IBM.

# Set up a Web server cluster in 5 easy steps

**Get up and running with the Linux Virtual Server and Linux-HA.org's Heartbeat**

Level: Intermediate

Eli M. Dow (emdow@us.ibm.com), Software Engineer, IBM
Frank LeFevre (lefevre@us.ibm.com), Senior Software Engineer, <a href="http://www.ibm.com/developerWorks">IBM</a>

22 Aug 2007

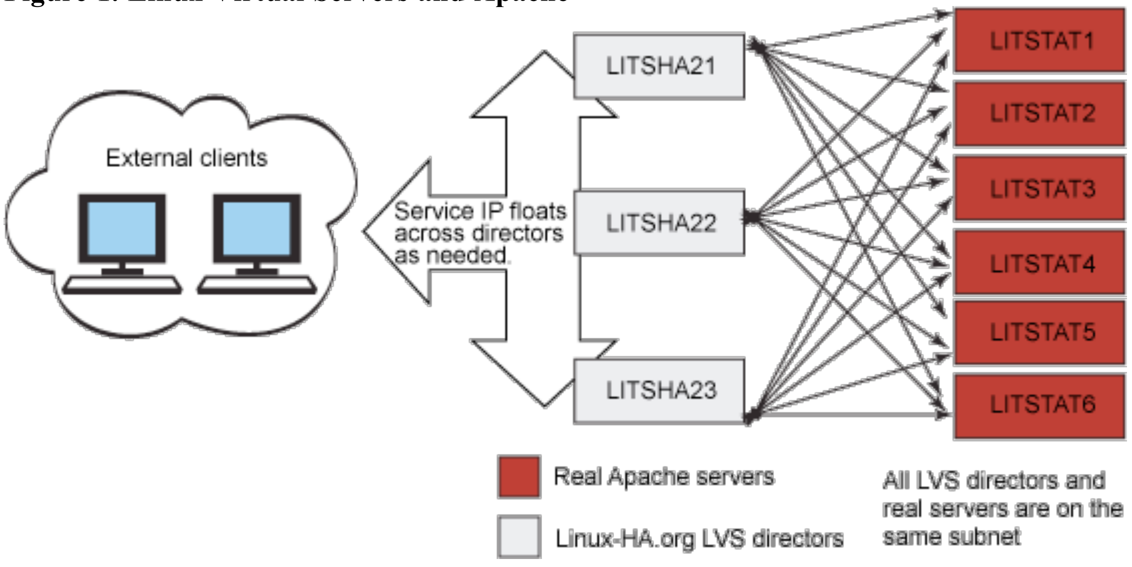
Construct a highly available Apache Web server cluster that spans multiple physical or virtual Linux® servers in 5 easy steps with Linux Virtual Server and Heartbeat v2.

Spreading a workload across multiple processors, coupled with various software recovery techniques, provides a highly available environment and enhances overall RAS (Reliability, Availability, and Serviceability) of the environment. Benefits include faster recovery from unplanned outages, as well as minimal effects of planned outages on the end user.

To get the most out of this article, you should be familiar with Linux and basic networking, and you should have Apache servers already configured. Our examples are based on standard SUSE Linux Enterprise Server 10 (SLES10) installations, but savvy users of other distributions should be able to adapt the methods shown here.

This article illustrates the robust Apache Web server stack with 6 Apache server nodes (though 3 nodes is sufficient for following the steps outlined here) as well as 3 Linux Virtual Server (LVS) directors. We used 6 Apache server nodes to drive higher workload throughputs during testing and thereby simulate larger deployments. The architecture presented here should scale to many more directors and backend Apache servers as your resources permit, but we haven't tried anything larger ourselves. Figure 1 shows our implementation using the Linux Virtual Server and the linux-ha.org components.

**Figure 1. Linux Virtual Servers and Apache**


External clients
Service IP floats across directors as needed
LITSHA21
LITSHA22
LITSHA23
LITSTAT1
LITSTAT2
LITSTAT3
LITSTAT4
LITSTAT5
LITSTAT6
Real Apache servers
Linux-HA.org LVS directors
All LVS directors and real servers are on the same subnet

As shown in Figure 1, the external clients send traffic to a single IP address, which may exist on any of the LVS director machines. The director machines actively monitor the pool of Web servers they relay work to.

Note that the workload progresses from the left side of Figure 1 toward the right. The floating resource address for this cluster will reside on one of the LVS director instances at any given time. The service address may be moved manually through a graphical configuration utility, or (more commonly) it can be self-managing, depending on the state of the LVS directors. Should any director become ineligible (due to loss of connectivity, software failure, or similar) the service address will be relocated automatically to an eligible director.

The floating service address must span two or more discrete hardware instances in order to continue operation with the loss of one physical machine. With the configuration decisions presented in this article, each LVS director is able to forward packets to any real Apache Web server regardless of physical location or proximity to the active director providing the floating service address. This article shows how each of the LVS directors can actively monitor the Apache servers in order to ensure requests are sent only to operational back-end servers.

With this configuration, practitioners have successfully failed entire Linux instances with no interruption of service to the consumers of the services enabled on the floating service address (typically http and https Web requests).

You can duplicate our configuration using an entirely open source software stack consisting of Heartbeat technology components provided by linux-ha.org, and server monitoring via mon and Apache. As stated, we used SUSE Linux Enterprise Server for testing our configuration.

All of the machines used in the LVS scenario reside on the same subnet and use the Network Address Translation (NAT) configuration. Numerous other network topographies are described at the Linux Virtual Server Web site (see Resources); we favor NAT for simplicity. For added security, you should limit traffic across firewalls to only the floating IP address that is passed between the LVS directors.

The Linux Virtual Server suite provides a few different methods to accomplish a transparent HA back-end infrastructure. Each method has advantages and disadvantages. LVS-NAT operates on a director server by grabbing incoming packets that are destined for configuration-specified ports and rewriting the destination address in the packet header dynamically. The director does not process the data content of the packets itself, but rather relays them on to the realservers. The destination address in the packets is rewritten to point to a given realserver from the cluster. The packet is then placed back on the network for delivery to the realserver, and the realserver is unaware that anything has gone on. As far as the realserver is concerned, it has simply received a request directly from the outside world. The replies from the realserver are then sent back to the director where they are again rewritten to have the source address of the floating IP address that clients are pointed at, and are sent along to the original client.

Using the LVS-NAT approach means the realservers require simple TCP/IP functionality. The other modes of LVS operation, namely LVS-DR and LVS-Tun require more complex networking concepts. The major benefit behind the choice of LVS-NAT is that very little alteration is required to the configuration of the realservers. In fact, the hardest part is remembering to set the routing statements properly.

**New to Linux Virtual Server terminology**

**LVS directors:** Linux Virtual Server directors are systems that accept arbitrary incoming traffic and pass it on to any number of realservers. They then accept the response from the realservers and pass it back to the clients who initiated the request. The directors need to perform their task in a transparent fashion such that clients never know that realservers are doing the actual workload processing.

LVS directors themselves need the ability to float resources (specifically, a virtual IP address on which they listen for incoming traffic) between one another in order to not become a single point of failure. LVS directors accomplish floating IP addresses by leveraging the Heartbeat component from LVS. This allows each configured director that is running Heartbeat to ensure one, and only one, of the directors lays claim to the virtual IP address servicing incoming requests.

Beyond the ability to float a service IP address, the directors need to be able to monitor the status of the realservers that are doing the actual workload processing. The directors must keep a working knowledge of what realservers are available for processing at all times. In order to monitor the realservers, the mon package is used. Read on for details

## Step 1: Building realserver images

Begin by making a pool of Linux server instances, each running Apache Web server, and ensure that the servers are working as designed by pointing a Web browser to each of the realserver's IP addresses. Typically, a standard install will be configured to listen on port 80 on its own IP address (in other words, on a different IP for each realserver).

Next, configure the default Web page on each server to display a static page containing the hostname of the machine serving the page. This ensures that you always know which machine you are connecting to during testing.

