OpenFlow-Based Dynamic Server Cluster Load Balancing with Measurement Support

Qingwei Du and Huaidong Zhuang
College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
Email: duqingwei@nuaa.edu.cn; zhhdong@nuaa.edu.cn

Abstract —In the current cloud computing environment, the size of the server cluster in the data center is growing in response to the increasing traffic. Due to the use of multiple replicas in the server cluster to provide the same services, effective load balancing as a key technology is very important. In this paper we implement and evaluate an alternative load-balancing architecture using OpenFlow switches connected to a controller, which gains high flexibility without additional equipment, and has the potential to be more robust than traditional load balancing approach. The system could measure network and server status in real-time and dynamic set weights of server according to the server's processing capability. Our load balancer installs wildcard rules in the switches proactively to direct requests of large groups of clients without involving the controller which effectively saves the flow table space and reduces the delay of the network. Our implementation uses the OpenFlow controller Floodlight and network emulator Mininet to verify the validity of this algorithm. The preliminary evaluation results demonstrate that our dynamic load balancing scheme is superior to not only the random load balancing algorithm but also the round robin load balancing algorithm.

Index Terms—Dynamic load balancing, OpenFlow, SDN

I. INTRODUCTION

With the growing scale of online services， using server cluster to provide network services has become a basic model of cloud computing. Multiple servers get together to provide the same services, which can greatly increase computing capacity as well as reduce single points of failure, thus providing higher availability. Load balancing in computer networks is a technique used to spread the workload across multiple network links or computers [1], [2]. In order to serve more clients with a minimum of latency and a maximum of throughput, Load Balancer distributes the incoming workload to a series of replicated servers. Traditional load balancing uses a dedicated hardware device to divide the network traffic into different server replicas. Although it is fast, but is expensive and lack of flexibility in the configuration, making the configuration cannot be dynamically adjusted based on the network status.

As an innovative networking technology that offers high programmability and practical way of user control in computer networks, OpenFlow [3] has been applied to many load balancing system [4]-[7]. A typical OpenFlow network consists of three major components, an OpenFlow controller, OpenFlow switches and hosts. The controller and switches communicate via OpenFlow messages. An OpenFlow switch consists of one or more flow tables that maintaining packet-handling rules. Each rule performs certain actions (such as forwarding, dropping, modifying the packets, or sending them to the controller) on a subset of the traffic that matches a rule. Each rule contains a pattern that matches fields of the packet header, and a priority field to distinguish between rules with overlapping patterns. The pattern supports exactly matching all the relevant header fields (that is a wildcard rule), or matching the wildcard rule with some "don't care" bits in the fields.

There are two ways for the controller to install rules in the switches. One way is reactive rule installation: When a new flow comes into the switch, it does a lookup in the flow tables. If the flow table is not matched, the switch creates an OpenFlow packet-in packet and forwards it to the controller for instructions. Then the switch installs a rule in the flow table based on the instruction.

Another way is proactive rule installation: the controller populate the flow tables ahead of time for all traffic that will come into the network, rather than reacting to a packet. By pre-defining all of flows and rules ahead of time in the OpenFlow switches flow tables, the packet-in event will never occur. The result is that all the packets are forwarded at line rate. If the flow table is in TCAM, it requires only a simple lookup. Proactive rule installation eliminates or drastically reduces the latency caused by the consulting a controller for every flow. Depending on the rules installed by a controller application, an OpenFlow switch can act as a switch, router, firewall, network address translator or load balancer [8].

sFlow is a general purpose network traffic measurement technology [9]. It is designed to not only provide the complete L2 to L4 information but also acquire the whole network statistics. All the traffic throughout the network can be accurately characterized

Manuscript received May 6, 2015; revised August 18, 2015.
This work was supported by the National Nature Science Foundation of China under Grant No. 61202350.
Corresponding author email: zhhdong@nuaa.edu.cn.
doi:10.12720/jcm.10.8.572-578.

and monitored, that allows analyzing performance or trends of network traffic in real time. We propose a novel approach to measure network and server status based on sFlow, which could avoid the use of the OpenFlow flow statistics counters. Therefore, there is no need to access the packet counters of each flow entry and aggregated flows can be used for forwarding purposes without affecting the operation of the load balancing mechanism.

