Future Generation Computer Systems 80 (2018) 409–416

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Reprint of "LBBSRT: An efficient SDN load balancing scheme based on server response time"☆

Hong Zhong, Yaming Fang, Jie Cui *

School of Computer Science and Technology, Anhui University, Hefei, 230039, China

ARTICLE INFO

Article history:
Available online 13 November 2017

Keywords:
SDN
OpenFlow
Load balancing
Server response time

ABSTRACT

The response time is the most important factor determining user experiences in the service provision model involving server clusters. However, traditional server cluster load balancing scheme are limited by the hardware conditions, and cannot completely exploit the server response times for load balancing. In order to effectively resolve the traditional load balancing schemes, we propose a load balancing scheme based on server response times by using the advantage of SDN flexibility, named LBBSRT. Using the real-time response time of each server measured by the controller for load balancing, we process user requests by obtaining an evenly balanced server loads. Simulation experiments show that our scheme exhibits a better load balancing effect and process requests with a minimum average server response times. In addition, our scheme is easy to implement, and exhibits good scalability and low cost characteristics.

1. Introduction

Achieving optimum load balancing is of significant importance whilst combating network overhead issues in any distributed processing architectures. Service availability is paramount in measuring end user satisfaction [1], which is heavily impacted by the level of achievable load balancing among the process clusters. In general, a well-balanced load in the network helps to optimize the utilization of the available resource by the ways of maximizing the throughput, minimizing the response time, and avoiding overloading resources in the network [2]. For the purposes of alleviating heavy-traffic network flux and reducing the risk of single server becoming the main overhead contributor, many datacentres adopt dedicated hardware resources to achieve load balancing whilst supporting a large number of users [3]. However, the increasing costs and technical complications in the deployment of such hardware systems often require human intervention to ensure consistent functioning of such strategies [4].

Software-Defined networking (SDN) is one of the notable forms of computer networking [5,6], facilitating a simple and conveniently maneuverable network flow control method requiring minimal investment costs whilst availing maximum benefits for a massive number of users. SDN controls the data transportation by deploying the network switches as a software implementation, whereby a flow table lookup operation will be carried out whenever a data flow arrives at the switches. Flow tables [7] ([Header: Counters: Actions]) are widely used in SDN. The headers and counters of the flow table are updated accordingly whenever actions relevant to flow changes are imposed. During this update process, the header information is usually recorded onto the database and the OpenFlow switches process the data flow in accordance with the header records. Based on the SDN model with a centralized controller, an OpenFlow switch [8] is designed with different rules to control the network traffic using the header records. Balancing the network load at the software tier is now practically realizable using the SDN facilitated flow control system. To this end, Handigol [9] proposed plug, a load balancing model based on SDN. Based on the Openflow environment, Kaur [10] achieved network load balancing using polling algorithm. Further, Zhang [11] achieved the minimum number of connections in the network using the polling algorithm of load balancing under the SDN framework. Shang [12] incorporated a middlebox based on the SDN architecture to achieve load balancing by collecting the server information. Despite the existing implementations of SDN to resolve high cost and poor flexibility issues in achieving effective load balancing, notable drawbacks are still prevalent in the aforementioned schemes. To add

DOI of original article: http://dx.doi.org/10.1016/j.future.2016.10.001.

☆ This article is a reprint of a previously published article. A publishers' error resulted in this article appearing in the wrong issue. The article is reprinted here for the reader's convenience and for the continuity of the special issue. For citation purposes, please use the original publication details; H. Zhong et al. / Future Generation Computer Systems 68 (2017) 183–190.

* Corresponding author.
E-mail address: cuijie@mail.ustc.edu.cn (J. Cui).

http://dx.doi.org/10.1016/j.future.2017.11.012
0167-739X/

a few, Kaur and Zhang [10,11] applied traditional load balancing algorithms to the SDN architecture, and so the two schemes cannot effectively reduce the server response time. Though Shang [12] can effectively reduce the server response time, this scheme relies on the server information which increases the complexities of the server architecture.

This paper proposes a new method of load balancing in SDN networks with the motivation of enhancing the load balancing effect by reducing the server response time. In this paper, the server response time is defined by the interval that begins from accepting user requests to responding to user requests for server. If servers in a server cluster have several similar performances and provide the same service, then for each server, the higher the load is, the longer the response time is. Correspondingly, the longer the response time is, the higher the corresponding load is. Therefore we propose a load balancing scheme based upon server response time. It can solve the problem of the load balancing in the server cluster based on the server response time. Our proposed approach effectively overcomes the drawbacks of the traditional methods, including high cost, low reliability and poor extensibility. The contributions of the paper include:

- An effective load balancing scheme based on SDN architecture, using the real-time response time of each server measured by an SDN controller.
- Realizing the potential implementation of our scheme by incorporating a floodlight controller module in the scheme.
- Proving the effectiveness of our proposed scheme by evaluating the response time and resource utilization metrics against the traditional schemes.

