Tuesday, March 19. 2013

Less known Solaris Features: Highly available loadbalancing.

Preview. Will change in the next evenings for typo correction, inclusion of pictures and so on. It's a draft, i just put it here this early, because a customer asked me for such a tutorial and i didn't wanted to reformat it for readability as a Word or LaTeX document . The commandlines are final, the language in between will be proof-read soon. That said, all suggestions are really welcome

As you may know, Solaris contains an integrated load balancer. It's really easy to configure. Not that well known is the point, that you can make it higly-available in an easy way as well. The following tutorial will give you an overview on the configuration of this feature.
VRRPIn this example, we won't use a cluster software to make the load balancer highly available. It's much easier. We use the VRRP feature included in Solaris 11. VRRP stands for Virtual Router Redundancy protocol. If you want to learn more about it, just look into the Wikipedia.
The idea behind using VRRP in this case is quite simple: Don't make the loadbalancer logic itself highly available, do this with the IP-addresses it is using.

So, what do you have to make highly available: At first obviously the virtual IP (short: VIP). The VIP is the one outside users are using to access you service. It's obvious that you to protect it. But at least when you are not using "Direct Server Return"-modes (on the return path the loadbalancer is totally circumvented) you have to protect the availability of the default router as well. I will explain the exact reason later, but you have to ensure that packets are going through the same server on their way back to the client. The easiest way is to make it the default router of your servers, however you have to protect the availability of this default gateway.

Network overviewSo i want you to give a complete walkthrough to the configuration. At first we have to do some IP numbering stuff. We need two different subnets. The reason will be obvious later on. Given that we need two subnets, the IP numbering scheme of our testbed is as following:

Loadbalancer 1
outside interface
10.0.1.10/24

Loadbalancer 2outside interface10.0.1.20/24
Loadbalancer cluster virtual IP 10.0.1.100/24
Loadbalancer 1 inside interface 10.0.10.10/24
Loadbalancer 2 inside interface 10.0.10.20/24
Loadbalancer cluster virtual default gateway 10.0.10.100/24
Real Server 1 single interface 10.0.10.200
Real Server 2 single interface 10.0.10.210

Or to visualize it in a picture:

Setting up the virtual machines
Now we need some hardware for the task. In my case i will use virtual hardware based on Virtualbox. Please configure Virtualbox described in the following steps. I assume that you have a already installed Solaris 11 in a VM.

At first i will install some of the nescessary packages to make life easier . I don't need all of them in all virtual machines, i'm just lazy. To be honest, i need both of them on two nodes and none of them on the other nodes.

```
# pkg install vrrp
           Packages to install:  1
       Create boot environment: No
```

```
Create backup boot environment: No
        Services to change:  1

DOWNLOAD                          PKGS         FILES    XFER (MB)   SPEED
Completed                       1/1        17/17      0.2/0.2 34.5k/s

PHASE                                ITEMS
Installing new actions                 46/46
Updating package state database              Done
Updating image state                   Done
Creating fast lookup database              Done
```

As we want to configure a loadbalancer, we obviously have to install the ilb package:

```
# pkg install ilb
```

Okay, shutdown the vm and go into the shell of you host system. We now clone the four VMs for our testbed. At first we use the command line tool VBoxManage to show all virtual machines.

```
$ VBoxManage list vms
"Solaris 11.1" {e9be8152-8eb0-419e-b7d5-a4879f5db835}
```

I will use the "Solaris 11.1" vm. Make a snapshot of the current state of this vm.

```
$ VBoxManage snapshot  e9be8152-8eb0-419e-b7d5-a4879f5db835 take "ilbdemo_basesnapshot"
```

This snapshot will be the foundation of all further VMs we are using in this tutorial. Okay, i need four vms. So i will just clone them. I will use linked clones in order to save some storage capacity.

```
$ VBoxManage clonevm e9be8152-8eb0-419e-b7d5-a4879f5db835 --snapshot "ilbdemo_basesnapshot"  --options link
--name "ilbdemo_ilb1" --register
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "ilbdemo_ilb1"
```

