Load balancing and high-availability clusters with iptables

STRENGTH IN NUMBERS

Iptables gives admins the ability to set up clusters and distribute the load. But what about failover? **BY MICHAEL SCHWARTZKOPFF**

Load sharing technologies often rely on a central system or application that distributes the work evenly over the members of the cluster. The Linux Virtual Server [1] project implements this on Linux. To avoid a single point of failure, the central instances should be highly available and continuously monitored by a routine that checks the systems and responds to errors or lost signals. If you prefer to avoid a central load sharing instance entirely, the iptables CLUSTERIP target is an alternative. CLUSTERIP is a simple and inexpensive technique for load sharing that is already part of the Netfilter code, and although this feature is not entirely stable, the technology is quite impressive.

In CLUSTERIP, the cluster nodes share a common address, and each node uses a hash algorithm to decide whether it is responsible for a connection. Admins can assign responsibilities to a node via */proc/net/ipt_CLUSTERIP*, influencing load sharing, or switching interactively or by means of dynamic scripting. Stonesoft [2] products have had this functionality for a while, and it works well.

Iptables clusters do not have a built-in heartbeat mechanism to check the health state of the nodes, remove broken systems from the cluster, or tell other nodes to take over the load of the failed system. Many failures are heralded by tell-tale signs, however, that give the ailing node the ability to voluntarily leave the cluster in good time. In this article, I show the possibilities of combining the CLUSTERIP target of iptables with a script controlling the cluster.

Cluster Example

The cluster shown in Figure 1 is made up of two nodes. Each node has an interface on the LAN (*eth0*) and another interface on the management network (*eth1*). The nodes use the second interface to exchange messages. A simple crossover cable is all you need for a two-node cluster. Each interface has an additional virtual cluster address on the LAN; it is this address that clients use to talk to the cluster. Each node autonomously decides whether it is responsible for a connection; of course, it needs to see the packet first to be able to do this.

To allow this to happen, the cluster has a shared IP address and a shared MAC address, which is the same for all nodes. This only works with multicast MAC addresses, which are identifiable in that the low-order bit in the higher order byte is set. The multicast address prevents address conflicts.

This approach has one drawback, however: RFC 1812 [3] states that a router should not trust an ARP (Address Resolution Protocol) reply if it assigns a multicast or broadcast MAC address as the IP address. Routers that strictly observe the RFC will need a static entry in their ARP tables.

Network Wizardry

Switches normally forward any incoming multicast packets to all interfaces. This can cause considerable confusion in the case of high-availability (HA) architectures with HSRP routers or with two switches, as in this example.

Switch 1, which is active, accepts the packet off the LAN and passes it on to the nodes, as well as to the second switch, to make sure that node 2 also receives the packet. Of course, switch 2 would normally pass the packet on to node 2 and back to the first switch. Some older switches do fall into this trap, and the LAN comes to a halt.

Table 1: Cluster Modes

Mode	Approach
sourceip	Only uses the source IP address for the hash and thus assigns exactly one node to each client.
sourceip-sourceport	Additionally uses information about the application's source port. This enhances load distribution between nodes.
sourceip-sourceport-destport	Additionally uses information about the application's destination port.

If a switch cannot learn multicast addresses by means of the normal mechanism, its configuration has to specify the incoming and outgoing ports responsible for multicast packets to make sure the switch forwards each packet exactly once to the LAN interface of each node. If both nodes of the cluster listen on interface 1 and 2, the command for Cisco switches would be:

```
mac-address-table static ⮒
01:02:03:04:05:06 interface ⮒
FastEthernet0/1 FastEthernet0/2
```

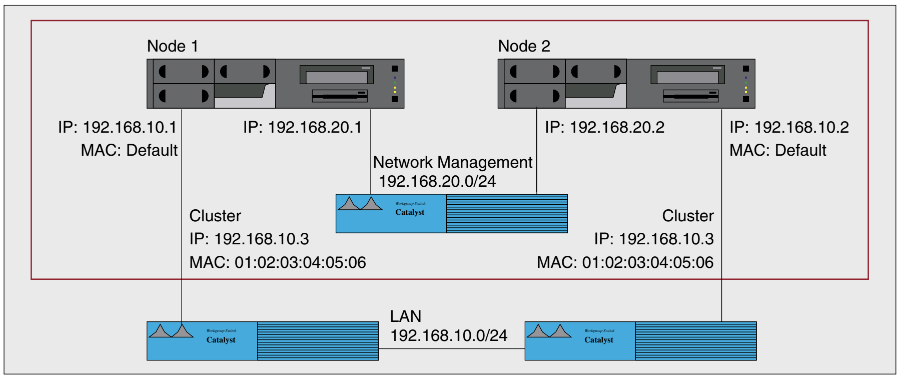

Figure 1: Two-node cluster. The LAN interfaces of all nodes are bonded to create a virtual cluster interface; the two nodes use a management network to talk to and control each other.