As a precaution, check that IP forwarding on these systems is OFF by issuing the following command:

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

If for any reason you need to disable it, issue this command:

```
# echo "0"
>/proc/sys/net/ipv4/ip_forward
```

An easy way to ensure that each of your realservers is properly listening on the http port (80) is to use an external system and perform a scan. From some other system with network connectivity to your server, you can use the nmap utility to make sure the server is listening.

on configuring Heartbeat and configuring mon.

**Realservers:** These systems are the actual Web server instances providing the HA service. It is vital to have more than one realserver providing the service you wish to make HA. In our environment, 6 realservers are implemented, but adding more is trivial once the rest of the LVS infrastructure is in place.

In this article, the realservers are all assumed to be running the Apache Web Server, but other services could just as easily have been implemented (in fact, it is trivially easy to enabled SSH serving as an additional test of the methodology presented here).

The realservers used are stock Apache Web servers with the notable exception that they were configured to respond as if it were the LVS director's floating IP address, or a virtual hostname corresponding to the floating IP address used by the directors. This is accomplished by altering a single line in the Apache configuration file.

**Listing 1. Using nmap to make sure the server is listening**

```
# nmap -P0 192.168.71.92

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2006-01-13 16:58 EST
Interesting ports on 192.168.71.92:
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE    SERVICE
22/tcp  open     ssh
80/tcp  filtered http
111/tcp open     rpcbind
631/tcp open     ipp
```

Be aware that some organizations frown on the use of port scanning tools such as nmap: make sure that your organization approves before using it.

Next, point your Web browser to each realserver's actual IP address to ensure each is serving the appropriate page as expected. Once this is completed, go to Step 2.

## Step 2: Installing and configuring the LVS directors

Now you are ready to construct the 3 LVS director instances needed. If you are doing a fresh install of SUSE Linux Enterprise Server 10 for each of the LVS directors, be sure to select the high availability packages relating to heartbeat, ipvsadm, and mon during the initial installation. If you have an existing installation, you can always use a package management tool, such as YAST, to add these packages after your base installation. It is strongly recommended that you add each of the realservers to the /etc/hosts file. This will ensure there is no DNS-related delay when servicing incoming requests.

At this time, double check that each of the directors are able to perform a timely ping to each of the realservers:

**Listing 2. Pinging the realservers**

```
# ping -c 1 $REAL_SERVER_IP_1

 # ping -c 1 $REAL_SERVER_IP_2

 # ping -c 1 $REAL_SERVER_IP_3

 # ping -c 1 $REAL_SERVER_IP_4

 # ping -c 1 $REAL_SERVER_IP_5

 # ping -c 1 $REAL_SERVER_IP_6
```

Once completed, install ipvsadm, Heartbeat, and mon from the native package management tools on the server. Recall that Heartbeat will be used for intra-director communication, and mon will be used by each director to maintain information about the status of each realserver.

## Step 3: Installing and configuring Heartbeat on the directors

If you have worked with LVS before, keep in mind that configuring Heartbeat Version 2 on SLES10 is quite a bit different than it was for Heartbeat Version 1 on SLES9. Where Heartbeat Version 1 used files (haresources, ha.cf, and authkeys) stored in the /etc/ha.d/ directory, Version 2 uses the new, XML-based Cluster Information Base (CIB). The recommended approach for upgrading is to use the haresources file to generate the new cib.xml file. The contents of a typical ha.cf file are shown in Listing 3.

We took the ha.cf file from a SLES9 system and added the bottom 3 lines (`respawn`, `pingd`, and `crm`) for Version 2. If you have an existing version 1 configuration, you may opt to do the same. If you are using these instructions for a new installation, you can copy Listing 3 and modify it to suit your production environment.

**Listing 3. A sample /etc/ha.d/ha.cf config file**

```
# Log to syslog as facility "daemon"
use_logd on
logfacility daemon

# List our cluster members (the realservers)
node litsha22
node litsha23
node litsha21

# Send one heartbeat each second
keepalive 3

# Warn when heartbeats are late
warntime 5

# Declare nodes dead after 10 seconds
deadtime 10

# Keep resources on their "preferred" hosts - needed for active/active
#auto_failback on


# The cluster nodes communicate on their heartbeat lan (.68.*) interfaces
ucast eth1 192.168.68.201
ucast eth1 192.168.68.202
ucast eth1 192.168.68.203

# Failover on network failures
# Make the default gateway on the public interface a node to ping
# (-m) -> For every connected node, add <integer> to the value set
#  in the CIB, * Default=1
# (-d) -> How long to wait for no further changes to occur before
#  updating the CIB with a changed attribute
# (-a) -> Name of the node attribute to set,  * Default=pingd
respawn hacluster /usr/lib/heartbeat/pingd -m 100 -d 5s

# Ping our router to monitor ethernet connectivity
ping litrout71_vip

#Enable version 2 functionality supporting clusters with  > 2 nodes
crm yes
```

The `respawn` directive is used to specify a program to run and monitor while it runs. If this program exits with anything other than exit code 100, it will be automatically restarted. The first parameter is the user id to run the program under, and the second parameter is the program to run. The `-m` parameter sets the attribute `pingd` to 100 times the number of ping nodes reachable from the current machine, and the `-d` parameter delays 5 seconds before modifying the `pingd` attribute in the CIB. The `ping` directive is given to declare the PingNode to Heartbeat, and the `crm` directive specifies whether Heartbeat should run the 1.x-style cluster manager or 2.x-style cluster manager that supports more than 2 nodes.

This file should be identical on all the directors. It is absolutely vital that you set the permissions appropriately such that the hacluster daemon can read the file. Failure to do so will cause a slew of warnings in your log files that may be difficult to debug.

For a release 1-style Heartbeat cluster, the haresources file specifies the node name and networking information (floating IP, associated interface, and broadcast). For us, this file remained unchanged:

```
litsha21  192.168.71.205/24/eth0/192.168.71.255
```

This file will be used only to generate the cib.xml file.

The authkeys file specifies a shared secret allowing directors to communicate with one another. The shared secret is simply a password that all the heartbeat nodes know and use to communicate with one another. The secret prevents unwanted parties from trying to influence the heartbeat server nodes. This file also remained unchanged:

```
auth  1

1  sha1  ca0e08148801f55794b23461eb4106db
```

The next few steps show you how to convert the version 1 haresources file to the new version 2 XML-based configuration format (cib.xml). Though it should be possible to simply copy and use the configuration file in Listing 4 as a starting point, it is strongly suggested that you follow along to tailor the configuration for your deployment.

To convert file formats to the XML-based CIB (Cluster Information Base) file you will use in deployment, issue the following command:

```
python  /usr/lib64/heartbeat/haresources2cib.py
/etc/ha.d/haresources  >  /var/lib/heartbeat/crm/test.xml
```

A configuration file similar to the one shown in Listing 4 will be generated and placed in /var/lib/heartbeat/crm/test.xml.