Load balancing is a classical problem; researchers have put forward some load balancing strategies based on OpenFlow nowadays [10]-[13]. Handigol *et al*. proposed using OpenFlow to implementation the load balancer [6]. Plug-n-server tried to minimize response time by controlling the load on the network and servers through using customized flow routing, but this reactive solution has scalability limitations. Furthermore the approach of proactively map blocks of source IP address to replica servers using the OpenFlow wildcard rules, so client requests are directly forwarded through the load balancing switch, have been reported in [4]. However, they assume the traffic volume is uniform across client IP address, which cannot represent the actual traffic. Chen Wenbo, *et al*. proposed a dynamic load balancing algorithm based on server running state and OpenFlow in virtualization environment [10]. The architecture not only can achieve real-time monitoring of load but also provide the flexibility to write modules in the controller for implementing the customizable policy set. They use Libvirt to implement virtual machine management module which is responsible for obtaining the running status of each virtual machine in a fixed period, but they also use a reactive flow entry installation.

We proposed the design and implementation of OpenFlow-based dynamic server cluster load balancing with measurement support using sFlow protocol. The architecture not only is inexpensive but also provides the flexibility to scale the network for increasing traffic. When the network size increases, it only needs to add switches rather than purchasing additional hardware. In the beginning, the proposed scheme will measure the state of the network and calculate every server's serving load, then get the detail statistics of the traffic for generation an exact match. Next, the system proactively installs wildcard rules of accurate measurement traffic in the switches to direct client requests without involving the controller. It is demonstrated in the paper that the proposed scheme through feedback server load, dynamic adjustment the weights of servers periodic will make the servers' loading be distributed more balanced and make the response time be shorter than other well-known mechanisms.

The remainder of the paper is organized as follows. The proposed dynamic server load balancing mechanism with measurement support is presented in Section II. It is followed by the experimental results compared with other load balancing schemes. Finally, we conclude the paper by summarizing the main contributions in Section IV.

II. The Design of Load Balancing Architecture

The load balancing architecture described in this paper mainly composes of switches, a controller, a collector and load balancing application running on the controller. AS shown in the Fig. 1, the OpenFlow switch connects multiple servers and communications with the controller via OpenFlow message, receives rules from the controller and modifies the packet header as the executor of the load balancing. The Controller installs rules in OpenFlow switches based on the load balancing policy. The collector obtains running status of the network from the sFlow agent (which combines interface counters and flow samples into sFlow datagrams that are sent across the network to an sFlow collector) in the switch, and then reports it to the controller regularly. Each server provides the same service to clients, and set a static IP address within the cluster. The controller maintains an IP address list of the server in the cluster and real-time network topology. The load balancer partitions the client requests among the servers and lets the clients access the cluster using a "virtual" IP address. From the client's perspective, the cluster is regarded as a single server that responses to these clients requests. With the traffic increasing, additional servers can be deployed to the cluster conveniently without modifying any configuration of the system.

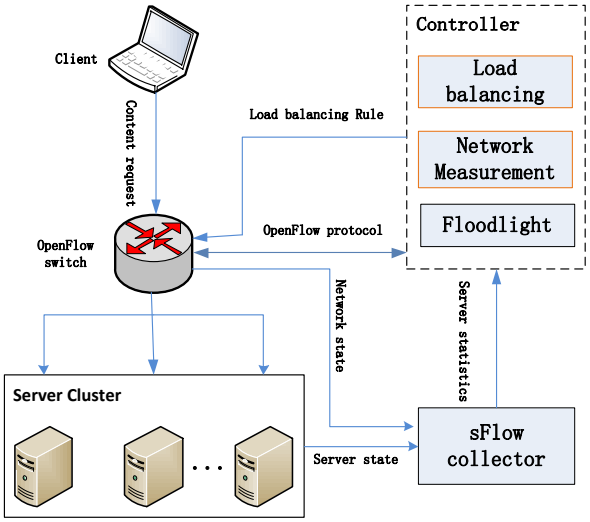

Fig. 1. The load balancing architecture in OpenFlow and sFlow environment.