I repeat this steps for the next three VMS.

```
$ VBoxManage clonevm e9be8152-8eb0-419e-b7d5-a4879f5db835 --snapshot "ilbdemo_basesnapshot"  --options link
--name "ilbdemo_ilb2" --register
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "ilbdemo_ilb2"
$ VBoxManage clonevm e9be8152-8eb0-419e-b7d5-a4879f5db835 --snapshot "ilbdemo_basesnapshot"  --options link
--name "ilbdemo_webserver1" --register
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "ilbdemo_webserver1"
$ VBoxManage clonevm e9be8152-8eb0-419e-b7d5-a4879f5db835 --snapshot "ilbdemo_basesnapshot"  --options link
--name "ilbdemo_webserver2" --register
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Machine has been successfully cloned as "ilbdemo_webserver2"
```

Now i have four virtual machines.

```
 ilbdemo_ilb1
 ilbdemo_ilb2
 ilbdemo_webserver1
 ilbdemo_webserver2
```

I will use my desktop as the client to check the ILB functionality. So i need an interface that is called "host-only interface" in Virtualbox speak. By doing so, i can connect from my local system to the VMs with a browser or a telnet client to simulate a browser.

```
# VBoxManage hostonlyif create ipconfig vboxnet2 --ip 10.0.1.2 --netmask 255.255.255.0
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Interface 'vboxnet2' was successfully created
```

Now let us confgure the VM. As GUIs are always somewhat difficult to explain without screenshots, i will use the command line.

At first put the NIC1 into host-only mode.

```
$ VBoxManage modifyvm "ilbdemo_ilb1" --nic1 hostonly Now put the NIC2 into the "internal network" mode.
$ VBoxManage modifyvm "ilbdemo_ilb1" --nic2 intnetNow i will configure the NIC-type as 82540EM for NIC1 and NIC2.
```

It should be the default. I'm just doing this to be sure.

$ VBoxManage modifyvm "ilbdemo_ilb1" --nictype1 82540EM
$ VBoxManage modifyvm "ilbdemo_ilb1" --nictype2 82540EM
Okay, we configured NIC1 as a host-only interface, now we have to assign it to a host-only network. In this case i will use the vboxnet2 network that i have just configured at the beginning:
$ VBoxManage modifyvm "ilbdemo_ilb1" --hostonlyadapter1 vboxnet2
NIC2 will be configured into the internal network ilbintnet0.

$ VBoxManage modifyvm "ilbdemo_ilb1" --intnet2 ilbintnet0
The next two steps represent a trick you have to use in oder to get this working at all in a VirtualBox environment. Without this two statements it will not work at all. It allows the guest OS to put the virtual NICs provided by Virtualbox to be set into promiscuous mode.
$ VBoxManage modifyvm "ilbdemo_ilb1" --nicpromisc1 allow-all
$ VBoxManage modifyvm "ilbdemo_ilb1" --nicpromisc2 allow-all

Repeat the steps for the second virtual machine used for the second loadbalancer:
$ VBoxManage modifyvm "ilbdemo_ilb2" --nic1 hostonly
$ VBoxManage modifyvm "ilbdemo_ilb2" --nic2 intnet
$ VBoxManage modifyvm "ilbdemo_ilb2" --nictype1 82540EM
$ VBoxManage modifyvm "ilbdemo_ilb2" --nictype2 82540EM
$ VBoxManage modifyvm "ilbdemo_ilb2" --hostonlyadapter1 vboxnet2
$ VBoxManage modifyvm "ilbdemo_ilb2" --intnet2 ilbintnet0
$ VBoxManage modifyvm "ilbdemo_ilb2" --nicpromisc1 allow-all
$ VBoxManage modifyvm "ilbdemo_ilb2" --nicpromisc2 allow-all
The VM configuration for the web servers is much simpler. Just put NIC1 into the "internal network" mode and put it into the internal network named "ilbintnet0". That's all.

$ VBoxManage modifyvm "ilbdemo_webserver1" --nic1 intnet
$ VBoxManage modifyvm "ilbdemo_webserver1" --intnet1 ilbintnet0
$ VBoxManage modifyvm "ilbdemo_webserver2" --nic1 intnet
$ VBoxManage modifyvm "ilbdemo_webserver2" --intnet1 ilbintnet0

Configuring the IP stacks
Let us start with the first webserver. At first we have to start the the virtual machine:
$ VBoxManage startvm "ilbdemo_webserver1"
Waiting for VM "ilbdemo_webserver1" to power on...
VM "ilbdemo_webserver1" has been successfully started.
Log into the system, get a root shell and check at the system at first, that you are able to see the network interface.
root@solaris:/# dladm show-link
LINK CLASS MTU STATE OVER
net0 phys 1500 unknown --

