

Simple quorum drive for the FreeBSD CTL HA and the BeaST storage system

Mikhail Zakharov zmey20000@yahoo.com

Preface

During our experiments on developing the [BeaST storage system](#) we have faced the lack of automatic LUN failover function of the CTL HA subsystem. Yes, we can switch LUNs with CTL HA, but we have to do it manually setting “Primary role” to the alive controller:

```
sysctl kern.camctl.ha_role=0
```

Also we have to do it fast enough, otherwise a client host may lose access to the drives of the storage system.

This issue can be solved if we involve an existing HA clustering software like [Pacemaker/Corosync](#) or [Heartbeat](#). Although both of these remarkable projects have Linux roots, you can find their skeletons in FreeBSD port collection (report if you succeed in installing and using [them](#)).

Currently for the BeaST project we do not need Linux compatibility or any fancy clustering abilities, so I have spent a couple of weeks writing a simple quorum device solution for our experiments with the BeaST, FreeBSD and CTL HA. The result of my work is the Beast Quorum (BQ) version 1.0 which you can [download](#) from Sourceforge site.

In order to test the Beast Quorum, I am going to use quite the [same environment](#) I have installed earlier for the BeaST storage and CTL HA experiments. We will run two storage controllers (ctrl-a, ctrl-b) and a client host (cln-1). But now we have to prepare two virtual SAS drives (da0 and da1), which are configured as “shareable” in Virtual Media Manager and simultaneously connected with both storage controllers. The da0 drive is for storing data, and da1 is for shared quorum. Actually, BeaST quorum device can be a shared drive, a slice or a partition of almost any size, recognizable by FreeBSD.

These IP addresses are assigned to the hosts:

Host	Private network	Public network
ctrl-a	192.168.10.101	192.168.20.101
ctrl-b	192.168.10.102	192.168.20.102
cln-1	-	192.168.20.103

We are not going to share quorum drive with clients so there is no need to include corresponding definitions into iSCSI configuration and /etc/ctld.conf:

ctrl-a	ctrl-b
portal-group pg0 { discovery-auth-group no-authentication	portal-group pg0 { discovery-auth-group no-authentication

<pre>listen 192.168.10.101 } target iqn.2016-01.local.beast:target0 { auth-group no-authentication portal-group pg0 lun 1 { path /dev/da0 } }</pre>	<pre>listen 192.168.10.102 } target iqn.2016-01.local.beast:target0 { auth-group no-authentication portal-group pg0 lun 1 { path /dev/da0 } }</pre>
--	--

Installation and configuration

Today BQ is in the early development stage, it supports only basic functions, which are essential to perform failover operations on storage controllers (nodes). Also there is no FreeBSD port for it, or even decent Makefile with “install” section yet. Obviously we have to perform most of the installation tasks by hands, luckily there are not many things to do.

So lets boot storage controllers (ctrl-a, ctrl-b) and run these commands on both of them to download BQ source code and compile it:

```
beast@ctrl-a:~/ % fetch http://netcologne.dl.sourceforge.net/project/bqorum/bq-1.0.tgz
bq-1.0.tgz                               100% of 6184  B   37 MBps 00m00s
beast@ctrl-a:~/ % tar zxvf bq-1.0.tgz
x bq-1.0/
x bq-1.0/bq.c
x bq-1.0/bq.h
x bq-1.0/bq.trigger.n0
x bq-1.0/bq.trigger.n1
x bq-1.0/README
x bq-1.0/Makefile
beast@ctrl-a:~/ % cd bq-1.0
beast@ctrl-a:~/bq-1.0 % make
cc -O2 -pipe -fstack-protector -fno-strict-aliasing -Wall -o bq bq.c
```

As everything has come off well, we can install quorum device. Therefore, gain root access and run:

```
root@ctrl-a:/home/beast/bq-1.0 # ./bq -I -d /dev/da1 -f 1 -t 10
BeaST Quorum is installed.
```

Where:

- I means “install” command;
- d specifies shared device;
- f is heartbeat frequency, default is 1 second;
- t heartbeat timeout in seconds. Default is 10 seconds.

Now we have configured BQ nodes to send heartbeats every second. Each node monitors these heartbeats and if the other node keeps silence within the period of 10 seconds, it means that the node is dead.

