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

Mikhail E. Zakharov zmey20000@yahoo.com

Version 1.0, 2020.08.28

Preface

The BeaST Grid is the FreeBSD based reliable storage system concept. It utilizes 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 backend which is also represented by three or more commodity servers. Using iSCSI (Fibre Channel in the future) both controllers have access to all storage nodes on the back-end. Depending on particular configuration it allows the BeaST Grid to create wide range of GEOM based software or hardware RAID array types along with ZFS storage pools.

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

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

Warning! This is alpha version of the BeaST Grid concept. It means not all the subsystems are implemented or work properly. Do not under any circumstances use this version in production.

Description

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

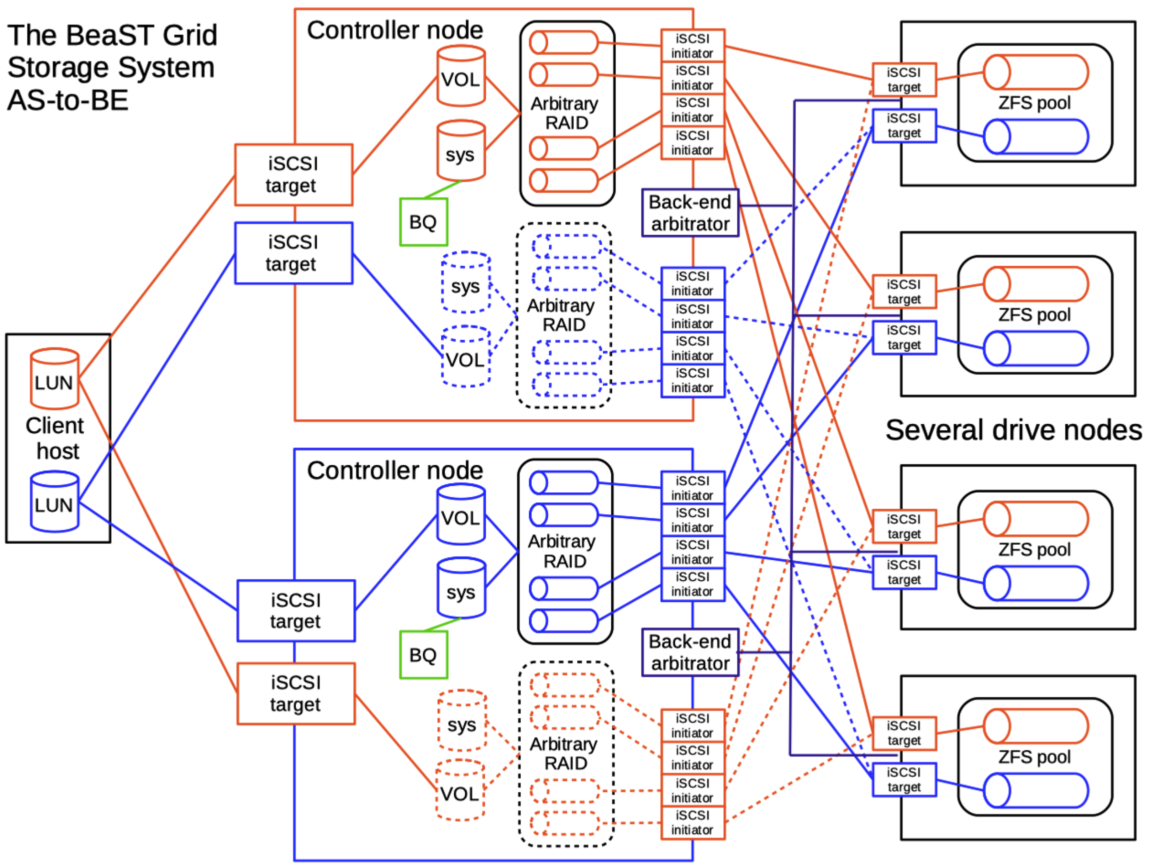
Two storage controllers (c1 and c2) are connected with two Internal LAN (host-only networks: 10.0.1.0/24, 10.0.2.0/24) to transfer all backend traffic and the Public LAN (host-only network 10.0.3.0/24) to allow the client host (h1) to reach LUNs on the BeaST Grid storage system.

