Container-based Management Platform for Scalable Service Clusters Provisioning

Cheng-Che Lee^{a,*}, Fu-Min Chang^b, Shang-Juh Kao^c

^{a,c}National Chung Hsing University, Taichung 402, Taiwan

^bChaoyang University of Technology, Taichung 413, Taiwan

*Corresponding Author: g103056143@mail.nchu.edu.tw

Abstract

In virtualized network environment, OpenFlow is widely used for centralized operation control due to its novel feature of the separation between control plane and data plane. In a large scale of services cluster, data forwarding among multiple servers has to pass through a number of OpenFlow switches, which are deployed within physical hosts. In order to provide an efficient service support, tunnels for the switches within a cluster are usually developed. However, the tunneling approach may results in massive connections over the cluster, which will lead to the inflexibility of scaling out.

In this paper, a container-based management platform for scalable service clusters in Software-Define Networking (SDN) environment is proposed. Our idea is to direct all the connections of OpenFlow switches to the controller without tunneling among switches. We containerize application services, controller, switches, and deploy them in the cluster. This method can massively reduce the number of communication links. Furthermore, by taking the benefits of quickly scaling out services from adopting the containers, expanding a services cluster and speeding up the network re-configuration can be easily accomplished. We run a simulation to demonstrate the feasibility of the approach and evaluate the performance in terms of bandwidth and latency. The simulation indicate that our value is higher than other scalable solutions.

Keywords: SDN, OpenFlow, container

1. Introduction

In a SDN (Software-Defined Networking) environment, a common solution for communication between two virtual

machines located at different hosts is to build a tunnel. In case of a cluster with a large amount of switches, building a channel between each pair of virtual machines will result in complicated channel management. The complication makes the management to be infeasible or even endures an extra transmission latency, which will eventually lead to the inflexibility of scaling out.

Many tools and projects have been proposed to reduce or avoid the complexity of channel management. Calico⁽¹⁾, developed by Metaswitch Networks and released under the Apache 2.0 License, is an open source solution for virtual networking in cloud data centers. Calico builds a virtual network modeled as a single micro-segmented flat VLAN with shared IP address space. However, in Calico, the IP address has to be coordinated by provider, so user cannot migrate their environment into its network. In addition, due to no overlapping IP address offered in Calico's virtual network, different user cannot use the same IP address ranges. Flannel⁽²⁾, proposed by CoreOS, aims to solve the problem of a host cannot get an entire subnet to itself by creating an overlay mesh network that provisions a subnet to each server. By cooperating with etcd, key and storage mechanism, Flannel could sync the network configuration and rule settings. But it will generate another overlaid network and then become a more complicated environment. Another layer 3 tool, weave⁽³⁾, also solve the connection between hosts without tunneling. Weave would deploy a peer router on every host machine. It uses gossip message to discovery each other and establish TCP connection. After topology has discovered, it transmits the data payload with encapsulation by UDP protocol. However, the uniqueness of packet format and discovery algorithm make it unable to be integrated with SDN.

In this paper, we propose a container-based management platform in Software-Defined Networking (SDN) environment. Although SDN is centralized-oriented, we include an abstraction by de-centralizing the network according to the specific services. This allows a loosely coupled overlay of SDN environment over the backend network, so that service can be efficiently provisioned and the controller's workload can be mitigated. In order to decouple the service's network from the infrastructure network, we provision the application service with its own switch and controller which is not shared with other service unit. In other words, application service, controller and switches containers are integrated into a cluster and deployed at the same time. The controller of each cluster manages its own network. Such mechanism can enhance the flexibility of the deployment of SDN environment. To speed up the deployment, we adopt the container but not virtual machines. The reason is that container won't emulate the operate system above the host operating system. This property minimize the initial time to deploy a cluster including network device that we needed within seconds. While the application service is containerized, it can be deployed as container, so does the two basic SDN components, OpenFlow controller and Open vSwitch.

The rest of this paper proceeds as follows. The proposed management platform is discussed in section 2. After that, system implementation and performance evaluation are given in section 3. Finally, we draw the conclusions.