For example, clustered servers in Fig. 1 work together in a distributed and parallel manner to serve requests from the Internet. When the client sends a request to the cluster, the gateway switch measures the traffic status of the network and reports it to the collector using sFlow protocol. The controller query statistics collected by the collector which analyzes the sFlow datagrams to produce a rich, network-wide view of traffic flows. Then the controller aggregates the requests, and generates wildcard rules to send to the switch. When the request arrives at load balancing switch, the switch uses packet header information to compare with flow entry in the flow table;

if packet header information matches up with a flow entry, the switch will modify the packet header according to actions in the flow entry to steer request to a specified server. If the packet does not match any flow entry，the OpenFlow switch will forward this packet to the controller, and let the controller determine how to distribute the request base on the load balancing policy.

A. Measurement of Network and Server

The measurement module use sFlow protocol to gather network status which includes the scale and IP address distribution of requests [14]. This module combines OpenFlow and sFlow technology to collect network status and export them to the controller. There are two different methods of network measurement. The first one is the basic OpenFlow method, in which controller queries flow statistics from switch periodical. The second one is using packet sampling to monitor flow in the OpenFlow network, sFlow was chosen as it is a low cost solution, and it has been implemented on a wide range of devices.

1) The basic OpenFlow method

For the purpose of obtaining the network status, the controller sends "Read-State" messages to collect statistics from the switch. Then, the switch replies the flow entries with their corresponding counters. The controller uses the counter of the rule to identify the imbalance of the traffic, and automatically adjusts the rule for rebalancing the traffic distribution. However, the controller needs to send a message to switch, and calculate the response information of the switch, which will cause huge overhead to the controller in large network, because of consumption a lot of resources.

2) The sFlow-based method

In order to overcome the blemish of the method we mentioned above, we take advantage of the packet sampling capability of sFlow in our method. Further, the sFlow technique decoupling the flow forwarding logic with statistics, since statistical information is no longer bound with flow entry, the packet sampling provides the necessary information of flow. Thus the measurement method based on the sFlow collects the statistical information of flow and update the corresponding counter, run as an individual application of the controller. Moreover, this method effectively reduces the required communication between switches and OpenFlow controllers, eliminates the potential control plane overloading in large scale data plane.

In this paper, the sFlow agent uses statistical packet-based sampling of switched packet flows to capture traffic statistics from the switch [9]. The packet flow sampling mechanism carried out by each sFlow instance must ensure that any packet observed at a data source has an equal chance of being sampled, irrespective of the packet flow(s) to which it belongs. Packet flow sampling works as follows: when a packet arrives on an interface, the switch makes a filtering decision to determine whether the packet should be dropped and whether or not to sample the packet. Samples are sent to the sFlow agent to process and get other packet information about the request, such as the source and destination interface, source and destination IP address, and next hop subnet. Then the collector aggregates the request information base on the IP address classification and reports to the controller. By this way, we can get the exact amount of the request IP address scale and distribution for the load balancing module, and which is not involved in the flow table. As the sFlow collector receives packet samples on the fly, it updates the corresponding counters inside the measurement module in a certain time period. Hence, there is no need to constantly maintain and compare detailed flow statistics for each flow entry of consecutive period. Consequently, this approach can reduce the complexity of the request measurement algorithm, thus requiring lower CPU resources.