The configuration summary is shown in the table below:

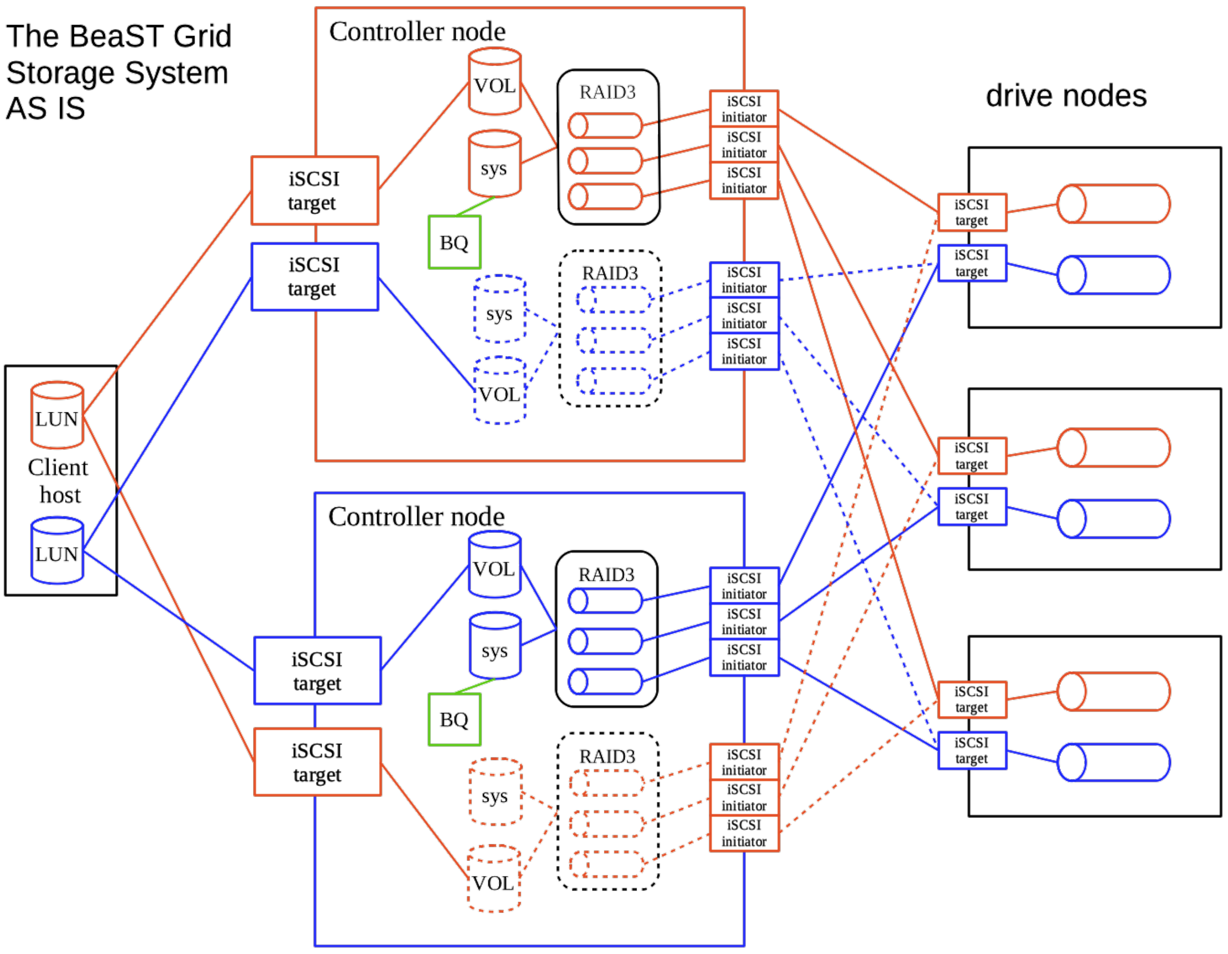
Node	Public LAN IPs	Internal LAN IP addresses	Drives	Base memory
c1	10.0.3.1/24	10.0.1.1/24, 10.0.2.1/24	System virtual drive on the IDE controller – ada0: 6-10GB to store FreeBSD OS	1024MB
c2	10.0.3.2/24	10.0.1.2/24,	System virtual drive on the IDE controller –	1024MB

		10.0.2.2/24	ada0: 6-10GB to store FreeBSD OS	
d1		10.0.1.10/24, 10.0.2.10/24	• System virtual drive on the IDE controller – ada0: 6-10GB to store FreeBSD OS • 2 virtual drives for the data storage – da0 and da1: 2GB each, connected to a virtual SAS or SATA controller	1024MB
d2		10.0.1.20/24, 10.0.2.20/24	• System virtual drive on the IDE controller – ada0: 6-10GB to store FreeBSD OS • 2 virtual drives for the data storage – da0 and da1: 2 GB each, connected to a virtual SAS or SATA controller	1024MB
d3		10.0.1.30/24, 10.0.2.30/24	• System virtual drive on the IDE controller – ada0: 6-10GB to store FreeBSD OS • 2 virtual drives for the data storage – da0 and da1: 2GB each, connected to a virtual SAS or SATA controller	1024MB
h1	10.0.3.3/24		System virtual drive on the IDE controller – ada0: 6-10GB to store FreeBSD OS	1024MB

