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

Mikhail E. Zakharov zmey20000@yahoo.com

Version 1.1 2017.05.23

Table of Contents

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 ZFS arrays and CTL HA.

The BeaST Classic with ZFS runs in two modes:

- Active-Active/Asymmetric – On the front-end, both controllers accept and serve requests. On the back-end, storage pools are balanced over controllers.
- Active/Hot-Standby – On the front-end, both controllers accept and serve requests. On the backend, storage pools are managed by an Active controller only.

When running in Active-Active/Asymmetric mode, the BeaST Classic with ZFS automatically fails to the Active/Hot-Standby on a controller failure. If a controller was successfully recovered from a failure, the system continues to work in the Active/Hot-Standby mode. A special procedure (see

Appendix A) which requires the storage system to be turned offline is used to enable Active-Active/Asymmetric mode.

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 tested configuration of the BeaST Classic storage system was build 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).
- With virtual SAS controllers to share four cache drives da0, da1, da2, da3 (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).

  Drives da0, da1 are used to store shared ZFS Intent Log (ZIL) and sa2, da3 drives for Level 2 Adaptive Replacement Cache (L2ARC).

  Cache drives can be replaced with software mirroring option, but it is strongly not recommended because of stability issues. This guide gives some hints on how to configure cache software mirroring option in the Appendix D.

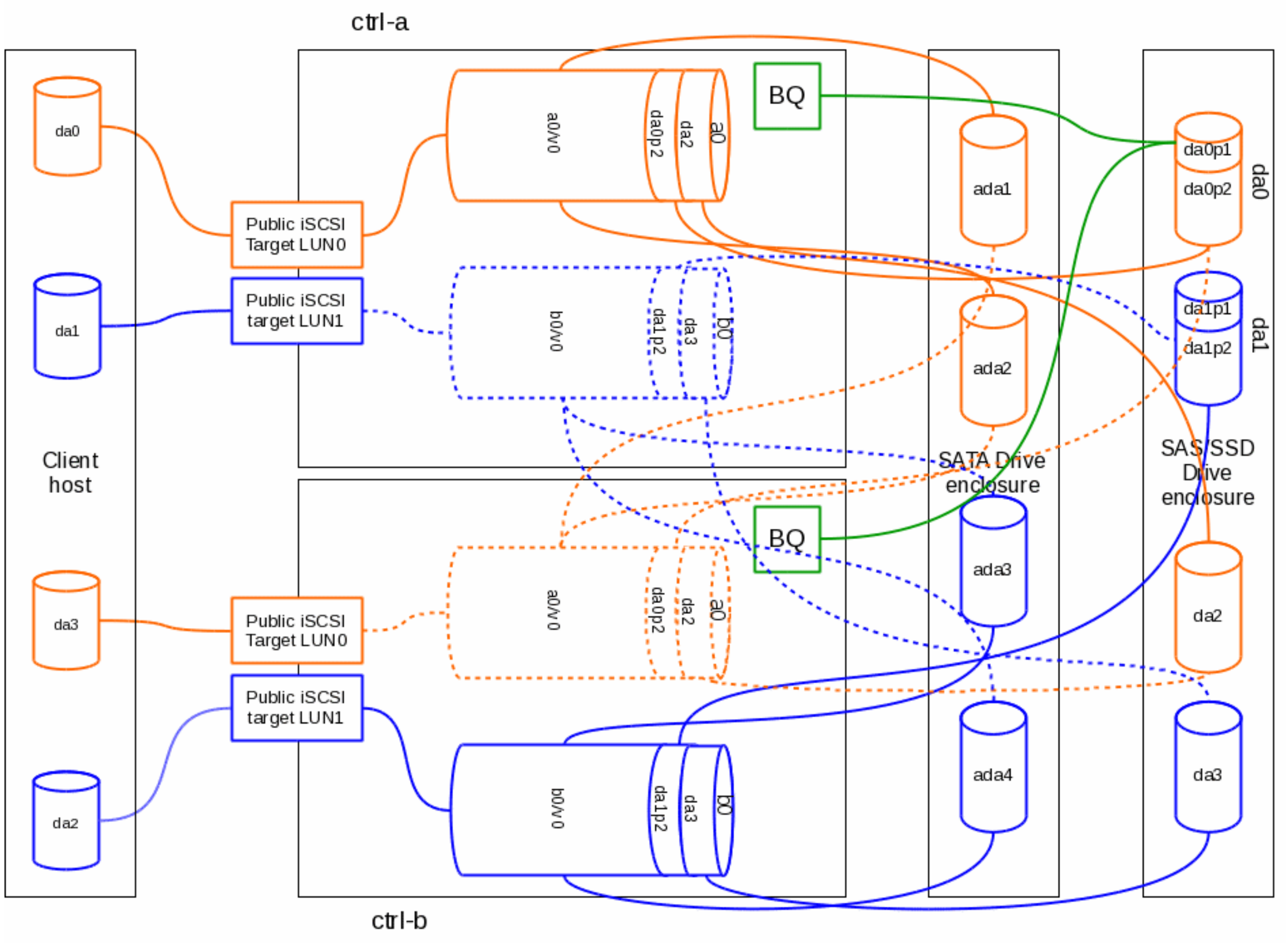
Both controllers are connected with the Cross-controller LAN (host-only network 192.168.10.0/24) to transfer all back-end traffic and the Public LAN (host-only network 192.168.12.0/24) to allow the client host (clnt-1) to reach LUNs on the BeaST Classic storage system.

