The BeaST Classic – dual-controller storage system with RAID arrays and CTL HA

Mikhail E. Zakharov zmey20000@yahoo.com

Version 1.1, 2018.06.03

Preface

The BeaST Classic is the FreeBSD based reliable storage system concept. It turns two commodity servers into a pair of redundant active-active/asymmetric storage controllers which use iSCSI protocol (Fibre Channel in the future) to provide clients with simultaneous access to volumes on the storage system. Both controllers have access to all drives on one or several SATA or SAS drive enclosures on the back-end. Depending on particular configuration it allows the BeaST Classic to create wide range of GEOM based software or hardware RAID array types along with ZFS storage pools.

Though the BeaST Classic may be used with different options, the scope of this document is limited to the The BeaST Classic configuration with RAID arrays and CTL HA.

The BeaST Classic does not have any installation script or wizard yet, therefore this document is intended to be the storage system installation and configuration guide.

Description

The minimal configuration of the BeaST Classic storage system was reproduced in the Oracle VM Virtual Box environment with the following specification.

Two storage controllers (ctrl-a and ctrl-b) are equipped with virtual SATA controllers to share four data drives ada1, ada2, ada3, ada4 (as seen by FreeBSD) which were created with the **fixed-sized** and **shareable** options **enabled** (see Oracle VM VirtualBox Virtual Media Manager for the details).

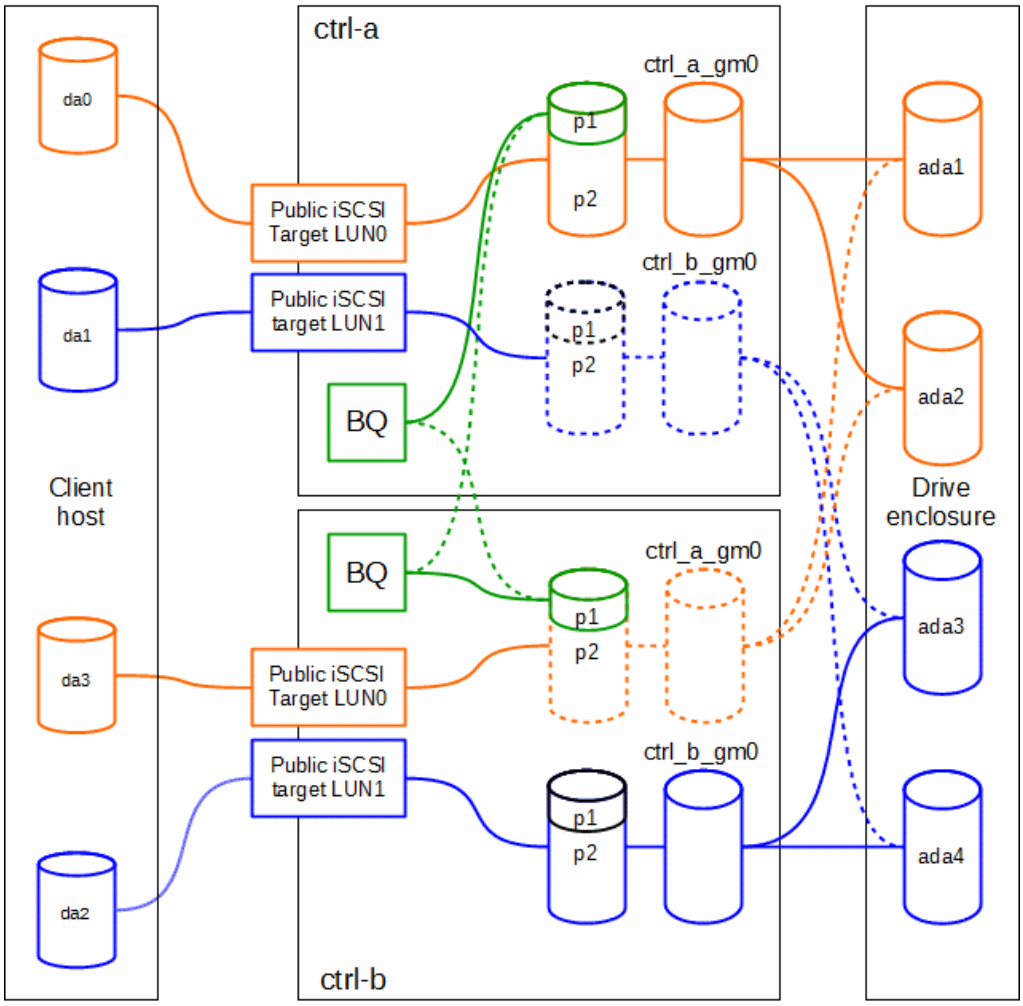
Both controllers are connected with the Cross-controller LAN (host-only network 192.168.10.0/24) to transfer all backend traffic and the Public LAN (host-only network 192.168.12.0/24) to allow the client host (clnt-1) to reach LUNs on the BeaST Classic storage system.