What's more, we can use sFlow to measure the server load without the need for additional equipment. The Host sFlow project provides an open source implementation of the sFlow standard. Host sFlow can be installed on the server to export physical and virtual server performance metrics, including CPU, memory, disk and network IO performance [9]. The sFlow protocol can be used to measure state of network and server at the same time, along with sFlow agents embedded within the switch form the integrated measurement system that provides a complete picture of network, system, and application performance which can scale to a large number of servers. Server load status is sent to the collector by the sFlow agent via sFlow protocol, and then the collector report load information to the controller. After the controller gets the server load information, it can dynamically adjust the weights according to the load condition of the server.

3) Partitioning the client traffic and rule generation

The most important idea of this paper is the generation of the pattern field (matching packet header fields) for the wildcard matching rule. After the measurement module obtain the request information, the load balancing module divides the traffic into small parts, besides, we process the actual traffic instead of assume the traffic distribution. Our goal is to generate a series of wildcard rules to divide the IP address space, and then associate with the weights of servers. The detail procedure is as follows:

4) Split the IP address space into subnets

At the beginning, we use a 24-bit netmask, with a range of 0.0.0.1/24-255.255.255.1/24 to split the IP address space. There is some different from traditional netmask in CIDR. If no wildcards are setting, the OpenFlow protocol match field exactly describes a flow, over the entire OpenFlow n-tuple. If all the wildcard flags are set, then every flow will match. The source and destination netmask is each specified in the wildcard description. It is interpreted similar to the CIDR suffix, but with the opposite meaning, since this is being used to indicate which bits of the IP address should be treated as "wild". For example, a CIDR suffix of "24" mean to use a netmask of "255.255.255.0". However, a wildcard mask

value of "24" means that the least-significant 24-bits is wild, so it forms a netmask of "255.0.0.0". So, we will get some different subnets contain the request IP address after the division of the subnet. It should be noted here that the subnet is not the subnet in the traditional sense; we are here to refer to a range of IP address space in the wildcard rules.

5) Calculate the number of IP addresses in each subnet

In this step we find actual requests in the subnet address space and calculate the number of the IP addresses for the next step to allocate server based on the weight of server.

6) Assign the request of the subnet to each server

First, we sort the subnet in accordance with the valid address in the wildcard rule, and then assign them to the specific replica server following the weight of the server. Then we select the biggest address space in a wildcard rule from a wide range of subnet, and assign it to the server which has the strongest processing capacity. Then we select the second large address space wildcard rule allocated to the server and go on until the final result is that, the wildcard rule is divided uniformly according to the server weight.

7) Divide the network into smaller subnetworks

If there is too many clients' traffic in a subnet, a server cannot afford to response so many requests, and then we divide the subnet again. Such as the original subnet is 128.0.0.1/24, which will be divided into smaller subnets. Here is from 128.0.0.1/16 to 128.255.0.1/16. After this division, we calculate the number of actual request in each new subnet again to count the valid number of IP address. Then the system split a wildcard rules to a new one with a smaller number valid IP address which can have a more fine-grained adjustments.

If the number of IP addresses within the subnet is too many, the wildcard rule can continue to be divided until it can be assigned to a server. The number of the wildcard rule and the divided subnet is equal; if we want to get less number of rules require minimal subnetting. Subnet mask is made by setting network bits to all "1"s and setting host bits to all "0"s, and the greater the subnet range is generated, the more the number of IP addresses that may be contained in the subnet.

8) Install wildcard rules in the switches proactively

The load balancing module generates wildcard rules flow entries base on the server's processing capacity, proactively install the rules in the load balancing switches to direct requests for large groups of clients without involving the controller.

B. Load Balancing Policies and the Weight of the Server

In this section, we propose a novel load balancing algorithm that takes into account not only the load on the server but also the performance differences between servers in heterogeneous cluster and different type of client request. For example the computationally intensive request and I/O intensive request consumption of CPU resources are different. In traditional load balancing scheme, the weights based on the server's performance are fixed. But in fact, the processing capacity of the server is dynamic changing. For example, with the increase of the load on the server, its processing capacity is reduced, on this condition, using a fixed value of weights may not be optimally distributing the load. Meanwhile, the type of client request is different, for example compute-intensive requests consume more CPU resources, and I/O intensive requests require faster disk speeds. If a fixed value is used without considering the differences, the algorithm cannot accurately reflect the server load when the request change, so using a dynamic adjustment algorithms base on the server status is essential.

