5 - 5 * IO & 50\% < IO \leq 70\% \\ 0 & IO > 70\% \end{cases} \tag{8}$$

$$\mu_h(IO) = \begin{cases} 0 & IO \leq 30\% \\ 2.5 * IO - 0.75 & 30\% < IO \leq 70\% \\ 1 & IO > 70\% \end{cases} \tag{9}$$

After fuzzy processing of the four input variables by their respective membership functions, we comprehensively evaluate the load status of the virtual servers. The only output of the fuzzy logic inference system is the probability that a request should be

allocated to a virtual server, indicated by R. The set of factors is taken to be $E = \{C, M, IO\}$ and the collection of comments as {high, medium, low} when comprehensive fuzzy evaluation is used for internet quality of service.

3.2 Fuzzy Reasoning

Fuzzy reasoning is the core of the fuzzy controller, based on the relation of the fuzzy logic and the rule of inference. It provides the ability to simulate based on the fuzzy concept. The fuzzy control rules database is the most important component of fuzzy logic inference. The classical fuzzy rules are composed of numerical or linguistic variables. We construct a fuzzy matrix to represent memberships of the input parameters in each fuzzy subset, and perform comprehensive fuzzy evaluation according to the following steps:

(1) In a parallel manner, we establish a comprehensive fuzzy evaluation model to evaluate a single factor of each index. The output of the fuzzy logic is the possibility of assigning the network request to the virtual server in the case of overload. The evaluation results of each factor have the following fuzzy vectors:

$$R1 = [u_h(C), u_m(C), u_l(C)],$$
$$R2 = [u_h(M), u_m(M), u_l(M)],$$
$$R3 = [u_h(IO), u_m(IO), u_l(IO)],$$

where the three vectors constitute a fuzzy matrix from the factor set to the comment set, $R = [R1, R2, R3]$.

(2) We determine weight vector $P = [p1, p2, p3]$, where $p1$, $p2$ and $p3$ represent the importance of CPU, memory and I/O, respectively, in u, and $p1 + p2 + p3 = 1$.

(3) We define a fuzzy transformation $Q = P \cdot R$, where Q is the evaluation result of each virtual server in the comment collection of fuzzy vector $F = (L, M, H)$. Three of these components represent the extent to which the virtualserver is a candidate.

3.3 Defuzzification

Because the output of fuzzy inference is fuzzy vector F, it is necessary to solve the model to obtain the exact output value. We adopt the classical area center method for defuzzification. It takes the centroid of the membership function of each fuzzy rank as the exact output of the fuzzy grade. The corresponding centroid of the fuzzy grade L is 0.15, the corresponding centroid of the fuzzy grade M is 0.5, and the corresponding centroid of the fuzzy grade H is 0.85. Finally, we use the following formula to calculate the exact value of the output of the fuzzy logic:

$$Fuzzy\_out = \frac{\sum_i w_i M_i}{\sum_i M_i}, \tag{10}$$

where, M_i represents the centroid of each output fuzzy level, and wiindicates the weight of the corresponding output fuzzy level for M_i.

3.4 Load Balancing Strategy Based on SDN

Our proposed LBSFL is based on SDN. This strategy attains load balance among multiple servers while saving energy. When the overall load is low, the server with the lightest load is set to sleep; when the overall load is high, the server is restarted. The following is the specific implementation strategy:

(1) Initialize the OpenFlow network. The system responds to the web requests through a classic polling algorithm. The load balancing module obtains server status information through SDN switches and calculates the load of the server through a classic polling algorithm, then calculates the load balancing parameter of the OpenFlow network.

(2) Set the load balance adjustment threshold. The minimum threshold of the server's average load is assumed to be 0.2 and the maximum threshold to be 0.8. The adjustment threshold for load balancing will be obtained experimentally. The fuzzy logic algorithm adjusts the load balance. When δ is greater than the threshold value, the current web request is forwarded to the server with the lowest load. When δ is lower than the threshold and F_{avg} is lower than 0.2, the server load is idle and the server migration strategy [5] is executed to sleep the server with the smallest load. When δ is lower than the threshold and the F_{avg} is greater than 0.8, the server load is saturated and the server migration strategy [5] is executed to restart a virtual server. When δ is lower than the threshold and the F_{avg} is between 0.2 and 0.8, it shows that the current load of the virtual server is balanced, and the status of the server continues to be monitored. The execution flow chart of the strategy is shown in Fig. 2:

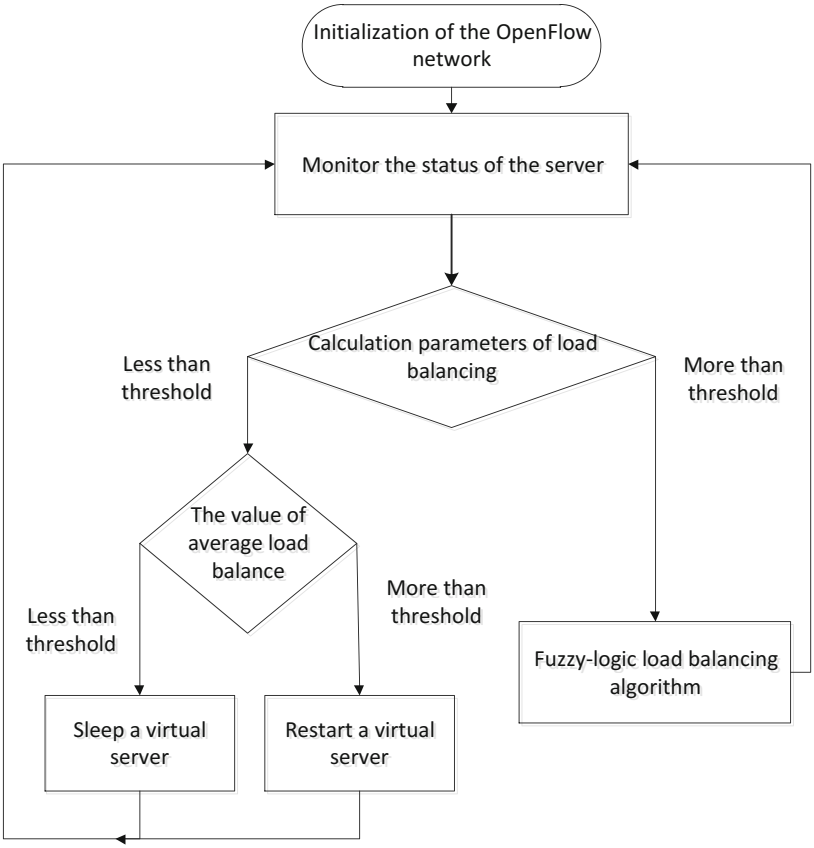

Fig. 2. Proposed load balancing strategy based on fuzzy logic

4 Experimental Analysis

To review the performance of the LBSFL based on SDN, we used the following experimental environment: Ubuntu version 11.04, running on an Intel Pentium E2180 dual-core 4-GHz processor. We used open source software Mininet 2.0 to build the OpenFlow network, the H3C5510_34C switch that supports the OpenFlow1.3 protocol, and a Java-based OpenDayLight controller to implement the load balancing strategy. To test the performance of the algorithm, we installed iperf, a network performance testing tool, which generated traffic pressure in the Mininet environment.

The simulation testing system structure is shown in Fig. 3. Four virtual machines with identical configurations were assigned as web servers. Taking into account that frequently sleeping and restarting the server impacted the performance of the system, we set a minimum of four virtual servers.

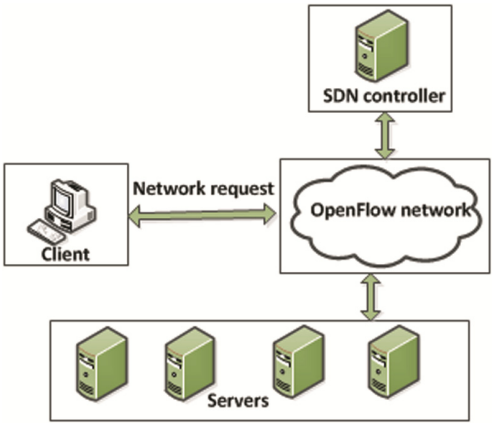

Fig. 3. Hardware simulation platform

We first evaluated the efficiency of LBSFL against traditional round-robin (WRR) and weighted least connections (WLC) schemes. The input data of the fuzzy system included CPU utilization, memory utilization and I/O utilization. The empirical values of the three parameters were 0.4, 0.3 and 0.3, respectively. In the test, the load was increased linearly for the first 10 min, adding 2000 connection requests per minute; followed by a linear reduction by 2000 connection requests per minute for the next 10 min. The total experimental time was 20 min. Samples were taken every 1 min, and the test was repeated five times. The average of these values was assumed to be the value of system response time at that time. The test results are shown in Fig. 4.

