Approaching online zpool switching between two FreeBSD machines

Mikhail E. Zakharov zmey20000@yahoo.com

Version 1.1 2017.04.10

We all know that ZFS pool is designed to be used by a single machine only at any particular point of time, and when we want to access a zpool from another system we have to export it and import to the new server. This disadvantage makes simultaneous direct usage of the pool quite impossible for reliable dual-controllered storage systems.

Earlier with the articles related to the BeaST storage system concept I have posted several solutions on how to bypass this architectural obstacle and make failover to another storage controller. But these hints turned out really dangerous to the pools. Testing has showed that in most cases it is impossible to make fail-back when controller boots again. Even worse, zpool often becomes permanently damaged.

Luckily it forced me to continue studies and finally I have discovered a plain way to make a safer online or, honestly saying, near-online failover/failback operations for ZFS storage pools. And yes, it now looks really simple comparing to what I have seen before. Also if everything goes well, this method with several essential improvements regarding ZIL mirroring will be implemented in the BeaST storage system for ZFS.

Such operations became possible due to CTL HA implementation, which controls access to the LUNs and prevents simultaneous attempts to use zpool at any moment including the time of switching controllers.

Now let's call Ockham to cut off all unnecessary stuff from the configuration and leave only the essence of the idea.

I have built a lab similar to what I normally use for the BeaST storage experiments. It is Oracle VM VirtualBox environment with one virtual machine clnt-1 for a client host and two virtual servers for storage controllers: ctrl-a and ctrl-b respectively. For the sake of simplicity on storage machines I use only three **shareable, fixed-sized**[1] virtual drives: ada1/ada2 connected with SATA virtual controller for client data, and da0 on SAS controller for ZFS Intent Log. FreeBSD 11.0 Release is installed on ada0 – a **normal** from the VirtualBox point of view **dynamically allocated** virtual drive on all of these three machines.

[1] See Oracle VM VirtualBox documentation for the virtual storage: https://www.virtualbox.org/manual/ch05.html

Two networks are configured: 192.168.55.0/24 for client access and 192.168.56.0/24 for cross-controller data path and system communications.

The dual-controller configuration

First of all, let's configure basics of FreeBSD system. The /etc/rc.conf on controllers is elementary:

ctrl-a	ctrl-b

`hostname="ctrl-a"`
 	
`# Cross-controller link`
`ifconfig_em0="inet 192.168.56.103 netmask 0xffffff00"`
`# Client network`
`ifconfig_em1="inet 192.168.55.103 netmask 0xffffff00"`
 	
`# VirtualBox guest additions`
`vboxguest_enable="YES"`
`vboxservice_enable="YES"`
	`hostname="ctrl-b"`
	
	`# Cross-controller link`
	`ifconfig_em0="inet 192.168.56.104 netmask 0xffffff00"`
	`# Client network`
	`ifconfig_em1="inet 192.168.55.104 netmask 0xffffff00"`
	
	`# VirtualBox guest additions`
	`vboxguest_enable="YES"`
	`vboxservice_enable="YES"`

Then edit /boot/loader.conf to configure boot time parameters of CTL HA:

ctrl-a	ctrl-b

`ctl_load="YES"`
 	
`kern.cam.ctl.ha_id=1`
`kern.cam.ctl.ha_mode=2`
`kern.cam.ctl.ha_role=0`
	`ctl_load="YES"`
	
	`kern.cam.ctl.ha_id=2`
	`kern.cam.ctl.ha_mode=2`
	`kern.cam.ctl.ha_role=1`

And reboot both controllers:

```
# reboot
```

You can see, I have set “`kern.cam.ctl.ha_mode=2`” which means that secondary CTL HA node accepts all requests and data to the LUN but then forwards everything to primary node, so it works in ALUA (Asymmetric Logical Unit Acces) mode. Full active-active access “`kern.cam.ctl.ha_mode=1`” may be dangerous when dealing with our dual-controller architecture and ZFS especially.

Now we can create a zpool (beast) with separated ZIL (da0) and a volume (v0) on the ctrl-a machine. The appropriate task on ctrl-b controller is only to import the “beast” pool. Options “-m none” on ctrl-a and “-N” on ctrl-b prevent pool mounting:

ctrl-a	ctrl-b

`zpool create -m none beast mirror /dev/ada1 /dev/ada2`
`zpool add beast log /dev/da0`
`zfs create -V 100M beast/v0`
	`zpool import -N beast`

zfs set sync=always beast	

The “zfs set sync=always beast” option is needed to push all transactions though ZIL device as we do not want to lose client data in the RAM memory of the dead controller in the case of failure.

When talking about production systems, ZIL device must be secure and as fast as possible. Ideally it should be a non-volatile cache memory (connected to- or mirrored between- both controllers) or shared by controllers fast NVMe, SSD drive at least. The BeaST storage system has ZIL mirroring function for that case, but for simplicity of our small Virtual Box lab we created a shared SAS drive for that ZIL device.

