Linux Virtual

Building load-balancing clusters with LVS is as good at keeping out the cold as chopping logs on a winter's day, maintains **Dr Chris Brown**.

Our expert

Dr Chris Brown
A Unix user for over 25 years, his own Interactive Digital Learning company provides Linux training, content and consultancy. He also specialises in computer-based classroom training delivery systems.

This month's Hardcore tutorial shows you how to provide a load-balanced cluster of web servers, with a scalable performance that far outstrips the capability of an individual server. We'll be using the Red Hat cluster software, but the core functionality described here is based on the Linux virtual server kernel module and is not specific to Red Hat.

If you want to follow along with the tutorial for real, you'll need at least four computers, as shown in Figure 1 (opposite page). I realise this might put it out of the reach of some readers as a practical project, but hopefully it can remain as a thought experiment, like that thing Schrödinger did with his cat. If you've got some old machines lying about the place, why not refer to this month's cover feature to find out how to get them up and running again – then network them?

Machine 1 is simply a client used for testing – it could be anything that has a web browser. In my case, I commandeered my wife's machine, which runs Vista. Machine 2 is where the real action is. This machine must be running Linux, and should preferably have two network interfaces, as shown in the figure. (It's possible to construct the cluster using just a single network, but this arrangement wouldn't be used on a production system, and it needs a bit of IP-level trickery to make it work).

This machine performs load balancing and routing into the cluster, and is known as the "director" in some LVS documentation. In my case, machine 2 was running CentOS5 (essentially equivalent to RHEL5). Machines 3 and 4 are the backend servers; in principle you could use anything that offers a web server. In my case I used a couple of laptops, one running SUSE 10.3 and one running Ubuntu 7.04. In reality you'd probably want more than two backend servers, but two is enough to prove the point.

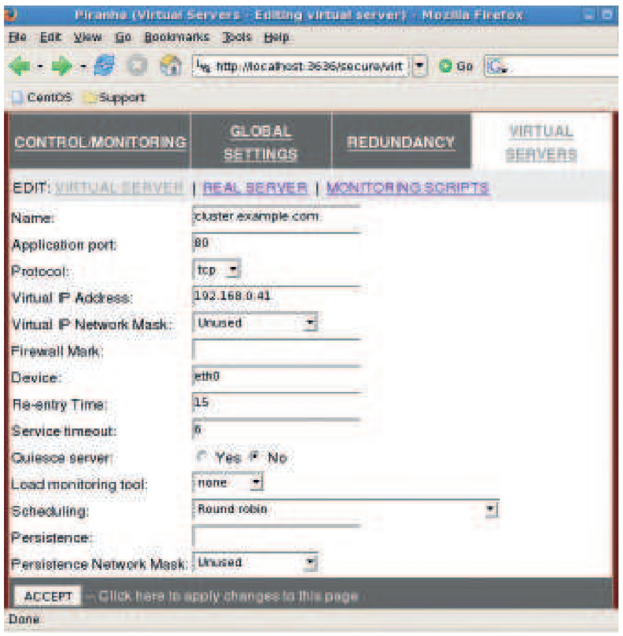
You might well be wondering why I'm running a different operating system on each of the four computers. It's a fair question, and one I often ask myself! Unfortunately, it's simply a fact of life for those of us who make our living by messing around with Linux.

How a load-balancing cluster works

Client machines send service requests (for example an **HTTP GET** request for a web page) to the public IP address of the director (**192.168.0.41** on the figure; as far as the clients are concerned, this is the IP address at which the service is provided).

The director chooses one of the backend servers to forward the request to. It rewrites the destination IP address of the packet to be that of the chosen backend server and forwards the packet out onto the internal network.

The chosen backend server processes the request and sends an HTTP response, which hopefully includes the requested page. This response is addressed to the client machine (**192.168.0.2**) but is initially sent back to the director (**10.0.0.1**), because the director is configured as the default gateway for the backend server. The director rewrites the source


Piranha (Virtual Servers - Editing virtual server) - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:3636/secure/virt
Go
CentOS
Support
CONTROL/MONITORING
GLOBAL SETTINGS
REDUNDANCY
VIRTUAL SERVERS
EDIT: VIRTUAL SERVER | REAL SERVER | MONITORING SCRIPTS
ACCEPT
-- Click here to apply changes to this page
Done

Name:	cluster.example.com
Application port:	80
Protocol:	tcp
Virtual IP Address:	192.168.0.41
Virtual IP Network Mask:	Unused
Firewall Mark:	
Device:	eth0
Re-entry Time:	15
Service timeout:	6
Quiesce server:	Yes [✓ No]
Load monitoring tool:	none
Scheduling:	Round robin
Persistence:	
Persistence Network Mask:	Unused

› **Figure 2: Configuring a virtual server with *Piranha*.**

» **Last month** Use 'heartbeat' to provide automatic failover to a backup server.