The rest of the paper is organized as follows: Section 2 reviews the existing traditional load balancing schemes and introduces the background of SDN. Section 3 details the design of our proposed scheme, LBBSRT (Load Balancing by Server Response Time). The performance evaluation of LBBSRT is presented in Section 4. Section 5 concludes the paper.

2. Related works

2.1. The traditional load balancing scheme

The traditional load balancing schemes are categorized into four major types [12] such as based on the client, based on the middle layer, based on the DNS, and based on the transport layer.

In the load balancing scheme based on the client side, clients primarily collect every server running parameters from the server clusters either periodically or non-periodically, and send a request to different servers to achieve load balancing. Although this method can achieve a certain degree of load balancing, it loses grip in large-scale server clusters due to a high degree of coupling between client and the server.

The most common method of load balancing based on the middle layer uses the reverse proxy server. This proxy server requests the back-end servers to balance the load in the server clusters, and also sends the cached data directly back to the clients. In some sense, this acceleration mode accelerates the access speed of the static pages. The reverse proxy server can combine the load balancing technology with the caching technology to enhance the performance. However, developing a reverse proxy for each service is often a substantial demand. Usually reverse proxy servers should maintain two connections: one connects to the client, and another connects to server clusters. With the increasing number of concurrent connections, the reverse proxy server itself will become the bottleneck of the system.

Load balancing scheme [13] based on DNS configures a single domain for multiple IP addresses in the server clusters. When a client requests a domain name service, the domain name server uses the method of polling to allocate different servers for different clients, so as to achieve the goal of load balancing. Load balancing based on DNS is simple and convenient, but susceptible of several issues. For instance, DNS server is not aware of the difference among the servers, and so cannot reflect the current state of the servers. But it is possible to send a lightweight access to the idle servers while there is an increasing server load on currently utilized servers.

The load balancing scheme based on transport layer sends the client's requests directly to the load balancing server. The load balancing server will then forward the requests to the back-end servers according to policies such as LVS [14], VS/NAT, etc. Although this approach balances the server load, it often demands additional hardware resources and thus proven costly.

Due to such hardware limitations, the load balancing among the server clusters is not only complex but also expensive, and is susceptible of poor scalability. The emergence of SDN architecture facilitates effective strategies to counteract such load balancing issues.

2.2. The SDN architecture

SDN encompasses a decoupling layered architecture which segregates the data level access from the control level access [15]. The control level includes the network operating system and applications, while the data level incorporates standard protocol supports for the network hardware equipment. The SDN process architecture can be categorized into a three-layer system structure [16] including application, control and data. Rather than a simple extension of the traditional network architecture, this three-layer structure of SDN is a disruptive innovation [17]. The centralized network control of SDN is effective in resolving the susceptible issues of the traditional network devices. Moreover, SDN supports independent programming of the network control system in the management mode, and instantaneous upgrading of which can also be achieved by the network application interface. The application layer of the SDN architecture provides users with a rich variety of API interfaces, which can be used to develop our own development module with desired functionalities [18] based on individual business needs. OpenFlow, one of the SDN mainstream southbound interface protocol [19], is one of the fundamental elements required for building SDN solutions. OpenFlow is the first standard communication protocol defined between the control layer and the infrastructure layer in SDN architecture [20,21]. OpenFlow uses the concept of flows to identify network traffic based on matching rules that can either be statically or dynamically programmed by the SDN control software. Switches are responsible for applying appropriate actions on the arriving packets and update every action on their flow table entry. Using such entries in the flow tables, switches simple forward packets without considering to construct or modify the flow tables. The SDN controller will create and install a rule in the flow tables for the necessary packets. Also, the SDN controller can manage the flow tables of all the switches simultaneously. OpenFlow-based SDN architectures provide extremely granular control privileges, by the way of enabling the network to react to real-time changes of both the application and the service users [22]. OpenFlow-based SDN technologies can enhance the network bandwidth capabilities, can effectively handle the dynamicity of the applications and can significantly reduce the operation and management complexity [23]. The distinctive and innovative features of SDN support the development and testing of novel forwarding strategies and network protocols effectively. There are three types of message in OpenFlow: Controller-to-Switch, Asynchronous and Symmetric. Each message has multiple sub message types, and two kinds of them are used

in this paper which are Packet_out and Packet_in respectively. Packet_out is the message that the controller sends to the switch. In many cases, the controller needs to send packets to data plane. These packets it can be sent to the switch in the way of being encapsulated as the Packet_out message. Packet_in is an important message type in OpenFlow. If the data packet received by the switch is not matched with the flow table or the controller is given as a specified port in the term of matched flow table as forwarding action, the switch will encapsulate Packet_in message and send this data packet to the controller [24].

