First look at the renewed CTL High Availability implementation in FreeBSD

Version 1.1 20 Oct 2016

Mikhail Zakharov zmey20000@yahoo.com

This enhancement looks extremely important for the BeaST storage system as implementation of high available native ALUA in FreeBSD can potentially replace the BeaST arbitration mechanism (“Arbitrator”), which is completely described in the papers on the BeaST project page.

So lets see what has happened. According to the ctl(4) man page: “The ctl subsystem provides SCSI disk and processor emulation” and “serves as a kernel component of the native iSCSI target”. Among other features it has now reimplemented “High Availability clustering support with ALUA”. See source code revision 287621 by Alexander Motin for more details on the change. Actually this revision was done a year ago and the feature is available both in FreeBSD 11.0 and in 10.3 releases.

ALUA in storage world terminology means Asymmetric Logical Unit Assignment. In simple words this set of technologies allows a host to access any LUN via both controllers of a storage system. FreeBSD ctl(4) man-page claims all possible modes are now available:

```
“kern.cam.ctl.ha_mode
        Specifies High Availability cluster operation mode:
              0   Active/Standby -- primary node has backend access and
                  processes requests, while secondary can only do basic
                  LUN discovery and reservation;
              1   Active/Active -- both nodes have backend access and
                  process requests, while secondary node synchronizes
                  processing with primary one;
              2   Active/Active -- primary node has backend access and
                  processes requests, while secondary node forwards all
                  requests and data to primary one;”
```

Now lets see if it is reasonable to use this new HA storage functionality in the BeaST project.

As I still do not have any real hardware drive-enclosures, we will use Oracle Virtual Box and iSCSI protocol. I have already deployed this environment for the BeaST development, so we can use the similar, yet more simplified template for the renewed CTL HA testing purpose.

We will run two storage controllers (ctrl-a, ctrl-b) and a host (cln-1). A virtual SAS drive (da0) of 256 MB is configured as “shareable” in Virtual Media Manager and simultaneously connected with both storage controllers.

Two virtual LANs are configured:

- 192.168.10.0/24 is a private network for HA interconnect;
- 192.168.20.0/24 is for the public access to the front-end and LUNs.

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 will use the fresh FreeBSD 11.0 release:

```
% uname -a
FreeBSD ctrl-a 11.0-RELEASE-p1 FreeBSD 11.0-RELEASE-p1 #0 r306420: Thu Sep 29
01:43:23 UTC 2016     root@releng2.nyi.freebsd.org:/usr/obj/usr/src/sys/GENERIC
amd64
```

Before doing anything else, lets save CTL frontend ports states of both controllers to refer to them later during our experiments:

```
root@ctrl-a:/home/beast # ctladm portlist
Port Online Frontend Name     pp vp
0    YES    ioctl    ioctl    0  0
1    YES    tpc      tpc      0  0
2    NO     camsim   camsim   0  0  naa.50000006e879fb03

root@ctrl-b:/home/ beast # ctladm portlist
Port Online Frontend Name     pp vp
128  YES    ioctl    ioctl    0  0
129  YES    tpc      tpc      0  0
130  NO     camsim   camsim   0  0  naa.50000001f740eb83
```

Now we can start configuring controllers. First of all, add essential variables to the /boot/loader.conf, then reboot both controllers:

ctrl-a	ctrl-b

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

Where:

ctl_load="YES" loads the CTL driver as module.

kern.cam.ctl.ha_id – specifies the node ID (1 or 2) and 0 disables HA functionality.

kern.cam.ctl.ha_mode – sets operational mode. See the description of it at the beginning of the article. For our purposes we are interested in Active/Active modes only.

kern.cam.ctl.ha_role – configures default role for the node. So ctrl-a is set as 0 (primary node), ctrl-b – 1 (secondary node). The role also can be specified on per-LUN basis which allows to distribute LUNs over both controllers evenly.

kern.cam.ctl.iscsi.ping_timeout – is a number of seconds to wait for initiator’s response on NOP-In PDU which is issued by the target when the traffic is inactive. By default, the session is forcibly terminated after 5 second. But for testing purposes we disable it by specifying 0.

Note, kern.cam.ctl.ha_id and kern.cam.ctl.ha_mode are read-only parameters and must be set only via the /boot/loader.conf file. Other useful variables we can put in /etc/sysctl.conf but I choose only kern.cam.ctl.debug=1:

```
beast@ctrl-a:~ % grep kern.cam.ctl /etc/sysctl.conf
kern.cam.ctl.debug=1
#kern.cam.ctl.ha_peer="connect 192.168.10.102:7777"

beast@ctrl-b:~ % grep kern.cam.ctl /etc/sysctl.conf
kern.cam.ctl.debug=1
#kern.cam.ctl.ha_peer="listen 192.168.10.102:7777"
```

As you can see, there is a remark sign before kern.cam.ctl.ha_peer variable. It is done to prevent an attempt to start CTL HA connection at boot: at this point LAN interfaces are not up yet, so the connection will fail.

But we can lately start CTL HA interconnect manually from shell or by script:

ctrl-a	**ctrl-b**
`# sysctl kern.cam.ctl.ha_peer="connect 192.168.10.102:7777"`	`sysctl kern.cam.ctl.ha_peer="listen 192.168.10.102:7777"`

If everything is OK, we will see these messages in logs on both controllers:

```
root@ctrl-b:/home/beast # dmesg | tail -2
CTL: HA link status changed from 0 to 1
CTL: HA link status changed from 1 to 2
```

The link states can be: 0 – not configured, 1 – configured but not established and 2 – established. The link state information can be also checked via sysctl:

```
root@ctrl-a:/home/beast # sysctl kern.cam.ctl.ha_link
kern.cam.ctl.ha_link: 2
```

As we got link state 2, we can check what is going on the CTL frontend:

```
root@ctrl-a:/home/beast # ctladm portlist
Port Online Frontend Name     pp vp
0    YES    ioctl    ioctl    0  0
1    YES    tpc      tpc      0  0
2    NO     camsim   camsim   0  0  naa.50000006e879fb03
128  YES    ha       2:ioctl  0  0
129  YES    ha       2:tpc    0  0
130  NO     ha       2:camsim 0  0  naa.50000001f740eb83

root@ctrl-b:/home/beast #  ctladm portlist
Port Online Frontend Name     pp vp
0    YES    ha       1:ioctl  0  0
1    YES    ha       1:tpc    0  0
2    NO     ha       1:camsim 0  0  naa.50000006e879fb03
128  YES    ioctl    ioctl    0  0
129  YES    tpc      tpc      0  0
130  NO     camsim   camsim   0  0  naa.50000001f740eb83
```