The configuration summary is shown in the table below:

Description	ctrl-a	ctrl-b	clnt-1
Cross-controller LAN	IP: 192.168.10.10 Mask: 255.255.255.0	IP: 192.168.10.11 Mask: 255.255.255.0	-
Public LAN	IP: 192.168.12.10 Mask: 255.255.255.0	IP: 192.168.12.11 Mask: 255.255.255.0	IP: 192.168.12.111 Mask: 255.255.255.0

Management LAN	DHCP	DHCP	DHCP
Shareable, fixed-sized virtual drives on the SATA controller - ada1, ada2, ada3, ada4	Each of four drives is 256MB size		-
System virtual drive (Dynamic-sized) on the IDE controller – ada0	At least 10 GB to store FreeBSD 11.1 Release default installation	At least 10 GB to store FreeBSD 11.1 Release default installation	At least 10 GB to store FreeBSD 11.1 Release default installation
Base memory	1024MB	1024MB	1024MB

Detailed layout of the BeaST Classic architecture with RAID arrays and fail-over Arbitrator is shown in the figure below. All the object names used in this figure are the same as in the Oracle VM Virtual Box lab specified above:



Initial controllers configuration

The latest FreeBSD 11.1 Release is installed on non-shareable ada0 drives of both storage controllers with the appropriate changes in /etc/rc.conf files:

ctrl-a	ctrl-b

hostname="ctrl-a"
 	
ifconfig_em0="DHCP"
ifconfig_em1="inet 192.168.10.10 netmask 0xffffff00" # Cross-controller
ifconfig_em2="inet 192.168.12.10 netmask 0xffffff00" # Client's LAN
 	
sshd_enable="YES"
# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable 	
dumpdev="AUTO"
 	
# VirtualBox guest additions 	
vboxguest_enable="YES"
vboxservice_enable="YES"
	hostname="ctrl-b"
	
	ifconfig_em0="DHCP"
	ifconfig_em1="inet 192.168.10.11 netmask 0xffffff00" # Cross-controller
	ifconfig_em2="inet 192.168.12.11 netmask 0xffffff00" # Client's LAN
	
	sshd_enable="YES"
	# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
	dumpdev="AUTO"
	
	# 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	ctl_load="YES" kern.cam.ctl.ha_id=2 kern.cam.ctl.ha_mode=2

And reboot both controllers:

```
# reboot
```

The “kern.cam.ctl.ha_mode=2” enables ALUA mode, which means that secondary CTL HA node accepts all requests and data to the LUN but then forwards everything to primary node. Full active-active access “kern.cam.ctl.ha_mode=1” is possible for the BeaST and may work with simple software GEOM configurations and in path-through hardware RAID modes, yet it is not tested well.

RAID and data partitions creation

The BeaST Classic may utilize quite all GEOM based RAID providers supported by FreeBSD. Hardware enforced RAID arrays should also work, thought these variants are not tested yet.

All shared drives are gathered into two groups. In the case of this document drives ada1/ada2 are normally owned by ctrl-a, whilst ada3/ada4 drives are managed by ctrl-b controller. In case of a controller failure all disks are managed by a single living controller.

For simplicity of the document a configuration with two GEOM based drive mirrors is chosen for this guide. Drives ada1/ada2 are used to form ctrl_a_gm0 mirror owned by ctrl-a controller and ada3/ada4 disks form ctrl_b_gm0 owned by ctrl-b controller.

ctrl-a	ctrl-b
gmirror load gmirror label -v ctrl_a_gm0 /dev/ada1 /dev/ada2	gmirror load gmirror label -v ctrl_b_gm0 /dev/ada3 /dev/ada4

Add geom_mirror_load="YES" to /boot/loader.conf on both controllers:

ctrl-a	ctrl-b
`geom_mirror_load="YES"`	`geom_mirror_load="YES"`

Note, the step above may not be needed when using hardware enforced arrays for creating RAID space. See the RAID configuration guide of the appropriate vendor for the details.