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: 192.168.11.10 Mask: 255.255.255.0	IP: 192.168.11.11 Mask: 255.255.255.0	IP: 192.168.11.111 Mask: 255.255.255.0
Management LAN	DHCP	DHCP	DHCP
Shareable, fixed-sized virtual drives on the SATA controller - ada1, ada2, ada3, ada4	Each drive is 256MB size		-
Shareable, fixed-sized virtual drives on the SAS controller - da0, da1, da2, da3	Each drive is 128MB size		-
System virtual drive	At least 10 GB to store	At least 10 GB to store	At least 10 GB to store

(Dynamic-sized) on the IDE controller – ada0	FreeBSD 11.0 Release default installation	FreeBSD 11.0 Release default installation	FreeBSD 11.0 Release default installation
Base memory	1024MB	1024MB	1024MB

Detailed layout of the BeaST Classic architecture with ZFS 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 for the Active-Active/Asymmetric mode

The latest FreeBSD 11.0 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 LAN`
`ifconfig_em2="inet 192.168.11.10`
`netmask 0xffffff00"    # Public LAN`
`ifconfig_em3="inet 192.168.12.10`
`netmask 0xffffff00"    # Cache mirror`
	`hostname="ctrl-b"`
	
	`ifconfig_em0="DHCP"`
	`ifconfig_em1="inet 192.168.10.11`
	`netmask 0xffffff00"    #`
	`Cross-controller LAN`
	`ifconfig_em2="inet 192.168.11.11`
	`netmask 0xffffff00"    # Public LAN`
	`ifconfig_em3="inet 192.168.12.11`
	`netmask 0xffffff00"    # Cache mirror`

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

The /boot/loader.conf sets boot time parameters for Active/Active Asymmetric mode 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

Start CTL HA:

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"

BeaST system partition is created for the BeaST Quorum software:

ctrl-a	ctrl-b
gpart create -s GPT /dev/da0 # BeaST Quorum partition (da0p1) gpart add -b 40 -s 10M -t freebsd-ufs /dev/da0 # Space for a0 pool ZIL (da0p2) gpart add -t freebsd-ufs -a 1m /dev/da0	gpart create -s GPT /dev/da1 # BeaST Quorum partition (da1p1) gpart add -b 40 -s 10M -t freebsd-ufs /dev/da1 # Space for b0 pool ZIL (da1p2) gpart add -t freebsd-ufs -a 1m /dev/da1
# sync partitions on both controllers reboot	# sync partitions on both controllers reboot

ZFS pools and volumes

ctrl-a	ctrl-b
zpool create -m none a0 mirror /dev/ada1 /dev/ada2 zpool set cachefile=/boot/zfs/zpool.cache.beast.a0 a0	zpool create -m none b0 mirror /dev/ada3 /dev/ada4 zpool set cachefile=/boot/zfs/zpool.cache.beast.b0 b0
# Sync ZFS cache-files on controllers: cp /boot/zfs/zpool.cache.beast.a0 / scp beast@192.168.10.11:/boot/zfs/zpool.cache.beast.b0 / cp /zpool.cache.beast.b0 /boot/zfs/	# Sync ZFS cache-files on controllers: cp /boot/zfs/zpool.cache.beast.b0 / scp beast@192.168.10.10:/boot/zfs/zpool.cache.beast.a0 / cp /zpool.cache.beast.a0 /boot/zfs/
# ZIL: zpool add -f a0 log /dev/da0p2 zfs set sync=always a0 # L2ARC: zpool add -f a0 cache /dev/da2 zfs set primarycache=all a0	# ZIL: zpool add -f b0 log /dev/da1p2 zfs set sync=always b0 # L2ARC: zpool add -f b0 cache /dev/da3 zfs set primarycache=all b0