2.3. The SDN controller

The controller or network operating system is the heart of the SDN, which is responsible for controlling and managing all the OpenFlow switches [25]. Some of the SDN controllers used widely in academia and industry [26] are summarized as follows:

NOX is the first SDN controller, which is developed by Nicira [27]. It is the foundation for many research projects during early SDN era.

Floodlight is created by Big Switch Network switches, and it is based on Beacon controller.

Ryu is originated from NTT in Japan. Ryu is based python and it is simple and easy to use.

OpenContrail is an open source controller written in C++ with a REST NBI, and it offers integration plug-ins for cloud services such as Amazon, Openstack, and Cloudstack.

OpenDaylight and floodlight is based JAVA and has two major technical characteristics. One is the use of OSGi architecture and the other is the introduction of SAL.

We deployed the Floodlight SDN controller in our scheme.

3. The design and implementation of LBBSRT

Usually, obtaining the response times of each server from a pool of servers is a tedious process using the traditional network equipment. Such traditional schemes do not incorporate the server response times whilst balancing the server loads. Instead, they simply ping the servers for obtaining their reply time. Such strategies may not obtain the actual reply time of the server and so are often not accurate. The segregation of the control plane and data plane in the SDN facilitates obtaining the server response times accurately and effectively. Now, the traffic load can be balanced among the servers utilizing the data obtained from the control plane. In this paper, we propose a novel approach of realizing effective load balancing using the server response times under the SDN architecture.

3.1. System model

We develop our system model in the OpenFlow environment, as shown in Fig. 1. The system is composed of three major parts described as follows. (1) Terminal equipment: server and client. Client is the user's machine, usually of a large quantity. Any number of servers belonging to a single server pool will be hosted on a single virtual IP address. For instance, there different servers in the same server pool with IP addresses 10.0.0.1, 10.0.0.2 and 10.0.0.3 respectively, will be referred with a common IP address as 10.0.0.1 for service provision. Users can obtain all the services provided by the server pool by accessing the IP 10.0.0.1, controlled by the control plane. All such control operations are transparent to the users. (2) Service network, which consists of OpenFlow switches (used to connect users with the server) along with the other common switches. Switches that connect to the server and the controller must support the OpenFlow protocol, which is not required for the datacentre switches and other external

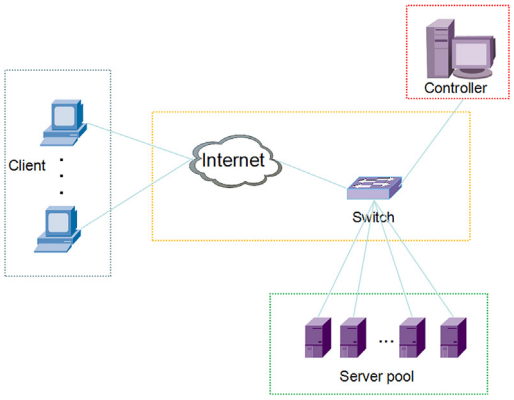


Fig. 1. System model.

network switches and routers. (3) Decision platform contains the SDN controller, including several modules for facilitating different functionalities. Decision platform includes the control plane of SDN, mainly composed one or more SDN controllers depending on the number of active switches. Some of the SDN controllers used in practice today include NOX, POX Floodlight, etc. Some of them are open source and some are commercial, the choice of the SDN controller is usually based on the server pools and the configuration switches. In this paper, we use the floodlight controller which is open source.

3.2. Scheme description

Traditional load balancing schemes mainly used Random, Round Robin, and Least Connections due to the limitations of the hardware equipment. Such load balancing algorithms can only be used in simple scenarios, and are not effective in a heterogeneous pool of servers. With the control plane and the data plane being separated in the OpenFlow environment, software configurations can be customized through the controller to achieve effective load balancing. Such strategy can be applied to existing types of servers for a better load balancing effect. In this paper, we design a load balancing algorithm in the OpenFlow environment, in which use the controller to obtain the real-time response times of each server, which is then utilized to choose the server with minimum or most stable response time. Because the server response time directly reflects the server load capability, selecting server based on the response times helps to send user requests to the servers operating under minimum server load to extract maximum performance. The concrete implementation process is explained as follows.

3.2.1. Real-time measurement of server response time

This section describes our strategy of obtaining the server response time.

Step 1: Send Packet_out message to the switches. Once the system is initiated, the controller sends multiple Packet_out messages to the switches with time interval t and records the transmission time. The number of messages sent out is the same as the number of servers in the resource pool. Each Packet_out message carries data packets whose source addresses are the controller IP address, while destination address is assigned with each server IP.

Step 2: Switches handles the Packet_out message. When the OpenFlow switch receives the Packet_out message sent by the controller, the switch will parse the data packets and sends these data packets to each server.