As you can see HA interconnection is established. It means we can add some LUNs to use them on our client host. Therefore create simple /etc/ctl.conf to add appropriate definitions for our iSCSI targets:

ctrl-a	**ctrl-b**

`portal-group pg0 {`
`        discovery-auth-group no-authentication`
`        listen 192.168.20.101`
`}`
`target iqn.2016-01.local.beast:target0 {`
`        auth-group no-authentication`
`        portal-group pg0`
 	
`        lun 1 {`
`                path /dev/da0`
`        }`
`}`
	`portal-group pg0 {`
	`        discovery-auth-group no-authentication`
	`        listen 192.168.20.102`
	`}`
	`target iqn.2016-01.local.beast:target0 {`
	`        auth-group no-authentication`
	`        portal-group pg0`
	
	`        lun 1 {`
	`                path /dev/da0`
	`        }`
	`}`

Then start ctld on both controllers:

```
# service ctld onestart
```

Now check what is registered on ports:

```
root@ctrl-a:/home/beast # ctladm portlist
Port Online Frontend Name     pp vp
0    YES    ioctl    ioctl    0  0
1    YES    tpc      tpc      0  0
2    NO     camsim   camsim   0  0  naa.5000000d5de41b03
3    YES    iscsi    iscsi    257 1  iqn.2016-01.local.beast:target0,t,0x0101
128  YES    ha       2:ioctl  0  0
129  YES    ha       2:tpc    0  0
130  NO     ha       2:camsim 0  0  naa.500000091b30b383
131  YES    ha       2:iscsi  257 1  iqn.2016-01.local.beast:target0,t,0x0101

root@ctrl-b:/home/beast # ctladm portlist
Port Online Frontend Name     pp vp
0    YES    ha       1:ioctl  0  0
1    YES    ha       1:tpc    0  0
2    NO     ha       1:camsim 0  0  naa.5000000d5de41b03
3    YES    ha       1:iscsi  257 1  iqn.2016-01.local.beast:target0,t,0x0101
128  YES    ioctl    ioctl    0  0
129  YES    tpc      tpc      0  0
130  NO     camsim   camsim   0  0  naa.500000091b30b383
131  YES    iscsi    iscsi    257 1  iqn.2016-01.local.beast:target0,t,0x0101
```

As new LUNs are shown, everything is going well right now. So we can start the client host and establish iSCSI connection with our storage:

```
root@cln-1:/home/beast # service iscsid onestart
root@cln-1:/home/beast # sysctl kern.iscsi.fail_on_disconnection=1
root@cln-1:/home/beast # iscsictl -A -p 192.168.20.101 -t iqn.2016-01.local.beast:target0
root@cln-1:/home/beast # iscsictl -A -p 192.168.20.102 -t iqn.2016-01.local.beast:target0
```

Note, sysctl kern.iscsi.fail_on_disconnection=1 on the client is needed to drop connection with one of the controllers in case of its failure.

During this operations we can see log updates with activity on iSCSI LUNs:

On ctrl-a:

```
(0:3:1/0): MODE SENSE(6). CDB: 1a 00 0a 00 18 00  Tag: 0x4/0
(0:3:1/0): CTL Status: SCSI Error
(0:3:1/0): SCSI Status: Check Condition
(0:3:1/0): SCSI sense: UNIT ATTENTION asc:29,1 (Power on occurred)
```

On ctrl-b:

```
(0:131:1/0): MODE SENSE(6). CDB: 1a 00 0a 00 18 00  Tag: 0x4/0
(0:131:1/0): CTL Status: SCSI Error
(0:131:1/0): SCSI Status: Check Condition
(0:131:1/0): SCSI sense: UNIT ATTENTION asc:29,1 (Power on occurred)
```

And on cln-1:

```
da0 at iscsi1 bus 0 scbus3 target 0 lun 1
da0: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da0: Serial Number MYSERIAL   0
da0: 150.000MB/s transfers
da0: Command Queueing enabled
da0: 256MB (524288 512 byte sectors)
da1 at iscsi2 bus 0 scbus4 target 0 lun 1
da1: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da1: Serial Number MYSERIAL   0
da1: 150.000MB/s transfers
da1: Command Queueing enabled
da1: 256MB (524288 512 byte sectors)
```

So we can state that the client has reached both LUNs (actually the client has accessed the same physical drive through connections with two different controllers).

As we know that da0 and da1 on the client are the same drive, we can put them under multipathing control:

```
root@cln-1:/home/beast #  gmultipath create -A HA /dev/da0 /dev/da1
```

Note, option -A enables Active/Active mode of multipathing, so the workload will be distributed over both paths and controllers.

Check if we succeeded:

```
root@cln-1:/home/beast # gmultipath status
        Name   Status  Components
multipath/HA  OPTIMAL  da0 (ACTIVE)
                       da1 (ACTIVE)
```

Well, now lets create a new filesystem and mount it:

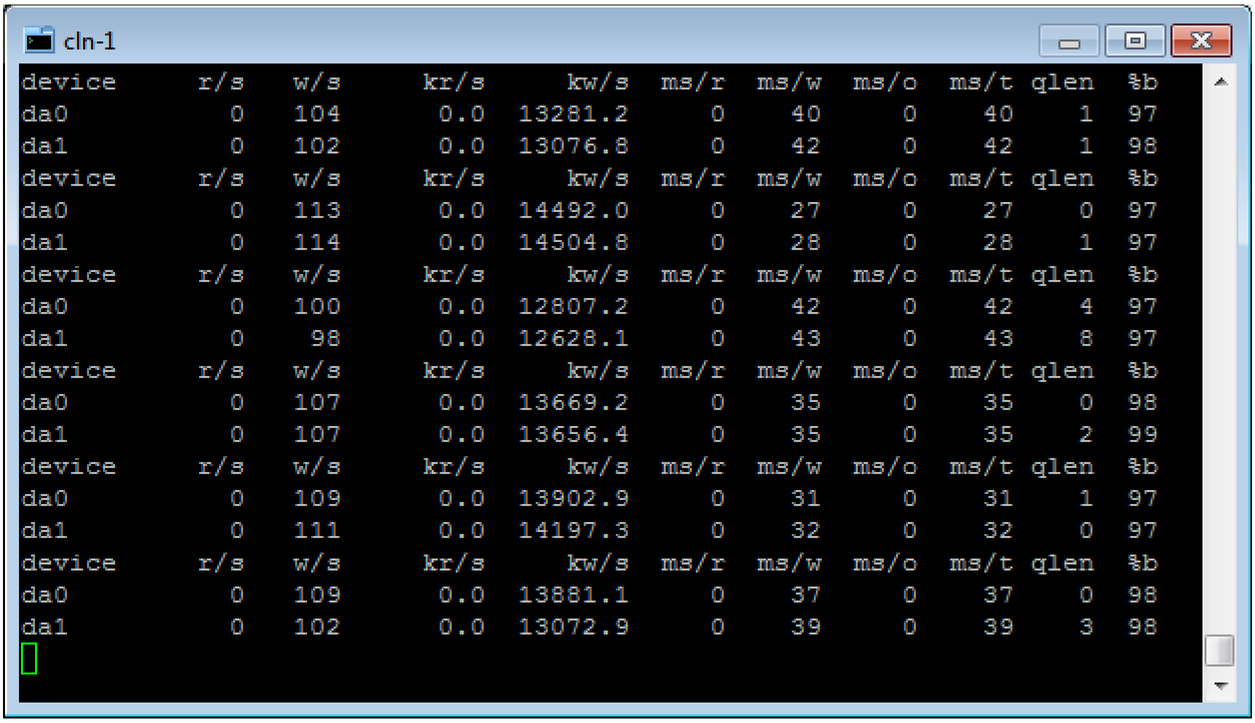
```
root@cln-1:/home/beast # newfs /dev/multipath/HA
root@cln-1:/home/beast # mount /dev/multipath/HA /mnt
```