`zfs set secondarycache=all a0`	`zfs set secondarycache=all b0`
`zfs create -V 128M a0/v0`	`zfs create -V 128M b0/v0`

Import cross-controller pools in read-only modes:

ctrl-a	ctrl-b
`zpool import -f -N -o readonly=on -o cachefile=/boot/zfs/zpool.cache.beast.b0 -c /boot/zfs/zpool.cache.beast.b0 b0`	`zpool import -f -N -o readonly=on -o cachefile=/boot/zfs/zpool.cache.beast.a0 -c /boot/zfs/zpool.cache.beast.a0 a0`

Configure /etc/ctl.conf to balance mode volumes over controllers:

ctrl-a	ctrl-b

# /etc/ctl.conf 	
 	
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 192.168.11.10
}
 	
target iqn.2016-01.local.beast:target0 {
        auth-group no-authentication
        portal-group pg0
 	
        lun 0 {
                path /dev/zvol/a0/v0
                option ha_role primary
                device-id "a0v0"
                serial "a0v0_0"
        }
 	
        lun 1 {
                path /dev/zvol/b0/v0
                option ha_role secondary
                device-id "b0v0"
                serial "b0v0_1"
        }
}
	# /etc/ctl.conf
	
	portal-group pg0 {
	        discovery-auth-group no-authentication
	        listen 192.168.11.11
	}
	
	target iqn.2016-01.local.beast:target0 {
	        auth-group no-authentication
	        portal-group pg0
	
	        lun 0 {
	                path /dev/zvol/a0/v0
	                option ha_role secondary
	                device-id "a0v0"
	                serial "a0v0_0"
	        }
	
	        lun 1 {
	                path /dev/zvol/b0/v0
	                option ha_role primary
	                device-id "b0v0"
	                serial "b0v0_1"
	        }
	}

Start front-end:

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

BeaST Quorum

The BeaST Quorum manages automatic Failover/Failback operations of the BeaST Classic storage system. In the BeaST Classic with ZFS storage system it enables Active/Hot-Standby mode.

Download the BeaST Quorum software, unpack it, compile and install:

```
# 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
```

Configure quorum drive using the BeaST shared system partition (`da0p1`). Heartbeat frequency (-f) is set to 1 second, alive timeout (-t) to 10 seconds:

```
# bq -I -d /dev/da0p1 -f 1 -t 10
```

Start the BeaST Quorum manually on both nodes:

ctrl-a	ctrl-b

`# /etc/rc.local`
 	
`bq -S -d /dev/da0p1 -n 0 -s /usr/local/etc/bq/bq.trigger.n0 -l /var/log/bq.log`
	`# /etc/rc.local`
	
	`bq -S -d /dev/da0p1 -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.
```

Contents of the /usr/local/etc/bq/bq.trigger.n0 and /usr/local/etc/bq/bq.trigger.n1 are shown in Appendixes B, C.

The state of BQ can be checked with the command:

```
# bq -L -d /dev/da0p1
```

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

ctrl-a	ctrl-b

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

The BeaST Classic with ZFS storage system is configured in Active-Active/Asymmetric mode. If Active/Hot-Standby mode is desired, reboot any of the nodes.

The client

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 192.168.11.111 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 192.168.11.10 -t iqn.2016-01.local.beast:target0
```

The client detects two new drives from ctrl-a:

```
da0   /dev/zvol/a0/v0   primary
da1   /dev/zvol/b0/v0   secondary
```

```
# iscsictl -A -p 192.168.11.11 -t iqn.2016-01.local.beast:target0
```

Two more drives from ctrl-b:

```
da2   /dev/zvol/a0/v0   secondary
da3   /dev/zvol/b0/v0   primary
```

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 bare-metal 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. Offline procedure to turn on the Active-Active/Asymmetric mode

This procedure requires the BeaST storage system to go offline. Unmount external drives and disconnect all hosts before proceeding further.

