The load balancing technology is widely used in current enterprise network to provide high quality and reliable service. Conventional load balancing technology is often achieved by specific hardware that is usually very expensive and lacks sufficient flexibility. Meanwhile, it is easy to become a single point of failure and would be restricted in virtualization environments. Thus, we propose a load balancing algorithm based on server running state, which can calculate comprehensive loading according to the CPU utilization, memory utilization, and network traffic of the servers. Furthermore, a load balancing solution based on software defined networks (SDN) technology is applied in this paper, and it is designed and implemented in OpenFlow network. We combine network management and server state monitor in this scheme, in which the OpenFlow switches forward the request to the least comprehensive loading server by modifying the packet.

1. Introduction

Currently, traffic on the network is very huge and is growing rapidly. Network congestion and server overload are becoming severe problems faced by enterprises. In particular, the technologies and concepts such as cloud computing, virtualization, and big data make the issue particularly important [1]. Typical enterprise networks are very complex, and with the growth of business, enterprises need to purchase more equipment, build more sophisticated networks, and handle more traffic.

Most internet service providers use load balancing technology to assign the user's requests to different computers in data center. In order to minimize response time of requests and enhance user experience, requests from different users are processed by different computing nodes [2]. Therefore, the amount of computation for each node is reduced. Load balancing technology is mainly for web service, FTP service, business-critical application, and other network applications [3]. Traditional load balancers are expensive, and the load balancer policy set needs to be formulated in advance, with its lack of flexibility leading to its inability to deal with emergency situations. Traditional load balancer requires dedicated administrators to be maintained and does not allow users to design flexible strategies based on their actual network conditions. Since all requests are passed through a single piece of hardware, any failures on the load balancer will cause the collapse of the entire service.

Various load balancing schemes have some deficiencies in current situations. The fundamental reason is that the traditional design of the Internet has some problems [4]. Under the impact of the new requirements, the bottleneck of traditional network architecture has been reflected in many aspects. People are looking for new options to meet changing business. Among many projects, SDN is the most influential and distinguished one, as the representative of SDN from the beginning, OpenFlow [5] has received wide attention from researchers. The goal of OpenFlow is to change the way of controlling traditional network devices. In traditional networks, network devices forward data in accordance with distributed data management and, in the process of forwarding data from source to destination, individual equipment determines how to forward its data independently. OpenFlow separates control module from the devices and puts it into an external server with a control program running on it; the server can send commands to
the OpenFlow switches to control the forwarding policy, and the control program can also provide an external application programming interface for network administrators to control the switch programmatically. This does not only reduce the need for manual configurations on switch, it can also provide greater flexibility in network management. The OpenFlow switch model is shown in Figure 1.

Using load balancing technology in OpenFlow network can well overcome some of the shortcomings of traditional load balancing and provide a simple and effective solution with high flexibility [6]. Due to the difference between the traditional Internet and OpenFlow network, they inevitably differ from each other in using load balancing techniques in traditional network and OpenFlow network. The traditional load balancing technology is not fully applicable to the OpenFlow network. New problems have emerged, such as load balancing module design, the server operating status monitoring, and how to ensure the flexibility of load balancing.

On the other hand, many businesses in the enterprise network are migrating to virtual environments because virtualization technology can help companies to save money, consolidate servers, and maximize the utilization of limited resources [7]. But now that load balancing technology is not mature enough in virtual environment, the application of traditional load balancing products is under restrictions in data center virtualization environments, which brought resistance to enterprise data center virtualization development.

We proposed the design and implementation of OpenFlow-based server clusters dynamic load balancing in a virtualized environment. The architecture not only is inexpensive but also provides the flexibility to write modules in the controller for implementing the customizable policy set. Internet applications can achieve real-time monitoring of load and timely access the appropriate resources by the flexible configuration capabilities of this architecture. An OpenFlow switch can connect to multiple controllers in OpenFlow1.3, we can use several servers as controller connecting to OpenFlow Switch, so as to improve the robustness of the system [8].

2. Related Work

Uppal and Brandon implemented and evaluated an alternative load balancing architecture using an OpenFlow switch connected to an NOX controller, which gains flexibility in policy and has the potential to be more robust to failure with future generations of switches [9]. OpenFlow load balancing architecture in this paper used an OpenFlow switch connected to the controller, the Internet, and a group of servers. Each server used a static IP address and the controller maintained the server list. NOX controller defined load balancing policy and added a new rule of the flow table to the OpenFlow switch using handler function. The module implemented the three load balancing strategies: Random, Round Robin, and Load-Based. Thus, all clients request to the same IP address; when a request arrives, NOX modifies the destination MAC address and the destination IP address of package based on the load balancing strategy. After modifying, the switch forwards the packet to a picked server. When the response from the server arrives, OpenFlow switch modifies the source MAC address and the source IP address into the destination MAC address and the destination IP address of the client request.