Server

IP address of the packet to refer to its own public IP address and sends the packet back to the client.

All of this IP-level tomfoolery is carried out by the LVS module within the Linux kernel. The director is not simply acting as a reverse web proxy, and commands such as **netstat -ant** will NOT show any user-level process listening on port 80.

The address re-writing is a form of NAT (Network Address Translation) and allows the director to masquerade as the real server, and to hide the presence of the internal network that's doing the real work.

Balancing the load

A key feature of LVS is load balancing – that is, making sure that each backend server receives roughly the same amount of work. LVS has several algorithms for doing this; we'll mention four.

» **Round-robin scheduling** is the simplest algorithm. The director just works its way around the backend servers in turn, starting round with the first one again when it reaches the end of the list. This is good for testing because it's easy to verify that all your backend servers are working, but it may not be best in a production environment.

» **Weighted round-robin** is similar but lets you assign a weight to each backend server to indicate the relative speed of each server. A server with a weight of two is assumed to be twice as powerful as a server with a weight of one and will receive twice as many requests.

» **Least-connection** load balancing tracks the number of currently active connections to each backend server and forwards the request to the one with the least number.

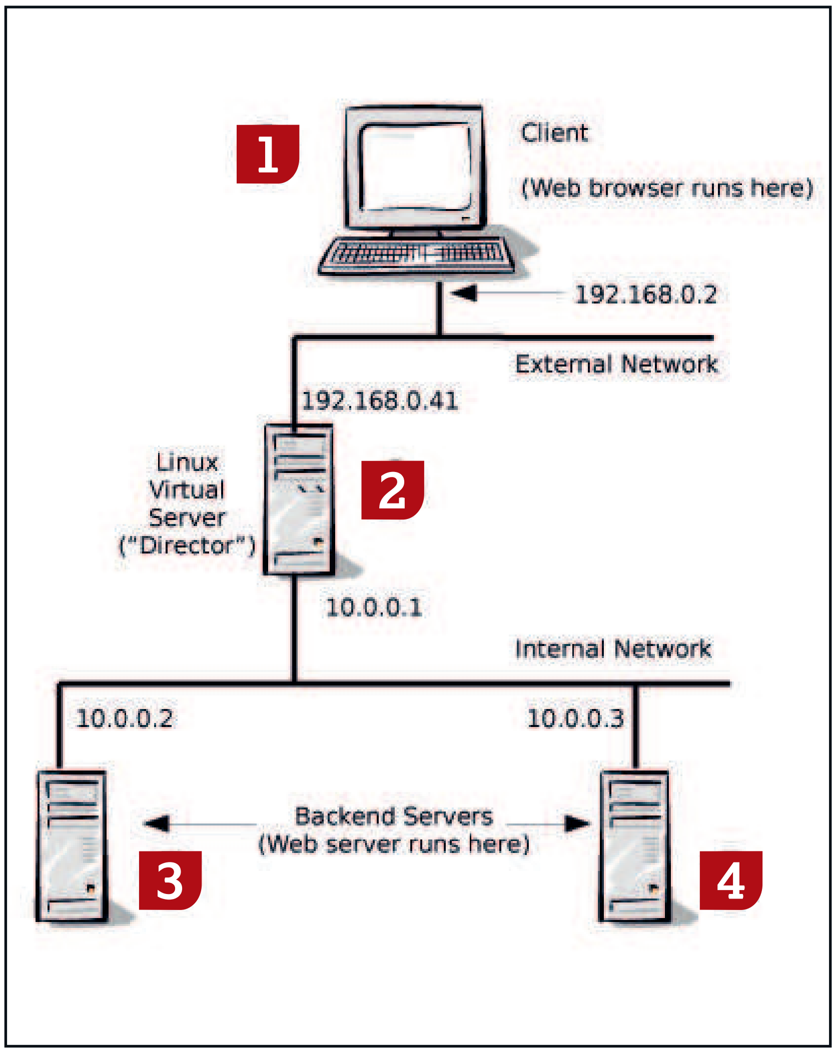
» **Weighted least connection** method also takes into account the relative processing power (weight) of the server. The weighted least connection method is good for production clusters because it works well when service requests take widely varying amounts of time to process and/or when the backend servers in the cluster are not equally powerful.

The latter two algorithms are dynamic; that is, they take account of the current load on the machines in the cluster.