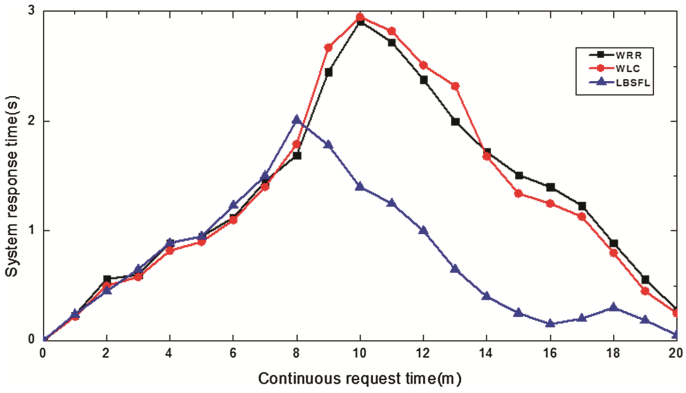

Fig. 4. Response time of system

At the same time, to achieve a more prominent load balancing effect in LBSFL, we also extracted the CPU, memory utilization, I/O average utilization rates of each server. Figures 5, 6 and 7 present the CPU, memory and I/O usage graphs of the four servers (h1, h2, h3, h4) under WRR, WLC and LBSFL schemes, respectively.

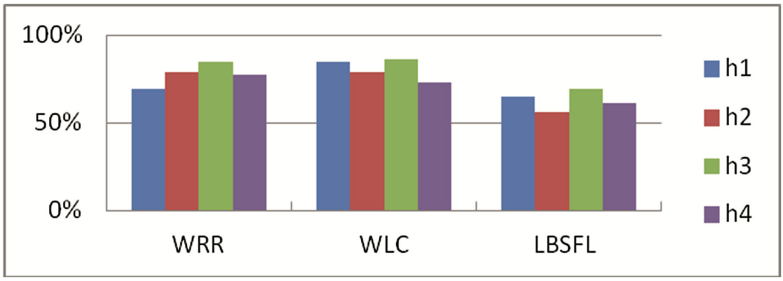

Fig. 5. CPU utilization

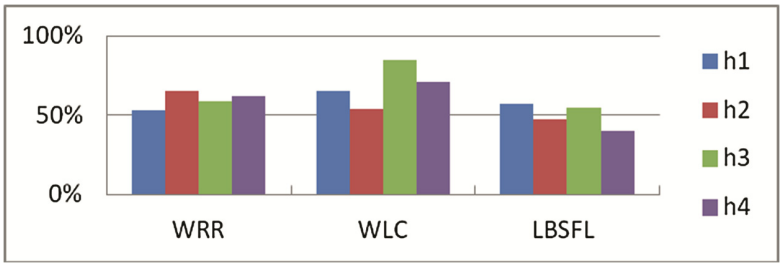

Fig. 6. Memory utilization

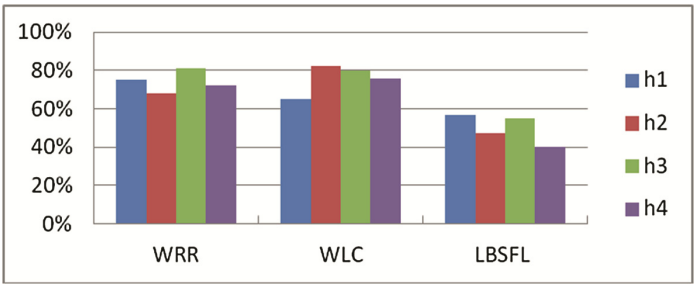

Fig. 7. I/O utilization

From the experimental results we can see:

(1) From Fig. 4, the average server response times for the three schemes, WRR, WLC and LBSFL were 1.434 s, 1.532 s and 0.83 s respectively. It is evident that the average server response times of the server in LBSFL was the lowest among the three schemes. This is because LBSFL always chooses the server characterized by a fuzzy logic algorithm to provide services to the users. Moreover, WRR and WLC do not consider the real-time status of the servers. From Figs. 5, 6 and 7, we also found that the load balancing effect of LBSFL is better than that of WRR and WLC.

(2) With the parameters shown in Fig. 4, the number of requests was small, the task management and scheduling was relatively simple, and there was not much

difference between the three strategies with respect to system response time. However, as the number of requests increased, task management and scheduling became more and more complex. The response time of LBSFL was shorter than those of WRR and WLC. With its simplified task management and scheduling, LBSFL provided more advantages than WRR and WLC.