**Listing 4. Sample CIB.xml file**

```
<cib admin_epoch="0" have_quorum="true" num_peers="3" cib_feature_revision="1.3"
 generated="true" ccm_transition="7" dc_uuid="114f3ad1-f18a-4bec-9f01-7ecc4d820f6c"
 epoch="280" num_updates="5205" cib-last-written="Tue Apr  3 16:03:33 2007">
   <configuration>
     <crm_config>
        <cluster_property_set id="cib-bootstrap-options">
          <attributes>
             <nvpair id="cib-bootstrap-options-symmetric_cluster"
                    name="symmetric_cluster" value="true"/>
             <nvpair id="cib-bootstrap-options-no_quorum_policy"
                    name="no_quorum_policy" value="stop"/>
             <nvpair id="cib-bootstrap-options-default_resource_stickiness"
                    name="default_resource_stickiness" value="0"/>
             <nvpair id="cib-bootstrap-options-stonith_enabled"
                    name="stonith_enabled" value="false"/>
             <nvpair id="cib-bootstrap-options-stop_orphan_resources"
                    name="stop_orphan_resources" value="true"/>
             <nvpair id="cib-bootstrap-options-stop_orphan_actions"
                    name="stop_orphan_actions" value="true"/>
             <nvpair id="cib-bootstrap-options-remove_after_stop"
                    name="remove_after_stop" value="false"/>
             <nvpair id="cib-bootstrap-options-transition_idle_timeout"
                    name="transition_idle_timeout" value="5min"/>
             <nvpair id="cib-bootstrap-options-is_managed_default"
                    name="is_managed_default" value="true"/>
          <attributes>
        <cluster_property_set>
     <crm_config>
     <nodes>
        <node uname="litsha21" type="normal" id="01ca9c3e-8876-4db5-ba33-a25cd46b72b3">
          <instance_attributes id="standby-01ca9c3e-8876-4db5-ba33-a25cd46b72b3">
            <attributes>
               <nvpair name="standby" id="standby-01ca9c3e-8876-4db5-ba33-a25cd46b72b3"
                     value="off"/>
            <attributes>
          <instance_attributes>
       <node>
        <node uname="litsha23" type="normal" id="dc9a784f-3325-4268-93af-96d2ab651eac">
          <instance_attributes id="standby-dc9a784f-3325-4268-93af-96d2ab651eac">
```

```
         <attributes>
            <nvpair name="standby" id="standby-dc9a784f-3325-4268-93af-96d2ab651eac"
                  value="off"/>
         <attributes>
        <instance_attributes>
      <node>
       <node uname="litsha22" type="normal" id="114f3ad1-f18a-4bec-9f01-7ecc4d820f6c">
         <instance_attributes id="standby-114f3ad1-f18a-4bec-9f01-7ecc4d820f6c">
         <attributes>
            <nvpair name="standby" id="standby-114f3ad1-f18a-4bec-9f01-7ecc4d820f6c"
                  value="off"/>
         <attributes>
        <instance_attributes>
      <node>
    <nodes>
    <resources>
       <primitive class="ocf" provider="heartbeat" type="IPaddr" id="IPaddr_1">
        <operations>
           <op id="IPaddr_1_mon" interval="5s" name="monitor" timeout="5s"/>
        <operations>
         <instance_attributes id="IPaddr_1_inst_attr">
         <attributes>
            <nvpair id="IPaddr_1_attr_0" name="ip" value="192.168.71.205"/>
            <nvpair id="IPaddr_1_attr_1" name="netmask" value="24"/>
            <nvpair id="IPaddr_1_attr_2" name="nic" value="eth0"/>
            <nvpair id="IPaddr_1_attr_3" name="broadcast" value="192.168.71.255"/>
         <attributes>
        <instance_attributes>
      <primitive>
    <resources>
    <constraints>
       <rsc_location id="rsc_location_IPaddr_1" rsc="IPaddr_1">
         <rule id="prefered_location_IPaddr_1" score="200">
           <expression attribute="#uname" id="prefered_location_IPaddr_1_expr"
                 operation="eq" value="litsha21"/>
        <rule>
      <rsc_location>
       <rsc_location id="my_resource:connected" rsc="IPaddr_1">
         <rule id="my_resource:connected:rule" score_attribute="pingd">
```

```
            <expression id="my_resource:connected:expr:defined" attribute="pingd"

                  operation="defined"/>

         <rule>

       <rsc_location>

     <constraints>

   <configuration>

 <cib>
```

Once your configuration file is generated, move test.xml to cib.xml, change the owner to hacluster and the group to haclient, and then restart the heartbeat process.

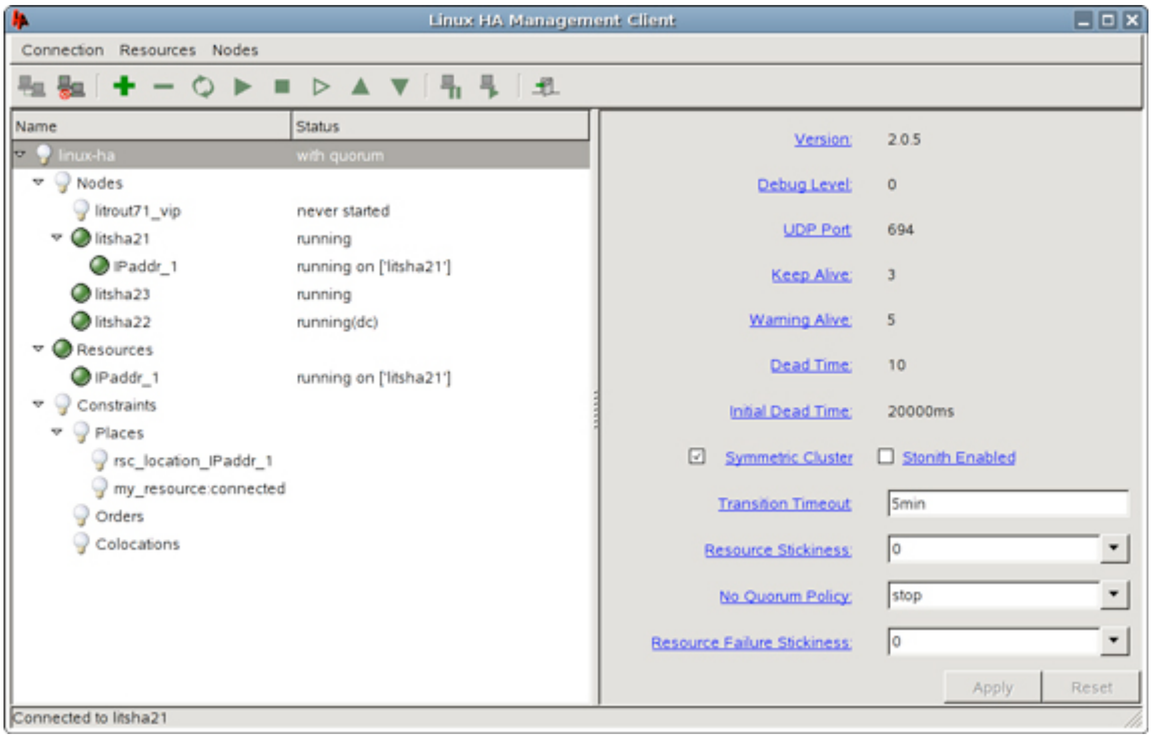
Now that the heartbeat configuration is complete, set heartbeat to start at boot time on each of the directors. To do this, Issue the following command (or equivalent for your distribution) on each director:

```
# chkconfig heartbeat on
```

Restart each of the LVS directors to ensure the heartbeat service starts properly at boot. By halting the machine that holds the floating resource IP address first, you can watch as the other LVS Director images establish quorum, and instantiate the service address on a newly-elected primary node within a matter of seconds. When you bring the halted director image back online, the machines will re-establish quorum across all nodes, at which time the floating resource IP may transfer back. The entire process should take only a few seconds.

Additionally, at this time you may wish to use the graphical utility for the heartbeat process, hb_gui (see Figure 2), to manually move the IP address around in the cluster by setting various nodes to the standby or active state. Retry these steps numerous times, disabling and re-enabling various machines that are active or inactive. With the choice of configuration policy selected earlier, as long as quorum can be established and at least one node is eligible, the floating resource IP address will remain operational. During your testing, you can use simple pings to ensure that no packet loss occurs. When you have finished experimenting, you should have a strong feel for how robust your configuration is. Make sure you are comfortable with the HA configuration of your floating resource IP before continuing on.