Detailed layout of the **future** version of the BeaST Grid architecture with RAID arrays and CTL HA is shown in the figure below:



Detailed layout of the **current** state of the BeaST Grid architecture with RAID arrays and CTL HA is shown in the figure below:



The current state is much simpler than the future one. For example, it lacks the essential Back-end arbitrator which manages online connection/disconnection of the drive-nodes and prevents data corruption at this point time on concurrent mutually exclusive operations on them.

Initial controllers configuration

The FreeBSD OS is installed on non-shareable ada0 drives of the controllers with the appropriate changes in system files described below.

/etc/rc.conf

c1	`hostname="c1"`
	`ifconfig_em0="DHCP"`
	`ifconfig_em0_ipv6="inet6 accept_rtadv"`
	`ifconfig_em1="inet 10.0.1.1 netmask 255.255.255.0"`
	`ifconfig_em2="inet 10.0.2.1 netmask 255.255.255.0"`
	
	`sshd_enable="YES"`
	`# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable`
	`dumpdev="AUTO"`

	`# iSCSI target`
	`ctld_enable="YES"`
	`# iSCSI Initiators`
	`iscsid_enable="YES"`
	`# VirtualBox guest additions`
	`vboxguest_enable="YES"`
	`vboxservice_enable="YES"`
c2	`hostname="c2"`
	`ifconfig_em0="DHCP"`
	`ifconfig_em0_ipv6="inet6 accept_rtadv"`
	`ifconfig_em1="inet 10.0.1.2 netmask 255.255.255.0"`
	`ifconfig_em2="inet 10.0.2.2 netmask 255.255.255.0"`
	`sshd_enable="YES"`
	`# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable`
	`dumpdev="AUTO"`
	`# iSCSI target`
	`ctld_enable="YES"`
	`# iSCSI Initiators`
	`iscsid_enable="YES"`
	`# VirtualBox guest additions`
	`vboxguest_enable="YES"`
	`vboxservice_enable="YES"`
d1	`hostname="d1"`
	`ifconfig_em0="DHCP"`
	`ifconfig_em0_ipv6="inet6 accept_rtadv"`
	`ifconfig_em1="inet 10.0.1.10 netmask 255.255.255.0"`
	`ifconfig_em2="inet 10.0.2.10 netmask 255.255.255.0"`
	`sshd_enable="YES"`
	`# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable`
	`dumpdev="AUTO"`
	`# iSCSI target`
	`ctld_enable="YES"`
	`# VirtualBox guest additions`
	`vboxguest_enable="YES"`
	`vboxservice_enable="YES"`
d2	`hostname="d2"`
	`ifconfig_em0="DHCP"`
	`ifconfig_em0_ipv6="inet6 accept_rtadv"`
	`ifconfig_em1="inet 10.0.1.20 netmask 255.255.255.0"`
	`ifconfig_em2="inet 10.0.2.20 netmask 255.255.255.0"`
	`sshd_enable="YES"`
	`# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable`
	`dumpdev="AUTO"`
	`# iSCSI target`
	`ctld_enable="YES"`
	`# VirtualBox guest additions`

	vboxguest_enable="YES"
	vboxservice_enable="YES"

d3	hostname="d3"
	ifconfig_em0="DHCP"
	ifconfig_em0_ipv6="inet6 accept_rtadv"
	ifconfig_em1="inet 10.0.1.30 netmask 255.255.255.0"
	ifconfig_em2="inet 10.0.2.30 netmask 255.255.255.0"
	
	sshd_enable="YES"
	# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
	dumpdev="AUTO"
	