1) Load computation of the server

There are many factors that would contribute to server's load, this paper considers the CPU utilization, memory utilization, disk I/O speed and bandwidth accessing rate. But this information does not directly represent the load of a server, and each ration has a certain influence on the load, so it needs a function to convert these indicators and the server's load can be expressed as below:

$$L(s) = w_1 L(CPU) + w_2 L(mem) + w_3 L(band) + w_4 L(disk) \quad (1)$$

where L(CPU) is the CPU usage rate, L(mem) is the memory usage rate, L(band) is the bandwidth usage rate and L(disk) is the utilization of I/O rate of the disk and the L(s) is the load of server s. Because different types of factors have different level of influence on the load, so we introduced the parameter w, used to indicate the influence degree of the load, and $\sum_{i=1}^{n} w_i = 1$. It can be adjusted according to the type of service and the performance of server. The value of L(s) indicates the degree of server load, the larger the value, the more heavily loaded; the smaller the value, the lighter the load.

2) Processing ability computation of the server

In heterogeneous server cluster, the server specification is different in performance, load balancing should not only consider the load, but also take into account the processing capacity of the server. In order to facilitate the realization of the algorithm, we calculate the server's processing capability based on the parameters: the CPU capacity, memory size, network throughput, and disk I/O speed. Therefore, server's processing capability can be expressed as a linear combination of these factors as given in the equation below,

$$C(s) = r_1 C(CPU) + r_2 C(mem) + r_3 C(band) + r_4 C(disk) \quad (2)$$

where the parameter r denote the weighting coefficient and $\sum_{i=1}^{n} r_i = 1$. The greater the processing capacity of the server, the server can able to handle the more client's request.

3) Server weight computation for the traffic distribution

After using wildcards to divide subnet, traffic is distributed to a specific server for each subnet based on the weight assigned to the server. In this article, weights are calculated taking into account not only the load on the server, but also the processing capabilities of the server. Through the above discussion, we give the definition of the server's current processing capacity as follows,

$$CC(s) = C(s) * (1 - L(s) / C(s)) \quad (3)$$

here, CC(s) means the server's current processing capacity to handle client requests currently. After computing of current processing capacity for each server, we use the formula (4) to calculate the current server weights.

$$W(s) = CC(s) / \sum_{s=1}^{n} CC(s) \quad (4)$$

Every time the load balancing module distributes traffic based on the weight of server, which is a dynamic value follow the server load, means the ratio of this server in the cluster. The server having a high-end hardware may have a low weight because of its high load, but the value W(s), can better reflect the relative importance of the server in the cluster.

C. Re-Partitioning Traffic to Replica Server

There are two cases need to re-partition the traffic. One case is that the measuring module detects the load between servers has a large difference; another case is the time goes beyond the default scheduling cycle.

There are several possible reasons that will cause load imbalance. Even the wildcard rule match up the most part of the traffic, the termination of connections or new request from client with a new IP address not in the wildcard rule, will change the scope and scale of IP address.

Meanwhile, different types of requests, such as compute-intensive or I/O intensive request consume different server load. In the re-partitioning period, a new request, if not match the existing wildcard rules, will be sent to the controller, then the controller select the lightest load server to process the request, until needing a new round traffic division.

Two questions need to be solved in the re-partitioning procedure. One is allowing ongoing TCP connections to complete, directing all TCP connections from one client IP address to the same host in the cluster. The other is tries to re-use the previous installed rules.