Now we change the hostname.
root@solaris:/# svccfg -s svc:/system/identity:node setprop config/nodename = "ilbws1"
root@solaris:/# svcadm refresh svc:/system/identity:node
root@solaris:/# svcadm restart svc:/system/identity:node
To check this part of the configaration leave the shell, and get a new root shell. The new name should appear. No we can start the configuration. At first switch off the automatic network configuration.
root@ilbws1:/# netadm enable -p ncp defaultfixed

No create an ip interface on the network interface net0. Afterwards assign an IP address to it, put name and ip into /etc/hosts and configure the default route. Finally enable the http server on the system as we want to loadbalance a webserver a http daemon is quite useful.

root@ilbws1:/# ipadm create-ip net0
root@ilbws1:/# ipadm create-addr -T static -a 10.0.10.200/24 net0/v4
root@ilbws1:/# echo "10.0.10.200 ilbws1" >> /etc/hosts
root@ilbws1:/# route -p add default 10.0.10.100
add net default: gateway 10.0.10.100

add persistent net default: gateway 10.0.10.100
root@ilbws1:/# svcadm enable http
root@ilbws1:/# echo "ilbdemo webserver1" > /var/apache2/2.2/htdocs/index.html

The last one is simply to check which server we are using. Of course in production you would have the same content on both servers and a more subtle scheme to tell the servers from each other. But for our purposes this echo is sufficient.
Now start the second webserver.
$ VBoxManage startvm "ilbdemo_webserver2"
Waiting for VM "ilbdemo_webserver2" to power on...
VM "ilbdemo_webserver2" has been successfully started.

Repeat all the steps we have done on the first webserver, of course with changed names and ip addresses on the second webserver.
root@solaris:/# svccfg -s svc:/system/identity:node setprop config/nodename = "ilbws2"
root@solaris:/# svcadm refresh svc:/system/identity:node
root@solaris:/# svcadm restart svc:/system/identity:nodeLeave the shell and get a new root shell

root@ilbws2:/# netadm enable -p ncp defaultfixed
root@ilbws2:/# ipadm create-ip net0
root@ilbws2:/# ipadm create-addr -T static -a 10.0.10.210/24 net0/v4
root@ilbws2:/# route -p add default 10.0.10.100
add net default: gateway 10.0.10.100
add persistent net default: gateway 10.0.10.100
root@ilbws2:/# echo "10.0.10.210 ilbws2" >> /etc/hosts
root@ilbws2:/# svcadm enable http
root@ilbws1:/# echo "ilbdemo webserver1" > /var/apache2/2.2/htdocs/index.html

Okay, our webservers are ready to go. Now we have to configure the ILB servers. So we start up the first one:
$ VBoxManage startvm "ilbdemo_ilb1"
Waiting for VM "ilbdemo_ilb1" to power on...
VM "ilbdemo_ilb1" has been successfully started.
Okay, get a root shell, set a new hostname. You know the drill.
root@solaris:# svccfg -s svc:/system/identity:node setprop config/nodename = "ilb1"
root@solaris:# svcadm refresh svc:/system/identity:node
root@solaris:# svcadm restart svc:/system/identity:node
Logout, login. Get a root shell. Now we have a difference. As we configured the VM with two network interfaces, we see them in the next step as well.
root@ilb1:# dladm show-link
LINK   CLASS   MTU   STATE   OVER
net0   phys   1500   unknown --
net1   phys   1500   unknown --
Deactivate the automatic network configuration, create two ip interfaces and assign ip addresses to them.
root@ilb1:/# netadm enable -p ncp defaultfixed
root@ilb1:/# ipadm create-ip net0
root@ilb1:/# ipadm create-ip net1
root@ilb1:/# ipadm create-addr -T static -a 10.0.1.10/24 net0/v4
root@ilb1:/# ipadm create-addr -T static -a 10.0.10.10/24 net1/v4
Okay, we repeat the same steps for the second ILB VM as well.
$ VBoxManage startvm "ilbdemo_ilb2"
Waiting for VM "ilbdemo_ilb2" to power on...
VM "ilbdemo_ilb2" has been successfully started.
Get a root shell, and rename the system.
jmoekamp@solaris:~$ svccfg -s svc:/system/identity:node setprop config/nodename = "ilb2"
jmoekamp@solaris:~$ svcadm refresh svc:/system/identity:node
jmoekamp@solaris:~$ svcadm restart svc:/system/identity:node
Check if both interfaces are visible to you.
root@ilb2:/# dladm show-link
LINK   CLASS   MTU   STATE   OVER
net0   phys   1500   unknown --
net1   phys   1500   unknown --