2. Container-based Management Platform

As we know, network communication devices usually have a higher view than the connecting end nodes. When the

network becomes complicated, the dynamic change of environment will affect the management operations of the entire network. By providing the services in separate clusters, we intend to downgrade the communication providers, such as switches and controller, into service providers in the proposed architecture. Within a cluster, the controller and switches deal with their service-associated network configurations only. This approach can improve the network independence of the service cluster and reduce the management complexity. The underlined environment could be a legacy network or a software-defined networking (SDN) network. And, through the provision of network independence, we are able to improve the portability of service cluster.

The proposed management system architecture is depicted in Fig.1. In the upper layer, the computing and network resources are virtualized in our container-based management platform, and every service cluster is isolated from each other. A service only exists when it is requested, and would be wiped out as soon as it completes. During this request-completion process, the beneath network behaves the same without any configuration change. In the lower layer, one switch has to be deployed for each host providing the service. Only one controller is necessary for each service cluster. However, more controllers may be included for redundancy purpose. The role of switch is to offer the communications for providing the service, either at the same host or not. Each switch connects to the controller without building mutual-switch tunnels. The controller installs the policy rules to determine routing alternations. Within a cluster, the network device container is at the same level as all application containers. Consequently, all service clusters are mutually independent. Each service user needs only to know the service gateway. In addition, containerizing

Fig. 1. System architecture.

Fig. 2. Experiment environment

network services make the system to be easily implemented, hence it is flexible to cope with the addition of services.

3. System implementation and performance evaluation

3.1 Experiment environment

In order to verify the feasibility of proposed architecture, we build up an experimental environment as depicted in Fig.2. The experimental environment consists of two physical hosts, which the operating system is Ubuntu 14.04. We adopt the Docker⁽⁴⁾, version 1.9, as our container. Docker has its convenient overlay network driver recently, for the flexibly to be applied to virtual machines and upstream compatibility with typical SDNs environment. We also reuse the controller and the switch, which is almost identical besides different configurations. Once an image of Docker is built, it would push to a public or private hub, and pull again whenever a specified image is needed. It's suitable for our system to deploy the network device as soon as the service is requested.

We deploy Ryu controller with version 3.29. The chosen of controller could be an issue. Different type of controller can uses an interface to deliver an integrated process for SDN deployment⁽⁵⁾. In our architecture, we adopt the Ryu controller for a lightweight development environment. Although OpenDayLight is also a popular controller, but it is more suitable to develop a function module inside the controller.

Besides, we use the Open vSwitch version 2.4 and MacVLAN between container and hosts. Without making use of the default Docker bridge "Docker0", we let all the service containers connect to Open vSwitch only. The normal service containers connect to Open vSwitch simply

by the link, which is created by net name space. Open vSwitch container can communicate beyond the host by MacVLAN. In each host, we bridge the MacVLAN to share the network in same cluster and isolate the network interface from others. Consequently, the tunnel between two local hosts does not need anymore. In addition, in this architecture, all the packets of service containers have to be forwarded by the switch to another service. Benefiting from this property, it is more convenient to monitor and apply the policy to the network by installing rules.

3.2 Predefined SDN

Because our system would pass through more switch than the conventional usage, and switch would communicate with controller first. To overcome the initial system delay, besides containerizing the Open vSwitch, we install the basic rules for switching in the cluster before we containerized it. This save the startup time because switch wouldn't needed to send a packet to ask the controller how to deliver the traffic. Until more complicated situation occurs, controller would compute the path with its overview of cluster.

Furthermore, we enhance the isolation and extensibility by change the MAC Address of packets. MAC Address management is more flexible in the virtual environment, we modified the MAC Address when the packet sending from application containers entering the switch, and modified back to its original MAC Address while it is about to reach the destination . By doing this, wherever the packet has been though, we can recognize it by the MAC Address. And we can define the meaning of MAC Address format, for example, most significant 3 bytes of MAC Address stands for organizational unique identifier (OUI) in the physical network environment, which is changeable in our approach.