**Figure 2. Graphical configuration utility for the heartbeat process, hb_gui**



| Name | Status |
|---|---|
| linux-ha | with quorum |
| Nodes | |
| litrout71_vip | never started |
| litsha21 | running |
| IPaddr_1 | running on ['litsha21'] |
| litsha23 | running |
| litsha22 | running(dc) |
| Resources | |
| IPaddr_1 | running on ['litsha21'] |
| Constraints | |
| Places | |
| rsc_location_IPaddr_1 | |
| my_resource:connected | |
| Orders | |
| Colocations | |

Version: 2.0.5
Debug Level: 0
UDP Port: 694
Keep Alive: 3
Warning Alive: 5
Dead Time: 10
Initial Dead Time: 20000ms
[✓] Symmetric Cluster ☐ Stonith Enabled
Transition Timeout: 5min
Resource Stickiness: 0
No Quorum Policy: stop
Resource Failure Stickiness: 0

Connected to litsha21

Figure 2 illustrates the graphical console as it appears after login, showing the managed resources and associated configuration options. Note that you must log into the hb_gui console when you first launch the application; the credentials used will depend on your deployment.

Notice in Figure 2 how the nodes in the cluster, the litsha2* systems, are each in the running state. The system labeled litsha21 is the current active node, as indicated by the addition of a resource displayed immediately below and indented (IPaddr_1).

Also note the selection labeled "No Quorum Policy" to the value "stop". This means that any isolated node releases resources it would otherwise own. The implication of that decision is that at any given time, 2 heartbeat nodes must be active to establish quorum (in other words, a voting majority). Even if a single active, 100% operational node loses connection to its peer systems due to network failure or if both the inactive peers halt simultaneously, the resource will be voluntarily released.

## Step 4: Creating LVS rules with the ipvsadm command

The next step is to take the floating resource IP address and build on it. Because LVS is intended to be transparent to remote Web browser clients, all Web requests must be funneled through the directors and passed on to one of the realservers. Then any results need to be relayed back to the director, which then returns the response to the client who initiated the Web page request.

To accomplish that flow of requests and responses, first configure each of the LVS directors to enable IP forwarding (thus allowing requests to be passed on to the realservers) by issuing the following commands:

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

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

If all was successful, the second command would return a "1" as output to your terminal. To add this permanently, add:

```
'' IP_FORWARD="yes"
```

to /etc/sysconfig/sysctl.

Next, to tell the directors to relay incoming HTTP requests to the HA floating IP address on to the realservers, use the `ipvsadm` command.

First, clear the old ipvsadm tables:

```
# /sbin/ipvsadm -C
```

Before you can configure the new tables, you need to decide what kind of workload distribution you want the LVS directors to use. On receipt of a connect request from a client, the director assigns a realserver to the client based on a "schedule," and you will set the scheduler type with the `ipvsadm` command. Available schedulers include:

- **Round Robin (RR)**: New incoming connections are assigned to each realserver in turn.
- **Weighted Round Robin (WRR)**: RR scheduling with additional weighting factor to compensate for differences in realserver capabilities such as additional CPUs, more memory, and so on.
- **Least Connected (LC)**: New connections go to the realserver with the least number of connections. This is not necessarily the least-busy realserver, but it is a step in that direction.
- **Weighted Least Connection** (WLC): LC with weighting.

It is a good idea to use RR scheduling for testing, as it is easy to confirm. You may want to add WRR and LC to your testing routine to confirm that they work as expected. The examples shown here assume RR scheduling and its variants.

Next, create a script to enable ipvsadm service forwarding to the realservers, and place a copy on each LVS director. This script will not be necessary when the later configuration of mon is done to automatically monitor for active realservers, but it aids in testing the ipvsadm component until then. Remember to double-check for proper network and http/https connectivity to each of your realservers before executing this script.

**Listing 5. The HA_CONFIG.sh file**

```
#!/bin/sh

# The virtual address on the director which acts as a cluster address

VIRTUAL_CLUSTER_ADDRESS=192.168.71.205

REAL_SERVER_IP_1=192.168.71.220

REAL_SERVER_IP_2=192.168.71.150

REAL_SERVER_IP_3=192.168.71.121

REAL_SERVER_IP_4=192.168.71.145

REAL_SERVER_IP_5=192.168.71.185

REAL_SERVER_IP_6=192.168.71.186

# set ip_forward ON for vs-nat director (1 on, 0 off).

cat /proc/sys/net/ipv4/ip_forward

echo "1" >/proc/sys/net/ipv4/ip_forward

# director acts as the gw for realservers

# Turn OFF icmp redirects (1 on, 0 off), if not the realservers might be clever and

#  not use the director as the gateway!

echo "0" >/proc/sys/net/ipv4/conf/all/send_redirects

echo "0" >/proc/sys/net/ipv4/conf/default/send_redirects
echo "0" >/proc/sys/net/ipv4/conf/eth0/send_redirects
# Clear ipvsadm tables (better safe than sorry)
/sbin/ipvsadm -C
# We install LVS services with ipvsadm for HTTP and HTTPS connections with RR
#  scheduling
/sbin/ipvsadm -A -t $VIRTUAL_CLUSTER_ADDRESS:http -s rr
/sbin/ipvsadm -A -t $VIRTUAL_CLUSTER_ADDRESS:https -s rr
# First realserver
# Forward HTTP to REAL_SERVER_IP_1 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_1:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_1:https -m -w 1
# Second realserver
# Forward HTTP to REAL_SERVER_IP_2 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_2:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_2:https -m -w 1
# Third realserver
# Forward HTTP to REAL_SERVER_IP_3 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_3:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_3:https -m -w 1
# Fourth realserver
# Forward HTTP to REAL_SERVER_IP_4 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_4:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_4:https -m -w 1
# Fifth realserver
# Forward HTTP to REAL_SERVER_IP_5 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_5:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_5:https -m -w 1
# Sixth realserver
# Forward HTTP to REAL_SERVER_IP_6 using LVS-NAT (-m), with weight=1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:http -r $REAL_SERVER_IP_6:http -m -w 1
/sbin/ipvsadm -a -t $VIRTUAL_CLUSTER_ADDRESS:https -r $REAL_SERVER_IP_6:https -m -w 1
# We print the new ipvsadm table for inspection
echo "NEW IPVSADM TABLE:"
/sbin/ipvsadm
```

As you can see in Listing 5, the script simply enables the ipvsadm services, then has virtually identical stanzas to forward Web and SSL requests to each of the individual realservers. We used the `-m` option to specify NAT, and weight each realserver equally with a weight of 1 (`-w 1`). The weights specified are superfluous when using normal round robin scheduling (as the default weight is always 1). The option is presented only so that you may deviate to select weighted round robin. To do so change `rr` to `wrr` on the 2 consecutive lines below the comment about using round robin, and of course do not forget to adjust the weights accordingly. For more information about the various schedulers, consult the man page for ipvsadm.

You have now configured each director to handle incoming Web and SSL requests to the floating service IP by rewriting them and passing the work on to the realservers in succession. But in order to get traffic back from the realservers, and do the reverse process before handing the requests back to the client who made the request, you need to alter a few of the networking settings on the directors. This is necessary because of the decision to implement LVS directors and realservers in a flat network topology (that is, all on the same subnet). We need to perform the following steps to force the Apache response traffic back through the directors rather than answering directly themselves:

```
echo "0" > /proc/sys/net/ipv4/conf/all/send_redirects
echo "0" > /proc/sys/net/ipv4/conf/default/send_redirects
echo "0" > /proc/sys/net/ipv4/conf/eth0/send_redirects
```

This was done to prevent the active LVS director from trying to take a TCP/IP shortcut by informing the realserver and floating service IP to talk directly to one another (since they are on the same subnet). Normally redirects are useful, as they improve performance by cutting out unnecessary middlemen in network connections. But in this case, it would have prevented the response traffic from being rewritten as is necessary for transparency to the client. In fact, if redirects were not disabled on the LVS director, the traffic being sent from the realserver directly to the client would appear to the client as an unsolicited network response and would be discarded.

At this point, it is time to set the default route of each of the realservers to point at the service floating IP address to ensure all responses are passed back to the director for packet rewriting before being passed back to the client that originated the request.

