A Study on Portable Load Balancer for Container Clusters

Kimitoshi Takahashi
The Graduate University for Advanced Studies
Chiyoda-ku, Tokyo, Japan
ktaka@nii.ac.jp

1 INTRODUCTION

Recently, Linux containers have drawn significant amount of attention because they are lightweight, portable, and repeatable. Linux containers are generally more lightweight than virtual machine (VM) clusters, because the containers share the kernel with the host operating system (OS), even though they maintain separate execution environments. They are generally portable because the process execution environments are archived into tar files, so whenever one attempts to run a container, the exact same file systems are restored from the archives even when totally different data centers are used. This means that containers can provide repeatable and portable execution environments. For the same reasons, Linux containers are attractive for web services as well, and it is expected that web services consisting of container clusters would be capable of being migrated easily for variety of purposes. For example disaster recovery, cost performance improvements, legal compliance, and shortening the geographical distance to customers are the main concerns for web service providers in e-commerce, gaming, Financial technology(Fintech) and Internet of Things(IoT) field.

Several container cluster management systems, including kuberenates, docker swarm etc., have been proposed, which enable easy deployment of container clusters. If these container cluster management systems could hide the differences in the base environments, users would be able to easily deploy a web service on different cloud providers or on on-premise data centers, without adjusting the container cluster configurations to the new environment. This allows a user to smoothly migrate a web service consisting of a container cluster even to the other side of the world.

However, in actuality, the current container management systems fail to provide uniform environments across the different cloud providers. One of the most prominent differences they fail to hide is the way they route the incoming traffic into the container cluster.

For example in the case of the Kubernetes, a load balancer is not included in a cluster management system, and is heavily dependent on external load balancers that are set up on the fly by cloud providers through their application protocol interfaces (APIs). These external load balancers distribute incoming traffic to every server that hosts containers. The traffic is then distributed again to destination containers using iptables destination network address translation (DNAT)[1, 2] rules in a round-robin manner. The problem happens in the environment with a load balancer that is not supported by the Kubernetes, e.g. in an on-premise data center with a bare metal load balancer. In such environments, the user needs to manually configure the static route for inbound traffic in an ad-hoc manner.

In order to address this issue, the author believes a rigorous study on providing the uniform environment for incoming traffic is critical. If we have the uniform environment across the different cloud providers including on-premise data centers, users can smoothly migrate their web services without adjusting them to the base environment. This means that there is no cloud vendor lock-ins and users have the freedom as to where to run their web services.

In order to address this issue, the author believes a rigorous study on providing the uniform environment for incoming traffic is critical. If we have the uniform environment across the different cloud providers including on-premise data centers, users can smoothly migrate their web services without adjusting them to the base environment. This means that there are no cloud vendor lock-ins and users will gain the freedom as to where to run their web services.

The goals of this study are the followings: 1) To clarify what type of the architecture for incoming traffic is feasible for web service migration. 2) To provide a software load balancer for such architecture. 3) To provide working prototype system for web service migration.

The rest of the paper is organized as follows. software load balancer containerization, and load balancer related tools within the context of the container technology. that we considered to be important in our experiment will be described in detail.

2 RELATED WORK

3 COMPARISON BETWEEN THE DIFFERENT ARCHITECTURE

The typical traditional on-premise data center used a set of load balancers to distribute the incoming traffic to cluster of servers thus providig scalability and redundancy of the service. The traffic is routed from the out side of the infrastructure to one of the load balancers using active-backup redundancy protocol OSPF, VRRP or HSRP. The traffic is then distributed to multiple of web servers.

redundant layer 4 swithes with redundatxxi This section discuss the different architecture.

In retrospect, either Google type config or

4 CONTAINERISED IPVS SOFTWARE LOAD BALANCER

The author and his collegue proposed a portable load balancer for the Kubernetes cluster systems that is aimed at facilitating migration of container clusters for web services.

We implemented a containerized software load balancer that is run by Kubernetes as a part of container cluster, using Linux kernel's IPVS, as a proof of concept. In order to discuss the feasibility of the proposed load balancer, we built a Kubernetes cluster system and conducted performance measurements. Our experimental results indicate that the IPVS based load balancer in container improves the portability of the Kubernetes cluster system while it shows the similar performance levels as the existing iptables DNAT based load balancer. We also clarified that choosing the right operating modes of overlay networks is important for the performance of load balancers. For example, in the case of flannel, only the vxlan and udp backend operation modes could be used in the cloud environment, and the udp backend significantly degraded their performance. Furthermore, we also learned that the distribution of packet processing among multiple CPUs was very important to obtain the maximum performance levels from load balancers.

The limitations of this work that authors aware of include the followings: 1) We have not discussed the load balancer redundancy. Routing traffic to one of the load balancers while keeping redundancy in the container environment is a complex issue, because standard Layer 2 rendandacy protocols, e.g. VRRP or OSPF[3] that uses multicast, can not be used in many cases. Further more, providing uniform methods independent of various cloud environments and on-premise datacenter is much more difficult. 2) Experiments are conducted only in a 1Gbps network environment. The experimental results indicate the performance of IPVS may be limited by the network bandwidth, 1Gbps, in our experiments. Thus, experiments with the faster network setting, e.g. 10Gigabit ethernet, are needed to investigate the feasibility of the proposed load balancer. 3) We have not yet compared the performance level of proposed load balance with those of cloud provider's load balancers. It shoud be fair to compare the performance of proposed load balancer with those of the combination of the cloud load balancer and the iptables DNAT. The authors leave these issues for future work and they will be discussed elsewhere.

Even though we could successfully containerised the load balancers and could deploy it as a part of web service, the performance we was not equivalent that of scalable load balancers provided by the GCP.

The next section will disscuss how to improve the performance levels of the open source software load balancer.

5 XDP LOADBALANCER

Comparison XDP, DPDK, ipvs, iptables.

DPDK: pros: bypass kernel network stack cons: only for Intel NIC, dedicated NIC required XDP: pros: hooks to the NIC driver redirect

6 CONCLUSION

aaa

7 FUTURE WORK

bbb

REFERENCES

[1] Victor Marmol, Rohit Jnagal, and Tim Hockin. 2015. Networking in Containers and Container Clusters. *Netdev* (2015).
[2] Martin A. Brown. 2007. Guide to IP Layer Network Administration with Linux. (2007), 5.5. Destination NAT with netfilter (DNAT) pages. http://linux-ip.net/html/index.html
[3] John Moy. 1997. OSPF version 2. (1997).