Step 3: The server sends the reply message, from which the controller obtains the server response time. After receiving the data packets sent from the controller, each server runs a simulation

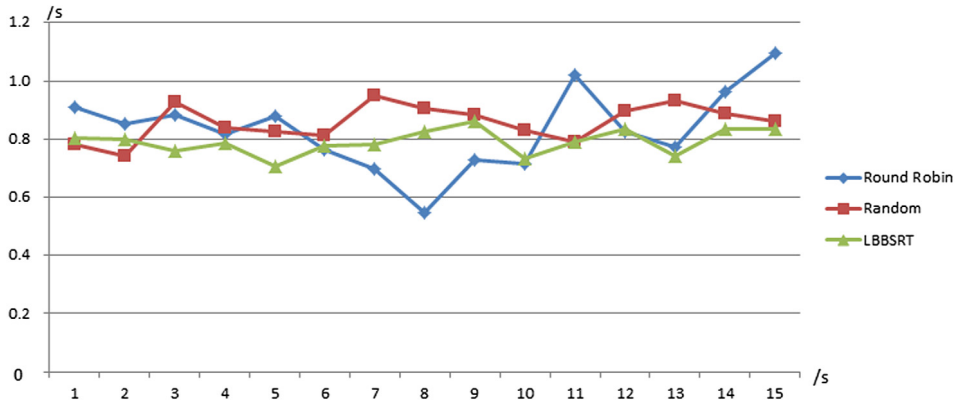


Fig. 2. The server's response time in the first case.

of the client request and then sends a data packet with the source address assigned as the server IP and the destination address assigned as the controller IP. Since this is a new event in the flow table, the switch needs to send Packet_in message to the controller. Now the controller obtains the arrival time of each server data packet by parsing the Packet_in message. As a result, the controller obtains the response time of each server, and updates the database accordingly.

Step 4: Repeat Step 1, step 2, and step 3.

The algorithm is described as follows (see Algorithm1):

Once the server response time is obtained by the controller, user requests are processed by balancing the serer load, as described in the following sections.

3.2.2. The process of user request

This section explains our strategy of load balancing using the server response time.

Step 1: The controller handles the ARP messages. Users will send ARP_broadcast message to the switches upon the first access since the switches do not contain flow tables for processing ARP in order to send Packet_in message to the controller. The controller will then construct a virtual MAC address, based on which, it sends a Packet_out message to the switches, and the switch sends the ARP reply packet to the user terminal.

Step 2: The controller handles the user request. After users receiving an ARP reply packet, they initiate a service request to the server, and the request process is similar to Step 1. The controller will also receive this user's request service packet, and select the server with minimum or stable response time according to obtained server data. The selection process is explained as follows.

① Using formula (1) and (2), we obtain the maximum value $T_{\max}$ and minimum value $T_{\min}$ of the server response times for the current server cluster.

$$T_{\max} = \mathrm{Max}\{T_{1,0}, T_{2,0}, T_{3,0}, \ldots, T_{n,0}\} \quad (1)$$

$$T_{\min} = \mathrm{Min}\{T_{1,0}, T_{2,0}, T_{3,0}, \ldots, T_{n,0}\} \quad (2)$$

where, $T_{i,j}$ is the response time of the ith server in the j time interval before the current time where each interval is t, and $T_{i,0}$ is the current response time of the ith server.

② According to the obtained $T_{\min}$ and $T_{\max}$, we calculate $|T_{\min} - T_{\max}|$. If $|T_{\min} - T_{\max}| < \lambda$, execute, ③ otherwise execute. ④ λ represents that the servers are of similar loads when the response time difference is in the range of λ.

③ Now we obtain the standard deviation of each server's response time by calculating the standard deviation of m historical data of each server's response time using formula 3. Then, we will select the server with minimum standard deviation $S_{\min}$, then execute⑤

$$S_i = \sqrt{(T_{i,0} - \bar{T})^2 + (T_{i,1} - \bar{T})^2 + \cdots + (T_{i,m-1} - \bar{T})^2} \quad (3)$$

where, $\bar{T}$ represents the average value of m historical data and S_i represents the standard deviation of historical data for the ith server.

④ Now, we select the server with $T_{\min}$.

⑤ The controller will send the flow table to the switches according to the selected server, then user requests will be sent to selected server.

The algorithm is described as follows (see Algorithm2):

4. Experiment result and performance analysis

In our experimental setup, the virtual switch is created by Open vSwitch. The floodlight is chosen as the SDN controller. Due to the floodlight is a free open source, and add and delete modules can be arbitrary used, so it provides much more convenience for our test. Three virtual machines with identical configurations are assigned as servers to provide web services. In this experiment, we let 30 clients to access to the server. Moreover, the access frequencies of different clients are usually not the same in the real world. So we set up two different access frequencies. One is that each client sends a continuous HTTP request to the server, and another is that each client sends a request every two seconds. WordPress is used to build a blog on three servers. And then in the following three cases, we start these clients in 2 min randomly. On another virtual machine, we use 30 clients to access the server under three different situations: (1) 12 clients send a service request to the server continuously, while 18 clients send a request discontinuously; (2) 15 clients send a service request to the server continuously, while 15 clients send a request discontinuously; (3) 18 clients send a service request to the server continuously, while 12 clients send a request discontinuously. In the controller, we add two modules, one is used to measure the server response time, and another is used to process the user requests and set flow table. Figs. 2, 3, 4 illustrate the server's response time for a period after the concurrent accesses reach the maximum value under the three situations respectively. There are a lot of servers in a server resource pool, in our experiment, we select the average response time of the servers as overall response time. The main purpose of load balancing is to avoid significant system load deviation in the long time of running so as to enhance the system efficiency and achieve a better user experience. Obviously, the effect of load balancing depends on the load and response time of the server. So we choose these two parameters to compare with other schemes. In this paper, we first evaluate the efficiency of our LBBSRT scheme against the traditional round-robin and random schemes.