	# iSCSI target
	ctld_enable="YES"
	
	# VirtualBox guest additions
	vboxguest_enable="YES"
	vboxservice_enable="YES"

/etc/sysctl.conf

c1	kern.iscsi.fail_on_disconnection=1
c2	kern.iscsi.fail_on_disconnection=1

/boot/loader.conf

c1	geom_multipath_load="YES"
	geom_raid3_load="YES"
	
	ctl_load="YES"
	kern.cam.ctl.ha_id=1
	# kern.cam.ctl.ha_role state is managed by BQ
	kern.cam.ctl.ha_role=1
	kern.cam.ctl.ha_mode=2

c2	geom_multipath_load="YES"
	geom_raid3_load="YES"
	
	ctl_load="YES"
	kern.cam.ctl.ha_id=2
	# kern.cam.ctl.ha_role state is managed by BQ
	kern.cam.ctl.ha_role=1
	kern.cam.ctl.ha_mode=2

d1	ctl_load="YES"
d2	ctl_load="YES"
d3	ctl_load="YES"

/etc/ctl.conf

c1	portal-group pg0 {
	        discovery-auth-group no-authentication
	        listen 10.0.3.1
	}

```
target iqn.2016-01.local.beast:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                device-id "BeaST-C1R3p2"
                path /dev/raid3/C1R3p2
                blocksize 1024

        }

        lun 1 {
                device-id "BeaST-C2R3p2"
                path /dev/raid3/C2R3p2
                blocksize 1024
        }
}
```

c2

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 10.0.3.2
}

target iqn.2016-01.local.beast:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                device-id "BeaST-C1R3p2"
                path /dev/raid3/C1R3p2
                blocksize 1024

        }

        lun 1 {
                device-id "BeaST-C2R3p2"
                path /dev/raid3/C2R3p2
                blocksize 1024
        }
}
```

d1

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 10.0.1.10
        listen 10.0.2.10
}

target iqn.2016-01.internal.beast:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                device-id "BeaST-D1L0"
                path /dev/da0
        }

        lun 1 {
                device-id "BeaST-D1L1"
                path /dev/da1
        }
}
```

d2

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 10.0.1.20
        listen 10.0.2.20
}

target iqn.2016-01.internal.beast:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
            device-id "BeaST-D2L0"
                path /dev/da0
        }

        lun 1 {
            device-id "BeaST-D2L1"
                path /dev/da1
        }
}
```

d3

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 10.0.1.30
        listen 10.0.2.30
}

target iqn.2016-01.internal.beast:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                device-id "BeaST-D3L0"
                path /dev/da0
        }

        lun 1 {
                device-id "BeaST-D3L1"
                path /dev/da1
        }
}
```

The “kern.cam.ctl.ha_mode=2” enables ALUA mode, which means that the secondary CTL HA node accepts all requests and data to the LUN but then forwards everything to the primary node.

RAID and data partitions creation

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

All LUNs imported from the drive-nodes are gathered into two logical groups. In the case of this document even numbered drives are normally “owned” by c1, whilst uneven ones are managed by c2 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 RAID3 arrays is chosen for this guide:

Configure gmultipath on C1 (changes are automatically seen on C2)

To utilize data storage from drive nodes configure multipathing for them:

```
# gmultipath label -vA D1L0 /dev/da0 /dev/da2
# gmultipath label -vA D1L1 /dev/da1 /dev/da3

# gmultipath label -vA D2L0 /dev/da4 /dev/da6
# gmultipath label -vA D2L1 /dev/da5 /dev/da7

# gmultipath label -vA D3L0 /dev/da8 /dev/da10
# gmultipath label -vA D3L1 /dev/da9 /dev/da11
```

RAID3 on controllers on C1 (changes are automatically seen on C2)

Build two RAID3 groups C1R3 is to be managed by C1 and C2R3 respectively by C2:

```
# graid3 label -v -r C1R3 /dev/multipath/D1L0 /dev/multipath/D2L0 /dev/multipath/D3L0
# graid3 label -v -r C2R3 /dev/multipath/D1L1 /dev/multipath/D2L1 /dev/multipath/D3L1
```

Partition RAID3 drives on C1 (changes are automatically seen on C2)

Create GPT scheme:

```
# gpart create -s GPT /dev/raid3/C1R3
# gpart create -s GPT /dev/raid3/C2R3
```

Create the BeaST Quorum partition:

```
# gpart add -b 40 -s 10M -t freebsd-ufs /dev/raid3/C1R3
# gpart add -b 40 -s 10M -t freebsd-ufs /dev/raid3/C2R3
```

And the client data partition:

```
# gpart add -t freebsd-ufs -a 1m /dev/raid3/C1R3
# gpart add -t freebsd-ufs -a 1m /dev/raid3/C2R3
```

