

Scalable Linux Clusters with LVS

Considerations and Implementation, Part I

Eric Searcy
Tag1 Consulting, Inc.
emsearcy@tag1consulting.com

April 2008

Abstract

Whether you are perusing mailing lists or reading the large LVS-HOWTO, much of the available information on creating a high-availability Linux cluster can be difficult to sift through. Relevant information shares space with obscure comments and implementation details that date back to version 2.0 of the Linux kernel.

This two-part article attempts to present an up-to-date tutorial on setting up LVS for your network, with an emphasis on designing a system that matches your infrastructure's needs.

1 Overview

Linux Virtual Server¹ is a fast, scalable way to load balance critical parts of your network, be it web, email, or application services. Combined with heartbeat², what may have been a single point-of-failure service can be quickly scaled up to be a fault tolerant, distributed service backed by several nodes on your network.

This article assumes a fairly in-depth understanding of networking as well as a familiarity with the Linux command line.³ Also, much of the implementation details vary depending on your distribution of Linux, so you should have a fairly good grasp of systems administration, such as package management and service configuration.

¹<http://www.linuxvirtualserver.org/>

²<http://www.linux-ha.org/>

³If you need a good reference on networking concepts from a Linux, command-line perspective, I recommend Scott Mann's *Linux TCP/IP Network Administration*.

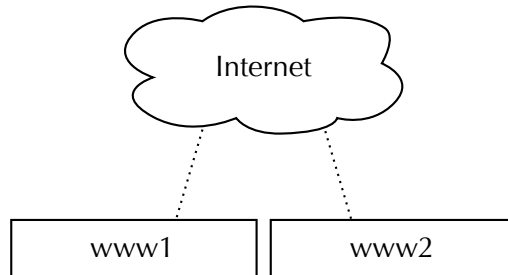


Figure 1: DNS Round-Robin

2 Linux Virtual Server

Figure 1 introduces a simple scenario this article will work with. Your web site is distributed via round-robin DNS to attempt to balance the load between multiple servers. Unfortunately, this is only an approximate balance, and if one of the servers goes down there is no fault tolerance; clients still will have the same IP cached in their browser if they were pointing at the now-downed node. This is a setup that LVS will easily improve upon.

LVS is broad term that refers to using Linux in order to abstract several servers behind one “virtual” server. What distinguishes LVS from using a Linux application proxy is that LVS solutions are provided by the kernel. Since the kernel already handles the networking layer, it is able to route connections efficiently without the overhead of a program running, resulting in lower latency for your service. For this article, the focus will be one of the more common applications of LVS: the Linux Layer 4 IP Virtual Server (IPVS).

There are three different modes in which IPVS can be set up. The simplest to set up is Network Address Translation mode. This uses NAT to rewrite incoming packets so that their destination addresses point to a cluster node. The return packets from the nodes reply back to the address of the load balancer. The load balancer then translates the return destination address back to the original client’s address. The next mode is direct routing. With direct routing, a packet arrives at the load balancer, whereon the destination hardware address is changed to the hardware address for one of the nodes, and this new packet is dropped back on the wire to be accepted this time by the chosen cluster node. The third mode is IP tunneling, which

encapsulates the original packet with a new IP header that directs the packet to the node selected.

Any one of these methods provides you with a load balancer that makes your service appear to originate on the load balancer, but in fact is passed on to one of the destination nodes.

3 Design Choices

At this point you need to choose between one of the aforementioned methods for delivering packets to one of your nodes. Your infrastructure should determine which is picked.

Consider NAT:

- Pro: simplest to set up
- Pro: destination nodes need no special configuration
- Con: overhead of encapsulation, de-encapsulation, and maintaining a state table on the load balancer

Next, direct routing:

- Pro: very fast, only has to rewrite hardware addresses
- Pro: load balancer only has to process packets going in one direction
- Con: more complex to understand and set up (dealing with ARP)
- Con: nodes must share subnet with load balancer

And tunneling:

- Pro: works well with geographically dispersed hosts
- Pro: load balancer only has to process packets going in one direction
- Con: more complex to understand and set up
- Con: requires setup of tunnel endpoint on cluster

To put these points in perspective, consider the three following example networks. Across each example, the assumption will be you want the service to have as low a latency as possible. In the first network, all the cluster nodes

are at the same data center on the same network. Here, the performance gained from doing hardware address rewrites would be a convincing reason to use direct routing (and this article aims to simplify its configuration).