Handigol et al. described a structure of the network load balancing system that addresses two aspects of load balancing concerns: network congestion and server load [10]. Plug-n-server tried to minimize response time by controlling the load of network and servers using customized flow routing. The system allowed the operator in any way to add computing resources and switch to increase the capacity of web services. Plug-n-server controller used an integrated optimization algorithm developed by Stanford, called LOBUS. In the demonstration, all servers were assigned the same IP alias. When the request whose destination IP address was the IP of the servers arrived, the controller determined which server and which route to be chosen to minimize response time. To achieve this goal, Plug-n-server completed the following tasks: (1) monitoring the current status of the network and servers, including network topology, network congestion, and load of servers and (2) selecting the appropriate server to respond to the request from client.

Yu and Pan proposed a dynamic routing algorithm for load balancing in the network data center to obtain high performance and low latency [11]. By comparison, dynamic load balancing routing could schedule network traffic according to the updated traffic statistics on each network device. Although dynamic load balancing is flexible and adaptive to real-time network statistics, it brings extra overheads for monitoring network statistics and scheduling flows. Their paper presented a load balancer for the fat-tree network with multipath support. They implemented a dynamic load balancing routing algorithm in the load balancer, which was
adaptive to network traffic and scheduled flows by examining the current available bandwidth on all alternative links.

Long et al. proposed a novel path-switching algorithm LABERIO to balance the traffic dynamically [12]. The purpose of LABERIO is to minimize network latency and transmission time and to maximize the throughput by utilizing resources better. The experiments performed in two kinds of network architectures show that LABERIO could get less response time than other typical load balancing algorithms. And they conducted some comparative study; it came that the algorithm was better than other methods within multiple transmission modes. And they compared some relative studies; it came that the algorithm was better than other methods within multiple transmission modes.

Ghaffarinejad and Syrotiuk developed a load balancing solution based on SDN for their campus network [13]. Their aim is to show that such a solution may cut costs and improve flexibility of network. In the solution, their application shared elements with the distributed control plane of Kandoo [14] and there were a root controller and multiple local controllers. When an event comes, if it is not required global information of network, the local controller will handle it; otherwise, the event will be forwarded to the root controller, which keeps network-wide state and could process it best.

3. Proposed Research

The load balancing architecture described in this paper consists of OpenFlow network with controller and server clusters, which is connected to the OpenFlow network, and a SAN storage network which provides shared storage for the server cluster. Controller rules the OpenFlow switches; at the same time, it obtains running status from the servers regularly.

As is shown in the Figure 2, each server, which provides external service, and the controller are equipped with a static IP address. The controller maintains a server address pool and generates current network topology in real-time. There is also a virtual IP address used by clients to request. From the client’s perspective, the server cluster reflects an IP-based system. All the servers share the virtual address and all the clients will send their requests to the virtual IP address. That
is to say, the clients treat the server cluster as one server whose IP address is the virtual IP address. When a client sends a request to the OpenFlow network, OpenFlow switches use the packet header information to compare with flow entry in the flow table; if packet header information matches up with a flow entry, OpenFlow switches will increase the counters and byte counters associated with the flow and forward the packet using actions in the flow entry. If there is no flow entry that matched up with the packet, OpenFlow switches will forward this packet to the OpenFlow controller, and then let the controller determine how the switch forwards the packet. The controller adds the corresponding flow entry to switch through OpenFlow protocol. Figure 3 shows the procedure of OpenFlow switch processing packet.

When a packet-in message comes into the controller, the controller will check if the IP address of the packet is the same as the virtual IP address, by which the servers provide network service. If not, the controller will forward the packet normally. Otherwise, the controller will select a server to respond to the client based on the load balancing strategy, and it will also add a flow entry to the OpenFlow switch which sent the packet-in message. In addition, the servers use a shared storage via iSCSI protocol to behave the same.

### 3.1. Load Balancing Strategies

#### 3.1.1. Random Algorithm
Immediately after each flow is forwarded to the controller, the controller randomly selects a server from the server list to process the client request [15].