Now we can force the construction to work, so lets continuously copy a file:

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

And check the results from the client side:

```
root@cln-1:/home/beast # iostat -xd 5 | grep '^d'
```



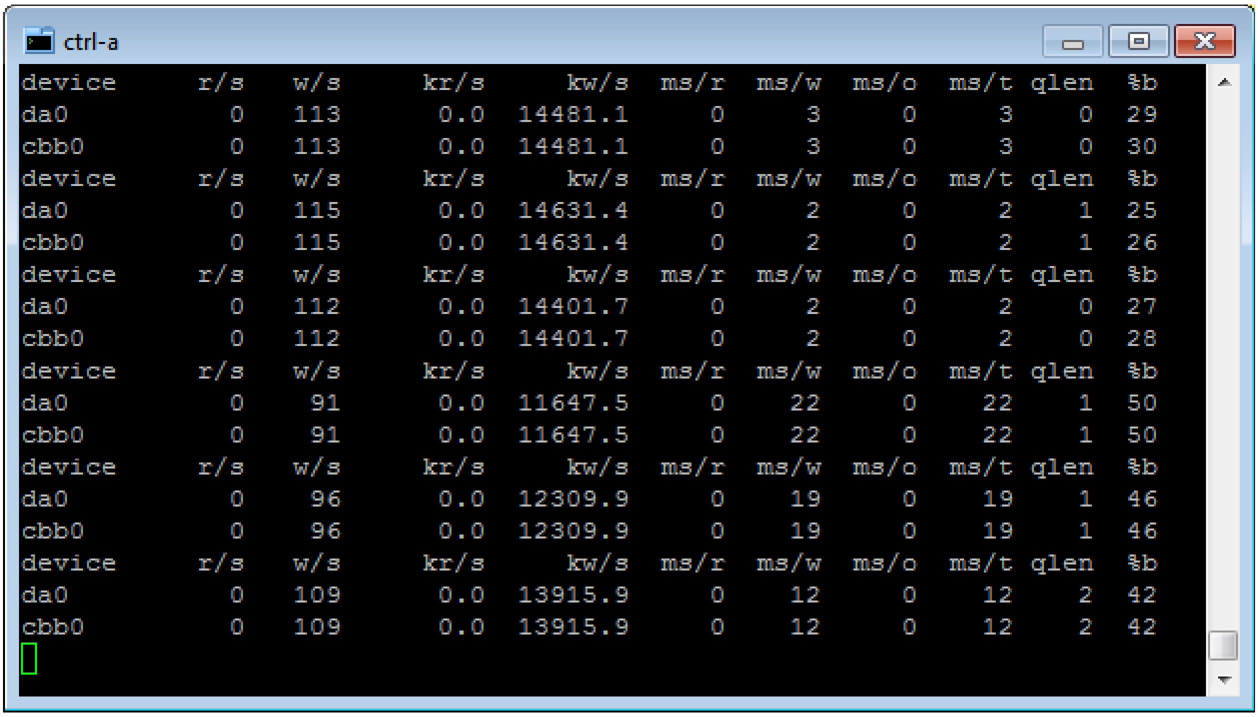
```
device     r/s   w/s    kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   104     0.0  13281.2     0    40     0    40    1  97
da1          0   102     0.0  13076.8     0    42     0    42    1  98
device     r/s   w/s    kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   113     0.0  14492.0     0    27     0    27    0  97
da1          0   114     0.0  14504.8     0    28     0    28    1  97
device     r/s   w/s    kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   100     0.0  12807.2     0    42     0    42    4  97
da1          0    98     0.0  12628.1     0    43     0    43    8  97
device     r/s   w/s    kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   107     0.0  13669.2     0    35     0    35    0  98
da1          0   107     0.0  13656.4     0    35     0    35    2  99
device     r/s   w/s    kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   109     0.0  13902.9     0    31     0    31    1  97
da1          0   111     0.0  14197.3     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   109     0.0  13881.1     0    37     0    37    0  98
da1          0   102     0.0  13072.9     0    39     0    39    3  98
```

According to the Active/Active multipathing mode both devices are working.