Once redirects are disabled on the directors, and the realservers are configured to route all traffic through the floating service IP, you may proceed to test your HA LVS environment. To test the work done thus far, point a Web browser on a remote client to the floating service address of the LVS directors.

For testing in the laboratory, we used a Gecko-based browser (Mozilla), though any browser should suffice. To ensure the deployment was successful, disable caching in the browser, and click the refresh button multiple times. With each refresh, you should see that the Web page displayed is one of the self-identifying pages configured on the realservers. If you are using RR scheduling, you should observe the page cycling through each of realservers in succession.

Are you now thinking of ensuring that the LVS configuration starts automatically at boot? Don't do that just yet! There is one more step needed (Step 5) to perform active monitoring of the realservers (thus keeping a dynamic list of which Apache nodes are eligible to service work request).

## Step 5: Installing and configuring mon on the LVS directors

So far, you have established a highly available service IP address and bound that to the pool of realserver instances. But you must never trust any of the individual Apache servers to be operational at any given time. By choosing RR scheduling, if any given realserver becomes disabled, or ceases to respond to network traffic in a timely fashion, 1/6th of the HTTP requests would be failures!

Thus it is necessary to implement monitoring of the realservers on each of the the LVS directors in order to dynamically add or remove them from the service pool. Another well-known open source package called mon is well suited for this task.

The mon solution is commonly used for monitoring LVS realnodes. Mon is relatively easy to configure, and is very extensible for people familiar with shell scripting. There are essentially three main steps to get everything working: installation, service monitoring configuration, and alert creation. Use your package management tool to handle the installation of mon. When finished with the installation, you need only to adjust the monitoring configuration, and create some alert scripts. The alert scripts are triggered when the monitors determine that a realserver has gone offline, or come back online.

Note that with heartbeat v2 installations, monitoring of realservers can be accomplished by making all the realserver services resources. Or, you can use the Heartbeat directord package.

By default, mon comes with several monitor mechanisms ready to be used. We altered sample configuration file in /etc/mon.cf to make use of the HTTP service.

In the mon configuration file, ensure the header reflects the proper paths. SLES10 is a 64-bit Linux image, but the sample configuration as shipped was for the default (31- or 32-bit) locations. The configuration file sample assumed the alerts and monitors are located /usr/lib, which was incorrect for our particular installation. The parameters we altered were as follows:

```
alertdir = /usr/lib64/mon/alert.d
mondir = /usr/lib64/mon/mon.d
```

As you can see, we simply changed `lib` to `lib64`. Such a change may not be necessary for your distribution.

The next change to the configuration file was to specify the list of realservers to monitor. This was done with the following 6 directives:

**Listing 6. Specifying realservers to monitor**

```
hostgroup litstat1 192.168.71.220 # realserver 1

hostgroup litstat2 192.168.71.150

hostgroup litstat3 192.168.71.121

hostgroup litstat4 192.168.71.145

hostgroup litstat5 192.168.71.185

hostgroup litstat6 192.168.71.186 # realserver 6
```

If you want to add additional realservers, simply add additional entries here.

Once you have all of your definitions in place, you need to tell mon how to watch for failure, and what to do in case of failure. To do this, add the following monitor sections (one for each realserver). When done, you will need to place both the mon configuration file and the alert on each of the LVS heartbeat nodes, enabling each heartbeat cluster node to independently monitor all of the realservers.

**Listing 7. The /etc/mon/mon.cf file**

```
#

# global options

#

cfbasedir    = /etc/mon

alertdir     = /usr/lib64/mon/alert.d

mondir       = /usr/lib64/mon/mon.d

statedir     = /var/lib/mon

logdir       = /var/log
```

```
maxprocs     = 20
histlength   = 100
historicfile = mon_history.log
randstart    = 60s
#
# authentication types:
#   getpwnam      standard Unix passwd, NOT for shadow passwords
#   shadow        Unix shadow passwords (not implemented)
#   userfile      "mon" user file
#
authtype = getpwnam
#
# downtime logging, uncomment to enable
#  if the server is running, don't forget to send a reset command
#  when you change this
#
#dtlogfile = downtime.log
dtlogging = yes
#
# NB:  hostgroup and watch entries are terminated with a blank line (or
#  end of file).  Don't forget the blank lines between them or you lose.
#
#
# group definitions (hostnames or IP addresses)
# example:
#
# hostgroup servers www mail pop server4 server5
#
# For simplicity we monitor each individual server as if it were a "group"
#  so we add only the hostname and the ip address of an individual node for each.
hostgroup litstat1 192.168.71.220
hostgroup litstat2 192.168.71.150
hostgroup litstat3 192.168.71.121
hostgroup litstat4 192.168.71.145
hostgroup litstat5 192.168.71.185
hostgroup litstat6 192.168.71.186
#
# Now we set identical watch definitions on each of our groups. They could be
#  customized to treat individual servers differently, but we have made the
#  configurations homogeneous here to match our homogeneous LVS configuration.
#
 watch litstat1
     service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
 watch litstat2
     service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
 watch litstat3
     service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
 watch litstat4
     service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
watch litstat5
    service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
watch litstat6
    service http
        description http check servers
        interval 6s
        monitor http.monitor -p 80 -u /index.html
        allow_empty_group
        period wd {Mon-Sun}
            alert dowem.down.alert -h
            upalert dowem.up.alert -h
            alertevery 600s
                alertafter 1
```

Listing 7 tells mon to use the http.monitor, which is shipped with mon by default. Additionally, port 80 is specified as the port to use. Listing 7 also provides the specific page to request; you may opt to transmit a more efficient small segment of html as proof of success rather than a complicated default html page for your Web server. The `alert` and `upalert` lines invoke scripts that must be placed in the `alertdir` specified at the top of the configuration file. The directory is typically something that is the distribution default, such as "/usr/lib64/mon/alert.d/". The alerts are responsible for telling LVS to add or remove Apache servers from the eligibility list (by invoking the `ipvsadm` command as we shall see in a moment). When one of the realservers fails the http test, the dowem.down.alert will be executed by mon with several arguments automatically. Likewise, when the monitors determine that a realserver has come back online, the mon process executes the dowem.up.alert with the numerous arguments automatically. Feel free to alter the names of the alert scripts to suit your own deployment.

Save this file, and create the alerts (using simple bash scripting) in the alertdir. Listing 8 shows a bash script alert that will be invoked by mon when a real server connection is re-established.

**Listing 8. Simple alert: we have connectivity**

```
#! /bin/bash
#   The h arg is followed by the hostname we are interested in acting on
#   So we skip ahead to get the -h option since we don't care about the others
REALSERVER=192.168.71.205
while [ $1 != "-h" ] ;
do
        shift
done
ADDHOST=$2
# For the HTTP service
/sbin/ipvsadm -a -t $REALSERVER:http -r $ADDHOST:http -m -w 1
# For the HTTPS service
/sbin/ipvsadm -a -t $REALSERVER:https -r $ADDHOST:https -m -w 1
```

Listing 9 shows a bash script alert that will be invoked by mon when a real server connection is lost.

**Listing 9. Simple alert: we have lost connectivity**

```
#! /bin/bash
#   The h arg is followed by the hostname we are interested in acting on
#   So we skip ahead to get the -h option since we dont care about the others
REALSERVER=192.168.71.205
while [ $1 != "-h" ] ;
do
        shift
done
BADHOST=$2
# For the HTTP service
/sbin/ipvsadm -d -t $REALSERVER:http -r $BADHOST
# For the HTTPS service
/sbin/ipvsadm -d -t $REALSERVER:https -r $BADHOST
```

Both of those scripts use of the `ipvsadm` command-line tool to dynamically add and remove realservers from the LVS tables. Note that these scripts are far from perfect. With mon monitoring only the http port for simple Web requests, the architecture as outlined here is vulnerable to situations where a given realserver might be operating correctly for http requests but not for SSL requests. Under those circumstances, we would fail to remove the offending realserver from the list of https candidates. Of course, this is easily remedied by making more advanced alerts specifically for each type of Web request in addition to enabling a second https monitor for each realserver in the mon configuration file. This is left as an exercise for the reader. Note that you are not forced to use bash scripts to implement your own alerts or monitors.

