Review on "A Study on Portable Load Balancer for Container Clusters" by Kimitoshi Takahashi

Danci Laurentiu-Cristian

December 12, 2021

Web services include search engines, social networks, email, blogs, video streaming and so on. Usually, a web application "lives" in a data center where multiple servers work together to process a client's request. As the number of users increases, so does the number of servers. Those collections of servers constitute a cluster. A load balancer is a specialized server that distribute user's request to multiple servers.

A web application should be portable - meaning that migrating it to another server should be fast and easy. If an application is portable, it is easy to add another cluster closer to a new country where a company wants to expand, or it is easy to move the application in case of disasters.

An ideal infrastructure for web services probably has the following features: a common middleware to manage the clusters, is capable to store data globally in a consistent storage and can routes traffic based on proximity to the client.

The paper addresses other information on web infrastructures, as per on-premise data centers where new servers are purchased as necessary. But the most relevant bits of information are on cloud computing. Basically, cloud computing uses virtual machines (VMs). Because a VM only uses a fraction of the physical server resources, a client can pay only for the resources used by his machines, thus the costs are lowered.

Recently, containers draw a significant amount of attention. A container is a process that runs in a separate environment, so two containers cannot see each other. All containers share the same hardware and the same kernel, but each container has its own resources (its own root filesystem, network device and so on). Simply, a container sits on top the of the host operating system and OS's kernel is shared (read only). On the other hand, a VM sits on top of an emulated hardware and has its own OS installed on it. The obvious advantage of a container is that it uses less space and it does not have the overhead of emulating a specific hardware architecture. Containers should behave exactly when ran multiple times on different hosts. Containers can have their own libraries and binaries independent from the host OS. Those benefits increase the portability of a web application - a web app can be deployed in different cloud infrastructures and should behave exactly the same.

A container orchestrator is a tool that manages clusters of containers. It launches new containers and creates internal routes between them in a scalable and redundant manner. Starting and running an orchestrator that manages multiple container clusters is simple as running a single command or program. Kubernetes is an orchestrator for containers.

To implement load balancing in containers, a proposed architecture could have the following characteristics: a cluster of load balancer containers (for redundancy and scalability), each load balancer is running as a pod and load balancing rules are dynamically updated based on the information about running pods. In Kubernetes terminology, a pod is a group of one or more containers. Kubernetes can restart any failed container.

Containerized load balancers can run in any environment - it is portable - without external load balancers.

IPVS load balancer is used to distribute incoming traffic for a single destination IP address to multiple servers. This load balancer usually works in two modes: NAT and tunneling. When working in Network Address Translation (NAT) mode, a request packet destined for an IP is analyzed by the load balancer. When a server is selected based on the scheduling algorithm, the original destination IP and port number are rewritten and forwarded to the right server. When the response is returned, the load balancer rewrites the source address and port number and send it back the the client. When used in tunneling mode, the requests are redirected to another IP address and the servers can send the response directly to the client - the load balancer encapsulates the original packet within a new packet and the servers decapsulate it and send the response directly to the client. The paper addresses the performance of those two: in a 10 Gbps network, the IPVS plateaus at roughly 300k requests per second, while IPVS tunneling has a throughput of over 700k requests. The performance difference is large because tunneling does not sends the responses back though the load balancer thus the overhead is eliminated.

Simply, a load balancer checks if a packet goes towards a IP that is a VIP (Virtual IP) and picks a real server.

IPVS balancer, that is included in the Linux kernel, can be used to implement a load balancer in a portable container. The other two components of the container are keepalived and a controller. All those components can be placed in a single container. Keepalived is a management program that performs health checks and manages IPVS balancing rules. Those rules are used for forwarding requests - the client send requests to a virtual IP and those are forwarded to real servers. The controller is a daemon that monitors the pod information (by consulting a master node) and performs actions when needed. Then, if a node is failing (the health checks fails), keepalived will remove the corresponding server from the IPVS rules. The controller also updates the rules when a container is created or deleted. The proposed load balancer improve the portability of a web application because the entire infrastructure can be managed by Kubernetes.

Routing the traffic to the load balancing containers is done by the router (the physical device that is connected to the internet). It distributes the traffic to multiple next hops (the load balancers) based on a hash of the following items: source IP, destination IP, source port, destination port and protocol - from what I understand, this hash is necessary so multiple packets from the same client are redirected to the same next hop (a single request can be made out of multiple packets and all packets must get to the same node). The multiple next hops have equal priority. This routing strategy is named Equal-cost multi-path routing (ECMP) and adds redundancy. This implies that the overall throughput is limited by the performance of the router. But it is worth to scale up the load balancer containers when the router is powerful and can handle a lot of traffic - for example, a single container may not have the same throughput as the router.

Kubernetes have their own load balancing features: Ingress that is used to expose HTTP routes from services and can do load balancing.

Other software load balancing for Kubernetes utilize ingress capability of Kubernetes. For example, Nginx-Ingress implements containerized Nginx proxies as load balancers. Nginx is a high performance web server that has the functionality of a layer-7 load balancer (application layer) - but it is slower than layer-4 load balancers (transport layer).

Large companies have developed their own cloud load balancing technologies. For example, Google uses Maglev for its GCP (Google Cloud Platform) but it cannot be used outside of those companies infrastructures and it is closed source. Other example is Facebook's Katran, a layer 4 load balancer.

Some benchmarks described in the paper show that, in case of containerized load balancers, the network setup for the containers adds an overhead - mainly the veth (virtual Ethernet devices), but containerized load balancers are deployed as a part of the web application which makes them more portable and easier to deploy on other infrastructures.

In conclusion, the paper presented related work regarding many aspects of web applications and software load balancers and proposed a portable load balancer for containers that works with Kubernetes. The load balancer differs from other because it is a part of the application. This way, issues can be resolved by the developers instead of the cluster administrators. Cloud load balancers tend to have the best performance, but those does not use standard technologies.