Now make a slight change to provide example two: you are running out of publicly-addressable IP space. If you choose to use direct routing, you may soon hit a point when your cluster will not be able to scale any more. Since you should always be thinking about scalability, future growth concerns are paramount. NAT gives you as much growth as you need. Put two network cards in your load balancer to make it a router and put your cluster nodes in a private network. There are a few consequences to this: if you are running other services on any of the nodes, they will not be accessible directly, you will have to use destination NAT (commonly called port forwarding) to access these.⁴

After those two related examples, the third example network is where your service nodes are spread out around the world; which mode should you use? Based on the above pros and cons, tunneling seems like the best option. Actually, you should avoid using any of the LVS modes for this network, if possible. With tunneling, each packet must travel through the load balancer to the destination, resulting in a performance decrease from the latency of two wide-area hops. If your service supports it, you can have each client reconnect directly to one of the nodes which is up (for example, with an HTTP redirect). The additional latency added of having to do the redirect in the user-space of your load balancer machine should be insignificant in comparison to the network latency savings.

With these pros, cons, and examples you should be able to take into consideration both your current needs and expected growth to make the best design choices now.

4 Implementing Direct Routing

For brevity, this article will focus only on the direct routing mode of LVS. NAT is a very accessible topic to learn about, and setting up a NAT LVS cluster does not have the same sort of pitfalls one runs into with Direct Routing. However, most of the setup, aside from the network interface section, is applicable to NAT LVS as well. LVS tunneling shares the interface setup, but requires the addition of tunnel interfaces, which will not be covered in this article.

⁴Perhaps you may consider this layer a security advantage.

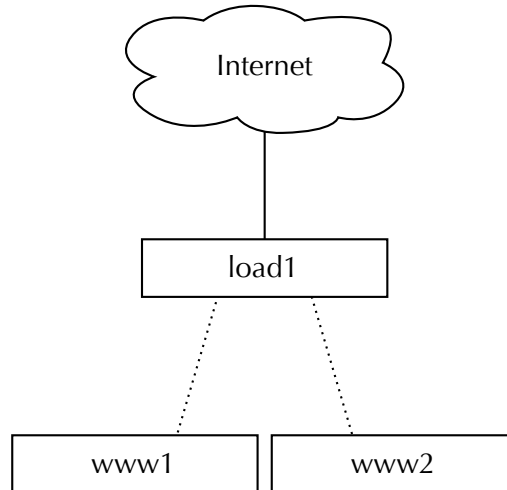


Figure 2: LVS Direct Routing

The goal of this section is shown in Figure 2. It is similar to DNS round-robin, except that all queries from the Internet come to the load balancer, which directs them to the nodes. Though it is not shown, the replies from the nodes are actually sent straight to the client; they do not route back through the load balancer.

4.1 IPVS Support

In order to use IPVS, you need support for it in the kernel. Assuming you are using a pre-compiled kernel, your distribution should have included kernel modules for LVS. Use the method for your distribution to enable the `ip_vs`, `ip_vs_wrr`, and `ip_vs_wlc` modules. If you compile your own kernels, you can find the options for these under “Networking options” in the kernel build configuration. Next, install the `ipvsadm` program. Again, this will be specific to your distribution; the package repository for your system should have it, otherwise you can compile it from source.⁵

⁵<http://www.linuxvirtualserver.org/software/ipvs.html>

4.2 Interfaces

The next step involves setting up the network interfaces. First, some terminology. The Virtual IP (VIP) is the IP address that is shared by the load balancer and each of the nodes. Each node and load balancer also has a Real IP (RIP). You need to add the load balancer to the same subnet as the nodes. The IPs that were already in use before setting up direct routing will be reused as the RIPs, and there will need to be one more IP dedicated for the VIP. By not changing the IP addresses on the nodes, you can leave your servers accessible via round-robin DNS until LVS is set up, allowing you to transition your service seamlessly.

Adding the VIP to the load balancer requires no special configuration apart from adding a virtual address. In part two of this article series, the heartbeat program will be adding and removing this address as a configured “resource,” but at present you will configure it manually. It is important that you set up the VIP in such a way that the default route out of the machine is still via the primary address (the RIP). This is done by defining the subnet mask to be 255.255.255.255 (32 in CIDR notation). Set it up as an additional address on `eth0`.