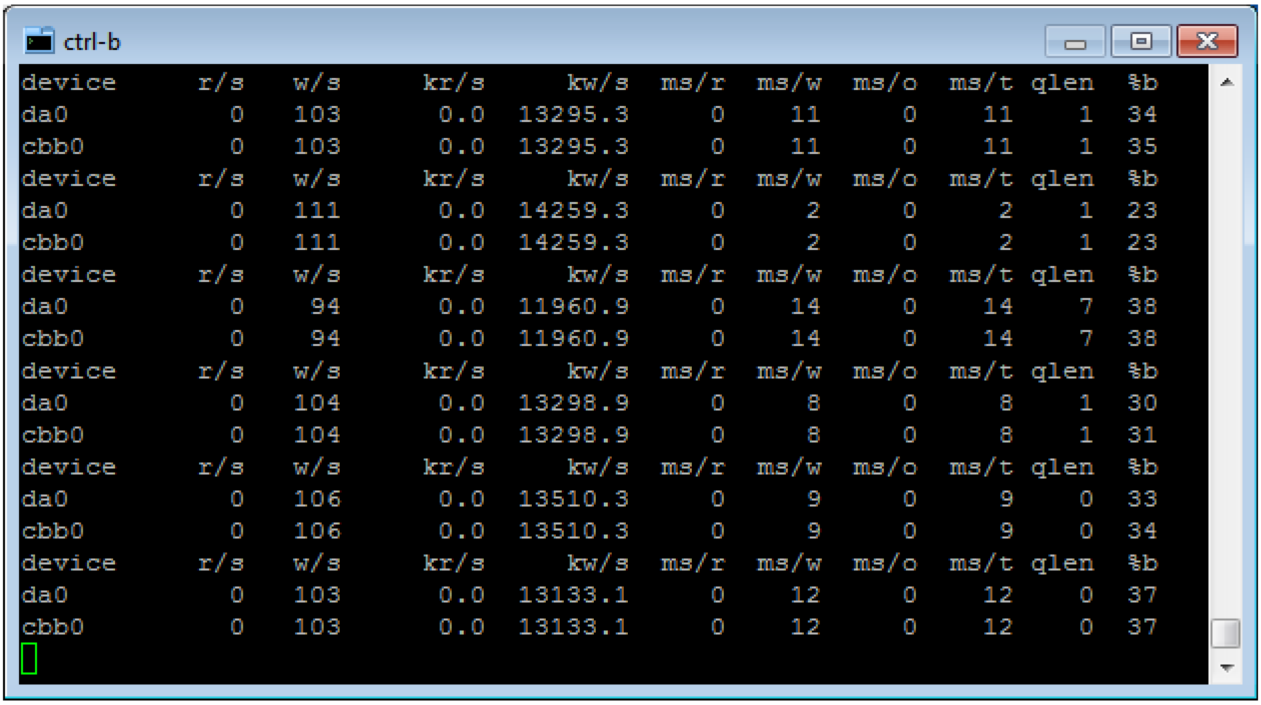
Now lets check what is going on storage controllers (ctrl-a and ctrl-b):

```
# iostat -xd 5 | egrep '^dev|^da0|^cb'
```

As we have set full Active/Active CTL HA mode (kern.cam.ctl.ha_mode=1) for the cluster, we can see the similar picture on both controllers:



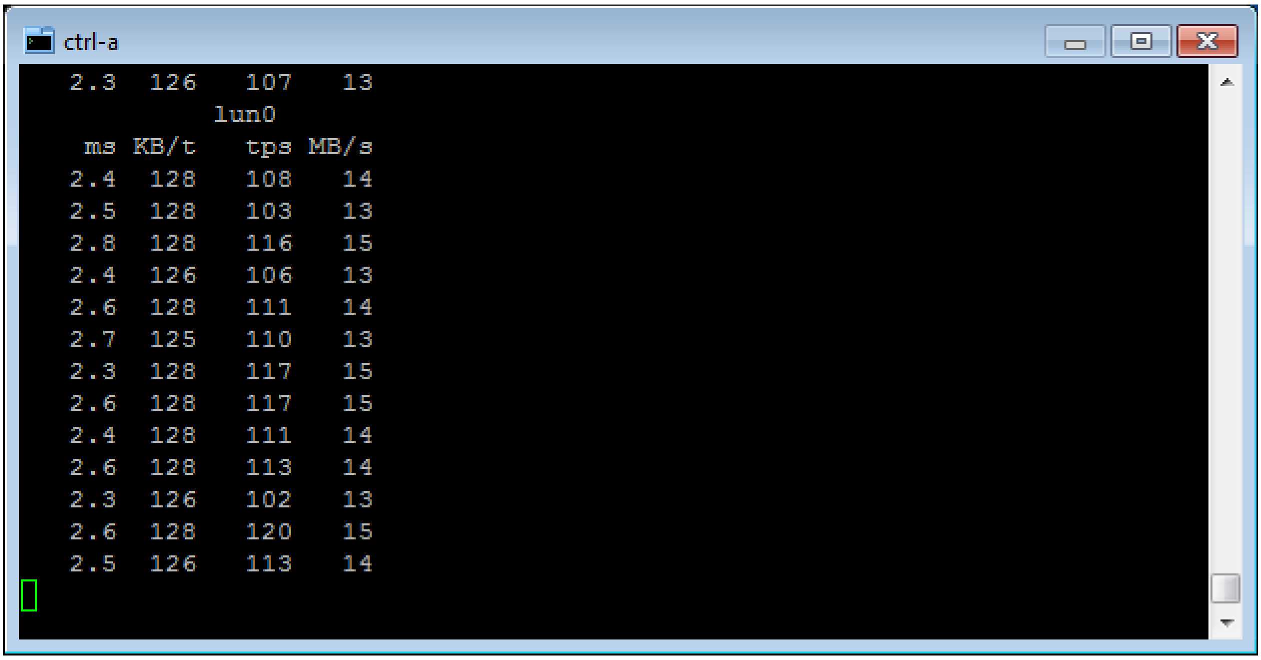
```
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   113      0.0   14481.1     0     3     0     3    0  29
cbb0           0   113      0.0   14481.1     0     3     0     3    0  30
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   115      0.0   14631.4     0     2     0     2    1  25
cbb0           0   115      0.0   14631.4     0     2     0     2    1  26
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   112      0.0   14401.7     0     2     0     2    0  27
cbb0           0   112      0.0   14401.7     0     2     0     2    0  28
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0    91      0.0   11647.5     0    22     0    22    1  50
cbb0           0    91      0.0   11647.5     0    22     0    22    1  50
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0    96      0.0   12309.9     0    19     0    19    1  46
cbb0           0    96      0.0   12309.9     0    19     0    19    1  46
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   109      0.0   13915.9     0    12     0    12    2  42
cbb0           0   109      0.0   13915.9     0    12     0    12    2  42
```