After quorum installation we can check BQ shared drive configuration, and -L key is used for this operation:

```
root@ctrl-a:/home/beast/bq-1.0 # ./bq -L -d /dev/da1
BQuorum Label:      BQ
BQuorum Version:    32
Heartbeat Frequency: 1
Heartbeat Timeout:  10
Node 0 Current State: 0
Node 1 Current State: 0
Node 0 Timestamp block: 1
Node 1 Timestamp block: 2
Node 0 Timestamp value: 0
Node 1 Timestamp value: 0
```

As we can see BQ header is installed but both nodes are currently offline: Node States – 0 and Timestamp values (heartbeats counters) – 0.

The command above reads the BeAST header structure from the shared drive, we get the same picture from the ctrl-b:

```
root@ctrl-b:/home/beast/bq-1.0 # ./bq -L -d /dev/da1
BQuorum Label:      BQ
BQuorum Version:    32
Heartbeat Frequency: 1
Heartbeat Timeout:  10
Node 0 Current State: 0
Node 1 Current State: 0
Node 0 Timestamp block: 1
Node 1 Timestamp block: 2
Node 0 Timestamp value: 0
Node 1 Timestamp value: 0
```

The same information from both controllers is a sign that the quorum header is installed correctly and both nodes are ready to run BQ operations. Now lets start BQ on both controllers in daemon mode:

```
root@ctrl-a:/home/beast/bq-1.0 # ./bq -S -d /dev/da1 -n 0 -s /home/beast/bq-1.0/bq.trigger.n0 -l /var/log/bq.log
```

```
root@ctrl-b:/home/beast/bq-1.0 # ./bq -S -d /dev/da1 -n 1 -s /home/beast/bq-1.0/bq.trigger.n1 -l /var/log/bq.log
```

Where:

- S mean “start” quorum operations;
- n Current node id 0 or 1;
- s command (trigger) to run on node alive/dead event;
- l path to the BQ log file.

The most interesting key here is -s, which specifies a trigger – a command or a script to run when BQ detects a change in the state of any node:

Currently BQ on each node recognizes three states:

Node	State	State number
------	-------	--------------

Current (this) node	alive	1
Other (that) node	dead	2
Other (that) node	alive	3

On these states BQ runs trigger command with three parameters: “reporting node id”, “node, which caused the trigger to run”, and “the new state”.

For example, in our case BQ can start a trigger like this:

```
/home/beast/bq-1.0/bq.trigger.n0 0 1 dead
```

And it means: Node 0 reports that Node 1 is now Dead.

For our tests I have written simple Bourne shell scripts (they are also included into BQ sources) to handle two main states, 1 and 2.

So we have two similar scripts bq.trigger.n0 and bq.trigger.n1 for each node. And the only difference between them is in the IP addresses of the nodes:

```
root@ctrl-a:/home/beast/bq-1.0 # diff bq.trigger.n0 bq.trigger.n1
39,40c39,40
< this_node="192.168.10.101:7940"
< that_node="192.168.10.102:7940"
---
> this_node="192.168.10.102:7940"
> that_node="192.168.10.101:7940"
```

These trigger scripts handle node failover operations manipulating ctld and setting proper kern.camctl.ha_role values on the controllers.

After both nodes have been started, lets once again check BQ status with -L key:

```
root@ctrl-a:/home/beast/bq-1.0 # ./bq -L -d /dev/da1
BQuorum Label:          BQ
BQuorum Version:       32
Heartbeat Frequency:    1
Heartbeat Timeout:     10
Node 0 Current State:   1
Node 1 Current State:   1
Node 0 Timestamp block: 1
Node 1 Timestamp block: 2
Node 0 Timestamp value: 1480787008
Node 1 Timestamp value: 1480787008
```

Now both nodes are alive (Node Current States) – 1, and the timestamps are updated every second.

So everything works for now and we can proceed with the client configuration. Lets start our cln-1 client and connect it with remote iSCSI drives:

```
root@cln-1:/ # sysctl kern.iscsi.fail_on_disconnection=1
root@cln-1:/ # iscsictl -A -p 192.168.20.101 -t iqn.2016-01.local.beast:target0
root@cln-1:/ # iscsictl -A -p 192.168.20.102 -t iqn.2016-01.local.beast:target0
root@cln-1:/ # gmultipath label -A HA /dev/da0 /dev/da1
root@cln-1:/ # newfs /dev/multipath/HA
```

```
root@cln-1:/ # mount /dev/multipath/HA /mnt
```

Most of the commands on the client were discussed in the previous articles, but there is one exception: `gmultipath` with “label” command writes multipathing metadata to the drives. It allows multipath to restore paths when drives are getting back.