Our approach is to compare the scope and scale of the current IP address with the last round to identify the change of traffic distribution. For the new IP address space, the new added parts will generate new wildcard rules; the original wildcard rules will be updated to remove the reduced parts. By recording the scope of IP addresses, and using a centralized control method, we can get the global optimal partition results while maintaining the continuity of TCP connection.

III. Experiment and Evaluation

In this section, we describe the evaluation environment, traffic design and measurement approaches. We conduct experiments with Mininet [15] on Ubuntu host and the Floodlight [16] OpenFlow controller. Then we measure the network throughput and latency under three different load balancing schemes—random, round robin and the mechanism proposed in Section II. Response latency refers to a time interval between the client sending an HTTP request to server and receiving the response of the server. Throughput of the system is the sum of the data rates processed by the server in the network.

Floodlight is a modular Java based controller, which is exploited as a high-level programmatic interface upon network events. Through the API of the Floodlight controller, we implemented all three components as separate Floodlight applications, responsible for measurement network and server, partitioning the client traffic and generation rules according to the load balancing policy.

A. Environment

The experiment configuration using the prototype system is shown in Fig. 2. Our load balancing experiment environment consists of seven machines. In the prototype system, three Web servers in the network map to the same virtual IP address and provide the same services. The detail hardware specifications are as follows, and the server1 has higher computing power than others.

Table I: The Detail Hardware Specification

Host Name	Web Server 1	Web Server 2	Web Server 3
CPU	Dual 3.2 GHz	Dual 2.6 GHz	Dual 2.6 GHz
Memory	4 GB	2 GB	2 GB
Disk	320G 5400rpm	320G 5400rpm	320G 5400rpm
GE NIC	Atheros AR8114	Atheros AR8114	Atheros AR8114

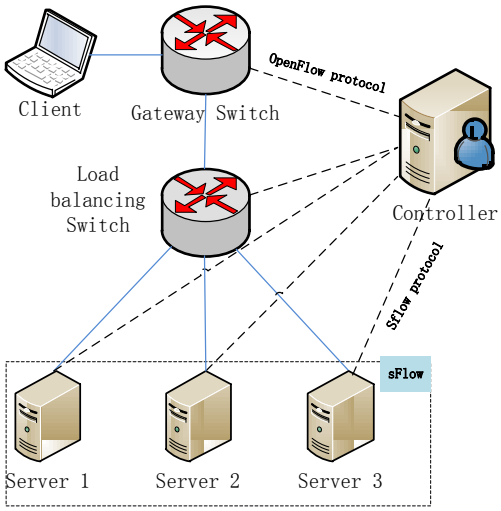

Fig. 2. Experiment environment.

Two PC install Openvswitch which support both OpenFlow and sFlow protocols, one as a gateway switch and the other as a load balancing switch which is responsible for distributing the request to the web server. The Controller and sFlow collector are installed on the same PC to reduce network transmission delay. We used the sFlow collector implemented as a Floodlight application along with traffic controlling application to measure the request. All servers have a fixed IP address and connect to the collector to report load information.

B. Traffic Design and Benchmark Algorithms

One main challenge of the network performance evaluation is how to generate real network traffic; we use English Wikipedia access trace [17] to generate traffic through Mininet, a network emulator, which creates a network of virtual hosts and switches. The hosts in Mininet set an IP address in the trace file which can reflect the scale and distribution of IP addresses in real network. Each host using "wget" to send request to server and one to ten processes are randomly selected to send request. In order to test the performance of the proposed algorithm, we use two other algorithms as comparisons.

The first is the random algorithm, each new request will be forwarded to the controller by load balancing switches, and the controller selects a server from the server cluster randomly to response client requests and generates a corresponding flow entry to send to the switch.

The second is the round robin algorithm. For each request that is forwarded to the controller by the switch, the controller selects a server from the server cluster sequentially, then the server respond to the client's request in turn, until the end of one cycle and then re-start the next round.