And now do the basic IP configuration.
root@ilb2:/# netadm enable -p ncp defaultfixed
root@ilb2:/# ipadm create-ip net0
root@ilb2:/# ipadm create-ip net1
root@ilb2:/# ipadm create-addr -T static -a 10.0.1.20/24 net0/v4
root@ilb2:/# ipadm create-addr -T static -a 10.0.10.20/24 net1/v4
Okay, the basics are done, now we can start the configure the load balancer.
Configuring the highly available loadbalancer

So, now all the basics have been configured. The configuration steps contain an important prerequisite for the configuration that is somewhat missing in the manual. You have to configure an ip address on the interfaces that are used provide a vrrp group. That is not expressed in a clear manner in the documentation. Without this IP-Addresses the interfaces cant send their advertisments, the VRRP stuff wouldn't work.

In addition to that, when you try this configuration with Virtualbox, you have to use a trick to get this working. VRRP depends on a Solaris VNIC. However you can't use a Solaris VNICs on top of a NIC provided by Virtualbox. However you can get it to work with a trick. You have to run snoop on the devices.

At first we have to do this trick on the first ILB node.
root@ilb1:/# nohup snoop -d net0 &
root@ilb1:/# nohup snoop -d net1 &
Then we repeat it on the second one.
root@ilb2:/# nohup snoop -d net0 &
root@ilb2:/# nohup snoop -d net1 &
Okay, now let's configure the load balancing itself. At first activate the ipv4 forwarding on the ILB note. As loadbalancing is a strange kind of packet forwarding, at the end it is just a special kind of NAT. So you have to activate it in order to make the ILB work.
root@ilb1:/# routeadm -u -e ipv4-forwarding
The next step is to enable the ILB service.
root@ilb1:/# svcadm enable ilb
For the next step you have to understand the administrative model of the ILB. You can visualise it like this:

At first you have to create a servergroup. This group is an group of server providing the same service. For example a group of webservers that all provide the same websites on port 80. This is important, because the loadbalancer balances requests to all servers in a servergroup. Let's assume you have two servers you want to load balance c0t0d0s0.org, both servers have to provide the service c0t0d0s0.org. It doesn't help you if one server provides only c0t0d0s0.org and the other for example the website of the dragonboating club. Then half of the requests would work, the other half would just deliver something else depending of your webserver configuration. So i'm creating a server group now containing the both webserver VMs we create before.
root@ilb1:/# ilbadm create-servergroup -s server=10.0.10.200,10.0.10.210 servergroup1
Now we configure the loadbalancing.

root@ilb1:/# ilbadm create-rule -ep -i vip=10.0.1.100,port=80,protocol=tcp -m lbalg=roundrobin,type=HALF-NAT,pmask=32 -o servergroup=servergroup1 rule1
Let me translate this: Everything that hits the loadbalancer on with the destination ip address 10.0.1.10 and destination port 80 as a tcp packet will be load balanced to the servers of servergroup servergroup1. The load balancing decision is based on round-robin. We have defined a persistency mask of 32. Persistency is important for many services. For example when you have a website with dynamic content that uses an on-server session context. When you pass the first request to server1, the session context is created by you website, however when you balance the second request on the other webserver, there is obviously no session context on this webserver. You application will fail, depends on your application. Persistency mask 32 means, that the persistency decision is made on the full ip-address (all 32 bits), for example when you set 24 here, all servers from the same Class-C sized fragment of the internet will hit the same server.

The load balancing algorithm is half-NAT. What is half-NAT? When the packet hits the loadbalancer on the virtual IP address, only the destination is changed to one of our servers in the server group. The source adress is still the same. On it's way back the the address of the server is changed back to the virtual IP address.

Doing so has the big advantage that you log files on the webserver and your access control lists based on IP make still sense, because when you are full-NAT both source (the loadbalancer inserts itself as the source) and destination are changed and so all requests would appear in the logfiles as sourced by the loadbalancer. However this puts and

important contraint on your network topology.

The webserver just puts it on its route to the client, most often the default gateway. You have to ensure that the packets back to your client has to pass through the same loadbalancer because just this one has the information to correctly reinsert the virtual IP of your service into the packet. Out of this reason it isn't possible to provide a load balanced service to a server in the same subnet as the server group. When you have a requirement that needs such a configuration, you have to put it into a different subnet. However people working with load loadbalancers already know this kind of constraint.