#### 3.1.2. Round Robin
For each flow that is forwarded to the controller just a moment, the controller selects a server to process the client's request according to a certain order [16].

#### 3.1.3. Server-Based Load Balancing Algorithm (SBLB)

1. Dynamic feedback of the current server load mainly collects CPU occupancy rate $L(C_i)$, memory occupancy rate $L(M_i)$, and response time $L(R_i)$, but this information does not directly represent the load of a node, so it needs a function to convert these indicators and then get the load of the server $L(S_i)$:

   $$L(S_i) = r_1 \cdot L(C_i) + r_2 \cdot L(M_i) + r_3 \cdot L(R_i),$$

   where $i = 0, 1, \ldots, n - 1$; $\sum r_i = 1$.

   Due to the fact that there are different service types, which can have different influences on each part of the node, we introduced the parameter $r$; it is used to emphasize different degrees of influence of different parts.

2. Processing computation ability of server node: when we compute load balancing, if the service nodes are heterogeneous, we not only should consider the node load but also must consider the node's processing capacity. For the processing ability $C(S_i)$ of node $S_i$, it is mainly considered from the following several indicators: CPU quantity $n_i$, CPU computation ability $C(C_i)$, CPU type, memory capacity $C(M_i)$, disk I/O rate $C(D_i)$, network throughput $C(N_i)$, and maximum process number $C(p_i)$:

   $$C(S_i) = k_1 \cdot n_i \cdot C(C_i) + k_2 \cdot C(M_i) + k_3 \cdot C(D_i) + k_4 \cdot C(N_i) + k_5 \cdot C(p_i),$$

   where $i = 0, 1, \ldots, n - 1$; $\sum(k_i) = 1$.

3. Calculating weight: when the server load balancing system comes into use, the administrator should set $\text{DEFAULTWEIGHT}$ as initial weight to every server; as the server load changes, the weight will be adjusted. In order to avoid that the weight becomes a too large value, we set a range of weights to $\text{DEFAULTWEIGHT} \cdot \text{SCALE} \cdot \text{DEFAULTWEIGHT}$, where $\text{SCALE}$ can be adjusted, and its default value to 10. Controller runs SNMP module periodically to query the load parameters and calculate the aggregate load value $\text{AGGREGATELOAD}$. And, using a sigmoid function to map aggregate load to $[0, +\infty)$, if the result is 1, we think the server is in the best condition. We introduce the following formula depending on the server aggregate load value to adjust its weight:

   $$w_i = w_j + A \cdot \sqrt{0.95 - \text{AGGREGATELOAD}}.$$  

In the formula, 0.95 is what the system utilization we want to achieve, and $A$ is an adjustable coefficient (the default value is 5). If the aggregate load is 0.95 then the server weight will not change, if the aggregate load is greater than 0.95, it will become smaller, and if the aggregate load is less than 0.95, it will become larger. If the new weight is greater than $\text{SCALE} \cdot \text{DEFAULTWEIGHT}$, we set the new weight of $\text{SCALE} \cdot \text{DEFAULTWEIGHT}$. This is a dynamic feedback formula that will adjust the weights to a stable point. When a system achieves the desired state, the weight will be constant. In practice, if the weight of every server is less than $\text{DEFAULTWEIGHT}$, the server cluster is considered overloaded. In this case, new nodes need to be added to the server cluster to address this issue. Conversely, if the weight of every server is close to $\text{SCALE} \cdot \text{DEFAULTWEIGHT}$, the current system load is considered very small.

4. For each new incoming request, the load balancer selects node to respond to the client in the following method. Firstly, find out the server $S_m$ that has the maximum value of weight, and then find out a server $S_i$, whose weight $W(S_i)$ fulfills $W(S_i) > W(S_m) - f$, $i = 0, 1, \ldots, n-1$, where $f$ is a constant. Then, add $S_i$ to
Main process:
(1) for $i = 1$ to length(servers)
(2) calculate $L(S_i)$, the load of server,
(3) calculate $C(S_i)$, the computation ability of server,
(4) Calculate $P(S_i)$ due to $C(S_i)$, the probability of task of server,
(5) end for
(6) find $n$ servers, which has least load
(7) selects a server from the $n$ servers based on the probability

Algorithm 1: Server-based load balancing algorithm.

Collect $f$. After that, we calculate the probability $P(S_k)$ for each server in the collection using the following:

$$ P(S_k) = \frac{C(S_k)}{\sum C(S_i)}. \quad (4) $$

Then the system selects a server based on the probability. The pseudocode is shown in Algorithm 1.

