A Study on Portable Load Balancer for Container Clusters

by

Kimitoshi Takahashi

Dissertation

submitted to the Department of Informatics

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

SOKENDAI (The Graduate University for Advanced Studies)

March 2019

Committee

Acknowledgments

I would like to thank ...

Abstract

This is a PhD thesis template for Sokendai students. Use *pdflatex* to build latex files. This file specifies the following:

- Font,
- Appearance of Text (Hyphenation and Letter-spacing),
- Page Layout,
- Theorem Environments,
- Header and Footer,
- Chapter Header,
- Table of Contents,
- Clickable PDF,
- Title Page and PDF-title.

Linux containers have become very popular these days due to their lightweight nature and portability. Numerous web services are now deployed as clusters of containers. Kubernetes is a popular container management system that enables users to deploy such web services easily, and hence, it facilitates web service migration to the other side of the world. However, since Kubernetes relies on external load balancers provided by cloud providers, it is difficult to use in environments where there are no supported load balancers. This is particularly true for on-premise data centers, or for all but the largest cloud providers. In this paper, we proposed a portable load balancer that was usable in any environment, and hence facilitated web services migration. We implemented a containerized software load balancer that is run by Kubernetes as a part of container cluster, using Linux kernel's Internet Protocol Virtual Server(IPVS). Then we compared the performance of our proposed load balancer with existing iptables Destination Network Address Translation (DNAT) and the Nginx load balancers. During our experiments, we also clarified the importance of two network conditions to derive the best performance: the first was the choice of the overlay network operation mode, and the second was distributing packet processing to multiple cores. The results indicated that our proposed IPVS load balancer improved portability of web services without sacrificing the performance.

Linux container technology and clusters of the containers are expected to make web services consisting of multiple web servers and a load balancer portable, and thus realize easy migration of web services across the different cloud providers and on-premise datacenters. This prevents service to be locked-in a single cloud provider or a single location and enables users to meet their business needs, e.g., preparing for a natural disaster. However existing container management systems lack the generic implementation to route the traffic from the internet into the web service consisting of container clusters. For example, Kubernetes, which is one of the most popular container management systems, is heavily dependent on cloud load balancers. If users use unsupported cloud providers or on-premise datacenters, it is up to users to route the traffic into their cluster while keeping the redundancy and scalability. This means that users could easily be locked-in the major cloud providers including GCP, AWS, and Azure. In this paper, we propose an architecture for a group of containerized load balancers with ECMP redundancy. We containerize Linux ipvs and exabgp, and then implement an experimental system using standard Linux boxes and open source software. We also reveal that our proposed system properly route the traffics with redundancy. Our proposed load balancers are usable even if the infrastructure does not have supported load balancers by Kubernetes and thus free users from lock-ins.

Contents

List of Figures

List of Tables

1 Introduction

Recently, launching web services on cloud computing infrastructure is getting more popular. Launching web services on the cloud is easier than launching them on on-premise data centers. A web service on the cloud becomes scalable, which means it can accommodate a large amount of traffic from the Internet by increasing the number of web servers, on demand. To distribute high volume traffic from the Internet to thousands of web servers load balancers are often used. Major cloud providers have developed software load balancers[?, ?] as part of their infrastructures, which they claim to have a high-performance level and scalability. In the case of on-premise data centers, one can use proprietary hardware load balancers. The actual implementation and the performance level of those existing load balancers are very different.

As for another issue regarding web services, there are also needs to use multiple of cloud providers or on-premise data centers seamlessly, which spread across the world, to prepare for the disaster, to lower the cost or to comply with the legal requirement. Linux container technology[?] facilitates these usages by providing container cluster management system as a middleware, where one can deploy a web service that consists of a cluster of containers without modification even on different infrastructures. However, in reality, it is difficult to do so, because existing load balancers are very different and are not included in the container cluster management system. Therefore, users should always manually adjust their web services to the infrastructure.

In short, there exist several software load balancers that are specific to cloud vendors and on-premise data centers. The differences among them are the major obstacles to provide uniform container cluster platform, which is necessary to realize web service migration across the different cloud providers and on-premise data centers.