Okay, we have to repeat the configuration on the next ILB as well:
root@ilb2:/# routeadm -u -e ipv4-forwarding
root@ilb2:/# svcadm enable ilb
root@ilb2:/# ilbadm create-servergroup -s server=10.0.10.200,10.0.10.210 servergroup1
root@ilb2:# ilbadm create-rule -ep -i vip=10.0.1.100,port=80,protocol=tcp -m
lbalg=roundrobin,type=HALF-NAT,pmask=32 -o servergroup=servergroup1 rule1

Making it highly available
As i wrote in the beginning, the trick of making the loadbalancer highly availble is to make the VIP highly available. In oder to do so, you can use VRRP. You have to configure it on both nodes, however you have to enable it first.
root@ilb1:/# svcadm enable vrrp
Now you can configure it. At first you have to create a special VNIC.
root@ilb1:/# dladm create-vnic -m vrrp -V 1 -A inet -l net0 vnic1
This vnic is home to the virtual MAC address. The virtual MAC address is derived from the the VRID (Virtual Router ID). For an IPv4 vrrp interface it's 00-00-5E-00-01-(VRID), so In this case it's 00-00-5E-00-01-01. Important: This is just the interface, the HA logic is missing. Configuring VRRP needs two steps. But before doing so, we create an IP interface on the VRRP vnic and assign the VIP on it.
root@ilb1:/# ipadm create-ip vnic1
root@ilb1:/# ipadm create-addr -T static -d -a 10.0.1.100/24 vnic1/lb1
Okay, now we create the HA stuff:
root@ilb1:/# vrrpadm create-router -V 1 -A inet -l net0 -p 255 vrrp1
With this command we configured a virtual router with the ID 1 and the priority 255. Priority? So simplify it a little bit: The higher the priority, the higher the probability that it is selected as the router owning the IP addresses. Per default the router with the higher priority wants its address ownership back as soon as it's back online.
Okay, repeat the same on the next next interface.
root@ilb1:/# dladm create-vnic -m vrrp -V 2 -A inet -l net1 vnic2
root@ilb1:/# ipadm create-ip vnic2
root@ilb1:/# ipadm create-addr -T static -d -a 10.0.10.100/24 vnic2/lb1
root@ilb1:/# vrrpadm create-router -V2 -A inet -l net1 -p 255 vrrp2
And now you have to do the same configuration on the second ILB VM. It looks almost the same but please note the differences. The name of the IP interface is slightly different and more important the priority is lower. So the virtual routers you will configure next are the backups for the ones on the first note.
root@ilb2:/# svcadm enable vrrp
root@ilb2:/# dladm create-vnic -m vrrp -V 1 -A inet -l net0 vnic1
root@ilb2:/# ipadm create-ip vnic1
root@ilb2:/# ipadm create-addr -T static -d -a 10.0.1.100/24 vnic1/lb2
root@ilb2:/# vrrpadm create-router -V 1 -A inet -l net0 -p 100 vrrp1
root@ilb2:/# dladm create-vnic -m vrrp -V 2 -A inet -l net1 vnic2
root@ilb2:/# ipadm create-ip vnic2
root@ilb2:/# ipadm create-addr -T static -d -a 10.0.10.100/24 vnic2/lb2
root@ilb2:/# vrrpadm create-router -V 2 -A inet -l net1 -p 100 vrrp2
Okay, let's check the configuration. On the first node we will notice that both virtual routers are in MASTER node.
root@ilb1:/# vrrpadm show-router
NAME VRID LINK AF PRIO ADV_INTV MODE STATE VNIC
vrrp2 2 net1 IPv4 255 1000 eopa- MASTER vnic2
vrrp1 1 net0 IPv4 255 1000 eopa- MASTER vnic1
When we look onto the second VM, both router are in backup mode.
root@ilb2:# vrrpadm show-router
NAME VRID LINK AF PRIO ADV_INTV MODE STATE VNIC
vrrp2 2 net1 IPv4 100 1000 e-pa- BACKUP vnic2
vrrp1 1 net0 IPv4 100 1000 e-pa- BACKUP vnic1
Now let us unplug a cable on the first node. Et voila. The virtual router on the second node takes over.

root@ilb2:/# vrrpadm show-router
NAME VRID LINK AF PRIO ADV_INTV MODE STATE VNIC
vrrp2 2 net1 IPv4 100 1000 e-pa- BACKUP vnic2
vrrp1 1 net0 IPv4 100 1000 e-pa- MASTER vnic1
Plug it back now.