There are other field in the packet format and tags can

be added in the OpenFlow protocol such as MPLS (*MPLS with a Simple OPEN Control Plane*) or vlan tags. MPLS tag can inherit the traditional properties like MPLS traffic engineering or MPLS VPN (*MPLS-TE and MPLS VPNs with OpenFlow*), but it need to modified the controller and the switch. Vlan tag is common usage for isolated the network, however, it will increase the size of frame. And the vlan ID is restricted by 4096 which is not enough in cloud environment, so the network of EHU⁽¹⁰⁾ OpenFlow Enabled Facility (EHU-OEF) solved this problem by defined its own Group ID.

(a) Bandwidth.

(b) Latency.

Fig. 3. Evaluation of Bandwidth and Latency

3.3 Performance Evaluation

For the purpose of performance comparison, we evaluate the proposed system MacVLAN equipping with an Open vSwitch container in each of two physical hosts and examine the performance of network bandwidth and latency. These performance metrics are measured by qperf⁽¹¹⁾ software, which is a well-known network performance testing tool. The target approaches for comparison include Weave, Flannel, and Project Calico. All tests are conducted between two containers in different hosts. One container

plays the role as the qperf server and the other one acts as a client. The test is initiated by the client with duration of 300 seconds and packet size of 512KB.

We take into account of network bandwidth and latency for performance evaluation. Bandwidth refers to the data transfer rate, MB/s, from the qperf-client to the qperf_server. Latency refers to the round trip time between both qperf containers. In the experiment, both performance metrics are calculated over a period of 300 seconds. Higher bandwidth implies better performance, but larger latency indicates longer packet transmission time, which includes data transfer and data handling cost. The comparison results are shown in Fig. 3. From the figure, MacVLAN significantly outperforms the weave and flannel in the bandwidth performance, but is worse than Calico. However, Calico needs more CPU consumption in the case of smaller packet size⁽¹²⁾, as adopted in our test. As of the latency test, MacVLAN is lower than Flannel and Calico, but is higher than weave. That's because weave enables automatic link discovery whenever the route of peer communications has been determined. Hence, the packet processing time along the path could be reduced.

4. Conclusions

In this paper, instead of dedicatedly installing the controller and switches on the hosts, we containerized them into the cluster. With our approach, the number of communication links within a cluster can be reduced massively. In addition, due to the ease of scaling out feature by adopting containers, the network configuration can be simplified when expanding a services cluster. By using the software Oper vSwitch, Docker, and MacVLAN, we built up an experiment environment and perform a simulation to demonstrate the feasibility of the approach. The simulation results revealed that our approach outperformed in overall system performance as compared with other scalable network solutions for containers such as Weave, Flannel, and Project Calico.

References

- (1) "Calico." [Online]. Available: www.projectcalico.org
- (2) "Flannel." [Online]. Available: <https://coreos.com/blog/introducing-rudder/>
- (3) "weave." [Online]. Available: <http://weave.works/>
- (4) "Docker." [Online]. Available: www.docker.com/

- (5) "Ryu." [Online]. Available: <https://osrg.github.io/ryu/>
- (6) Stritzke, Christian, Claudia Priesterjahn, and Pedro A. Aranda Gutierrez. "Towards a Method for end-to-end SDN App Development." *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*. IEEE, 2015.
- (7) "OpenDayLights." [Online]. Available: <https://www.opendaylight.org/>
- (8) Das, Saurav, et al. "MPLS with a simple OPEN control plane." *Optical Fiber Communication Conference*. Optical Society of America, 2011.
- (9) Sharafat, Ali Reza, et al. "Mpls-te and mpls vpns with openflow." *ACM SIGCOMM Computer Communication Review*. Vol. 41. No. 4. ACM, 2011.
- (10) Matias, Jon, et al. "Implementing layer 2 network virtualization using OpenFlow: Challenges and solutions." *Software Defined Networking (EWSDN), 2012 European Workshop on*. IEEE, 2012.
- (11) "qperf." [Online]. Available: linux.die.net/man/1/qperf
- (12) "calico-dataplane-performance" [Online]. Available: <http://www.projectcalico.org/calico-dataplane-performance/>