After RAID creation is done, the space is divided into partitions for system and client usage by running these commands:

ctrl-a	ctrl-b

`gpart create -s GPT /dev/mirror/ctrl_a_gm0`
 	
`# BeaST Quorum partition (ctrl_a_gm0p1)`
`gpart add -b 40 -s 10M -t freebsd-ufs`
`/dev/mirror/ctrl_a_gm0`
 	
`# Client data space (ctrl_a_gm0p2)`
`gpart add -t freebsd-ufs -a 1m /dev/mirror/ctrl_a_gm0`
	`gpart create -s GPT /dev/mirror/ctrl_b_gm0`
	
	`# BeaST Quorum partition (ctrl_b_gm0p1)`
	`gpart add -b 40 -s 10M -t freebsd-ufs`
	`/dev/mirror/ctrl_b_gm0`
	
	`# Client data space (ctrl_b_gm0p2)`
	`gpart add -t freebsd-ufs -a 1m /dev/mirror/ctrl_b_gm0`

Finally reboot both controllers:

```
# reboot
```

CTL HA and Front-End configuration

The BeaST Classic provides access through both controllers to the client data space. It also utilizes CTL HA to switch paths in case of one of the controllers' death.

Start CTL HA interconnection on the back-end:

ctrl-a	ctrl-b
`# sysctl kern.cam.ctl.ha_peer="listen 192.168.10.10:7777"`	`# sysctl kern.cam.ctl.ha_peer="connect 192.168.10.10:7777"`

Prepare front-end configuration in /etc/ctl.conf:

ctrl-a	ctrl-b

`portal-group pg0 {`
`  discovery-auth-group no-authentication`
`  listen 192.168.12.10`
`}`
 	
`target iqn.2016-01.local.beast:target0 {`
`  auth-group no-authentication`
`  portal-group pg0`
 	
`  lun 0 {`
`  path /dev/mirror/ctrl_a_gm0p2`
`  option ha_role primary`
`  }`
 	
`  lun 1 {`
`  path /dev/mirror/ctrl_b_gm0p2`
`  option ha_role secondary`
`  }`
`}`
	`portal-group pg0 {`
	`  discovery-auth-group no-authentication`
	`  listen 192.168.12.11`
	`}`
	
	`target iqn.2016-01.local.beast:target0 {`
	`  auth-group no-authentication`
	`  portal-group pg0`
	
	`  lun 0 {`
	`  path /dev/mirror/ctrl_a_gm0p2`
	`  option ha_role secondary`
	`  }`
	
	`  lun 1 {`
	`  path /dev/mirror/ctrl_b_gm0p2`
	`  option ha_role primary`
	`  }`
	`}`

Start ctld daemon to accept iSCSI connections:

ctrl-a	ctrl-b

service ctld start	service ctld start

Automated Fail-over/Fail-back configuration

Using Arbitrator mechanism the BeaST Classic performs automated failover task in case of a single controller's death. When the controller is going back online it must be included to the data routing paths by repeating most of configuration steps shown above.

These fail-back tasks are performed with BQ (the BeaST Quorum) software.

There is no FreeBSD port or package for BQ now, therefore installation is done manually:

```
# fetch --no-verify-peer http://downloads.sourceforge.net/project/bquorum/bq-1.1.tgz
# tar zxvf bq-1.1.tgz
# cd bq-1.1
# make install
```

The BeaST Quorum must label shared drives to use. The command below installs BQ header to the system partition (/dev/mirror/ctrl_a_gm0p1) on one of the RAID arrays. Heartbeat frequency (-f) is set to 1 second, alive timeout (-t) to 10 seconds:

```
# bq -I -d /dev/mirror/ctrl_a_gm0p1 -f 1 -t 10
```

Then start bq in daemon mode on each controller:

ctrl-a
bq -S -d /dev/mirror/ctrl_a_gm0p1 -n 0 -s /usr/local/etc/bq/bq.trigger.n0 -l /var/log/bq.log

ctrl-b
bq -S -d /dev/mirror/ctrl_a_gm0p1 -n 1 -s /usr/local/etc/bq/bq.trigger.n1 -l /var/log/bq.log

Where:
-S means "start" quorum operations
-n Current node id 0 or 1
-s trigger (script or command) to run in case the node becomes alive or dead
-l path to the BQ log file.

Examples of bq.trigger.n0 and bq.trigger.n1 scripts are shown in Appendix A and Appendix B.