Run these commands to enable the Active-Active/Asymmetric mode:

ctrl-a	ctrl-b
`service ctld onestop`	`service ctld onestop`
`zpool export a0`	`zpool export a0`
`zpool export b0`	`zpool export b0`
`cp /zpool.cache.beast.* /boot/zfs`	`cp /zpool.cache.beast.* /boot/zfs`
`zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.a0 -c /boot/zfs/zpool.cache.beast.a0 a0`	`zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.b0 -c /boot/zfs/zpool.cache.beast.b0 b0`
`zpool import -f -N -o readonly=on -o cachefile=/boot/zfs/zpool.cache.beast.b0 -c /boot/zfs/zpool.cache.beast.b0 b0`	`zpool import -f -N -o readonly=on -o cachefile=/boot/zfs/zpool.cache.beast.a0 -c /boot/zfs/zpool.cache.beast.a0 a0`
`service ctld onestart`	`service ctld onestart`
`ctladm modify -b block -l 0 -o ha_role=primary`	`ctladm modify -b block -l 0 -o ha_role=secondary`
`ctladm modify -b block -l 1 -o ha_role=secondary`	`ctladm modify -b block -l 1 -o ha_role=primary`

Appendix B. Sample bq.trigger.n0 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2017 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="192.168.10.10:7777"
that_node="192.168.10.11:7777"

lock="/var/run/bq.lock"
# --------------------------------------------------------------------------------

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

wait_lock()
{
      while [ -f $lock ]
      do
            printf "Waiting for the lock\n"
            sleep 1
      done
}

[ "$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 "$trigger_status" = "alive" ] &&
      {
            printf "Node: %s: Node %s is Alive. Setting Secondary\n" \
                  "$trigger_node" "$current_node";

            wait_lock;
            printf "Setting lock\n"
            touch $lock;

            printf "Setting HA peer\n";
            sysctl kern.cam.ctl.ha_peer="connect $that_node";

            printf "Re-importing zpools\n"
            zpool export a0;
            zpool export b0;
            cp /zpool.cache.beast.a0 /boot/zfs;
            cp /zpool.cache.beast.b0 /boot/zfs;
            zpool import -f -N -o readonly=on -o
cachefile=/boot/zfs/zpool.cache.beast.a0 -c /boot/zfs/zpool.cache.beast.a0 a0;
            zpool import -f -N -o readonly=on -o
cachefile=/boot/zfs/zpool.cache.beast.b0 -c /boot/zfs/zpool.cache.beast.b0 b0;

            printf "Starting ctld\n";
            service ctld onestart;

            printf "Setting LUNs to Secondary\n";
            ctladm modify -b block -l 0 -o ha_role=secondary;
            ctladm modify -b block -l 1 -o ha_role=secondary;

            printf "Secondary Done\n"
            rm $lock
      }
```

```
# 2. Cross node is dead: Current node != Trigger node; Trigger Status == dead
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "dead" ] &&
      {
            sleep 1;          # Let the first combination to start. It happens
on boot

            printf "Node: %s: Node %s is Dead. Setting Primary\n" \
                  "$trigger_node" "$current_node";

            wait_lock;
            printf "Setting lock\n";
            touch $lock;

            printf "Setting HA peer\n";
            sysctl kern.cam.ctl.ha_peer="listen $this_node";

            printf "Re-importing zpool: %s\n" "a0";
            zpool export a0 &&
            cp /zpool.cache.beast.a0 /boot/zfs &&
            zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.a0 -c
/boot/zfs/zpool.cache.beast.a0 a0;

            printf "Re-importing zpool: %s\n" "b0";
            zpool export b0 &&
            cp /zpool.cache.beast.b0 /boot/zfs &&
            zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.b0 -c
/boot/zfs/zpool.cache.beast.b0 b0;

            printf "Reloading ctld configuration\n";
            service ctld onereload;

            printf "Setting LUNs to Primary\n";
            ctladm modify -b block -l 0 -o ha_role=primary;
            ctladm modify -b block -l 1 -o ha_role=primary;

            printf "Starting scrub on pools\n";
            zpool scrub a0;
            zpool scrub b0;

            printf "Primary Done\n";
            rm $lock;
      }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
      {
            printf "Node: %s: Node %s is Alive. Nothing to do.\n" \
                  "$trigger_node" "$current_node";
      }
```

Appendix C. Sample bq.trigger.n1 script

```
#!/bin/sh
#
# Copyright (c) 2016, 2017 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="192.168.10.11:7777"
that_node="192.168.10.10:7777"

lock="/var/run/bq.lock"
# ---------------------------------------------------------------------------------