In high-available scenarios with double switches (one for each node), the spanning tree protocol can help to prevent loops on the path to the target.

CLUSTERIP Target

The cluster has to pass all the packets that belong to a specific connection to the same node. This method is the only way to prevent clients from receiving duplicate responses, or servers from discarding packets because they are not aware of the connection's history.

The cluster software in the Netfilter packet does this autonomously in an elegant way. At the configuration phase, each cluster is given a serial number; these are the numbers 1 and 2 in this example. For each connection, the iptables target uses Bob Jenkins' approach [4] to calculate a hash from the connection data and maps this to a range of 1 through 2 to discover whether the node is responsible for the connection.

The connection status (Netfilter's connection tracking function) makes sure that the connection stays assigned to the node. The type of data used by the hash mechanism depends on the cluster mode (see Table 1). As a test, I only want the cluster to respond to ICMP echo requests; in this context, *sourceip* is the only meaningful cluster mode setting – ICMP packets do not have a port number. *sourceip-sourceport* would be preferable for application clusters such as web server farms.

Table 2: Options

Switch	Function
-d	IP address of the cluster on the LAN
-i	LAN interface
--hashmode	Hash mode
--clustermac	Cluster MAC address
--total-nodes	Total number of nodes in the cluster
--local-node	Node to configure

Cluster Configuration

The cluster configuration consists of a single iptables command per node. Table 2 explains the options. The following example restricts the cluster to ICMP echo requests (ping):

```
iptables -I INPUT -d ⮒
192.168.10.3 ⮒
-p icmp --icmp-type ⮒
echo-request ⮒
-j CLUSTERIP --new ⮒
--hashmode sourceip ⮒
--clustermac 01:02:03:04:05:06 ⮒
--total-nodes 2 --local-node 1
```

The configuration for node 2 is almost identical; however, in this case, I need a value of *2* for *--local--node*.

From the outside (*ifconf* output), you can't tell that the interfaces belong to the cluster, and the kernel does not know that it has to react to the cluster's IP address. The best way to change this is to use an *ip* command:

```
ip address add ⮒
192.168.10.3/24 dev eth0
```

This command adds the cluster address as an extra IP for this interface. The cluster will respond to pings after doing so. If you are wondering what happened to the multicast MAC address, *iptables* takes care of this automatically.

The hash value for this computer – that is, the number the computer responds to – is stored in */proc/net/ipt_CLUSTERIP/192.168.10.3*. This number can be modified at run time to make node 2 responsible for node 1:

```
echo "+1" > ⮒
/proc/net/ipt_CLUSTERIP/⮒
192.168.10.3
```

Running this command on node 2 means that the cluster will respond to each ping with two echo reply packets. */proc/net/ipt_CLUSTERIP/192.168.10.3* on computer 2 now reads *1,2*. *echo "-1"*... removes responsibility for the hash value of *1*. This allows admins to switch the node off and assign responsibility for connections to the second computer.

Failover

A script that automatically manages node responsibilities has to:

- initialize the cluster;
- check the node for errors,
- in case of error, remove the node from the cluster while delegating responsibility to the other nodes; and
- recheck the node and move it back into the cluster if applicable.

The Bash script in Listing 1 covers these tasks, but the script is only a demonstration. The configuration section (lines 3–11) groups the settings for the nodes and the cluster. Following this, line 31 loads the *ipt_conntrack* kernel module.

The interface is then assigned the cluster address, and the *INPUT* chain is deleted for security reasons (line 33) before the call to *iptables* in the next line sets up the cluster. The infinite loop in lines 40 through 56 checks to see that the cluster interface is in *up* mode (with the *check_node* function in line 13). If not, *failover* (line 18) deletes the interface's IP address, takes responsibility away from the local node, and uses SSH to assign it to the other nodes. To allow this to happen, the nodes need the ability to use SSH to access each other without user interaction.