Automated Fail-over/Fail-back configuration

Using CTL HA mechanism the BeaST Grid performs automated failover task in case of a single controller’s death. When the controller is going back online it must be attached 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-1.4.tgz
# tar zxvf bq-1.4.tgz
# cd bq-1.4
# 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/raid3/C1R3p1 -f 1 -t 10
```

Scripts /usr/local/etc/bq/bq.trigger.n0 and /usr/local/etc/bq/bq.trigger.n1 are used to control Failover/ Failback operations. Examples of these files are shown in Appendix A and Appendix B. For the described configuration to work, change this_node and that_node variables in /usr/local/etc/bq/bq.trigger.n0 and /usr/local/etc/bq/bq.trigger.n1 scripts:

c1	this_node="10.0.1.1:7777"
	that_node="10.0.2.1:7777"

c2	this_node="10.0.2.1:7777"
	that_node="10.0.1.1:7777"

The state of BQ may be checked with the command:

```
# bq -L -d /dev/raid3/C1R3p1
```

Finally reboot both controllers one by one:

```
# reboot
```

After performing all the steps above the BeaST Grid 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 Grid storage system:

```
hostname="h-1"
# Management LAN
ifconfig_em0="DHCP"
# Public network
ifconfig_em1="inet 10.0.3.3 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 Grid storage system using iSCSI protocol:

```
# iscsictl -A -p 10.0.3.1 -t iqn.2016-01.local.beast:target0
# iscsictl -A -p 10.0.3.2 -t iqn.2016-01.local.beast:target0
```

Setup multipathing on the client:
```
# gmultipath create -vA C1 /dev/da0 /dev/da2
# gmultipath create -vA C2 /dev/da1 /dev/da3
```

Note, although the BeaST Grid controllers on the front-end have active-active connections, 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.

Create a striped volume:
```
# gstripe create BEAST /dev/multipath/C1 /dev/multipath/C2
```

And finally create a filesystem and mount it:
```
# newfs /dev/stripe/BEAST
# mount /dev/stripe/BEAST /mnt
```

Warning message

The BeaST Grid 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 Grid in production as there is no guarantee that you will not lose data.

Appendix A. Sample bq.trigger.n0 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2020 Mikhail E. 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="10.0.1.1:7777"
that_node="10.0.2.1:7777"
# -----------------------------------------------------------------------------

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 "$trigger_status" = "alive" ] &&
      {
            # Must check whether:
            # - the oppisite/cross node is alive;
            # - do failback of LUNs or not.
            printf "Node %s is Alive. Demoting Node %s to Secondary\n" \
                  "$trigger_node" "$current_node";

            /sbin/sysctl kern.cam.ctl.ha_peer="connect $that_node"
            service ctld onestatus
            if [ $? -eq 1 ] ; then
                  # ctld is not running: start it
                  service ctld onestart
            else
```

```
                # ctld is running: send it -HUP signal
                service ctld onereload
            fi
        }

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

            /sbin/sysctl kern.cam.ctl.ha_role=0;
            /sbin/sysctl kern.cam.ctl.ha_peer="listen $this_node"
            if [ $? -eq 1 ] ; then
                # ctld is not running: start it
                service ctld onestart
            else
                # ctld is running: send it -HUP signal
                service ctld onereload
            fi
        }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
            printf "Node %s is Alive. Reballancing LUNs\n" "$trigger_node";
            printf "Not implemented yet.\n"
        }
```

Appendix B. Sample bq.trigger.n1 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2020 Mikhail E. 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="10.0.2.1:7777"
that_node="10.0.1.1:7777"
# --------------------------------------------------------------------------------

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 "$trigger_status" = "alive" ] &&
      {
            # Must check whether:
            # - the oppisite/cross node is alive;
            # - do failback of LUNs or not.
            printf "Node %s is Alive. Demoting Node %s to Secondary\n" \
                  "$trigger_node" "$current_node";

            /sbin/sysctl kern.cam.ctl.ha_peer="connect $that_node"
            service ctld onestatus
            if [ $? -eq 1 ] ; then
                  # ctld is not running: start it
```

```
                service ctld onestart
            else
                # ctld is running: send it -HUP signal
                service ctld onereload
            fi
        }

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

            /sbin/sysctl kern.cam.ctl.ha_role=0;
            /sbin/sysctl kern.cam.ctl.ha_peer="listen $this_node"
            if [ $? -eq 1 ] ; then
                # ctld is not running: start it
                service ctld onestart
            else
                # ctld is running: send it -HUP signal
                service ctld onereload
            fi
        }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
        {
            printf "Node %s is Alive. Reballancing LUNs\n" "$trigger_node";
            printf "Not implemented yet.\n"
        }
```