Failover functional tests

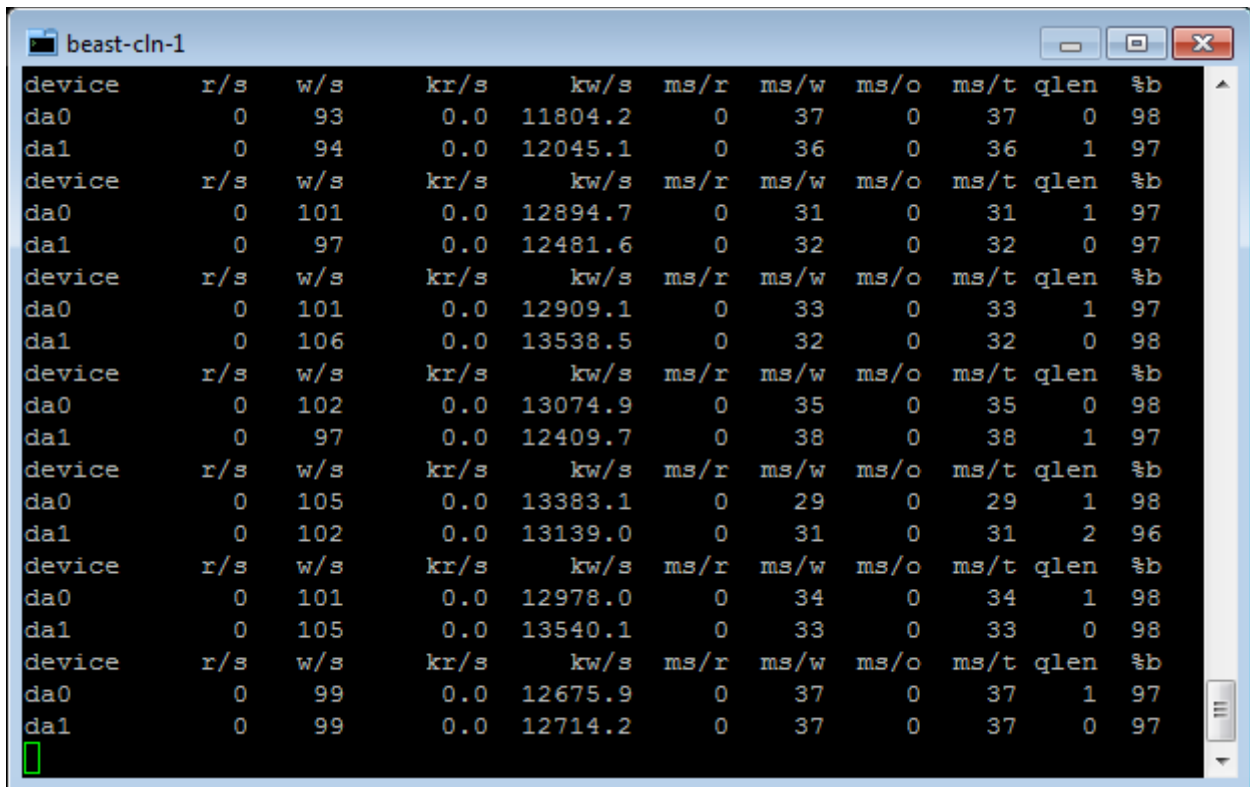
Like in previous tests we can push the whole construction to work by continuously copying a file:

```
# while true; do cp ports.tar.gz /mnt; done
```

Then start gathering statistics from the client and nodes:

```
root@cln-1:/ # iostat -xd 5 | grep '^d'
root@ctrl-a:/ # iostat -xd 5 | egrep '^dev|^da0|^cb'
root@ctrl-b:/ # iostat -xd 5 | egrep '^dev|^da0|^cb'
```