To ensure monitoring has been activated, enable and disable the Apache process on each of the realservers in sequence, observing each of the directors for their reaction to the events. Only when you have confirmed that each director is properly monitoring each realserver, should you use the `chkconfig` command to make sure that the mon process starts automatically at boot. The specific command used was `chkconfig mon on`, but this may vary based on your distribution.

With this last piece in place, you have finished the task of constructing a cross-system, highly-available Web server infrastructure. Of course, you might now opt to do more advanced work. For instance, you may have noticed that the mon daemon itself is not monitored (the heartbeat project can monitor mon for you), but with this last step, the basic foundation has been laid.

## Troubleshooting

There are many reasons why an active node could stop functioning properly in an HA cluster, either voluntarily or involuntarily. The node could lose network connectivity to the other nodes, the heartbeat process could be stopped, there might be any one of a number of environmental occurrences, and so on. To deliberately fail the active node, you can issue a halt on that node, or set it to standby mode using the `hb_gui` (clean take down) command. If you feel inclined to test the robustness of your environment, you might opt to be a bit more aggressive (yank the plug!).

### Indicators and failover

There are two types of log file indicators available to the system administrator responsible for configuring a Linux HA heartbeat system. The log files vary depending on whether or not a system is the recipient of the floating resource IP address.

On members of the cluster that did not receive the Floating Resource IP Address:

**Listing 10. Log results for also-rans**

```
litsha21:~ # cat  /var/log/messages

Jan 16 12:00:20 litsha21 heartbeat: [3057]: WARN: node litsha23: is dead

Jan 16 12:00:21 litsha21 cib: [3065]: info: mem_handle_event: Got an event

 OC_EV_MS_NOT_PRIMARY from ccm

Jan 16 12:00:21 litsha21 cib: [3065]: info: mem_handle_event: instance=13, nodes=3,

 new=1, lost=0, n_idx=0, new_idx=3, old_idx=6

Jan 16 12:00:21 litsha21 crmd: [3069]: info: mem_handle_event: Got an event

 OC_EV_MS_NOT_PRIMARY from ccm

Jan 16 12:00:21 litsha21 crmd: [3069]: info: mem_handle_event: instance=13, nodes=3,

 new=1, lost=0, n_idx=0, new_idx=3, old_idx=6

Jan 16 12:00:21 litsha21 crmd: [3069]: info: crmd_ccm_msg_callback:callbacks.c Quorum

 lost after event=NOT PRIMARY (id=13)

Jan 16 12:00:21 litsha21 heartbeat: [3057]: info: Link litsha23:eth1 dead.

Jan 16 12:00:38 litsha21 ccm: [3064]: debug: quorum plugin: majority

Jan 16 12:00:38 litsha21 ccm: [3064]: debug: cluster:linux-ha, member_count=2,

 member_quorum_votes=200
```

```
Jan 16 12:00:38 litsha21 ccm: [3064]: debug: total_node_count=3,
 total_quorum_votes=300
                   ................. Truncated For Brevity .................
Jan 16 12:00:40 litsha21 crmd: [3069]: info: update_dc:utils.c Set DC to litsha21
 (1.0.6)
Jan 16 12:00:41 litsha21 crmd: [3069]: info: do_state_transition:fsa.c litsha21:
 State transition S_INTEGRATION ->
S_FINALIZE_JOIN [ input=I_INTEGRATED cause=C_FSA_INTERNAL
 origin=check_join_state ]
Jan 16 12:00:41 litsha21 crmd: [3069]: info: do_state_transition:fsa.c All 2 cluster
 nodes responded to the join offer.
Jan 16 12:00:41 litsha21 crmd: [3069]: info: update_attrd:join_dc.c Connecting to
 attrd...
Jan 16 12:00:41 litsha21 cib: [3065]: info: sync_our_cib:messages.c Syncing CIB to
 all peers
Jan 16 12:00:41 litsha21 attrd: [3068]: info: attrd_local_callback:attrd.c Sending
 full refresh
                   ................. Truncated For Brevity .................
Jan 16 12:00:43 litsha21 pengine: [3112]: info: unpack_nodes:unpack.c Node litsha21
 is in standby-mode
Jan 16 12:00:43 litsha21 pengine: [3112]: info: determine_online_status:unpack.c Node
 litsha21 is online
Jan 16 12:00:43 litsha21 pengine: [3112]: info: determine_online_status:unpack.c Node
 litsha22 is online
Jan 16 12:00:43 litsha21 pengine: [3112]: info: IPaddr_1
        (heartbeat::ocf:IPaddr): Stopped
Jan 16 12:00:43 litsha21 pengine: [3112]: notice: StartRsc:native.c  litsha22
   Start IPaddr_1
Jan 16 12:00:43 litsha21 pengine: [3112]: notice: Recurring:native.c litsha22
      IPaddr_1_monitor_5000
Jan 16 12:00:43 litsha21 pengine: [3112]: notice: stage8:stages.c Created transition
 graph 0.
                   ................. Truncated For Brevity .................
Jan 16 12:00:46 litsha21 mgmtd: [3070]: debug: update cib finished
Jan 16 12:00:46 litsha21 crmd: [3069]: info: do_state_transition:fsa.c litsha21:
 State transition S_TRANSITION_ENGINE ->
 S_IDLE [ input=I_TE_SUCCESS cause=C_IPC_MESSAGE origin=do_msg_route ]
Jan 16 12:00:46 litsha21 cib: [3118]: info: write_cib_contents:io.c Wrote version
 0.53.593 of the CIB to disk (digest: 83b00c386e8b67c42d033a4141aaef90)
```

As you can see from Listing 10, a roll is taken, and sufficient members for quorum are available for the vote. A vote is taken, and normal operation is resumed with no further action needed.

On the member of the cluster who received the floating resource IP address:

**Listing 11. The log file of the resource holder**

```
litsha22:~ # cat  /var/log/messages

Jan 16 12:00:06 litsha22 syslog-ng[1276]: STATS: dropped 0

Jan 16 12:01:51 litsha22 heartbeat: [3892]: WARN: node litsha23: is dead

Jan 16 12:01:51 litsha22 heartbeat: [3892]: info: Link litsha23:eth1 dead.

Jan 16 12:01:51 litsha22 cib: [3900]: info: mem_handle_event: Got an event

 OC_EV_MS_NOT_PRIMARY from ccm

Jan 16 12:01:51 litsha22 cib: [3900]: info: mem_handle_event: instance=13, nodes=3,

 new=3, lost=0, n_idx=0, new_idx=0, old_idx=6

Jan 16 12:01:51 litsha22 crmd: [3904]: info: mem_handle_event: Got an event

 OC_EV_MS_NOT_PRIMARY from ccm

Jan 16 12:01:51 litsha22 crmd: [3904]: info: mem_handle_event: instance=13, nodes=3,

 new=3, lost=0, n_idx=0, new_idx=0, old_idx=6

Jan 16 12:01:51 litsha22 crmd: [3904]: info: crmd_ccm_msg_callback:callbacks.c Quorum

 lost after event=NOT PRIMARY (id=13)

Jan 16 12:02:09 litsha22 ccm: [3899]: debug: quorum plugin: majority

Jan 16 12:02:09 litsha22 crmd: [3904]: info: do_election_count_vote:election.c

 Election check: vote from litsha21

Jan 16 12:02:09 litsha22 ccm: [3899]: debug: cluster:linux-ha, member_count=2,

 member_quorum_votes=200

Jan 16 12:02:09 litsha22 ccm: [3899]: debug: total_node_count=3,

 total_quorum_votes=300

Jan 16 12:02:09 litsha22 cib: [3900]: info: mem_handle_event: Got an event

 OC_EV_MS_INVALID from ccm

Jan 16 12:02:09 litsha22 cib: [3900]: info: mem_handle_event: no mbr_track info

Jan 16 12:02:09 litsha22 cib: [3900]: info: mem_handle_event: Got an event

 OC_EV_MS_NEW_MEMBERSHIP from ccm

Jan 16 12:02:09 litsha22 cib: [3900]: info: mem_handle_event: instance=14, nodes=2,

 new=0, lost=1, n_idx=0, new_idx=2, old_idx=5

Jan 16 12:02:09 litsha22 cib: [3900]: info: cib_ccm_msg_callback:callbacks.c

 LOST: litsha23

Jan 16 12:02:09 litsha22 cib: [3900]: info: cib_ccm_msg_callback:callbacks.c

 PEER: litsha21

Jan 16 12:02:09 litsha22 cib: [3900]: info: cib_ccm_msg_callback:callbacks.c
```

```
 PEER: litsha22
                   ................. Truncated For Brevity .................
Jan 16 12:02:12 litsha22 crmd: [3904]: info: update_dc:utils.c Set DC to litsha21
 (1.0.6)
Jan 16 12:02:12 litsha22 crmd: [3904]: info: do_state_transition:fsa.c litsha22:
 State transition S_PENDING -> S_NOT_DC [ input=I_NOT_DC cause=C_HA_MESSAGE
 origin=do_cl_join_finalize_respond ]
Jan 16 12:02:12 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Update (client:
 3069, call:25): 0.52.585 ->
0.52.586 (ok)
                   ................. Truncated For Brevity .................
Jan 16 12:02:14 litsha22 IPaddr[3998]: INFO: /sbin/ifconfig eth0:0 192.168.71.205
 netmask 255.255.255.0 broadcast 192.168.71.255
Jan 16 12:02:14 litsha22 IPaddr[3998]: INFO: Sending Gratuitous Arp for
 192.168.71.205 on eth0:0 [eth0]
Jan 16 12:02:14 litsha22 IPaddr[3998]: INFO: /usr/lib64/heartbeat/send_arp -i 500 -r
 10 -p
/var/run/heartbeat/rsctmp/send_arp/send_arp-192.168.71.205 eth0 192.168.71.205 auto
 192.168.71.205 ffffffffffff
Jan 16 12:02:14 litsha22 crmd: [3904]: info: process_lrm_event:lrm.c LRM operation
 (3) start_0 on IPaddr_1 complete
Jan 16 12:02:14 litsha22 kernel: send_arp uses obsolete (PF_INET,SOCK_PACKET)
Jan 16 12:02:14 litsha22 kernel: klogd 1.4.1, ---------- state change ----------
Jan 16 12:02:14 litsha22 kernel: NET: Registered protocol family 17
Jan 16 12:02:15 litsha22 crmd: [3904]: info: do_lrm_rsc_op:lrm.c Performing op
 monitor on IPaddr_1 (interval=5000ms, key=0:f9d962f0-4ed6-462d-a28d-e27b6532884c)
Jan 16 12:02:15 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Update (client:
 3904, call:18): 0.53.591 ->
0.53.592
 (ok)
Jan 16 12:02:15 litsha22 mgmtd: [3905]: debug: update cib finished
```

As shown in Listing 11, the /var/log/messages file shows this node has acquired the floating resource. The `ifconfig` line shows the eth0:0 device being created dynamically to maintain service.

And as you can see from Listing 11, a roll is taken, and sufficient members for quorum are available for the vote. A vote is taken, followed by the `ifconfig` commands that are issued to claim the floating resource IP address.

As an additional means of indicating when a failure has occurred, a systems programmer may log in to any of the cluster members and execute the `hb_gui` command. Through this method, a system programmer can determine by visual inspection which system has the floating resource.

Lastly, we would be remiss if we did not illustrate a sample log file from a no-quorum situation. If any singular node cannot communicate with either of its peers, it has lost quorum (since 2/3 is the majority in a three-member voting party). In this situation, the node understands that it has lost quorum, and invokes the no quorum policy handler. Listing 12 shows an example of the log file from such an event. When quorum is lost, a log entry indicates it. The cluster node showing this log entry will disown the floating resource. The `ifconfig down` statement releases it.

**Listing 12. Log entry showing loss of quorum**

```
litsha22:~ # cat /var/log/messages
....................
Jan 16 12:06:12 litsha22 ccm: [3899]: debug: quorum plugin: majority
Jan 16 12:06:12 litsha22 ccm: [3899]: debug: cluster:linux-ha, member_count=1,
 member_quorum_votes=100
Jan 16 12:06:12 litsha22 ccm: [3899]: debug: total_node_count=3,
 total_quorum_votes=300
                    ................. Truncated For Brevity .................
Jan 16 12:06:12 litsha22 crmd: [3904]: info: crmd_ccm_msg_callback:callbacks.c Quorum
 lost after event=INVALID (id=15)
Jan 16 12:06:12 litsha22 crmd: [3904]: WARN: check_dead_member:ccm.c Our DC node
 (litsha21) left the cluster
                    ................. Truncated For Brevity .................
Jan 16 12:06:14 litsha22 IPaddr[5145]: INFO: /sbin/route -n del -host 192.168.71.205
Jan 16 12:06:15 litsha22 lrmd: [1619]: info: RA output: (IPaddr_1:stop:stderr)
 SIOCDELRT: No such process
Jan 16 12:06:15 litsha22 IPaddr[5145]: INFO: /sbin/ifconfig eth0:0 192.168.71.205
 down
Jan 16 12:06:15 litsha22 IPaddr[5145]: INFO: IP Address 192.168.71.205 released
Jan 16 12:06:15 litsha22 crmd: [3904]: info: process_lrm_event:lrm.c LRM operation
 (6) stop_0 on IPaddr_1 complete
Jan 16 12:06:15 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Update (client:
 3904, call:32): 0.54.599 ->
0.54.600 (ok)
Jan 16 12:06:15 litsha22 mgmtd: [3905]: debug: update cib finished
```

As you can see from the Listing 12, when quorum is lost for any given node, it relinquishes any resources as a result of the chosen no quorum policy configuration. The choice of no quorum policy is up to the systems programmer.

## Fail-back actions and messages

One of the more interesting implications of a properly-configured Linux HA system is that the systems programmer does not need to take any action to re-instantiate a cluster member. Simply activating the Linux instance is sufficient to let the node rejoin its peers automatically. If you have configured a primary node (that is, one that is favored to gain the floating resource above all others), it will regain the floating resources automatically. Non-favored systems will simply join the eligibility pool and proceed as normal.

Adding another Linux instance back into the pool will cause each node to take notice, and if possibly, re-establish quorum. The floating resources will be re-established on one of the nodes if quorum can be re-established.