The state of BQ may be checked with the command:

```
# bq -L -d /dev/mirror/ctrl_a_gm0p1
```

If everything is OK, create /usr/local/etc/rc.d/bq.sh with appropriate permissions to start BQ at boot-time:

ctrl-a
#!/bin/sh /usr/local/bin/bq -S -d /dev/mirror/ctrl_a_gm0p1 -n 0 -s /usr/local/etc/bq/bq.trigger.n0 -l /var/log/bq.log

```
# chmod 755 /usr/local/etc/rc.d/bq.sh
```

ctrl-b
#!/bin/sh /usr/local/bin/bq -S -d /dev/mirror/ctrl_a_gm0p1 -n 1 -s /usr/local/etc/bq/bq.trigger.n1 -l /var/log/bq.log
chmod 755 /usr/local/etc/rc.d/bq.sh

After performing all the steps above the BeaST classic storage system is fully configured with RAID arrays and fail-over Arbitrator mechanism.

Sample FreeBSD client configuration

Changes in /etc/rc.conf essential to work with the BeaST Classic storage system:

```
hostname="cln-1"
# Management LAN
ifconfig_em0="DHCP"
# Public network
ifconfig_em1="inet 192.168.12.12 netmask 0xffffff00"
# VirtualBox guest additions
vboxguest_enable="YES"
vboxservice_enable="YES"

# iSCSI Initiators
iscsid_enable="YES"
```

Add kernel modules to /boot/loader.conf to load them at boot time:

```
geom_multipath_load="YES"
geom_stripe_load="YES"
```

In /etc/sysctl.conf iSCSI “disconnection on fail” kernel variable is set to 1 to enable fail-over to the living controller in case of disaster:

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

The tasks needed to connect with the BeaST Classic storage system using iSCSI protocol:

```
# iscsictl -A -p 192.168.12.10 -t iqn.2016-01.local.beast:target0
```

The client detects two new drives from ctrl-a:

- da0 – /dev/mirror/ctrl_a_gm0p2
- da1 – /dev/mirror/ctrl_b_gm0p2

```
# iscsictl -A -p 192.168.12.11 -t iqn.2016-01.local.beast:target0
```

Two more drives from ctrl-b:

- da2 – /dev/mirror/ctrl_a_gm0p2
- da3 – /dev/mirror/ctrl_b_gm0p2

Configure appropriate multipath for the devices:

```
# gmultipath label CTRL_A /dev/da0 /dev/da2
# gmultipath label CTRL_B /dev/da3 /dev/da1
```

Note, although the BeaST Classic has active-active controllers, the path to the client LUN through the non-owner controller is longer and takes more time. Therefore, depending on the particular workload you may prefer to use active-passive (as shown in the example above) or active-read multipathing algorithms to active-active one.

Configure stripe over the LUNs of the BeaST Classic storage system:

```
# gstripe label BEAST /dev/multipath/CTRL_A /dev/multipath/CTRL_B
```

Finally, create a new filesystem and mount it to use for storing test only data:

```
newfs /dev/stripe/BEAST
mount /dev/stripe/BEAST /mnt
```

Warning message

The BeaST Classic is the study of the storage systems technology. All the ideas, algorithms and solutions are at concept, development and testing stages. Do not implement the BeaST Classic in production as there is no guarantee that you will not lose data.

Important implementation notes

When implementing the BeaST Classic in the real hardware, consider using port-trunking for back-end cross-controller links. It will increase bandwidth and prevent occasional split-brain condition when this interconnect is lost because of LAN issues. **Immediately shutdown one of the controllers if there is only one link left in the trunk**.

To increase reliability ob the BeaST Quorum **install watch-dog software to restart bq daemon if it hangs or stops running**.

Appendix A. Sample bq.trigger.n0 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2018 Mikhail Zakharov <zmey20000@yahoo.com>
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#    in this position and unchanged.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. The name of the author may not be used to endorse or promote products
#    derived from this software without specific prior written permission
```

```
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
# OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
# IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
# NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
# THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#

# ------------------------------------------------------------------------
#
# bq.trigger 0|1 0|1 alive|dead
#
# $1 - Current node number: 0|1
# $2 - Trigger node number: 0|1
# $3 - Trigger node event: alive|dead

# Set IP:port for both nodes ---------------------------------------------
this_node="192.168.10.10:7777"
that_node="192.168.10.11:7777"
this_luns="0"                   # Primary LUNs on This node (secondary on That)
that_luns="1"                   # Secondary LUNs on This node (primary on That)
# ------------------------------------------------------------------------

usage()
{
        printf "Usage: $0 0|1 0|1 alive|dead\n"
        exit 1
}

[ "$1" = "" -o "$2" = "" -o "$3" = "" ] &&
        {
                printf "Error: all parameters must be specified\n";
                usage;
        }
[ "$3" != "alive" -a "$3" != "dead" ] &&
        {
                printf "Error: Trigger event: alive|dead must be specified\n";
                usage;
        }

