an LFNETWORKING project

Yahoo! JAPAN Solutions Brief

FD.io Powered Scalable L4 Load Balancer

Yahoo! JAPAN addresses infrastructure scalability challenges through the adoption of commodity x86 servers combined with the Linux Foundation Networking Fast Data Project (FD.io).

This solution brief outlines key design choices and required functionality in the move from a traditional Enterprise load balancing implementation to an open software-based, Cloud-native scalable, load balancing implementation running on commodity x86 servers.

COMPANY

- 100+ Services
- 70+ Billion monthly page views
- 90+ Million daily browsers
- 60+ Million daily smart phone sessions

CHALLENGE

- Variety of hardware load balancers to support services
- 140K+ VMs
- Robust scalable LBaaS (Load Balancer as a Service)
- Hardware upgrades and downtime

SOLUTION

- L3DSR L4 software load balancer
- Scaling-in/out (N+1) LB capability on top of Clos
- Simple integration with BGP

BENEFITS

- Commodity x86 servers
- Built on open source with FD.io VPP data plane
- Sufficient wire performance
- Zero downtime for upgrades

“The rapid increase of service endpoints and traffic coming into our large-scale data center in conjunction with constant endpoint changes (e.g., bare-metal to VM or container) resulted in a need to re-architect our load balancing infrastructure to address scalability and on-demand deployment – without downtime. To respond to this challenge – and achieve networking agility – we built a scalable software L4 load balancer cluster and management system based on FD.io VPP (Vector Packet Processing). This solution provides not only wire-rate fast and scalable forwarding performance, but also fast deployment and stable operation of a LB cluster with zero downtime on a Clos architecture for data center.”

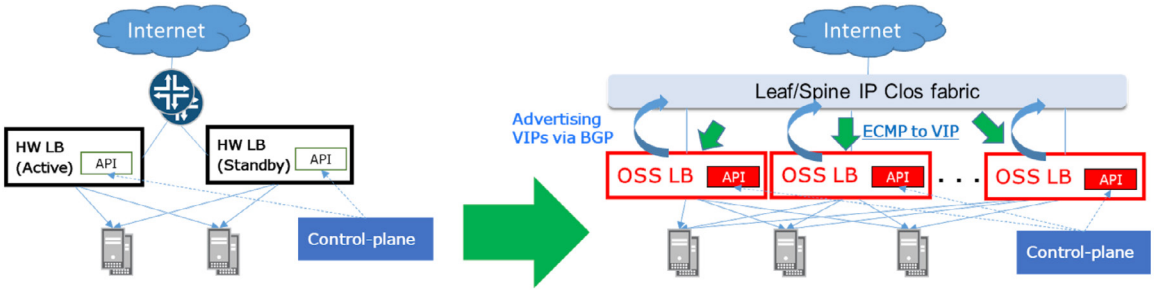
Yusuke Tatsumi, network infrastructure engineer at Yahoo! JAPAN

SOLUTION OVERVIEW

Our scalable cloud native L4 load balancer is built from the following components: commodity x86 servers, Linux, FD.io VPP-based load balancer nodes, and a centralized custom-built load balancer controller.

Given that traffic from our services back to the clients' is 2x – 10x greater than that coming from clients, we went with a load balancer choice of L3 Direct Server Return (L3DSR). This approach avoids the tunneling overhead associated with unused DSCP field - providing significant bandwidth savings.