When adding the VIP to the nodes, it is essential that the IP address is unresolvable to the network via ARP. If it were, the load balancer would become unreachable. In order to hide the address, you need to set some kernel “sysctl” parameters by editing `/etc/sysctl.conf`. Look in your distribution’s documentation to confirm this file is not auto-generated from other files or by a configuration utility. Set the following parameters:⁶

```
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
```

This ensures that interfaces will only answer ARP requests for IP addresses that belong to them, as opposed to all IP addresses on the machine. For example, if the VIP is a virtual address on the loopback device (`lo`), then the RIP (`eth0`) will not advertise it. Run `sysctl -p` as root, or, if you are familiar with it, use the `/proc/sys/` interface to set these values.

Now that you have set these parameters, you may add the VIP to `lo`. This will be similar to configuring the VIP on the load balancer, except that

⁶Much of the documentation online dealing with ARP behavior refers to old kernel versions that use different syntax. This is correct from 2.6.4 through the current version, 2.6.25. Refer to `Documentation/networking/ip-sysctl.txt` in the kernel source for changes.

the additional address is for `lo`, not `eth0`. Again, ensure that the netmask of the address is `255.255.255.255`.

Time to test. The service you are running on the nodes must be configured to listen on both the RIP and VIP addresses. Assuming your firewall policy allows pings, you should still be able to ping the RIP of each node from a third-party machine unrelated to the load balancer setup. Next, try pinging the RIP of each node from the load balancer; connectivity to the node from the load balancer will be necessary once you configure the load balancers to check the nodes for availability.

Lastly, pinging the VIP from off-network should result in a response from the load balancer.

4.3 IPVS Configuration

Next, you need to tell IPVS on the load balancer to forward a service to the nodes. There are multiple algorithms that can be used for node selection, and you loaded two as kernel modules earlier in the article: weighted-round-robin (`wrr`) and weighted-least-connections (`wlc`). As its name would suggest, with the former option each node is delegated new connections in a round-robin fashion proportional to each node's weight. The `wlc` algorithm, however, keeps the number of current connections on each node proportional to the weight. The `wlc` option keeps a more even load, but has the downside that when a new node comes up, that node is preferred to the exclusion of the others (to attempt to balance the number of current connections). With all the new connections going to the single new node, the connections begin to finish on the original nodes, and the load balancer begins bouncing back and forth between exclusively preferring either the old nodes or the new node.⁷ However, for services with a high variability of connection time, you may want to start with using `wlc` and only switch to `wrr` if you experience this problem.

You can now configure IPVS using `ipvsadm`. This is the manual way to add a service and related nodes; the next section of this article will introduce an automatic method based on availability. The following `ipvsadm` command sets up a TCP service to be load balanced:

```
ipvsadm -A -t VIP:port -s wrr
```

Then, configure the destination nodes for this service. Direct routing

⁷This is called the thundering herd problem, and is described in more detail in the LVS-HOWTO section 4.21, "Thundering herd problem."

is the default, though the following example explicitly specifies it with `-g`. Both nodes are given a weight of one:

```
ipvsadm -a -t VIP:port RIP_1:port -g -w 1
ipvsadm -a -t VIP:port RIP_2:port -g -w 1
```

Run these commands, filling in the port your service runs on and the proper IP addresses. Switch `-t` to `-u` for a UDP service. Run `ipvsadm` without any command-line options to see your configuration (you can also use this to see the connection counts for each node). To test this service, make a connection to your service on the VIP address; you should receive a response from a node.

5 What's Next

At this point, you could change the DNS entries for your service, replacing the round-robin of node IPs with a single VIP. This has a few advantages to the former DNS setup. First, changes you make when you add or remove nodes from your cluster via `ipvsadm` happen instantly, without the need for waiting for DNS entries to expire. Secondly, LVS provides a more accurate way to balance the load than the approximate balancing provided by multiple DNS entries.

However, while the LVS setup thus far has these scalability advantages, it can still be improved in the area of high availability. The companion part two of this article will extend the foundation that has been introduced here by configuring the load balancer to handle automatic node addition and removal based on the availability of the nodes. Furthermore, using heartbeat, you can gain redundancy with the addition of a second load balancer.



This work is licensed under a Creative Commons Attribution-Share Alike 3.0 United States License.
<http://creativecommons.org/licenses/by-sa/3.0/us/>