If the interface recovers from the error and comes back online, *recover()* (in line 24) restores the original configuration. The script does not take all possible scenarios into account. For example, the management interface might report a problem that affects communications between the nodes (split brain). This case has no effect as long as no other problems occur, because each of the nodes would go on working autonomously; however, things will go wrong on failover. Half of the communications would remain unanswered. A backup management interface (via a serial port or the LAN interface) would help to handle the situation.

Nodes in a production HA setup would need to monitor each other's health state, or one node might die without being able to warn the others (e.g., power supply failure). This would lead to another split brain situation, in which the nodes could not talk but still had network access. From the High-Availability Linux project (Linux-HA) [5], a fencing mechanism can handle this.

And Much More ...

The author is considering rewriting the cluster script in C to remove the need for UDP-socket-based communications between the nodes, to support more than two nodes, and to standardize the configuration. At the same time, tests could take parameters such as CPU load, disk space, the ability to reach external systems, or the application status into consideration. It also seems to make sense to let Linux-HA manage the cluster configuration as a resource and thus combine the benefits of HA and load sharing.

CLUSTERIP technology is also suitable for farms of high-available application servers, which could be implemented without the use of central load sharing. CLUSTERIP load sharing just goes to prove that a simple but cleverly used hash mechanism can work wonders. The Linux-HA project also provides the framework within which this concept could be added as a resource. Supervision of the health of the nodes, like *ping external systems*, hardware health, or fencing in case of problems are done by the heartbeat software, as well as the failover transferring a resource from a failed node to an active one.

My next article will describe utilizing the CLUSTERIP target of iptables in a resource agent of Linux-HA, thus creating a load-sharing cluster with up to 16 nodes, using plain Linux software. ■

INFO

[1] Linux Virtual Server: *http://www.linuxvirtualserver.org*

[2] Stonesoft: *http://www.stonesoft.com*

[3] RFC 1812, "Requirements for IP Version 4 Routers": *http://rfc.net/rfc1812.html*

[4] Bob Jenkins' "A Hash Function for Hash Table Lookup": *http://www.burtleburtle.net/bob/hash/doobs.html*

[5] High-Availability Linux project: *http://www.linux-ha.org*

Listing 1: Cluster Script

```
01 #!/bin/bash
02
03 # Node configuration
04 MYNODE=1
05 DEVICE=eth0
06 OTHERNODE=192.168.20.2
07
08 # Cluster configuration
09 CLUSTERIP=192.168.10.3
10 CLUSTERMAC=01:02:03:04:05:06
11 ONLINE=0
12
13 check_node () {
14   ip link list dev $DEVICE | grep -q UP
15   return $?
16 }
17
18 failover () {
19   ip address delete $CLUSTERIP/24 dev $DEVICE
20   echo "-$MYNODE" > /proc/net/ipt_CLUSTERIP/$CLUSTERIP
21   ssh $OTHERNODE "echo '+$MYNODE' > /proc/net/ipt_CLUSTERIP/$CLUSTERIP"
22 }
23
24 recover () {
25   ssh $OTHERNODE "echo '-$MYNODE' > /proc/net/ipt_CLUSTERIP/$CLUSTERIP"
26   ip address add $CLUSTERIP/24 dev $DEVICE
27   echo "+$MYNODE" > /proc/net/ipt_CLUSTERIP/$CLUSTERIP
28 }
29
30 # Initialize node
31 modprobe ipt_conntrack
32 ip address add $CLUSTERIP/24 dev $DEVICE
33 iptables -F INPUT
34 iptables -I INPUT -d $CLUSTERIP -i $DEVICE \
35   -p icmp --icmp-type echo-request -j CLUSTERIP --new \
36   --hashmode sourceip --clustermac $CLUSTERMAC \
37   --total-nodes 2 --local-node $MYNODE
38
39 # Test if cluster interface is working
40 ONLINE=1
41 while (true); do
42   if ( check_node ) then
43     echo "Interface up"
44     if [ $ONLINE -eq 0 ]; then
45       recover
46       ONLINE=1
47     fi
48   else
49     echo "Interface down"
50     if [ $ONLINE -eq 1 ]; then
51       failover
52       ONLINE=0
53     fi
54   fi
55   sleep 1
56 done
57
58 exit 0
```