In this case, the average server's response time of the three schemes, Round Robin, Random and LBBSRT are 0.831 s, 0.857 s, and 0.791 s respectively.

Algorithm1: Measure server's response time

```
While system startup do
    If current time % t == 0 do
        Send Packet_out to switches and record sending time T_send ;
    End if
    If receive a Packet_in message then
        Parse message;
        If the source address of the received packet is the server, the destination
        address is the controller then
            Record the time T_arrive of received message;
            Calculate the response time by the formula T_response = T_arrive - T_send ;
            Store response time;
        Else
            Send to other modules;
        End if
    End if
End while
```

Algorithm2: Handle user requests

```
If the controller receives a Packet_in message then
    Parse message;
    If data package for ARP request then
        Controller sends Packet_out message reply ARP;
    End if
    If packet for user service request then
        Based the formula (1), (2) obtain the current server in the cluster server response time
        maximum and minimum value, T_max , T_min ;
        If |T_min - T_max| < λ then
            By the formula (3) select the standard deviation of the minimum value
            corresponding to the server;
        Else
            Select the T_min corresponding to the server;
        End if
    Else
        Send to other modules;
    End if
End if
```

The average server's response time of the three schemes, Round Robin, Random and LBBSRT are 0.954 s, 0.996 s and 0.892 s respectively.

The average server's response time of the three schemes, Round Robin, Random and LBBSRT are 1.236 s, 1.366 s and 1.119 s respectively. It is evident that the average server response times of the server in LBBSRT is the minimum among the three schemes under the above three scenarios. Also the polylines fluctuation of our LBBSRT scheme is stable and minimum, as shown in Figs. 2–4. This is because our scheme always chooses the server characterized with minimum response time in order to provide services to the users. Moreover, round robin algorithm and random algorithm does not consider the real-time status of the servers. As a result, the load among all the server is balanced evenly and the server response time is reduced to minimum.

LBBSRT has significant advantages compared to other schemes in terms of the overall response times. In order to achieve a more prominent load balancing effect in LBBSRT, we also extract the CPU and memory utilization rates of each server when the system reaches the maximum number of concurrent access. Figs. 5, 6 presents the CPU and memory usage graphs of the three servers named h1, h2, h3 under Robin Round, Random and LBBSRT schemes.

From Figs. 5, 6, we observe a slight difference in memory utilization and CPU utilization at 50% and 75% for the three servers respectively. The reason is that in our scheme, we can use controller get the real-time response time of each server, and it is more difficult to achieve in the traditional scheme. Our scheme always choose the server with minimal or the most stable response time. The response time of the server is smaller or more stable, the

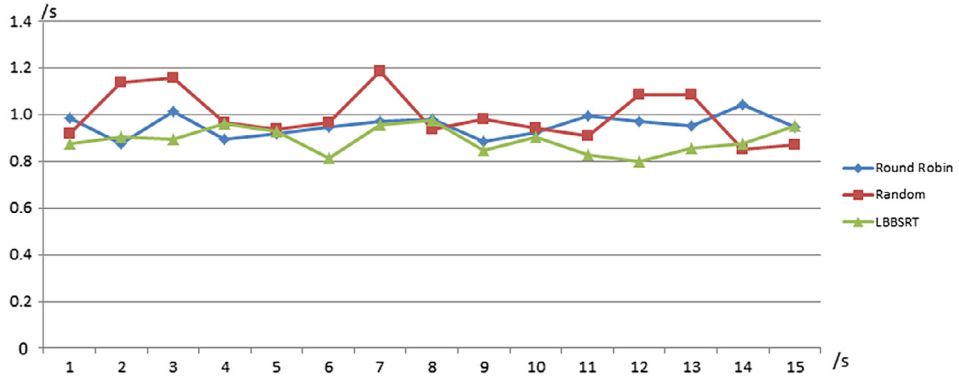


Fig. 3. The server's response time in the second case.