The flow entry's action is to modify the destination MAC and IP address of the request packet with the selected server's MAC and IP address. After the packet's header is modified, the switch forwards the packet to the output port of the switch. In a word, the controller generates a flow entry, the switch modify the source IP address of the packet to the load balancing virtual IP address, the server responds to the client's request.

C. Results

Each measurement is performed 10 times for the rather stable results. Fig. 3 shows the average response time of the system at different rate of requests using three load balancing strategies. The Random and round robin methods have similar response time, and each request can obtain relatively fast response, when the request is at a low rate. But with the requests rate increasing, the proposed algorithm has a significant advantage on response time. Since the proposed method does not require the first packet of each flow to forward to the controller, the delay is significantly reduced. But after a period of time or a large amount of traffic does not match the installed rule, the delay will increase, because the packet has to forward to the controller like the original. In addition, an important reason for the delay is that the software switches rewriting the packet header is very slow, but performance will be better if using a hardware switch.

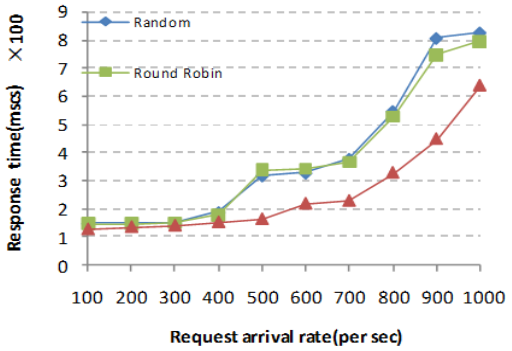

Fig. 3. Response time of 3 different load balancing strategies.

Fig. 4 depicts the changes in throughput of the server against the request arrival rate. The x-axis represents the client request arrival rate per second; the y-axis represents the throughput of the server cluster. As can be seen from the figure, the proposed dynamic load balancing algorithm can achieve higher throughput than random and round robin algorithm. With the request rate increasing, throughput gradually increased until it reaches a maximum at 600 requests per second, then throughput begins to decline. But the dynamic load balancing algorithm is still able to get a higher throughput than other algorithms under the same conditions.

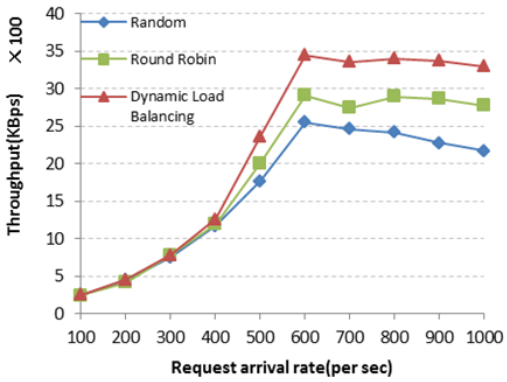

Fig. 4. Server's throughput of experiment demonstrating.

Fig. 5 shows the CPU usage of three algorithms when the number of requests is 500 per second. The CPU load is used here to represent the server's load, because the CPU is the most important factor accounting for the weight of a server. When we use the random algorithm, CPU utilization of the server is most unequal, and using the proposed dynamic load balancing algorithm can make the servers have almost same CPU utilization. Because server 1 has a higher computing power, it' CPU utilization is relatively low when using other algorithms. But the proposed dynamic server load balancing algorithm can distribute more requests to server1 and make full use of its performance. The dynamic load

balancing algorithm has a better resource scheduling ability to avoid unbalanced load among servers, thus improving overall resource utilization of the system. The above result confirms that the proposed method successfully improves performance by controlling the traffic load on servers.

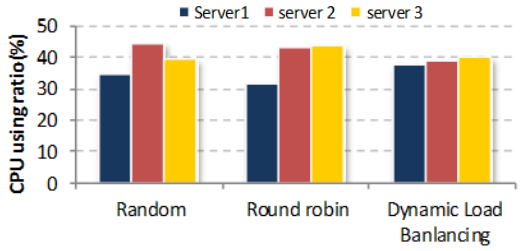

Fig. 5. Server's CPU using ratio of 3 load balancing strategies.