So all paths and a physical drive as well are utilized. See the appropriate screenshots below.



```
beast-cln-1
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0   93   0.0 11804.2  0    37    0    37   0  98
da1     0   94   0.0 12045.1  0    36    0    36   1  97
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0  101   0.0 12894.7  0    31    0    31   1  97
da1     0   97   0.0 12481.6  0    32    0    32   0  97
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0  101   0.0 12909.1  0    33    0    33   1  97
da1     0  106   0.0 13538.5  0    32    0    32   0  98
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0  102   0.0 13074.9  0    35    0    35   0  98
da1     0   97   0.0 12409.7  0    38    0    38   1  97
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0  105   0.0 13383.1  0    29    0    29   1  98
da1     0  102   0.0 13139.0  0    31    0    31   2  96
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0  101   0.0 12978.0  0    34    0    34   1  98
da1     0  105   0.0 13540.1  0    33    0    33   0  98
device  r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0     0   99   0.0 12675.9  0    37    0    37   1  97
da1     0   99   0.0 12714.2  0    37    0    37   0  97
```

```

beast-ctrl-a
da0      0  105    0.0 13391.2    0    6    0    6    0  30
cbb0     0  105    0.0 13391.2    0    6    0    6    0  31
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  105    0.0 13467.4    0    2    0    2    1  24
cbb0     0  105    0.0 13467.4    0    2    0    2    1  24
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  102    0.0 13035.9    0    3    0    3    0  27
cbb0     0  102    0.0 13035.9    0    3    0    3    0  27
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  102    0.0 13021.5    0    2    0    2    0  26
cbb0     0  102    0.0 13021.5    0    2    0    2    0  26
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  102    0.0 12966.7    0    3    0    3    0  28
cbb0     0  102    0.0 12966.7    0    3    0    3    0  28
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  105    0.0 13417.9    0    2    0    2    0  26
cbb0     0  105    0.0 13417.9    0    2    0    2    0  26
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  106    0.0 13604.3    0    2    0    2    1  24
cbb0     0  106    0.0 13604.3    0    2    0    2    1  25

```

```

beast-ctrl-b
cbb0     0  100    0.0 12802.4    0    5    0    5    1  28
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0   93    0.0 11952.7    0    9    0    9    0  32
cbb0     0   93    0.0 11952.7    0    9    0    9    0  32
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  104    0.0 13417.8    0    3    0    3    1  22
cbb0     0  104    0.0 13417.8    0    3    0    3    1  23
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  100    0.0 12777.7    0    6    0    6    0  29
cbb0     0  100    0.0 12777.7    0    6    0    6    0  30
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0   98    0.0 12559.1    0    2    0    2    0  21
cbb0     0   98    0.0 12559.1    0    2    0    2    0  21
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0   97    0.0 12456.5    0    4    0    4    0  24
cbb0     0   97    0.0 12456.5    0    4    0    4    0  25
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0   94    0.0 12067.7    0    4    0    4    0  26
cbb0     0   94    0.0 12067.7    0    4    0    4    0  26
device   r/s  w/s    kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0   97    0.0 12501.1    0    2    0    2    0  21
cbb0     0   97    0.0 12501.1    0    2    0    2    0  21

```

As the traffic is going through both controllers, lets find the “Primary” CTL HA node:

```

root@ctrl-a:/ # sysctl kern.camctl.ha_role
kern.camctl.ha_role: 0
root@ctrl-b:/ # sysctl kern.camctl.ha_role
kern.camctl.ha_role: 1

```

We can see the “Primary” CTL HA node is ctrl-a. We remember that bare CTL HA needs user interaction to change HA role and make controller failover. Lets reboot ctrl-a controller and see if the Beast Quorum can automatically do the job and failover to ctrl-b:

```
root@ctrl-a:/ # reboot
```

Below there are screenshots from cln-1 and ctrl-b which I have captured just a few second after ctrl-a disappearance.