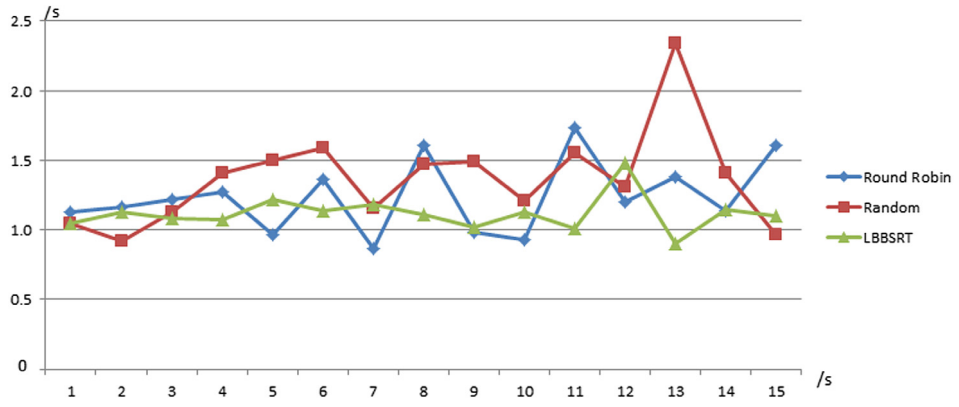


Fig. 4. The server's response time in the third case.

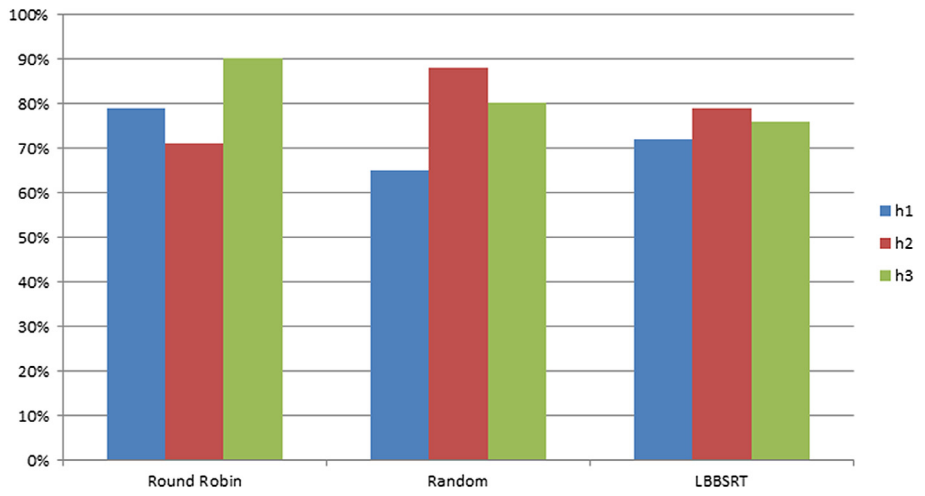


Fig. 5. CPU utilization.

corresponding server load is lower or more stable. In comparison with the Round Robin and Random schemes, our scheme exploits the server resources completely and thus achieves a much better effect of load balancing. After comparing with the static load balancing scheme, we further add an experimental contrast with zhang [11] based on the second cases. The experimental results are shown below.

From Fig. 7, the average value of the server response time of zhang [11] is 0.936 s which is lower than the Round Robin and Random scheme. But compared to LBBSRT, it is still high. From Figs. 5, 6 and 8, we can also find that the load balancing effect of LBBSRT is better than zhang [11].

5. Conclusions and future work

The emergence of SDN architecture provides us with a new train of novel prospects for solving the prevailing issues in the traditional load balancing network. In order to solve the problems of lower efficiency and higher deployments costs of

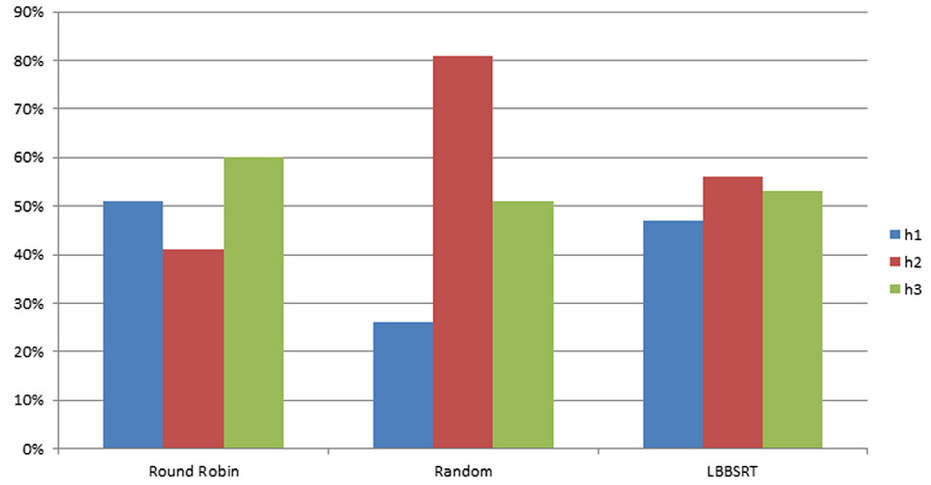


Fig. 6. Memory utilization.