IV. Conclusions

In this paper, we propose a dynamic server load balancing algorithm based on OpenFlow and sFlow to efficiently distribute traffic among servers of cluster. The algorithm utilizes the wildcard rules to aggregates traffic of the server replicas, and makes decisions based on real-time traffic statistics obtained via the sFlow protocol. Furthermore, our load balancer proactively installs wildcard rules on the switches to direct requests for a large group of clients without involving the controller, which will reduce the number of rule and reduce the network latency. We have implemented the algorithm as a module of the Floodlight controller application, and use the Mininet network emulator and Openvswitch to evaluate the effectiveness of the algorithm. By comparing with the random and the round robin algorithms, it proves that our algorithm can improve the throughput and obtain lower latency while the server load will be more balanced. Due to limitations of the experimental conditions, the network size is small and only software switch is used. In our future work, we plan to apply our algorithm to the actual environment with the hardware switch and real traffic, it will get better results.

Acknowledgment

The authors would like to thank the anonymous reviewers for their comments that improved the content and the presentation of this paper. This work was supported by the National Nature Science Foundation of China (No. 61202350).

References

[1] Microsoft's Network Load Balancing. [Online]. Available: https://msdn.microsoft.com/en-us/library/bb742455.aspx

[2] Q. Zhang, A. Riska, W. Sun, *et al*, "Workload-aware load balancing for clustered web servers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 3, pp. 219-233, 2005.

[3] N. McKeown, T. Anderson, H. Balakrishnan, *et al*., "OpenFlow: Enabling innovation in cam pus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[4] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," in *Proc. 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Boston, Mass, USA, April 2011.

[5] H. Nikhil, F. Mario, S. Srini, J. Ramesh, and N. McKeown, "Aster* x: Load-balancing as a network primitive," in *Proc. 9th GENI Engineering Conference (Plenary)*, 2010.

[6] N. Handigol, *et al*., "Plug-n-Serve: Load-balancing web traffic using OpenFlow," *ACM Sigcomm Demo*, 2009.

[7] M. Koerner and O. Kao, "Multiple service load-balancing with OpenFlow," in *Proc. IEEE 13th International Conference on High Performance Switching and Routing*, 2012.

[8] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87-98, 2014.

[9] P. Phaal. (July 2004). sFlow Specification Version 5. [Online]. Available: http://sflow.org/sflow_version_5.txt

[10] W. B. Chen, *et al*., "Dynamic server cluster load balancing in virtualization environment with OpenFlow," *International Journal of Distributed Sensor Networks*, 2014.

[11] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in *Proc. 27th IEEE International Conference on Advanced Information Networking and Applications*, March 2013, pp. 290–297.

[12] A. Ghaffrinejad and V. R. Syrotiuk, "Load balancing in a campus network using softare defied networking," in *Proc. 3rd GENI Research and Educational Experiment Workshop*, 2014.

[13] L. Yu and D. Pan, "OpenFlow based load balancing for fat-tree networks with multipath support," in *Proc. 12th IEEE International Conference on Communications*, Budapest, Hungary, 2013.

[14] Host-sFlow Project. [Online]. Available: http://host-sflow.sourceforge.net

[15] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM SIGCOMM HotNets Workshop*, 2010.

[16] Floodlight Openflow Controller. [Online]. Available: http://floodlight.openflowhub.org/

[17] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830-1845, 2009.

Qingwei Du was born in 1974. He received the Ph.D. degree from Southeast University. He is currently working as a vice professor of Nanjing University of Aeronautics and Astronautics. His research interests include wireless sensor networks, mobile computing and software define network.

Huaidong Zhuang was born in Suzhou, China in 1989. He received his B.S. degree in Computer Science and Technology from Linyi University, China in 2012. He is currently a graduate student of Nanjing University of Aeronautics and Astronautics. His research interests include software define network, traffic schedule and load balancing systems.