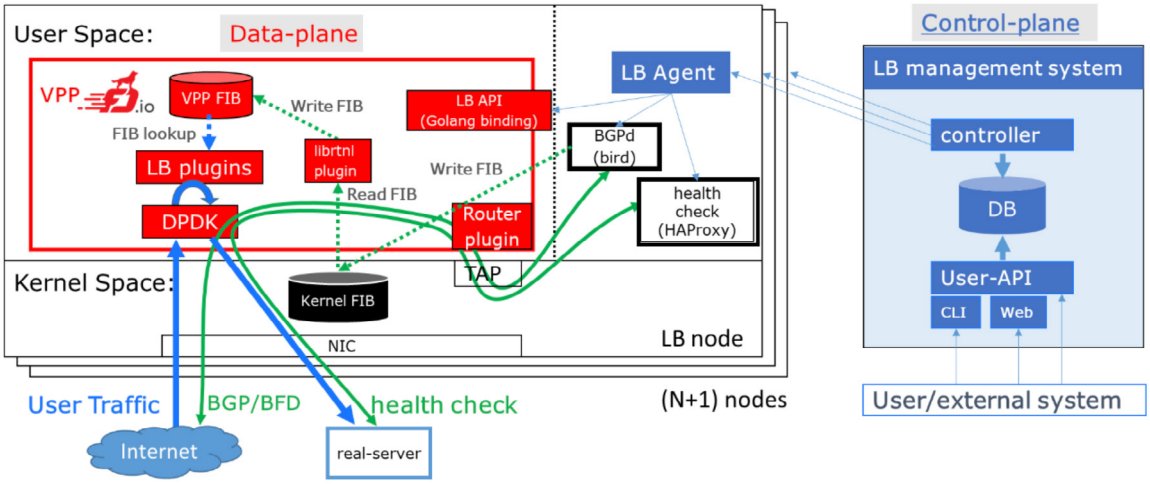
PRE AND POST SOLUTION
HIGH LEVEL DIAGRAMS



Current Infrastructure (Enterprise approach)	Post Solution Infrastructure (Cloud native approach)
☐ Active-backup (2N) LB capability on top MLAG	☐ Scaling-in/out (N+1) LB capability on top Clos
☐ Hard to scale-out	☐ Commodity server/NIC + OSS (White box)
☐ Vendor proprietary (Black box)	☐ Simpler operation to add/change LB with BGP integration
☐ Lead time & End of Life concerns	
☐ Complex operation (plan/exec) to add capacity	

Our solution has two major components: the data plane LB nodes and the centralized controller.

SOLUTION HIGH LEVEL DIAGRAM



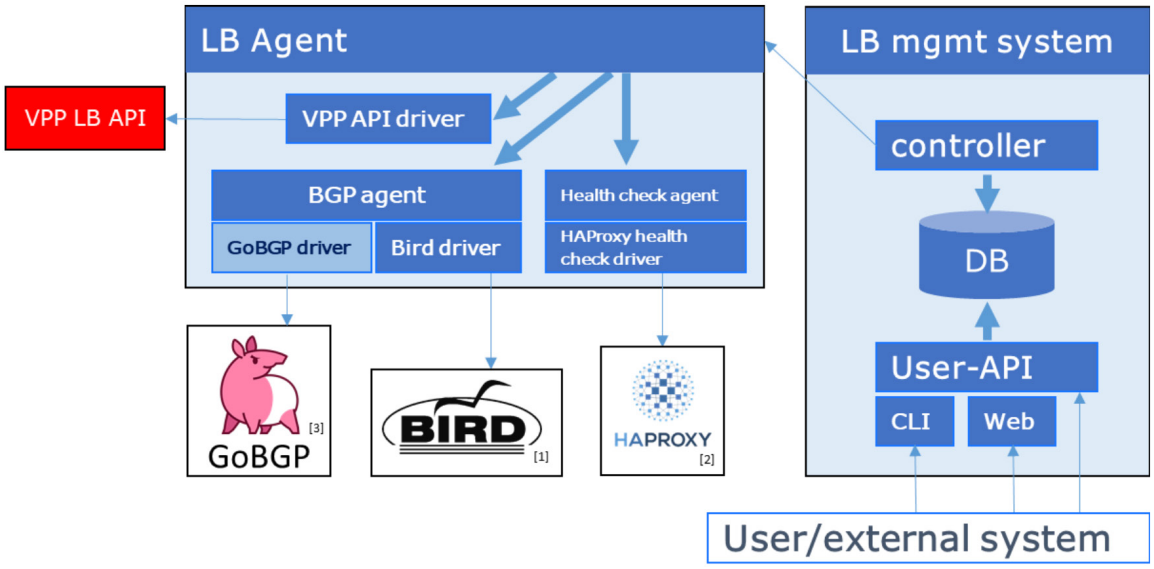
The load balancer packet forwarding plane is built as a plug-in to the foundation provided by the FD.io VPP project. The use of plug-ins also facilitates connecting our load balancer data plane to other networking OS services, such as BIRD for routing, and HAProxy for failure detection.