Now setup CTL HA interlink. Run:

ctrl-a	ctrl-b
sysctl kern.cam.ctl.ha_peer="listen 192.168.56.103:7777"	sysctl kern.cam.ctl.ha_peer="connect 192.168.56.103:7777"

Check if sysctl kern.cam.ctl.ha_link value equals 2. It means that the link is established:

```
# sysctl kern.cam.ctl.ha_link
kern.cam.ctl.ha_link: 2
```

Prepare simple /etc/ctl.conf configuration for a single LUN definition with our ZFS volume:

ctrl-a	ctrl-b

portal-group pg0 {
        discovery-auth-group no-authentication
        listen 192.168.55.103
}
 	
target iqn.2016-01.local.beast:target0 {
    auth-group no-authentication
    portal-group pg0
 	
    lun 0 {
        path /dev/zvol/beast/v0
    }
}
	portal-group pg0 {
	        discovery-auth-group no-authentication
	        listen 192.168.55.104
	}
	
	target iqn.2016-01.local.beast:target0 {
	        auth-group no-authentication
	        portal-group pg0
	
	        lun 0 {
	                path /dev/zvol/beast/v0
	        }
	}

And finally start ctld daemon:

ctrl-a	ctrl-b
service ctld onestart	service ctld onestart

The client

This part is the least interesting one to create and to read, so I just post obvious configuration files and commands here.

Update /etc/rc.conf on the client:

```
hostname="cln-1"
ifconfig_em0="inet 192.168.55.111 netmask 0xffffff00"   # Public LAN

# VirtualBox guest additions
vboxguest_enable="YES"
vboxservice_enable="YES"

# iSCSI
iscsid_enable="YES"                # Initiators
```

Set this kernel variable so the client is able to lose iSCSI paths. Contents of /etc/sysctl.conf:

```
kern.iscsi.fail_on_disconnection=1
```

Now connect controllers by running:

```
# iscsictl -A -p 192.168.55.103 -t iqn.2016-01.local.beast:target0
# iscsictl -A -p 192.168.55.104 -t iqn.2016-01.local.beast:target0
```

Then create multipathing device to access the same LUN from both controllers:

```
# gmultipath label beast /dev/da0 /dev/da1
```

The command above sets active-passive mode on multipath device. For testing purposes you may want to create it in active-active:

```
# gmultipath label -A beast /dev/da0 /dev/da1
```

From the client’s point of view, all paths are active. But the path for ctrl-b will respond slowly as in fact CTL HA forwards all data to ctrl-a and on production systems, so active-active mode may have negative effect on client’s performance.

Create and mount a filesystem:

```
# newfs /dev/multipath/beast
# mount /dev/multipath/beast /mnt
```

And finally put some load on it:

```
# dd if=/dev/random of=data.rnd bs=1M count=90
# sh -c "while true; do cp data.rnd /mnt; md5 /mnt/data.rnd; done"
```

Now we can consider the system ready for failover/failback tests.

Fail there and back again

In the BeaST storage system we use simple BeaST Quorum (BQ) software to detect controller failure and react accordingly. But today we will do everything by our own hands just to see what is really going on during our experiments.

Let’s go! Switch off the power of ctrl-a controller and immediately run to console of ctrl-b and type:

```
# zpool export beast && zpool import -f -N beast && sysctl kern.cam.ctl.ha_role=0
```

These commands will reattach the “beast” pool to ctrl-b and make this controller primary for CTL HA operations.

The client has to wait for these operations to complete. And in real life it may take quite a long period of time! It mostly depends on the pool size and volumes, workload and other options, so be careful with it and check the timeout settings. But we should not worry in our tiny configuration it will be fast for sure.

Nevertheless, if everything is done properly, client will continue to work with the volume and pool through the ctrl-b controller.

To restore ctrl-a boot it and run:

```
# sysctl kern.cam.ctl.ha_role=1
# zpool status
# sysctl kern.cam.ctl.ha_peer="listen 192.168.56.103:7777"
# service ctld onestart
```

It makes ctrl-a to start operating and to be the secondary member of the CTL HA cluster, so the controller will serve requests by forwarding data to the ctrl-b controller.

The same procedure applies to the ctrl-b controller if we decide to fail it. Actually, we can have fun crushing controllers, one after another, all day long, but it’s better to use the BeaST Quorum or some other more advanced quorum software to automate the recovering process. Details on how-to install and configure the BeaST Quorum are fully described in the Simple quorum drive for the FreeBSD CTL HA and the BeaST storage system article.

Conclusions

Although this experiment looks like quite a rough example of the fail-over/fail-back procedures for ZFS data pools, actually it is a breakthrough for our storage system development.

After several enhancements to support reliability and workload balancing, we will use two storage pools (one pool per controller) in conjunction with the BeaST Quorum to automate zpool switching between both active controllers.

Now cross your fingers for good luck, as I am one step from implementing it in the BeaST storage system.