Testcases
So ... we configured a lot of stuff. Let us try the loadbalancing. Okay, at first let us stop a webserver. A first a check that load balancing is working at all.
$ telnet 10.0.1.100 80
Trying 10.0.1.100...
Connected to 10.0.1.100.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 19 Mar 2013 09:11:36 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/1.0.0j DAV/2
Last-Modified: Tue, 19 Mar 2013 09:05:38 GMT
ETag: "45dd-13-4d84368b7aa18"
Accept-Ranges: bytes
Content-Length: 19
Connection: close
Content-Type: text/html

ilbdemo webserver1
Connection closed by foreign host.
Nice, we see the string ilbdemo webserver1. Now i will kill this webserver. Checking for the UUID.
$ VBoxManage list vms | grep "ilbdemo_webserver1"
"ilbdemo_webserver1" {3dba1781-932e-4d82-be53-59b39758aedc}
And now i'm just removing the power.
$ VBoxManage controlvm 3dba1781-932e-4d82-be53-59b39758aedc poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Next check.
$ telnet 10.0.1.100 80
Trying 10.0.1.100...
Connected to 10.0.1.100.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 19 Mar 2013 09:14:45 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/1.0.0j DAV/2
Last-Modified: Tue, 19 Mar 2013 09:06:22 GMT
ETag: "45dd-13-4d8436b5adee1"
Accept-Ranges: bytes
Content-Length: 19
Connection: close
Content-Type: text/html

ilbdemo webserver2
Connection closed by foreign host.

Webserver is still reacting. But this time the answer comes from the second server. Works as we planed. We will just restart this webserver now.
$ VBoxManage startvm 3dba1781-932e-4d82-be53-59b39758aedc
Waiting for VM "3dba1781-932e-4d82-be53-59b39758aedc" to power on...

VM "3dba1781-932e-4d82-be53-59b39758aedc" has been successfully started.
Next testcase, stopping an ILB VM. We know by our configuration that ilb2 is the master. So we stop this VM.
$ VBoxManage list vms | grep "ilbdemo_ilb1"

"ilbdemo_ilb1" {89e44d1f-7fdf-44c2-b22a-12e30d787403}
$ VBoxManage controlvm 89e44d1f-7fdf-44c2-b22a-12e30d787403 poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%

Okay, one load balancer just went way, let us check the availability of our service.
$ telnet 10.0.1.100 80
Trying 10.0.1.100...
Connected to 10.0.1.100.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 19 Mar 2013 09:48:33 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/1.0.0j DAV/2
Last-Modified: Tue, 19 Mar 2013 09:06:22 GMT
ETag: "45dd-13-4d8436b5adee1"
Accept-Ranges: bytes
Content-Length: 19
Connection: close
Content-Type: text/html

ilbdemo webserver2
Connection closed by foreign host.

Now a short check on the second node.
root@ilb2:/# vrrpadm show-router
NAME VRID LINK AF PRIO ADV_INTV MODE STATE VNIC
vrrp2 2 net1 IPv4 100 1000 e-pa- MASTER vnic2
vrrp1 1 net0 IPv4 100 1000 e-pa- MASTER vnic1
Great both interfaces are now in MASTER state
Constraints
To be honest, there are some constraints on the ILB functionality and the way HA mechanism is working, there are some constraint. At first there is no synchronisation of state tables between the both nodes. There are two totally different loadbalancers just doing the same. To load balancing decisions may be different on both nodes. Out of this reason we should do everything to keep the loadbalancing on the same server we are using for loadbalancing.

The second constraint is, that both virtual routers are independent. So to have a load balancer failover the problem must impact both interfaces of the ILB. However when you just unplug one cable or the switch goes down, one virtual router is in a perfect state, and has no reason to switch, the other router has to failover. The output that i've used earlier is a problem when you want to protect against a link failure.
root@ilb2:/# vrrpadm show-router
NAME VRID LINK AF PRIO ADV_INTV MODE STATE VNIC
vrrp2 2 net1 IPv4 100 1000 e-pa- BACKUP vnic2
vrrp1 1 net0 IPv4 100 1000 e-pa- MASTER vnic1
The VIP is on the second loadbalancer, the default gate on the first loadbalancer. Remember the stuff i have told you about "same way back". Isn't the case now, so the load balancer will stop working.