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

wait_lock()
```

```
{
      while [ -f $lock ]
      do
            printf "Waiting for the lock\n"
            sleep 1
      done
}

[ "$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 "$trigger_status" = "alive" ] &&
      {
            printf "Node: %s: Node %s is Alive. Setting Secondary\n" \
                  "$trigger_node" "$current_node";

            wait_lock;
            printf "Setting lock\n"
            touch $lock;

            printf "Setting HA peer\n";
            sysctl kern.cam.ctl.ha_peer="connect $that_node";

            printf "Re-importing zpools\n"
            zpool export a0;
            zpool export b0;
            cp /zpool.cache.beast.a0 /boot/zfs;
            cp /zpool.cache.beast.b0 /boot/zfs;
            zpool import -f -N -o readonly=on -o
cachefile=/boot/zfs/zpool.cache.beast.a0 -c /boot/zfs/zpool.cache.beast.a0 a0;
            zpool import -f -N -o readonly=on -o
cachefile=/boot/zfs/zpool.cache.beast.b0 -c /boot/zfs/zpool.cache.beast.b0 b0;

            printf "Starting ctld\n";
            service ctld onestart;

            printf "Setting LUNs to Secondary\n";
            ctladm modify -b block -l 0 -o ha_role=secondary;
            ctladm modify -b block -l 1 -o ha_role=secondary;

            printf "Secondary Done\n"
            rm $lock
      }

# 2. Cross node is dead: Current node != Trigger node; Trigger Status == dead
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "dead" ] &&
      {
            sleep 1;          # Let the first combination to start. It happens
on boot

            printf "Node: %s: Node %s is Dead. Setting Primary\n" \
                  "$trigger_node" "$current_node";

            wait_lock;
            printf "Setting lock\n";
            touch $lock;

            printf "Setting HA peer\n";
            sysctl kern.cam.ctl.ha_peer="listen $this_node";

            printf "Re-importing zpool: %s\n" "a0";
            zpool export a0 &&
            cp /zpool.cache.beast.a0 /boot/zfs &&
            zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.a0 -c
/boot/zfs/zpool.cache.beast.a0 a0;

            printf "Re-importing zpool: %s\n" "b0";
            zpool export b0 &&
            cp /zpool.cache.beast.b0 /boot/zfs &&
            zpool import -f -N -o cachefile=/boot/zfs/zpool.cache.beast.b0 -c
/boot/zfs/zpool.cache.beast.b0 b0;

            printf "Reloading ctld configuration\n";
            service ctld onereload;

            printf "Setting LUNs to Primary\n";
            ctladm modify -b block -l 0 -o ha_role=primary;
            ctladm modify -b block -l 1 -o ha_role=primary;

            printf "Starting scrub on pools\n";
            zpool scrub a0;
            zpool scrub b0;