3.2. OpenFlow Based Load Balancer Implementation. Controller forwards the packet which accesses the virtual host to a server whose aggregated load is the smallest in the server pool by adding flow. It works as follows: all clients use the virtual IP address for accessing the servers, which is known as an address for the network service. Client should set an ARP request for the virtual address before request. When OpenFlow switch receives the ARP request and sends a packet-in message to controller, controller will send an ARP reply packet through a packet-out message, which contains a virtual MAC address associated with the virtual IP address. When a request arrives to OpenFlow switch, then the controller designates one server to serve it according to the load balancing strategy then adds the flow entry to the flow table, which will modify the request packet’s destination MAC and IP address into the selected server’s MAC and IP address and forward packet to the selected server. Meanwhile, it also adds a flow for the response of server, which will modify the response packet’s source MAC and IP address into the virtual MAC and IP address. Then the connection between the client and the server is established; all packets will be forwarded at line speed.

In this paper, our load balancer consists of three modules as follows: virtual machine management, load balancing policy module, and Floodlight, as shown in Figure 4. In the experiments, we use Libvirt to implement virtual machine management module. This module is responsible for obtaining the running status of each virtual machine in fixed period, including CPU usage, memory usage, system I/O usage and response time of web server, and feeding the result back to the load balancing policy module. The responsibility of the load balancing policy module is to select a server for the client according to the SBBL algorithm when a request arrives, which will process the request from the client. In order to achieve that, the load balancing policy module calculates the aggregated load of each server according to the feedback information from virtual machine management module and then selects the smallest loading server based on the calculated result. The selected server will process the client’s request. Floodlight module establishes a connection between the client and the server that serves the client by adding flows. When a client request arrives, Floodlight module will notify the load balancing policy module. Load balancing module will then select a server and let Floodlight know. Lastly, Floodlight module will establish network connection between the server and client so that the client and the server can resume normal communication. The whole request process is therefore completed.

4. Experiment

4.1. Experiment Environment. The main purpose of this experiment is to compare the performance of using different load balancing strategies in virtualization environment based on OpenFlow. As shown in Figure 5, there are 4 KVM virtual machines as web servers and 3 Open vSwitch instances as OpenFlow switches, which connect to the servers. The servers in the network map to the same virtual address and provide the same service. Since we must guarantee the consistency of the servers’ content so that they offer the same web service, another KVM virtual machine is used as a shared storage node that interfaces with other nodes as an iSCSI target in the form of a shared disk. The KVM servers are set as iSCSI client that mount the shared disk so that they could access the same storage to provide the same service.

4.2. Experiment Results. In order to store the running state of the server, we store all the results into MySQL databases that are collected by virtual machine management module. Load balancing policy module gets data from the database to calculate the current load and informs Floodlight. The Ubuntu based client runs Jmeter, a load test tool, to simulate 100 users accessing the service concurrently. And there are the same configuration and hardware for all servers in the experiment.

To see the advantage of SBLB load balancing strategy in performance more intuitively, we analyzed the system response time by three different load balancing strategies, SBLB, Random, and Round Robin. Figure 6 shows the system response time by three different load balancing strategies. The horizontal axis represents
the number of clients accessing and the vertical axis represents the system’s response time in seconds. As shown in the Figure 6, compared to the Round Robin and Random, SBLB algorithm has great advantages: it significantly reduced the response time and provided a better user experience in the experiment.

To illustrate, SBLB load balancing strategy is more efficient than Round Robin and Random in utilization of network resource; we analyzed the throughput of the entire load balancing architecture when different load balancing strategies are used. Throughput refers to the number of client requests that had been processed per unit time. Figure 7 shows the throughput chart of using three different load balancing strategies. The graph shows the value of the amount of received data from web server per second.

As is shown in Figure 7, the client gets about 375 KB data per second when SBLB algorithm is used. And the client could only get about 300 KB data when Round Robin or Random algorithm is used. It is clear that SBLB algorithm did improve the throughput of the system.

We also analyzed the CPU and memory utilization of each server by the three load balancing strategies to illustrate that SBLB algorithm could make better use of each server’s computing ability.

Figure 8 shows the four servers’ memory utilization when SBLB strategy is used.
As can be seen from Figures 8–10, when SBLB strategy is used, the servers’ memory utilization is closer. When Round Robin or Random algorithm is used, there is a gap in the servers’ memory utilization. It means that the policy can allocate user requests more evenly and make rational use of memory for each server.