That said, you have to find a solution for this and since 11.1 there is a simple one. You can't work around it with IPMP because VRRP doesn't work on an IPMP interface, a LACP ethernet trunk would work, however without propritary extensions to LACP this kind of link aggregation will just work with a single switch. This way you would be still toast when the switch fails on the other side. You just moved the single point of failure.

However since Solaris 11.1 there is a second trunking model that can do multipathing on the datalink layer and can work with member interfaces that terminates on different switches. It's called data link multi pathing or short DLMP.

Let us configure one of our loadbalancers to use DLMP. At first shut down the vm with "ilbdemo_ilb1". When you have followed the tutorial so far, it should already down. Now add additional networking interfaces:

$ VBoxManage modifyvm "ilbdemo_ilb1" --nic3 hostonly
$ VBoxManage modifyvm "ilbdemo_ilb1" --nic4 intnet
$ VBoxManage modifyvm "ilbdemo_ilb1" --nictype3 82540EM

```
$ VBoxManage modifyvm "ilbdemo_ilb1" --nictype4 82540EM
$ VBoxManage modifyvm "ilbdemo_ilb1" --hostonlyadapter3 vboxnet2
$ VBoxManage modifyvm "ilbdemo_ilb1" --intnet4 ilbintnet0
$ VBoxManage modifyvm "ilbdemo_ilb1" --nicpromisc3 allow-all
$ VBoxManage modifyvm "ilbdemo_ilb1" --nicpromisc4 allow-all
```

The configuration is a copy of the steps we made to get the basic ilb vm ready, just that everything we did for net0 is repeated net3 and everything with net1 is repeated for net4. Now start the system:

```
$ VBoxManage startvm 89e44d1f-7fdf-44c2-b22a-12e30d787403
Waiting for VM "89e44d1f-7fdf-44c2-b22a-12e30d787403" to power on...
VM "89e44d1f-7fdf-44c2-b22a-12e30d787403" has been successfully started.
```

As soon as the system is running, you can get a shell on it and check if everything went well:

```
root@ilb1:/# dladm show-link
LINK              CLASS     MTU    STATE    OVER
net0              phys      1500   up       --
net1              phys      1500   up       --
vnic1             vnic      1500   up       net0
vnic2             vnic      1500   up       net1
net2              phys      1500   unknown  --
net3              phys      1500   unknown  --
```

Okay, now we will delete the old configuration. We will do the configuration form scratch.

```
root@ilb1:/# vrrpadm delete-router vrrp1
root@ilb1:/# vrrpadm delete-router vrrp2
root@ilb1:/# ipadm delete-ip vnic1
root@ilb1:/# ipadm delete-ip vnic2
root@ilb1:/# ipadm delete-ip net0
root@ilb1:/# ipadm delete-ip net1
root@ilb1:/# dladm delete-vnic vnic1
root@ilb1:/# dladm delete-vnic vnic2
```

Now start the snoops

```
root@ilb1:/# nohup snoop -d net0 &
root@ilb1:/# nohup snoop -d net1 &
root@ilb1:/# nohup snoop -d net3 &
root@ilb1:/# nohup snoop -d net4 &
```

At first we create the DLMP aggregations. You may know the syntax from normal link aggregation, however we have to add the option -m dlmp to tell dladm that we want a DLMP aggregation.

```
root@ilb1:/# dladm create-aggr -m dlmp -l net0 -l net2 outside0
root@ilb1:/# dladm create-aggr -m dlmp -l net1 -l net3 inside1
```

You should see those aggregation when checking for the links available in the system.

```
root@ilb1:/# dladm show-link
LINK              CLASS     MTU    STATE    OVER
net0              phys      1500   up       --
net1              phys      1500   up       --
net2              phys      1500   up       --
net3              phys      1500   up       --
outside0          aggr      1500   up       net0 net2
inside1           aggr      1500   up       net1 net3
```

Now create the IP interfaces on both aggregations. It's a simple conversion: The address of net0 will now reside on outside0 and the address of net1 on inside1:

```
root@ilb1:/# ipadm create-ip outside0
root@ilb1:/# ipadm create-ip inside1
root@ilb1:/# ipadm create-addr -T static -a 10.0.1.10/24 outside0/v4
root@ilb1:/# ipadm create-addr -T static -a 10.0.10.10/24 inside1/v4
```