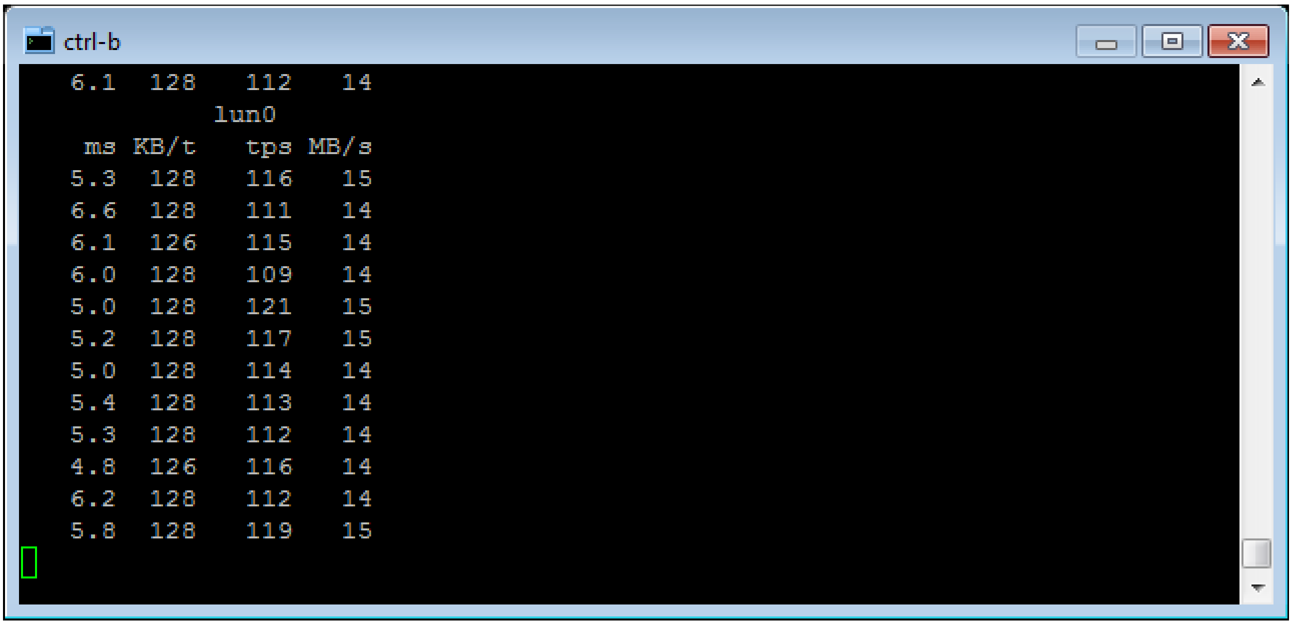
```
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   103      0.0   13295.3     0    11     0    11    1  34
cbb0           0   103      0.0   13295.3     0    11     0    11    1  35
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   111      0.0   14259.3     0     2     0     2    1  23
cbb0           0   111      0.0   14259.3     0     2     0     2    1  23
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0    94      0.0   11960.9     0    14     0    14    7  38
cbb0           0    94      0.0   11960.9     0    14     0    14    7  38
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   104      0.0   13298.9     0     8     0     8    1  30
cbb0           0   104      0.0   13298.9     0     8     0     8    1  31
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   106      0.0   13510.3     0     9     0     9    0  33
cbb0           0   106      0.0   13510.3     0     9     0     9    0  34
device       r/s   w/s     kr/s      kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0            0   103      0.0   13133.1     0    12     0    12    0  37
cbb0           0   103      0.0   13133.1     0    12     0    12    0  37
```

Now lets see CTL LUN statistics on both controllers. So run:

```
# ctlstat -C
```



```
   2.3  126   107   13
            lun0
    ms  KB/t   tps  MB/s
   2.4  128   108   14
   2.5  128   103   13
   2.8  128   116   15
   2.4  126   106   13
   2.6  128   111   14
   2.7  125   110   13
   2.3  128   117   15
   2.6  128   117   15
   2.4  128   111   14
   2.6  128   113   14
   2.3  126   102   13
   2.6  128   120   15
   2.5  126   113   14
```



```
   6.1  128   112   14
            lun0
    ms  KB/t   tps  MB/s
   5.3  128   116   15
   6.6  128   111   14
   6.1  126   115   14
   6.0  128   109   14
   5.0  128   121   15
   5.2  128   117   15
   5.0  128   114   14
   5.4  128   113   14
   5.3  128   112   14
   4.8  126   116   14
   6.2  128   112   14
   5.8  128   119   15
```

And it seems, the second node for the LUN responds slower (two times slower in our example).

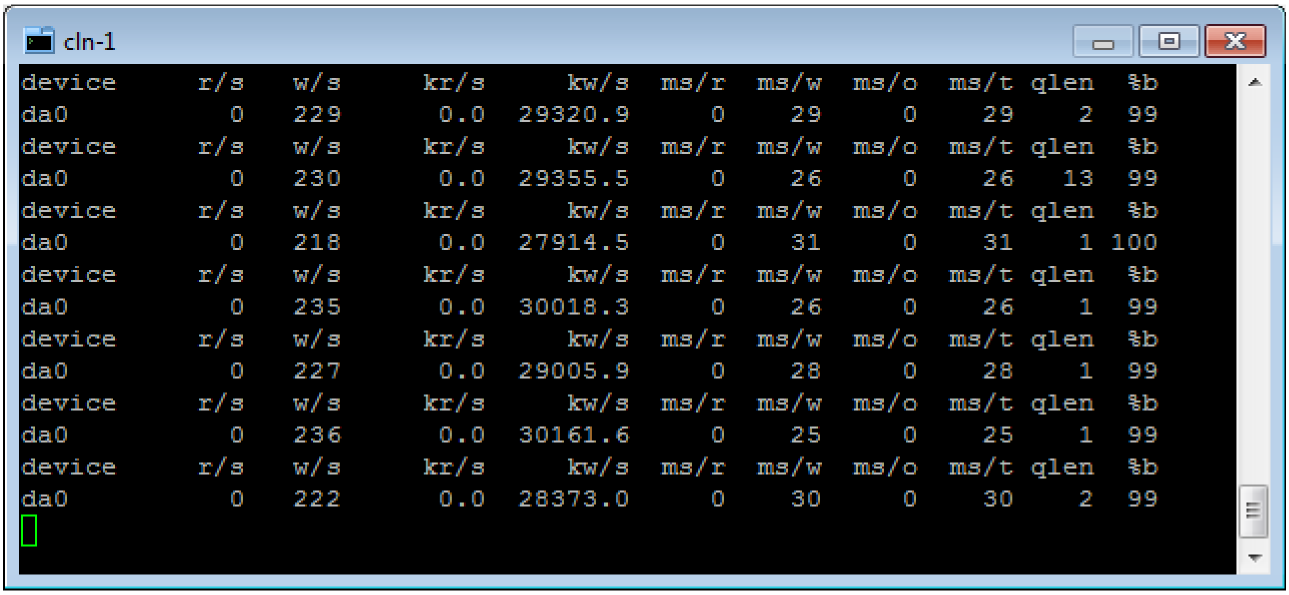
Remember that kern.cam.ctl.ha_mode=2 enables Active/Active frontend, while secondary node forwards all requests and data to primary one. I do not show screenshots, so you have to believe me, but if we set this mode, the overall picture will be similar except that iostat -xd 5 on secondary node shows zero disk activity and primary node processes all the workload.

Anyway the full Active/Active CTL HA configuration (kern.cam.ctl.ha_mode=1) runs well, utilizing all available paths through both controllers. Now it's time to test high availability of the new CTL HA subsystem.