**Listing 13. Quorum is re-established**

```
litsha22:~ # tail -f /var/log/messages
Jan 16 12:09:02 litsha22 heartbeat: [3892]: info: Heartbeat restart on node litsha21
Jan 16 12:09:02 litsha22 heartbeat: [3892]: info: Link litsha21:eth1 up.
Jan 16 12:09:02 litsha22 heartbeat: [3892]: info: Status update for node litsha21:
 status init
Jan 16 12:09:02 litsha22 heartbeat: [3892]: info: Status update for node litsha21:
 status up
Jan 16 12:09:22 litsha22 heartbeat: [3892]: debug: get_delnodelist: delnodelist=
Jan 16 12:09:22 litsha22 heartbeat: [3892]: info: Status update for node litsha21:
 status active
Jan 16 12:09:22 litsha22 cib: [3900]: info: cib_client_status_callback:callbacks.c
 Status update: Client litsha21/cib now has status [join]
Jan 16 12:09:23 litsha22 heartbeat: [3892]: WARN: 1 lost packet(s) for [litsha21]
 [36:38]
Jan 16 12:09:23 litsha22 heartbeat: [3892]: info: No pkts missing from litsha21!
Jan 16 12:09:23 litsha22 crmd: [3904]: notice:
 crmd_client_status_callback:callbacks.c Status update: Client litsha21/crmd now has
 status [online]
....................
Jan 16 12:09:31 litsha22 crmd: [3904]: info: crmd_ccm_msg_callback:callbacks.c Quorum
 (re)attained after event=NEW MEMBERSHIP (id=16)
Jan 16 12:09:31 litsha22 crmd: [3904]: info: ccm_event_detail:ccm.c NEW MEMBERSHIP:
 trans=16, nodes=2, new=1, lost=0 n_idx=0, new_idx=2, old_idx=5
Jan 16 12:09:31 litsha22 crmd: [3904]: info: ccm_event_detail:ccm.c     CURRENT:
 litsha22 [nodeid=1, born=13]
Jan 16 12:09:31 litsha22 crmd: [3904]: info: ccm_event_detail:ccm.c     CURRENT:
 litsha21 [nodeid=0, born=16]
Jan 16 12:09:31 litsha22 crmd: [3904]: info: ccm_event_detail:ccm.c     NEW:
     litsha21 [nodeid=0, born=16]
Jan 16 12:09:31 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Local-only
 Change (client:3904, call: 35):
0.54.600 (ok)
Jan 16 12:09:31 litsha22 mgmtd: [3905]: debug: update cib finished
```

```
....................
Jan 16 12:09:34 litsha22 crmd: [3904]: info: update_dc:utils.c Set DC to litsha22
 (1.0.6)
Jan 16 12:09:35 litsha22 cib: [3900]: info: sync_our_cib:messages.c Syncing CIB to
 litsha21
Jan 16 12:09:35 litsha22 crmd: [3904]: info: do_state_transition:fsa.c litsha22:
 State transition S_INTEGRATION ->
 S_FINALIZE_JOIN [ input=I_INTEGRATED cause=C_FSA_INTERNAL origin=check_join_state ]
Jan 16 12:09:35 litsha22 crmd: [3904]: info: do_state_transition:fsa.c All 2 cluster
 nodes responded to the join offer.
Jan 16 12:09:35 litsha22 attrd: [3903]: info: attrd_local_callback:attrd.c Sending
 full refresh
Jan 16 12:09:35 litsha22 cib: [3900]: info: sync_our_cib:messages.c Syncing CIB to
 all peers
.........................
Jan 16 12:09:37 litsha22 tengine: [5119]: info: send_rsc_command:actions.c Initiating
 action 4: IPaddr_1_start_0 on litsha22
Jan 16 12:09:37 litsha22 tengine: [5119]: info: send_rsc_command:actions.c Initiating
 action 2: probe_complete on litsha21
Jan 16 12:09:37 litsha22 crmd: [3904]: info: do_lrm_rsc_op:lrm.c Performing op start
 on IPaddr_1 (interval=0ms,
 key=2:c5131d14-a9d9-400c-a4b1-60d8f5fbbcce)
Jan 16 12:09:37 litsha22 pengine: [5120]: info: process_pe_message:pengine.c
 Transition 2: PEngine Input stored in: /var/lib/heartbeat/pengine/pe-input-72.bz2
Jan 16 12:09:37 litsha22 IPaddr[5196]: INFO: /sbin/ifconfig eth0:0 192.168.71.205
 netmask 255.255.255.0 broadcast 192.168.71.255
Jan 16 12:09:37 litsha22 IPaddr[5196]: INFO: Sending Gratuitous Arp for
 192.168.71.205 on eth0:0 [eth0]
Jan 16 12:09:37 litsha22 IPaddr[5196]: INFO: /usr/lib64/heartbeat/send_arp -i 500 -r
 10 -p
 /var/run/heartbeat/rsctmp/send_arp/send_arp-192.168.71.205 eth0 192.168.71.205 auto
 192.168.71.205 ffffffffffff
Jan 16 12:09:37 litsha22 crmd: [3904]: info: process_lrm_event:lrm.c LRM operation
 (7) start_0 on IPaddr_1 complete
Jan 16 12:09:37 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Update (client:
 3904, call:46): 0.55.607 -> 0.55.608 (ok)
Jan 16 12:09:37 litsha22 mgmtd: [3905]: debug: update cib finished
Jan 16 12:09:37 litsha22 tengine: [5119]: info: te_update_diff:callbacks.c Processing
 diff (cib_update): 0.55.607 -> 0.55.608
```

```
Jan 16 12:09:37 litsha22 tengine: [5119]: info: match_graph_event:events.c Action
 IPaddr_1_start_0 (4) confirmed
Jan 16 12:09:37 litsha22 tengine: [5119]: info: send_rsc_command:actions.c Initiating
 action 5: IPaddr_1_monitor_5000 on litsha22
Jan 16 12:09:37 litsha22 crmd: [3904]: info: do_lrm_rsc_op:lrm.c Performing op
 monitor on IPaddr_1 (interval=5000ms, key=2:c5131d14-a9d9-400c-a4b1-60d8f5fbbcce)
Jan 16 12:09:37 litsha22 cib: [5268]: info: write_cib_contents:io.c Wrote version
 0.55.608 of the CIB to disk (digest: 98cb6685c25d14131c49a998dbbd0c35)
Jan 16 12:09:37 litsha22 crmd: [3904]: info: process_lrm_event:lrm.c LRM operation
 (8) monitor_5000 on IPaddr_1 complete
Jan 16 12:09:38 litsha22 cib: [3900]: info: cib_diff_notify:notify.c Update (client:
 3904, call:47): 0.55.608 -> 0.55.609 (ok)
Jan 16 12:09:38 litsha22 mgmtd: [3905]: debug: update cib finished
```

In Listing 13, you see that quorum has been re-established. When quorum is re-established, a vote is performed and litsha22 becomes the active node with the floating resource.

## Next steps

High availability is best seen as a series of challenges, and the solution outlined here describes the first step. From here, there are many ways to move forward with your environment: you may choose to install redundant networking, a cluster file system to support the realservers, or more advanced middleware that supports clustering directly.

**Share this...**

- Digg this story
- Post to del.icio.us
- Slashdot it!

## Resources

**Learn**

- Wikipedia's article on basic networking is a good resource for learning about basic computer topology and networking tasks. Wikipedia has more information about Network Address Translation (NAT), too.
- Visit the home page of the Apache HTTP server project if you are new to basic Apache configuration. The site offers useful documentation and howto information. See also your distribution-provided Apache man pages for more.
- The Linux Virtual Server (LVS) project provides the LVS technology used in this article, including `ipvsadm`. The site contains a wealth of tutorials and other useful documentation, and provides a mailing list.
- The Linux-HA Web site contains much useful information on high availability, as well as the Heartbeat component described in this article.
- mon is a framework for monitoring service outages and their recovery—check it out.
- In the developerWorks Linux zone, find more resources for Linux developers, including Linux

tutorials, as well as our readers' favorite Linux articles and tutorials over the last month.

- Stay current with developerWorks technical events and Webcasts.

**Get products and technologies**

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- With IBM trial software, available for download directly from developerWorks, build your next development project on Linux.

**Discuss**

- Get involved in the developerWorks community through our developer blogs, forums, podcasts, and community topics in our new developerWorks spaces.

## About the authors

Eli M. Dow is a software engineer in the IBM Test and Integration Center for Linux in Poughkeepsie, NY. He holds a B.S. degree in computer science and psychology and a master's of computer science from Clarkson University. He is an alumnus of the Clarkson Open Source Institute. His interests include the GNOME desktop, human-computer interaction, virtualization, and Linux systems programming. He is the coauthor of an IBM Redbook *Linux for IBM System z9 and IBM zSeries*.

Frank LeFevre is a Senior Software Engineer in the IBM Systems and Technology Group in Poughkeepsie, NY. He has over 28 years of experience with IBM mainframe hardware and operating systems. He is currently the Team Leader for the Linux Virtual Server Platform Evaluation Test team.