As we can see, the cln-1 client lost access to the ctrl-a drive (da0), and the traffic stopped on (da1) but when BQ turned on CTL HA “Primary” role on the ctrl-b, the traffic began to run again:

```

beast-cln-1
da0      0   97   0.0 12325.3   0   36   0   36   0  97
da1      0   91   0.0 11640.8   0   40   0   40   1  97
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  102   0.0 13098.2   0   32   0   32   2  98
da1      0  101   0.0 13047.0   0   31   0   31   7  97
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  109   0.0 13958.1   0   32   0   32   0  97
da1      0  102   0.0 13002.1   0   36   0   36   1  98
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1     -2147483648 -2147483648 10.1 3575663633939611.9 0 0
0        0   16   66
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1      0   0   0.0   0.0    0    0    0    0   16   0
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1      0  30   0.0  3962.4   0 1098   0 1098   3 218
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1      0 210   0.0 26767.5   0   35   0   35   4  99
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1      0 196   0.0 25112.2   0   45   0   45   5  99
device   r/s  w/s   kr/s   kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1      0 189   0.0 24078.6   0   47   0   47  16  99

```

The corresponding picture from ctrl-b:

```

beast-ctrl-b
cbb0      0  97  0.0 12426.1  0  6  0  6  0  29
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0 106  0.0 13529.2  0  2  0  2  0  21
cbb0      0 106  0.0 13529.2  0  2  0  2  0  21
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  91  0.0 11659.5  0  6  0  6  0  28
cbb0      0  91  0.0 11659.5  0  6  0  6  0  29
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0 101  0.0 12978.1  0  4  0  4  7  26
cbb0      0 101  0.0 12978.1  0  4  0  4  7  26
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0 102  0.0 13058.3  0  3  0  3  1  23
cbb0      0 102  0.0 13058.3  0  3  0  3  1  23
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  82  0.0 10476.7  0  3  0  3  0  19
cbb0      0  82  0.0 10476.7  0  3  0  3  0  20
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  0  0.0  0.0  0  0  0  0  0  0
cbb0      0  0  0.0  0.0  0  0  0  0  0  0
device   r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0      0  34  0.0  4425.3  0  4  0  4  1  10
cbb0      0  34  0.0  4425.3  0  5  0  5  1  10

```

We can state, BQ correctly detects controllers death and automatically switches node roles.

Now lets test what happens when the node comes back.

Therefore, boot the ctrl-a again and start BQ on it:

```

root@ctrl-a:/home/beast/bq-1.0 # ./bq -S -d /dev/da1 -n 0 -s /home/beast/bq-1.0/bq.trigger.n0 -l /var/log/bq.log

```

And the BeAST Quorum has done everything right to put the node online. We can see that da0 drive on the ctrl-a begins to receive traffic:


```

beast-ctrl-a
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0    0    0.0   0.0   0     0     0     0     0     0
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    1    93   1.0  11915.2  2     6     0     6     0    27
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da0    0   100   0.0  12794.6  0     5     0     5     0    28

```

Multipathing on the cln-1 client also works well and we can see da0 is online again:

```

beast-cln-1
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  109   0.0  13975.4  0    29     0    29     1   96
da0    0  104   0.0  13271.6  0    32     0    32     0   97
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  104   0.0  13288.7  0    33     0    33     3   98
da0    0  105   0.0  13492.5  0    32     0    32     1   97
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  107   0.0  13695.6  0    32     0    32     1   97
da0    0  103   0.0  13221.5  0    35     0    35     2   98
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  104   0.0  13284.6  0    32     0    32     0   97
da0    0  107   0.0  13725.7  0    32     0    32     1   97
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  107   0.0  13731.2  0    33     0    33     1   97
da0    0  101   0.0  12948.8  0    36     0    36     0   97
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0   88   0.0  11245.4  0    40     0    40     2   97
da0    0   90   0.0  11603.2  0    40     0    40     0   98
device  r/s  w/s  kr/s  kw/s  ms/r  ms/w  ms/o  ms/t  qlen  %b
da1    0  107   0.0  13657.3  0    31     0    31     1   98
da0    0  100   0.0  12826.1  0    33     0    33     0   97

```

Conclusions

Yes, it works! The BeaST Quorum successfully does the job for automatic CTL HA controller/node failover operations. Without BQ we have to do it manually, that is nonsense for the reliable storage systems.

But there are lot of things to do to improve BQ. For example, we must take care of reliable mechanisms of restarting BQ daemon if it has been occasionally killed while controller continues to run; add failover support for individual LUNs; improve heartbeat processing and implement sliding redundant heartbeat blocks; fix log procedures to prevent simultaneous writes into the same file; and much more including typographic errors.

Note the BeaST Quorum is currently in the early development stage! Use it for testing purposes only! Do not put it in production as you can lose your data!