Now we have to configure the VRRP stuff. We use the same translation rule again to get to this configuration net0 is now outside0 and net1 is now net0

```
root@ilb1:/# dladm create-vnic -m vrrp -V 2 -A inet -l inside1 vnic2
root@ilb1:/# dladm create-vnic -m vrrp -V 1 -A inet -l outside0 vnic1
```

```
root@ilb1:/# ipadm create-ip vnic1
root@ilb1:/# ipadm create-addr -T static -d -a 10.0.1.100/24 vnic1/lb1
root@ilb1:/# vrrpadm create-router -V 1 -A inet -l outside0 -p 255 vrrp1
root@ilb1:/# ipadm create-ip vnic2
root@ilb1:/# ipadm create-addr -T static -d -a 10.0.10.100/24 vnic2/lb1
root@ilb1:/# vrrpadm create-router -V2 -A inet -l inside1 -p 255 vrrp2
```

Okay, let us check the aggregations first:

```
root@ilb1:/# dladm show-aggr -x
LINK        PORT          SPEED DUPLEX   STATE     ADDRESS           PORTSTATE
outside0    --            1000Mb full    up        8:0:27:29:a2:89   --
            net0          1000Mb full    up        8:0:27:29:a2:89   attached
            net2          1000Mb full    up        8:0:27:ec:8a:7e   attached
inside1     --            1000Mb full    up        8:0:27:d5:aa:2    --
            net1          1000Mb full    up        8:0:27:d5:aa:2    attached
            net3          1000Mb full    up        8:0:27:f0:17:7d   attached
```

All links are up and running. Now we will "disconnect" one of the "cables".

```
$ VBoxManage controlvm 89e44d1f-7fdf-44c2-b22a-12e30d787403 setlinkstate1 off
```

The outside0 interface is still up, just one member interface is down as shown by the output of dladm.

```
root@ilb1:/# dladm show-aggr -x
LINK        PORT          SPEED DUPLEX   STATE     ADDRESS           PORTSTATE
outside0    --            1000Mb full    up        8:0:27:ec:8a:7e   --
            net0          0Mb  unknown   down      8:0:27:29:a2:89   standby
            net2          1000Mb full    up        8:0:27:ec:8a:7e   attached
inside1     --            1000Mb full    up        8:0:27:d5:aa:2    --
            net1          1000Mb full    up        8:0:27:d5:aa:2    attached
            net3          1000Mb full    up        8:0:27:f0:17:7d   attached
```

Of course we have to check the status of our virtual router now.

```
root@ilb1:/# vrrpadm show-router
NAME    VRID LINK    AF   PRIO ADV_INTV MODE  STATE VNIC
vrrp2   2    inside1 IPv4 255  1000     eopa- MASTER vnic2
vrrp1   1    outside0 IPv4 255 1000     eopa- MASTER vnic1
```

No switch took place. Nice. Let's check the answer of the loadbalancer:

```
$ telnet 10.0.1.100 80
Trying 10.0.1.100...
Connected to 10.0.1.100.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 19 Mar 2013 11:10:19 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/1.0.0j DAV/2
Last-Modified: Tue, 19 Mar 2013 09:05:38 GMT
ETag: "45dd-13-4d84368b7aa18"
Accept-Ranges: bytes
Content-Length: 19
Connection: close
Content-Type: text/html

ilbdemo webserver1
Connection closed by foreign host.
```

Cool, as expected

Conclusion

Okay, this was a long article how to configure the loadbalancer on Solaris 11.1, however you now have full walk through. I hope this can help you in order to use this feature at your site. The next steps would be the usage of the health-checking, however this is part of another tutorial.

Do you want to learn more?

Oracle Doc - Managing Oracle Solaris 11.1 Network Performance
Administering Link Aggregations
Virtual Router Redundancy Protocol (Overview)
Integrated Load Balancer (Overview)

Posted by Joerg Moellenkamp in 2blogsoraclecom, English, Solaris at 16:55

very useful! thanks a million!
Anonymous on Mar 20 2013, 15:36

Thanks Joerg, very interesting entry.
Anonymous on Mar 20 2013, 20:24

Im wondering what zeus is up to these days. They used to have a pretty nifty loadbalancer running on solaris. Sold by SUN too back in the day.
Anonymous on Mar 21 2013, 07:22

This is good, I don't know Solaris have this tool yet.
Anonymous on Mar 21 2013, 11:23

Zeus was bought by Riverbed some time ago (and was renamed "stingray" in the process).
Anonymous on Apr 1 2013, 08:31