I have no idea of how to fail backend virtual link between the drive and SAS controller in VirtualBox environment, but we can easily simulate a crash of a whole controller. So, lets just shutdown a secondary node (ctrl-b) and see if the cluster can survive it:

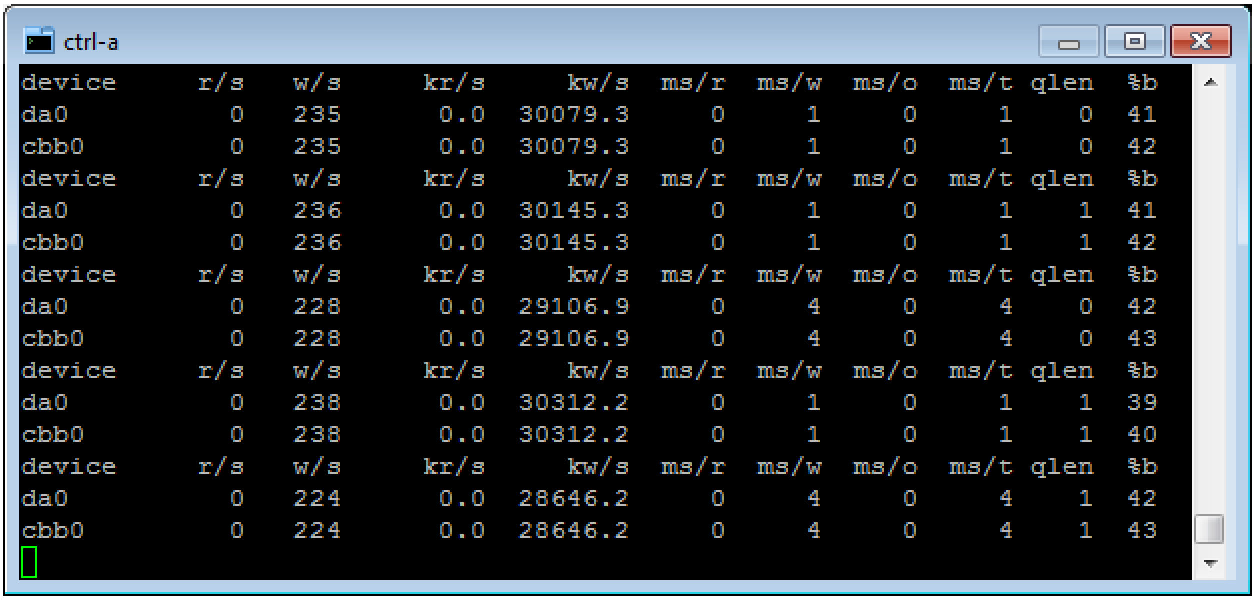
```
root@ctrl-b:/home/beast # shutdown -p now
```

And nothing bad has happened to the client. It has lost one iSCSI path (for the da1) but multipathing works well, it is now forwarding all the data through the primary node (ctrl-a):



```
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   229      0.0  29320.9     0    29     0    29    2  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   230      0.0  29355.5     0    26     0    26   13  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   218      0.0  27914.5     0    31     0    31    1 100
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   235      0.0  30018.3     0    26     0    26    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   227      0.0  29005.9     0    28     0    28    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   236      0.0  30161.6     0    25     0    25    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   222      0.0  28373.0     0    30     0    30    2  99
```

And on ctrl-a (primary node of CTL HA cluster) we can see that data is going to the da0:



```
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   235      0.0  30079.3     0     1     0     1    0  41
cbb0         0   235      0.0  30079.3     0     1     0     1    0  42
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   236      0.0  30145.3     0     1     0     1    1  41
cbb0         0   236      0.0  30145.3     0     1     0     1    1  42
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   228      0.0  29106.9     0     4     0     4    0  42
cbb0         0   228      0.0  29106.9     0     4     0     4    0  43
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   238      0.0  30312.2     0     1     0     1    1  39
cbb0         0   238      0.0  30312.2     0     1     0     1    1  40
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da0          0   224      0.0  28646.2     0     4     0     4    1  42
cbb0         0   224      0.0  28646.2     0     4     0     4    1  43
```

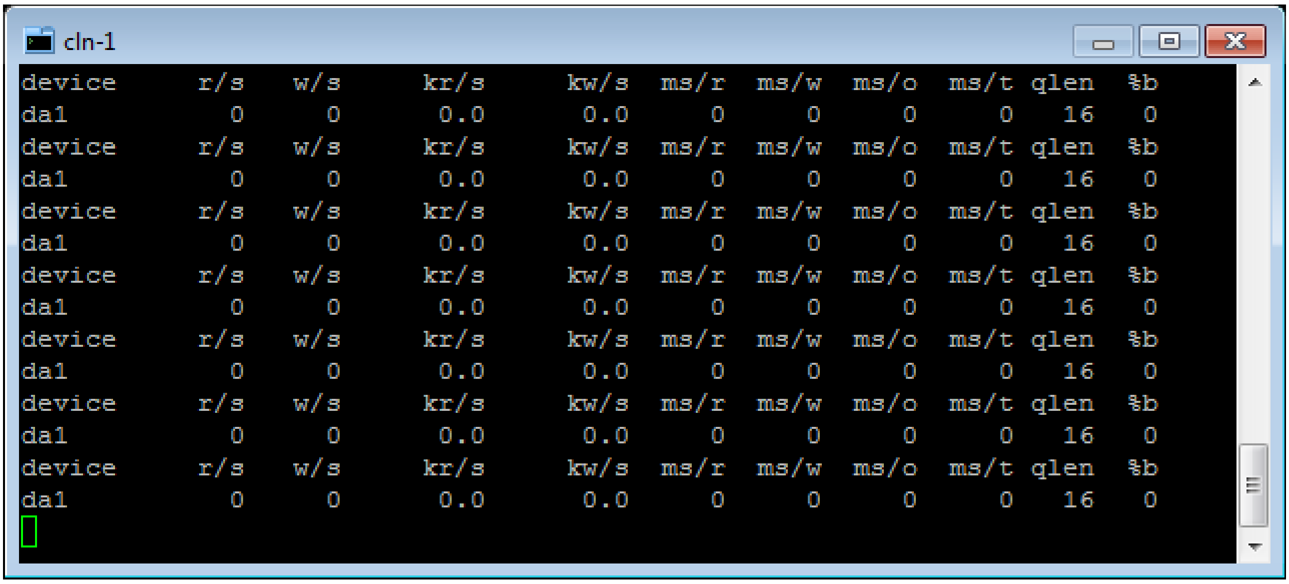
Our next steps will be to boot the ctrl-b controller, restore our CTL HA cluster and start copying ports.tar.gz file once again. But I’m not going to show it here to save time and space.