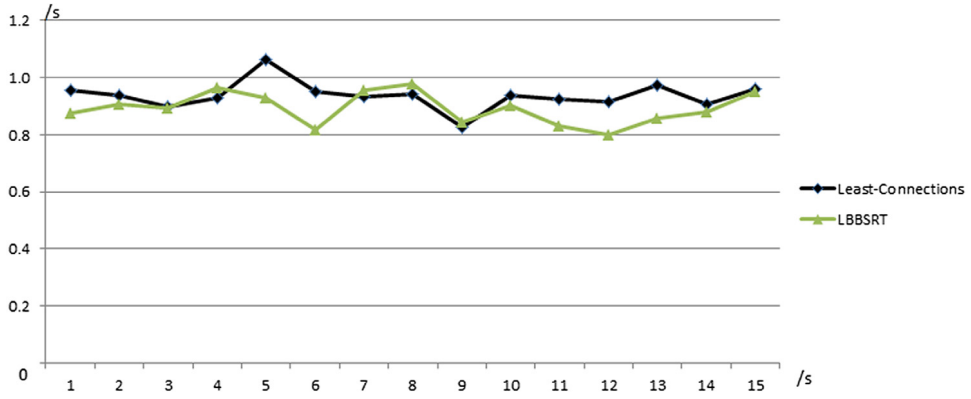


Fig. 7. The server's response time.

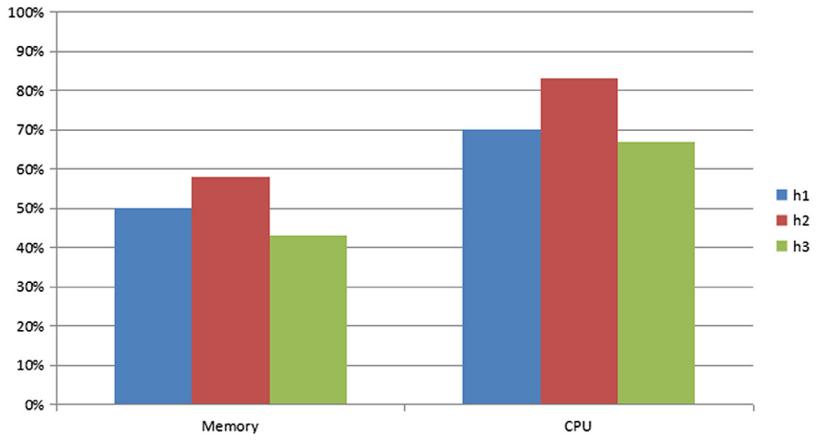


Fig. 8. Resource utilization of least-connections.

load balancing in the traditional networks, this paper proposes a dynamic load balancing scheme under the SDN architecture, using the controller to obtain the real response time of each server ultimately to select a server with minimum or the most stable response time. Our proposed scheme exploits the server resources and achieves a much better load balancing effect in comparison with the traditional Round Robin and Random schemes. Also, our schemes is cost effective than the traditional schemes, since we reduce the requirements of hardware equipment by the way of software customizing approach. Although our scheme solves the load balancing problem efficiently, there are still some limitations. We do not take into account the issue of energy saving in the load balancing on the server. Accordingly, we will further study how to save energy when achieve a balanced load so as to make a lot of more sense in the future.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (No. 61572001, No. 61502008), the Research Fund for the Doctoral Program of Higher Education (No. 20133401110004), the Educational Commission of Anhui Province,

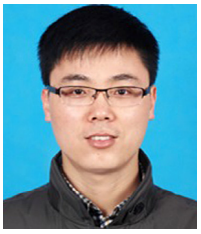
China (No. KJ2013A017), the Natural Science Foundation of Anhui Province (No. 1508085QF132), the Tender Project of the Co-Innovation Center for Information Supply & Assurance Technology of Anhui University (No. ADXXBZ2014-7), and the Doctoral Research Start-up Funds Project of Anhui University (No. J01001903). The authors are very grateful to the anonymous referees for their detailed comments and suggestions regarding this paper.