The tools for the job

To make all this work, the kernel maintains a virtual server table which contains, among other things, the IP addresses of the backend servers. The command-line tool *ipvsadm* is used to maintain and inspect this table.

Assuming that your kernel is built to include the LVS module (and most of them are these days) *ipvsadm* is the only program you absolutely must have to make LVS work; however there are some other toolsets around that make life easier. (The situation is similar to the packet-filtering machinery in the kernel. In theory we only need the command-line tool *iptables* to manage this and create a packet-filtering firewall, in practice most of us use higher-



› **Figure 1: A simple load-balancing cluster.**

Worried about bottlenecks?

Since all the traffic into and out of the cluster passes through the director, you might be wondering if these machines will create a performance bottleneck. In fact this is unlikely to be the case. Remember, the director is performing only simple packet header manipulation inside the kernel, and it turns out that even a modest machine is capable of saturating a 100 MBps network. It's much more likely that the network bandwidth into the site will be the bottleneck. For lots more information about using old PCs for tasks like networking, see this month's cover feature starting on page 45.

level tools – often graphical – to build our firewall rulesets.) For this tutorial we'll use the clustering toolset from Red Hat which includes a browser-based configuration tool called *Piranha*.

Now, the configuration shown in Figure 1 has one major shortcoming – the director presents a single point of failure within the cluster. To overcome this, it's possible to use a primary and backup server pair as the director, using a *heartbeat* message exchange to allow the backup to detect the failure of the primary and take over its resources.

For this tutorial I've ignored this issue, partly because I discussed it in detail last month and partly because I would have needed a fifth machine to construct the cluster, and I only have four! It should be noted, however, that the clustering toolset from Red Hat does indeed include functionality to failover from a primary to a backup director.

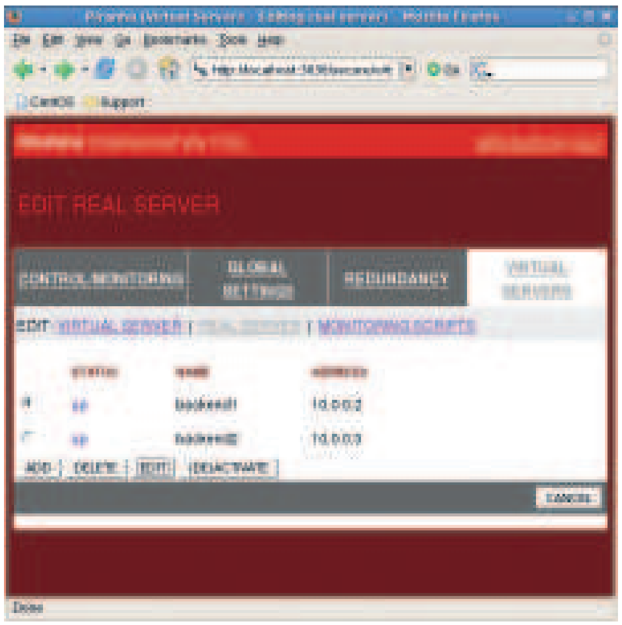
Setting up the backend servers

If you want to actually build the cluster described in this tutorial, start with the backend servers. These can be any machines able to offer HTTP-based web service on port 80; in my case they were running Linux and *Apache*.

You will need to create some content in the **DocumentRoot** directory of each server, and for testing purposes it's a good idea to make the content different on each machine so we can easily distinguish which one actually served the request. For example, on backend server 1, I created a file called **proverb.html** with the line "A stitch in time saves nine". On server 2 the line was "Look before you leap". Of course in reality you'd need to ensure that all the backend servers were 'in sync' and serving the same content – an issue we'll return to later.

Give the machines static IP addresses appropriate for the internal network. In my case I used **10.0.0.2** and **10.0.0.3**. Set the default route on these machines to be the private IP address of the director (**10.0.0.1**).

"You'll need at least four computers to follow along for real; otherwise see this as a thought experiment"


EDIT REAL SERVER
CONTROL/MONITORING
GLOBAL SETTINGS
REDUNDANCY
VIRTUAL SERVERS
EDIT: VIRTUAL SERVER | REAL SERVER | MONITORING SCRIPTS
ADD | DELETE | EDIT | (DE)ACTIVATE
CANCEL
Done

up	backend1	10.0.0.2
up	backend2	10.0.0.3

Figure 3: Specifying the real (backend) servers.