[ $1 -lt 0 -o $1 -gt 1 ] &&
        {
                printf "Error: Current node number must be 0 or 1\n";
                usage;
        }
[ $2 -lt 0 -o $2 -gt 1 ] &&
        {
                printf "Error: Trigger node number must be 0 or 1\n";
                usage;
        }

current_node=$1
trigger_node=$2
trigger_status=$3

# With current BeaST Quorum implementation we have three possible combinations:
```

```
# 1. Self status alive: Current node == Trigger node; Trigger Status == alive
[ "$current_node" = "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
                printf "Node %s is Alive. Taking back LUNs\n" "$trigger_node"

                /sbin/sysctl kern.cam.ctl.ha_peer="listen $this_node"
            service ctld onestart
                for lun in $this_luns; do
                        ctladm modify -b block -l $lun -o ha_role=primary
                done
        }

# 2. Cross node is dead: Current node != Trigger node; Trigger Status == dead
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "dead" ] &&
        {
                printf "Node %s is Dead. Node %s takes ownership\n" \
                        "$trigger_node" "$current_node";

                for lun in $that_luns; do
                        ctladm modify -b block -l $lun -o ha_role=primary
                done
        }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
                printf "Node %s is Alive. Node %s returns LUNs\n" \
                        "$trigger_node" "$current_node";

                for lun in $that_luns; do
                        ctladm modify -b block -l $lun -o ha_role=secondary
                done
        }
```

Appendix B. Sample bq.trigger.n1 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2018 Mikhail Zakharov <zmey20000@yahoo.com>
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#    in this position and unchanged.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in the
#    documentation and/or other materials provided with the distribution.
# 3. The name of the author may not be used to endorse or promote products
#    derived from this software without specific prior written permission
#
# THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
# OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
# IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
# NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
```

```
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
# THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#

# ------------------------------------------------------------------------
#
# bq.trigger 0|1 0|1 alive|dead
#
# $1 - Current node number: 0|1
# $2 - Trigger node number: 0|1
# $3 - Trigger node event: alive|dead

# Set IP:port for both nodes ----------------------------------------------
this_node="192.168.10.11:7777"
that_node="192.168.10.10:7777"
this_luns="1"                   # Primary LUNs on This node (secondary on That)
that_luns="0"                   # Secondary LUNs on This node (primary on That)
# ------------------------------------------------------------------------

usage()
{
        printf "Usage: $0 0|1 0|1 alive|dead\n"
        exit 1
}

[ "$1" = "" -o "$2" = "" -o "$3" = "" ] &&
        {
                printf "Error: all parameters must be specified\n";
                usage;
        }
[ "$3" != "alive" -a "$3" != "dead" ] &&
        {
                printf "Error: Trigger event: alive|dead must be specified\n";
                usage;
        }

[ $1 -lt 0 -o $1 -gt 1 ] &&
        {
                printf "Error: Current node number must be 0 or 1\n";
                usage;
        }
[ $2 -lt 0 -o $2 -gt 1 ] &&
        {
                printf "Error: Trigger node number must be 0 or 1\n";
                usage;
        }

current_node=$1
trigger_node=$2
trigger_status=$3

# With current BeaST Quorum implementation we have three possible combinations:

# 1. Self status alive: Current node == Trigger node; Trigger Status == alive
[ "$current_node" = "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
                printf "Node %s is Alive. Taking back LUNs\n" "$trigger_node";

                /sbin/sysctl kern.cam.ctl.ha_peer="connect $that_node";
```

```
            service ctld onestart;
                for lun in $this_luns; do
                        ctladm modify -b block -l $lun -o ha_role=primary
                done
        }

# 2. Cross node is dead: Current node != Trigger node; Trigger Status == dead
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "dead" ] &&
        {
                printf "Node %s is Dead. Node %s takes ownership\n" \
                        "$trigger_node" "$current_node";

                for lun in $that_luns; do
                        ctladm modify -b block -l $lun -o ha_role=primary
                done
        }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
                printf "Node %s is Alive. Node %s returns LUNs\n" \
                        "$trigger_node" "$current_node";

                for lun in $that_luns; do
                        ctladm modify -b block -l $lun -o ha_role=secondary
                done
        }
```