Working with the FD.io community, we were able to quickly arrive at a design approach, identify missing functionality across the base framework, protocol support, and management API. The VPP project provided a great foundation for our load balancer. We were able to build on a highly-scalable framework and the richness of networking protocol support. The following table identifies a few key existing components, modifications, and additions to the VPP project necessary to complete our load balancer requirements.

Category	Feature	Description
Framework	Consistent Hash	Brings great scalability and redundancy for LB nodes
	Multi-Threading	Performance scalability with CPU cores
Protocols	GRE4/GRE6	Encap with GRE IPv4 and IPv6
	L3DSR *	Layer 3 Direct Server Return
	NAT4/NAT6	NAT
	Port # aware *	Care TCP/UDP port number for LBaaS integration
API	Manipulation *	Set/Get VIP/member over API from controller
	Statistics *	Collect more detailed statistics like number of connections

* *Added by this project*

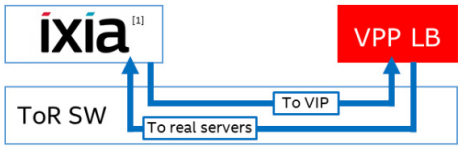
In considering existing controller options within an OpenStack environment, we found them to be either overly complex, or lacking in functionality. As a result, we built the load balancer controller from scratch. It is written completely in Go with a total of 22K lines of code. High-level components are shown in the following diagram:

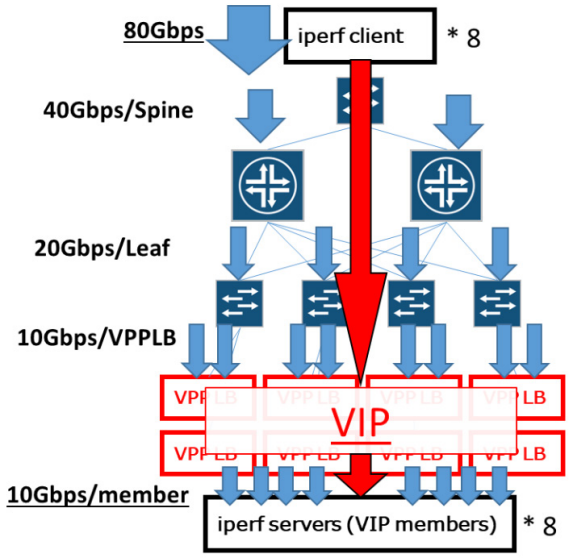


SOLUTION PERFORMANCE

Performance evaluation focused on single node performance, as well as scalability via additional load balancer nodes. A single node produces 5.5Mpps (84B IP packet) per CPU core. More interestingly, that performance scales linearly based on multi-threaded design. As a result, 4 CPU cores are enough to serve a 10G wire rate – even for small packet sizes. In addition to core scaling, node scaling was confirmed by creating an 8 node LB cluster, which yielded over 80 Gbps of throughput. The design approach introduces no additional overhead as nodes are added. Currently, Yahoo! JAPAN deployed this solution with an initial cluster size of 16 nodes.

- **Traffic generator: Ixia IxNetwork**
- **VPP Server Spec:**
  - **CPU: Intel® Xeon® Processor E5-2650L v3 * 2S**
  - **Memory: 384GB**
  - **NIC: Intel® Ethernet Network Connection X540-AT2 (10Gbps*2)**





SUMMARY

The objective of modernizing Yahoo! JAPAN's network load balancing infrastructure was successfully achieved - leveraging both FD.io VPP and its community. We found the FD.io community both easy to work with, and instrumental in resolving real-world networking challenges required for a deployable solution. For more insight into this use case, please view our ONS North America presentation and engage the FD.io community.