            printf "Primary Done\n";
            rm $lock;
      }

# 3. Cross node is alive: Current node != Trigger node; Trigger Status == alive
[ "$current_node" != "$trigger_node" -a "$trigger_status" = "alive" ] &&
      {
            printf "Node: %s: Node %s is Alive. Nothing to do.\n" \
                  "$trigger_node" "$current_node";
      }
```

Appendix D. Sample cache software mirroring options

There is a number of ways to configure cache mirroring. During the BeaST development ZIL mirroring with gmirror + istgt (/usr/ports/net/istgt) iSCSI legacy target software showed the best results. Nevertheless, all software mirroring options do not work stable enough to be recommended.

Below, the istgt sample configuration is presented:

ctrl-a	ctrl-b
# Load gmirror module	# Load gmirror module
gmirror load	gmirror load
# Create memory drives for ZIL storage:	# Create memory drives for ZIL storage:
mdconfig -a -t swap -s 128m -u 0	mdconfig -a -t swap -s 128m -u 0
# md0 - ctrl-a ZIL	# md0 - ctrl-a ZIL
mdconfig -a -t swap -s 128m -u 1	mdconfig -a -t swap -s 128m -u 1
# md1 - ctrl-b ZIL	# md1 - ctrl-b ZIL
# Start istgt to export memory drives for ZIL mirror:	# Start istgt to export memory drives for ZIL mirror:
/usr/local/bin/istgt	/usr/local/bin/istgt

ZFS ZIL mirror:

ctrl-a	ctrl-b
# /usr/local/etc/istgt/istgt.conf	# /usr/local/etc/istgt/istgt.conf
[Global]	[Global]
Comment "Global section"	Comment "Global section"
NodeBase "iqn.2016-01.local.beast.private"	NodeBase "iqn.2016-01.local.beast.private"
PidFile /var/run/istgt.pid	PidFile /var/run/istgt.pid
AuthFile /usr/local/etc/istgt/auth.conf	AuthFile /usr/local/etc/istgt/auth.conf
MediaDirectory /var/istgt	MediaDirectory /var/istgt
LogFacility "local7"	LogFacility "local7"
Timeout 30	Timeout 30
NopInInterval 20	NopInInterval 20

DiscoveryAuthMethod Auto	DiscoveryAuthMethod Auto
MaxSessions 16	MaxSessions 16
MaxConnections 4	MaxConnections 4
MaxR2T 32	MaxR2T 32
MaxOutstandingR2T 16	MaxOutstandingR2T 16
DefaultTime2Wait 2	DefaultTime2Wait 2
DefaultTime2Retain 60	DefaultTime2Retain 60
FirstBurstLength 262144	FirstBurstLength 262144
MaxBurstLength 1048576	MaxBurstLength 1048576
MaxRecvDataSegmentLength 262144	MaxRecvDataSegmentLength 262144
InitialR2T Yes	InitialR2T Yes
ImmediateData Yes	ImmediateData Yes
DataPDUInOrder Yes	DataPDUInOrder Yes
DataSequenceInOrder Yes	DataSequenceInOrder Yes
ErrorRecoveryLevel 0	ErrorRecoveryLevel 0
[UnitControl]	[UnitControl]
Comment "Internal Logical Unit Controller"	Comment "Internal Logical Unit Controller"
AuthMethod CHAP Mutual	AuthMethod CHAP Mutual
AuthGroup AuthGroup10000	AuthGroup AuthGroup10000
Portal UC1 127.0.0.1:3261	Portal UC1 127.0.0.1:3261
Netmask 127.0.0.1	Netmask 127.0.0.1
[PortalGroup1]	[PortalGroup1]
Comment "ZIL port group"	Comment "ZIL port group"
Portal DA1 192.168.11.10:3260	Portal DA1 192.168.11.11:3260
[InitiatorGroup1]	[InitiatorGroup1]
Comment "Initiator Group1"	Comment "Initiator Group1"
InitiatorName "ALL"	InitiatorName "ALL"
Netmask 192.168.11.0/24	Netmask 192.168.11.0/24
[LogicalUnit1]	[LogicalUnit1]
Comment "ZIL devices"	Comment "ZIL devices"
TargetName target0	TargetName target0
TargetAlias "ZIL Devices"	TargetAlias "ZIL Devices"
Mapping PortalGroup1 InitiatorGroup1	Mapping PortalGroup1 InitiatorGroup1
AuthMethod None	AuthMethod None
UseDigest Auto	UseDigest Auto
UnitType Disk	UnitType Disk
LUN0 Storage /dev/md0 Auto	LUN0 Storage /dev/md0 Auto
LUN1 Storage /dev/md1 Auto	LUN1 Storage /dev/md1 Auto

Build mirror:

ctrl-a	**ctrl-b**
iscsictl -A -p 192.168.11.10 -t iqn.2016-01.local.beast.private:target0	iscsictl -A -p 192.168.11.11 -t iqn.2016-01.local.beast.private:target0
# da2 – ctrl-a ZIL from ctrl-a	# da2 – ctrl-a ZIL from ctrl-b
# da3 – ctrl-b ZIL from ctrl-a	# da3 – ctrl-b ZIL from ctrl-b
iscsictl -A -p 192.168.12.11 -t iqn.2016-01.local.beast.private:target0	iscsictl -A -p 192.168.12.10 -t iqn.2016-01.local.beast.private:target0
# da4 – ctrl-a ZIL from ctrl-b	# da4 – ctrl-a ZIL from ctrl-a
# da5 – ctrl-b ZIL from ctrl-b	# da5 – ctrl-b ZIL from ctrl-a

`gmirror label zil-a /dev/da4 /dev/da2`	`gmirror label zil-a /dev/da4 /dev/da2`
`gmirror label zil-b /dev/da5 /dev/da3`	`gmirror label zil-b /dev/da5 /dev/da3`

If needed, the similar definitions must be added for L2ARC configuration.