Setting up the director

On the director machine, begin by downloading and installing the Red Hat clustering tools. In my case (remember that I'm running CentOS5 on this machine) I simply used the graphical install tool (*pirut*) to install the packages *ipvsadm* and *Piranha* from the CentOS repository. My next step was to run the command **piranha-passwd** to set a password for the *piranha* configuration tool:

```
# /etc/init.d/piranha-gui start
```

This service listens on port 3636 and provides a browser-based interface for configuring the clustering toolset, so once it's running I can point my browser at **http://localhost:3636**. I'll need to log in, using the username piranha and the password I just set. From here I'm presented with four main screens: Control/Monitoring, Global Settings, Redundancy and Virtual Servers (you can see the links to these in Figure 3 (above). To begin, go to the

If it doesn't work

Chances are good that your cluster won't work the first time you try it. You have a couple of choices at this point. First, you could hurl one of the laptops across the room and curse Microsoft.

These are not, of course, productive responses, even if they do make you feel better. And Microsoft really shouldn't be blamed since we're not actually using any of its software. The second response is to take some deep, calming breaths, then conduct a few diagnostic tests:

A good first step would be to verify connectivity from the LVS machine, using good ol' ping. First, make sure you can ping your test machine (**192.168.0.2**). Then check that you can ping both your backend servers (**10.0.0.2** and **10.0.0.3**). If this doesn't work, run **ifconfig -a** and verify that eth0 has IP address **192.168.0.41** and eth1 has address **10.0.0.1**.

Also, run **route -n** to check the routing table – verify that you have routes out onto the **192.168.0.0** network and the **10.0.0.0** network via the appropriate interfaces.

If you can ping your backend servers, fire up the web browser on the LVS machine (assuming there is a browser installed there – you don't really need one). Verify that you can reach the URLs **http://10.0.0.2** and **http://10.0.0.3**, and see the web content served by these two machines. (If the pings are OK but the web servers aren't accessible, make sure that the web servers are actually running on the backend servers, and that the backend servers don't have firewall rules in place that prevent them from being accessed.)

You should also carefully check the LVS routing table displayed by the Control/Monitoring page of *Piranha* and verify that your backend servers show up there. If you're still having no luck, verify that IP forwarding is enabled on the LVS machine. You can do this with the command:

```
# cat /proc/sys/net/ipv4/ip_forward
```

If it reports '**1**', that's fine, but if it reports '**0**' you'll need to enable IP forwarding like this:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

You might also verify that your kernel is actually configured to use LVS. If your distro provides a copy of the config file used to build the kernel (it will likely be **/boot/config-something-or-other**) use **grep** to look for the string **CONFIG_IP_VS** in this file, and verify that you see a line like **CONFIG_IP_VS=m**, among others. You might also try:

```
# lsmod | grep vs
```

to verify that the *ip_vs* module is loaded. If you can't find evidence for virtual server support in the kernel you may have to reconfigure and rebuild your kernel, an activity that is beyond the scope of this tutorial.

If all these tests look OK and things still don't work, you may now hurl one of the laptops across the room. This won't make your cluster work, but at least you'll have a better idea of why it's broken ....

Virtual Servers screen and add a new service. Figure 2 (page 98) shows the form you'll fill in. Among other things, you need to give the service a name, specify the port number and interface it will accept request packets on, and select the scheduling algorithm (for initial testing I chose 'Round Robin'). Clicking the Real Server link at the top of this page takes you to the screen shown in Figure 3. Here you can specify the names, IP addresses and weights of your backend servers.

Behind the scenes, most of the configuration captured by *Piranha* is stored in the config file ***/etc/sysconfig/ha/lvs.cf***. Other tools in the clustering toolset read this file; it's plain text and there's no reason why you can't just hand-edit it directly if you prefer. With this configuration in place, you should be good to go. Launch the clustering service from the command line with:

```
# /etc/init.d/pulse start
```

(On a production system you'd have this service start automatically at boot time.)

Now go to the *Piranha* control/monitoring screen, shown in Figure 4 (below). Look carefully at the LVS routing table. You should see an entry there for your virtual service (that's the line starting **TCP...**) and below that, a line for each of your backend servers. You can obtain the same information from the command line with the command

```
# ipvsadm -L
```

The periodic health check

Also on the control/monitoring screen there's an LVS process table. Here you'll see a couple of 'nanny' processes. These are responsible for verifying that the backend servers are still up and running, and there's one nanny for each server. They work by periodically sending a simple **HTTP GET** request and verifying that there's a reply. If you look carefully at the **-s** and **-x** options specified to nanny, you'll see the send and expect strings that are used for this test. (You can customise these if you want, using the Virtual Servers > Monitoring Scripts page of *Piranha*.)