Figure 11 shows the servers’ CPU utilization when SBLB strategy is used. Figure 12 shows the servers’ CPU utilization when Round Robin strategy is used. Figure 13 shows the servers’ CPU utilization when Random strategy is used. As can be seen from Figures 11–13, when SBLB algorithm is used, the servers’ CPU utilization is closer. When Round Robin or Random algorithm is used, there is a gap in the servers’ CPU utilization. It means that SBLB algorithm can allocate user requests better and make rational utilization of CPU for each server.
5. Conclusion

This paper described a server cluster dynamic load balancing method based on OpenFlow technology in virtual environment to solve the problem of how to take load balancing into network virtualization in data center. The experiments show that it is possible to make a powerful, flexible, and cost effective load balancing methods by OpenFlow. OpenFlow technology provides flexibility for the realization of different load balancing strategies, which makes it convenient to use software-defined method to achieve different load balancing strategies in different network environments. Experimental results show that, compared with the traditional algorithms, such as Round Robin and Random, the SBLB load balancing algorithm which we proposed can not only decrease the response time to provide a better user experience but also make the utilization of server CPU and memory more efficient and offer a more effective load balancing method.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] T. Benson, A. Anand, A. Akella, and M. Zhang, “Understanding data center traffic characteristics,” ACM SIGCOMM Computer Communication Review, vol. 40, no. 1, pp. 92–99, 2010.
[2] R. Schoonderwoerd, J. L. Bruten, O. E. Holland, and L. J. M. Rothkrantz, “Ant-based load balancing in telecommunications networks,” Adaptive Behavior, vol. 5, no. 2, pp. 169–207, 1997.
[3] V. Cardellini, M. Colajanni, and P. S. Yu, “Dynamic load balancing on web-server systems,” IEEE Internet Computing, vol. 3, no. 3, pp. 28–39, 1999.
[4] B. Raghavan, T. Koponen, A. Ghodsi, M. Casado, S. Ratnasamy, and S. Shenker, “Software-defined internet architecture: decoupling architecture from infrastructure,” in Proceedings of the 11th ACM Workshop on Hot Topics in Networks (HotNets’12), pp. 43–48, ACM, October 2012.
[5] N. McKeown, T. Anderson, H. Balakrishnan et al., “OpenFlow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.
[6] R. Wang, D. Butnariu, and J. Rexford, “OpenFlow-based server load balancing gone wild,” in Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE ’11), Boston, Mass, USA, April 2011.
[7] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the internet impasse through virtualization,” Computer, vol. 38, no. 4, pp. 34–41, 2005.
[8] M. Koerner and O. Kao, “Multiple service load-balancing with openflow,” in Proceedings of the IEEE 13th International Conference on High Performance Switching and Routing (HPSR ’12), pp. 210–214, Belgrade, Serbia, June 2012.
[9] H. Uppal and D. Brandon, “OpenFlow based load balancing,” Project Report CSE561: Networking, University of Washington, 2010.
[10] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: load-balancing web traffic using OpenFlow," in *Proceedings of the ACM SIGCOMM Conference on Data Communication (SIGCOMM ’09)*, Barcelona, Spain, 2009.

[11] L. Yu and D. Pan, “OpenFlow based load balancing for fat-tree networks with multipath support,” in *Proceedings of the 12th IEEE International Conference on Communications (ICC ’13)*, Budapest, Hungary, 2013.

[12] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: dynamic load-balanced routing in OpenFlow-enabled networks," in *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA ’13)*, pp. 290–297, IEEE, March 2013.

[13] A. Ghaffarinejad and V. R. Syrotiuk, "Load balancing in a campus network using software defined networking," in *Proceedings of the 3rd GENI Research and Educational Experiment Workshop (GREE ’14)*, IEEE, 2014.

[14] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks (HotSDN ’12)*, pp. 19–24, New York, NY, US, August 2012.

[15] Y. Murata, T. Inaba, H. Takizawa, and H. Kobayashi, "A distributed and cooperative load balancing mechanism for large-scale P2P systems," in *Proceedings of the International Symposium on Applications and the Internet Workshops (SAINT ’06)*, pp. 126–129, IEEE, January 2006.

[16] S. Adabala, V. Chadha, P. Chawla et al., "From virtualized resources to virtual computing grids: the in-VIGO system," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896–909, 2005.

[17] Specification, OpenFlow Switch. "v1.3.0.", 2012.