References

[1] Z. Huang, J. Liu, Q. Shen, et al., A threshold-based multi-traffic load balance mechanism in LTE-A networks, in: Wireless Communications and Networking Conference (WCNC), 2015 IEEE, IEEE, 2015, pp. 1273–1278.
[2] J. Zha, J. Wang, R. Han, et al., Research on load balance of service capability interaction management, in: 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology, (IC-BNMT), IEEE, 2010, pp. 212–217.
[3] W. Tian, M. Xu, Y. Chen, et al., Prepartition: A new paradigm for the load balance of virtual machine reservations in data centers, in: 2014 IEEE International Conference on Communications, (ICC), IEEE, 2014, pp. 4017–4022.
[4] E. Musoll, Hardware-based load balancing for massive multicore architectures implementing power gating, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (1) (2010) 493–497.
[5] F.A. Lopes, M. Santos, R. Fidalgo, et al., A software engineering perspective on SDN programmability, IEEE Commun. Surv. Tutor. 18 (2) (2016) 1255–1272.
[6] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openFlow: from concept to implementation, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2181–2206.
[7] P. Bosshart, G. Gibb, H.S. Kim, et al., Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 99–110. ACM.
[8] J. Naous, D. Erickson, G.A. Covington, et al., Implementing an OpenFlow switch on the NetFPGA platform, in: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM, 2008, pp. 1–9.
[9] N. Handigol, S. Seetharaman, M. Flajslik, et al., Plug-n-Serve: Load-balancing web traffic using OpenFlow, ACM Sigcomm Demo 4 (5) (2009) 6.
[10] S. Kaur, J. Singh, K. Kumar, et al., Round-robin based load balancing in Software Defined Networking, in: 2015 2nd International Conference on Computing for Sustainable Global Development, (INDIACom), IEEE, 2015, pp. 2136–2139.
[11] H. Zhang, X. Guo, SDN-based load balancing strategy for server cluster, in: 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems, (CCIS), IEEE, 2014, pp. 662–667.
[12] Z. Shang, W. Chen, Q. Ma, 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), IEEE, 2013, pp. 691–697.
[13] Z. Xu, R. Huang, L.N. Bhuyan, Load balancing of dns-based distributed web server systems with page caching, in: Proceedings. Tenth International Conference on Parallel and Distributed Systems, 2004. ICPADS 2004, IEEE, 2004, pp. 587–594.
[14] R. Tong, X. Zhu, A load balancing strategy based on the combination of static and dynamic, in: 2010 2nd International Workshop on Database Technology and Applications, (DBTA), IEEE, 2010, pp. 1–4.
[15] S. Yu, IEEE approves new ieee 802.1 aq™shortest path bridging standard[EB/OL]. [2012-05-08] https://www.techpowerup.com/165594/ieee-approves-new-ieee-802-1aq-shortest-path-bridging-standard.
[16] M.K. Shin, K.H. Nam, H.J. Kim, Software-defined networking (SDN): A reference architecture and open APIs, in: 2012 International Conference on ICT Convergence, (ICTC), IEEE, 2012, pp. 360–361.
[17] A. Malishevskiy, D. Gurkan, L. Dane, et al., OpenFlow-Based Network Management with Visualization of Managed Elements, in: Research and Educational Experiment Workshop (GREE), 2014 Third GENI, IEEE, 2014, pp. 73–74.
[18] S. Huang, J. Griffioen, K.L. Calvert, Network Hypervisors: Enhancing SDN Infrastructure, Comput. Commun. 46 (2014) 87–96.
[19] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: A survey, IEEE Commun. Surv. Tutor. 16 (1) (2014) 493–512.
[20] N. McKeown, T. Anderson, H. Balakrishnan, et al., OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[21] C. Rotsos, N. Sarrar, S. Uhlig, et al., OFLOPS: An open framework for OpenFlow switch evaluation, in: Passive and Active Measurement, Springer, Berlin, Heidelberg, 2012, pp. 85–95.
[22] M. Kobayashi, S. Seetharaman, G. Parulkar, et al., Maturing of OpenFlow and Software-defined Networking through deployments, Comput. Netw. 61 (2014) 151–175.
[23] H. Yin, T. Zou, H. Xie, Defining data flow paths in software-defined networks with application-layer traffic optimization: U.S. Patent Application 13/915,410[P]. 2013–6-11.
[24] F. Pakzad, M. Portmann, W.L. Tan, et al., Efficient topology discovery in OpenFlow-based Software Defined Networks, Comput. Commun. 77 (2016) 52–61.
[25] S. Scott-Hayward, Design and deployment of secure, robust, and resilient SDN Controllers, in: 2015 1st IEEE Conference on Network Softwarization, (NetSoft), IEEE, 2015, pp. 1–5.
[26] D.B. Hoang, M. Pham, On software-defined networking and the design of SDN controllers, in: 2015 6th International Conference on the Network of the Future, (NOF), IEEE, 2015, pp. 1–3.
[27] N. Gude, T. Koponen, J. Pettit, et al., NOX: towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (1) (2008) 105–110.

Hong Zhong is a Professor (from 2009) and Executive Dean of the School of Computer Science and Technology, Anhui University, China. She received Ph.D. degree in University of Science and Technology of China in 2005. Her research interests cover network and information security.

Yaming Fang is now a research student in the School of Computer Science and Technology, Anhui University. His research interest is Software Defined Networking.



Jie Cui is now an Associate Professor in the School of Computer Science and Technology, Anhui University. He received Ph.D. degree in University of Science and Technology of China in 2012. He has published over 30 papers. His research interests include network and information security.