To address this problem, we propose a portable and scalable software load balancer that can be used in any environment including cloud providers and in on-premise data centers. We can include this load balancer as a part of a container cluster management system, e.g., Kubernetes[?], which acts as a common middleware on which web services run. Users now do not need manual adjustment of their services to the infrastructures. We will implement the proposed software load balancer using following technologies; 1) To make the load balancer usable in any environment, we containerize ipvs[?] using Linux container technology[?]. 2) To make the load balancer scalable, we make it capable of being run in parallel using Equal Cost Multi-Path(ECMP) technique[?]. In order to make the load balancer's performance level to meet the need for 10Gbps network speed, a software load balancer that better performs than ipvs is required. We also implement the novel load balancer using eXpress Data Plane(XDP) technology[?].

The outcome of our study will benefit users who want to deploy their web services on any cloud provider where no scalable load balancer is provided, to achieve high scalability. Moreover, the result of our study will potentially benefit users who want to use a group of different cloud providers and on-premise data centers across the globe, as if it were a single computer on which their web services run.

The rest of the paper is organized as follows. Section 2.1 highlights work that deals specifically with container cluster migration, software load balancer containerization, and load balancer related tools within the context of the container technology. Section **??** will explain existing architecture problems and propose our solutions. In Section **??**, experimental results for containerized ipvs are discussed, which is followed by a summary of our work in Section **??**.

2 Related work

2.1 Related Work

This section highlights related work, especially that dealing with container cluster migration, software load balancer containerization, load balancer tools within the context of the container technology and scalable load balancer in the cloud providers.

Container cluster migration: Kubernetes developers are trying to add federation[?] capability for handling situations where multiple Kubernetes clusters[1] are deployed on multiple cloud providers or on-premise data centers, and are managed via the Kubernetes federation API server (federation-apiserver). However, how each Kubernetes cluster is run on different types of cloud providers and/or on-premise data centers, especially when the load balancers of such environments are not supported by Kubernetes, seems beyond the scope of that project. The main scope of this paper is

[1]The *Kubernetes cluster* refers to a server cluster controlled by the Kubernetes container management system, in this paper.

to make Kubernetes usable in environments without supported load balancers by providing a containerized software load balancer.

Software load balancer containerization: As far as load balancer containerization is concerned, the following related work has been identified: Nginx-ingress[?, ?] utilizes the ingress[?] capability of Kubernetes, to implement containerized Nginx proxy as a load balancer. Nginx itself is famous as a high-performance web server program that also has the functionality of a Layer-7 load balancer. Nginx is capable of handling Transport Layer Security(TLS) encryption, as well as Uniform Resource Identifier(URI) based switching. However, the flip side of Nginx is that it is much slower than Layer-4 switching. We compared the performance between Nginx as a load balancer and our proposed load balancer in this paper. Meanwhile, the kube-keepalived-vip[?] project is trying to use Linux kernel's IPVS[?] load balancer capabilities by containerizing the keepalived[?]. The kernel IPVS function is set up in the host OS's net name spaces and is shared among multiple web services, as if it is part of the Kubernetes cluster infrastructure. Our approach differs in that the IPVS rules are set up in container's net name spaces and function as a part of the web service container cluster itself. The load balancers are configurable one by one, and are movable with the cluster once the migration is needed. The kube-keepalived-vip's approach lacks flexibility and portability whereas ours provide them. The swarm mode of the Docker[?, ?] also uses IPVS for internal load balancing, but it is also considered as part of Docker swarm infrastructure, and thus lacks the portability that our proposal aims to provide.

Load balancer tools in the container context: There are several other projects where efforts have been made to utilize IPVS in the context of container environment. For example, GORB[?] and clusterf[?] are daemons that setup IPVS rules in the kernel inside the Docker container. They utilize running container information stored in key-value storages like Core OS etcd[?] and HashiCorp's Consul[?]. Although these were usable to implement a containerized load balancer in our proposal, we did not use them, since Kubernetes ingress framework already provided the methods to retrieve running container information through standard API.

Cloud load balancers: As far as the cloud load balancers are concerned, two articles have been identified. Google's maglev[?] is a software load balancer used in Google Cloud Platform(GCP)[?]. Maglev uses modern technologies including per flow ECMP and kernel bypass for userspace packet processing. Maglev serves as the GCP's load balancer that is used by the Kubernetes. Maglev can be solely used in GCP, and the users need open source software load balancer that is runnable even in on-premise data centers. Microsoft's Ananta[?] is another software load balancer implementation using ECMP and windows network stack. Ananta can be solely used in Microsoft's Azure cloud infrastructure[?]. The proposed load balancer by the author is different in that it is aimed to be used in every cloud provider and on-premise data centers.

7

Bibliography