(3) The LBSFL system response time curve shows two obvious wave peaks, the first at about 8 min. With increasing requests, the current load capacity of all virtual servers peaked. To reach system dynamic balance, the controller triggered the load scheduling mechanism to start a virtual server, after which the response time of the system appeared to decline and stabilize. The second peak was at about 18 min. With decreasing requests, the current server load continually reduced until the controller triggered the load scheduling mechanism to sleep a virtual server. After this, the system response time again appeared to decline and stabilize. Because of the sleep of a server, the system request processing ability was weakened, so there was a small wave crest, and with the task management and scheduling becoming simpler, the response time of the system appeared to decline. Server sleep and restart also resulted in system energy saving.

5 Summary

Traditional load balancing hardware is expensive and lacks adequate scalability and flexibility. We propose a load balancing strategy in SDN networks that successfully enhanced the load balancing effect and improved network resource utilization.

Acknowledgments. The author would thank the support from projects of the national "863Program" (NO. 2015BAF09B02-3); the natural science fund of Tianjin city (NO. 17JCQNJC00500); Tianjin education science planning project of 13th five-year plan (HE3045); the fundamental research fund for the university in Tianjin, Tianjin Chengjian university (2016CJ12) and the fund of Tianjin Education Committee (20110813).

References

1. McKeown, N.: Software-defined networking. INFOCOM Keynote Talk **17**(2), 30–32 (2009)
2. McKeown, N., Anderson, T., Balakrishnan, H., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
3. Namal, S., Ahmad, I., Gurtov, A., et al.: SDN based inter-technology load balancing leveraged by flow admission control. In: 2013 IEEE SDN for Future Networks and Services (SDN4FNS), pp. 1–5. IEEE (2013)
4. Marconett, D., Liu, L., Yoo, S.J.B.: Optical FlowBroker: load-balancing in software-defined multi-domain optical networks. In: Optical Fiber Communication Conference. Optical Society of America (2014): W2A. 44
5. Muñoz, P., Barco, R., de la Bandera, I.: Load balancing and handover joint optimization in LTE networks using fuzzy logic and reinforcement learning. Comput. Netw. **76**, 112–125 (2015)

6. Zhong, H., Fang, Y., Cui, J.: LBBSRT: an efficient SDN load balancing scheme based on server response time. Future Gener. Comput. Syst. **68**, 183–190 (2017)
7. Gandhi, R., Liu, H.H., Hu, Y.C., et al.: Duet: cloud scale load balancing with hardware and software. ACM SIGCOMM Comput. Commun. Rev. **44**(4), 27–38 (2015)
8. Wang, Y., Zhang, Y., Chen, J.: SDNPS: a load-balanced topic-based publish/subscribe system in software-defined networking. Appl. Sci. **6**(4), 91 (2016)
9. Handigol, N., Seetharaman, S., Flajslik, M., et al.: Plug-n-serve: load-balancing web traffic using OpenFlow. ACM Sigcomm Demo **4**(5), 6 (2009)
10. Kaur, S., Singh, J., Kumar, K., et al.: Round-robin based load balancing in software defined networking. In: 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2136–2139. IEEE (2015)
11. Zhang, H., Guo, X.: SDN-based load balancing strategy for server cluster. In: 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 662–667. IEEE (2014)
12. Shang, Z., Chen, W., Ma, Q., et al.: Design and implementation of server cluster dynamic load balancing based on OpenFlow. In: 2013 International Joint Conference on Awareness Science and Technology and Ubi-Media Computing (iCAST-UMEDIA), pp. 691–697. IEEE (2013)
13. Pakzad, F., Portmann, M., Tan, W.L., et al.: Efficient topology discovery in OpenFlow based software defined networks. Comput. Commun. **77**, 52–61 (2016)
14. Scott-Hayward, S.: Design and deployment of secure, robust, and resilient SDN controllers. In: 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1–5. IEEE (2015)
15. Hoang, D.B., Pham, M.: On software-defined networking and the design of SDN controllers. In: 2015 6th International Conference on the Network of the Future (NOF), pp. 1–3. IEEE (2015)
16. Kang, S.B., Kwon, G.I.: Load balancing strategy of SDN controller based on genetic algorithm. Mech. Eng. **129**, 219–222 (2016)