If you look at the *Apache* access logs on the backend servers, you'll see these probe requests arriving every six seconds. If a nanny process detects that its server has stopped responding, it will invoke **ipvsadm** to remove that machine's entry from the LVS routing table, so that the director will no longer forward any requests to that machine. The nanny will continue to probe the server, however, and should it come back up, the nanny will run **ipvsadm** again to reinstate the routing entry.

You can observe this behaviour by unplugging the network cable from a backend server (or simply stopping the **httpd** daemon) and examining the LVS routing table. You should see your dead server's entry disappear. If you reconnect the network cable, or restart the server, the entry will re-appear. Be patient, though, it can take 20 seconds or so for these changes to show up.

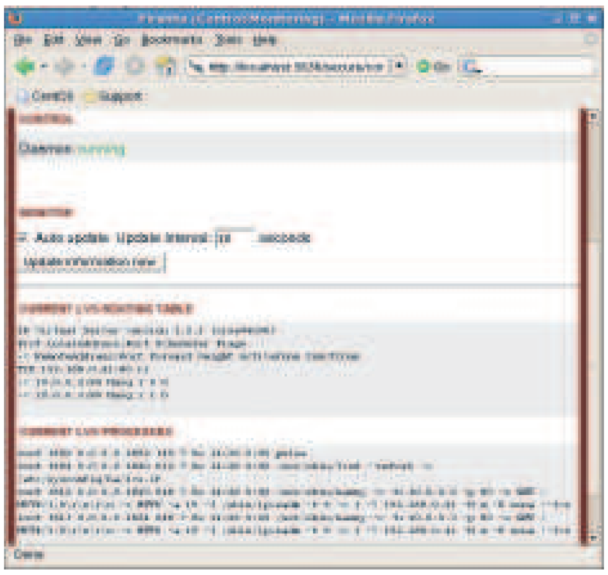
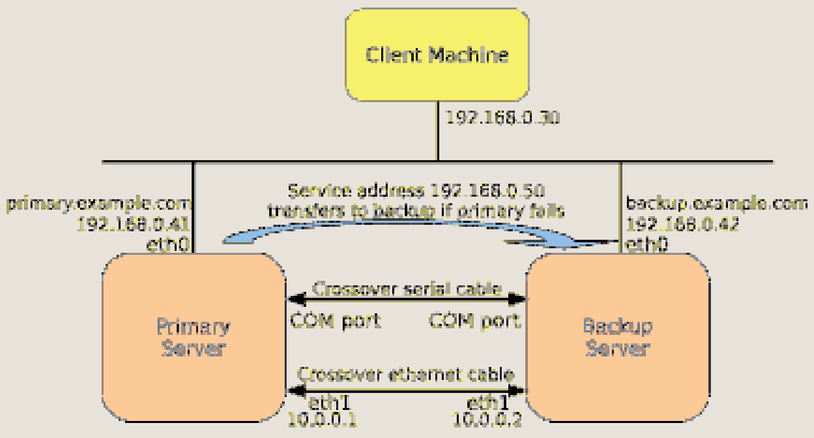

File Edit View Go Bookmarks Tools Help
CentOS Support
Daemon: running
Auto update Update Interval: 10 seconds
Update information now
Close

Figure 4: The control and monitor screen of *Piranha*.

ERRATA: Figure 1 from LXF101



In LXF101's Clustering tutorial, we managed to print the wrong figure. Apologies to any readers who struggled to make sense of the diagram in the context of the tutorial. Here's the Figure 1 that should have appeared last month!

The Denouement

If all seems well, it's time to go to the client machine (machine 1 on the first figure) and try to access the page in the browser. In this example, you'd browse to **http://192.168.0.41/proverbs.html**, and you should see the Proverbs page served from one of your backend servers. If you force a page refresh, you'll see the proverbs page from the next server in the round-robin sequence. You should also be able to verify the round-robin behaviour by examining the apache access logs of the individual backend servers. (If you look carefully at the log file entries, you see that these accesses come from **192.168.0.2** whereas the probes from the nannies come from **10.0.0.1**.) If all of this works, congratulations! You've just built your first load-balancing Linux cluster.

Postscript

Let's pause and consider what we've demonstrated in these two tutorials. We've seen how to build failover clustering solutions that add another nine onto the availability of your service (for example, to go from 99.9% to 99.99% availability); and we've seen how to build load-balancing clusters able to far outstrip the performance of an individual web server. In both cases we've used free, open source software running on a free, open source operating system. I sometimes wonder what, as he gets into bed at night with his mug of Horlicks, Steve Ballmer makes of it all. It sure impresses the hell out of me. LXF

Next month Distribute workload of a 'virtual' server across multiple backend servers.