After restoring the cluster and starting iostat -xd 5 along with ctlstat -C we can finally crash the primary node (ctrl-a):

```
root@ctrl-a:/home/beast # shutdown -p now
```

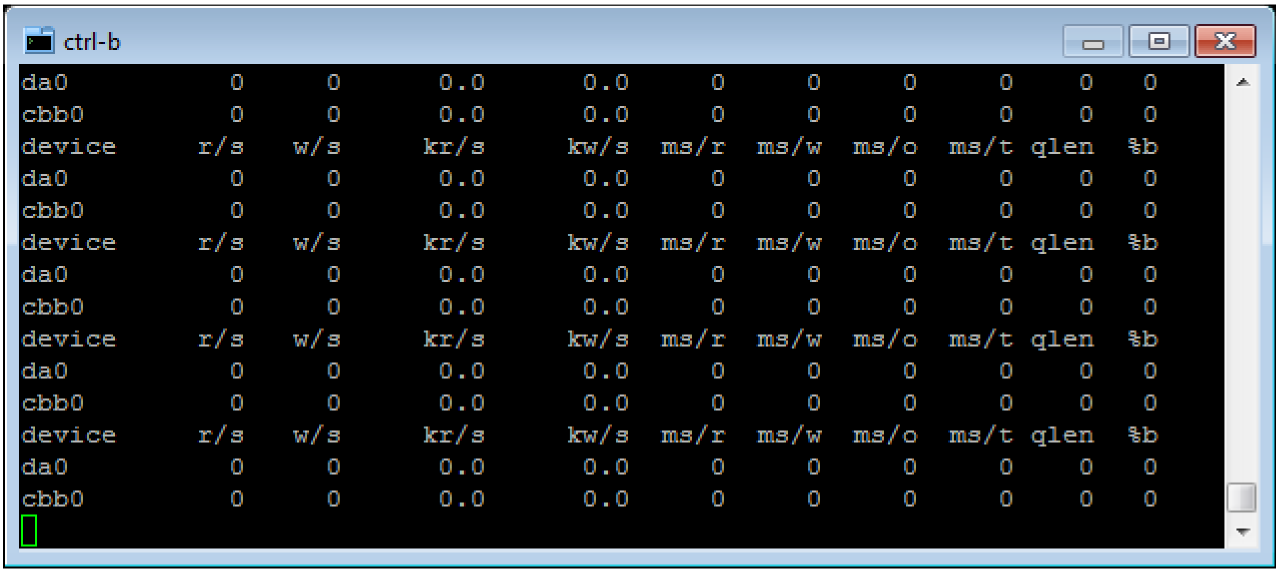
And the things turn really bad now.

First of all, we have an active path, but traffic has absolutely stopped on the client (cln-1):



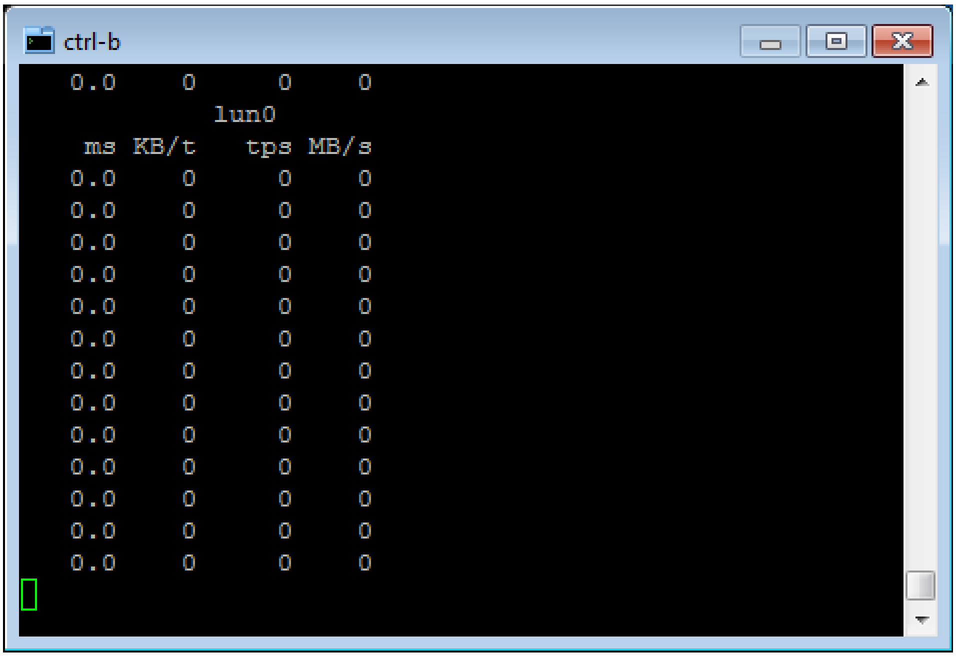
```
device       r/s    w/s      kr/s      kw/s   ms/r   ms/w   ms/o   ms/t qlen   %b
da1            0      0       0.0       0.0      0      0      0      0   16    0
device       r/s    w/s      kr/s      kw/s   ms/r   ms/w   ms/o   ms/t qlen   %b
da1            0      0       0.0       0.0      0      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      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      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      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      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      0       0.0       0.0      0      0      0      0   16    0
```

Output of iostat -xd5 on ctrl-b is also empty:



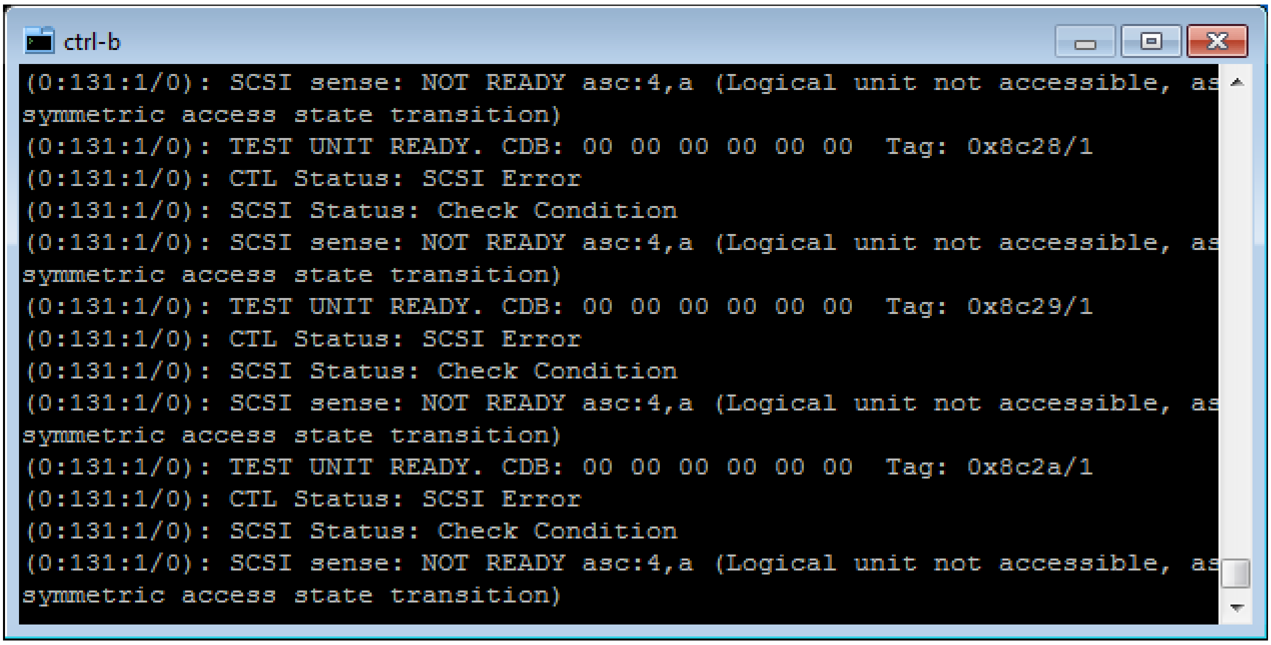
```
da0            0      0       0.0       0.0      0      0      0      0    0    0
cbb0           0      0       0.0       0.0      0      0      0      0    0    0
device       r/s    w/s      kr/s      kw/s   ms/r   ms/w   ms/o   ms/t qlen   %b
da0            0      0       0.0       0.0      0      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      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      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      0       0.0       0.0      0      0      0      0    0    0
cbb0           0      0       0.0       0.0      0      0      0      0    0    0
```

And the same picture of the LUN statistics on ctrl-b:



```
   0.0     0      0     0
                lun0
    ms  KB/t    tps  MB/s
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
   0.0     0      0     0
```

And finally ctrl-b log is overwhelmed by such kind of messages:



```
(0:131:1/0): SCSI sense: NOT READY asc:4,a (Logical unit not accessible, as
symmetric access state transition)
(0:131:1/0): TEST UNIT READY. CDB: 00 00 00 00 00 00  Tag: 0x8c28/1
(0:131:1/0): CTL Status: SCSI Error
(0:131:1/0): SCSI Status: Check Condition
(0:131:1/0): SCSI sense: NOT READY asc:4,a (Logical unit not accessible, as
symmetric access state transition)
(0:131:1/0): TEST UNIT READY. CDB: 00 00 00 00 00 00  Tag: 0x8c29/1
(0:131:1/0): CTL Status: SCSI Error
(0:131:1/0): SCSI Status: Check Condition
(0:131:1/0): SCSI sense: NOT READY asc:4,a (Logical unit not accessible, as
symmetric access state transition)
(0:131:1/0): TEST UNIT READY. CDB: 00 00 00 00 00 00  Tag: 0x8c2a/1
(0:131:1/0): CTL Status: SCSI Error
(0:131:1/0): SCSI Status: Check Condition
(0:131:1/0): SCSI sense: NOT READY asc:4,a (Logical unit not accessible, as
symmetric access state transition)
```

Note, that this message “Logical unit not accessible, asymmetric access state transition” is described in the ctl(4) man page:

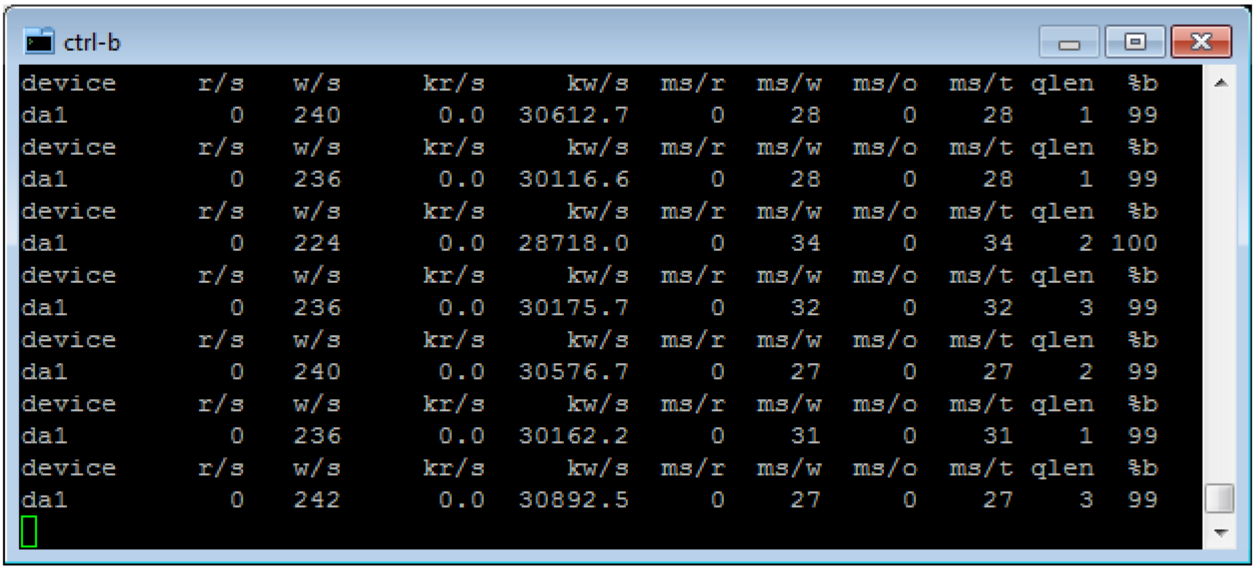
> *“If there is no primary node (both nodes are secondary, or secondary node has no connection to primary one), secondary node(s) report Transitioning state.”*

Therefore, it looks like a “normal” (kern.cam.ctl.ha_mode=2 shows the same results) behavior of CTL HA cluster in a case of disaster and loss of the primary node. It also means that a very lucky administrator can restore the failed primary controller before timeouts are elapsed.

Talking seriously, the failover can be done by setting primary role to the survived controller. So:

`root@ctrl-b:/home/beast #` sysctl kern.cam.ctl.ha_role=0

And the traffic starts flowing once again:



```
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   240      0.0  30612.7     0    28     0    28    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   236      0.0  30116.6     0    28     0    28    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   224      0.0  28718.0     0    34     0    34    2 100
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   236      0.0  30175.7     0    32     0    32    3  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   240      0.0  30576.7     0    27     0    27    2  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   236      0.0  30162.2     0    31     0    31    1  99
device     r/s   w/s     kr/s     kw/s  ms/r  ms/w  ms/o  ms/t qlen  %b
da1          0   242      0.0  30892.5     0    27     0    27    3  99
```