FreeBSD based dual-controller storage system concept

Mikhail E. Zakharov, zmey20000@yahoo.com

Nowadays most of the modern and powerful block-level storage systems around the world are built to work in expensive Fibre Channel or more cheaper iSCSI SAN environments. Independent of their class, capacity and performance they are created on well-known principles, technologies and architectures. Furthermore some of these systems are using common servers for their controller hardware with Linux or even AIX as storage operating systems on-board.

So theoretically nothing can stop us from developing our own reliable block-level storage system based on FreeBSD especially keeping in mind existence of successful FreeNAS, iXsystems and NetApp projects. Now lets see what FreeBSD-10.2 could bring us just out of the box, so we do not have to write any additional kernel modules to achieve the goal.

Although our simple model is intended to work on my laptop within the Oralce VirtualBox environment which is used to run virtual machines for storage controllers along with a client host, we will try to follow the basic principles of reliability, availability, and serviceability (RAS) firstly introduced by IBM. And may the Beastie be with us!

For our experiments we will need two nearly identical virtual machines (ctrl-a and ctrl-b) for storage controllers and one machine (clnt-1) for a client host. For obvious reasons we can't test Fibre Channel HBAs on the laptop so we will use iSCSI for our block-level access to the FreeBSD storage system model.

As we don't use Fibre Channel in our environment, the controllers will have 2 LAN connections: one (private) for inter-controller and another (public) for host-controller communications. In my case this would be host-only networks:

ctrl-a	**ctrl-b**
Private LAN IP-address: 192.168.56.10	Private network IP-address: 192.168.56.11
Public LAN IP-address: 192.168.55.10	Public LAN IP-address: 192.168.55.11

We also need to prepare and configure “shared drives” on both of controllers. This special feature allows to share physical drives between virtual machines so do not confuse it with “shared folders” which is a completely different technology. In our case these “shared drives” will simulate a dummy enclosure with four drives (d0, d1, d2, d3) for our storage system model.

It's better to use a dedicated controller for these shared drives to separate them from the system drive. In the virtual environment it's not very important which one to choose, but it will definitely work with SATA controller emulation. Therefore:

- Using VirtualBox Manager create a new SATA controller with 4 **fixed-sized VDI** drives: d0, d1, d2, d3 on ctrl-a machine;
- Then go to VirtualBox Virtual Media Manager and set “**shareable**” flag for each drive;
- Finally attach drives to the new SATA controller on ctrl-b virtual machine.

Install FreeBSD on ctrl-a and ctrl-b virtual machines using default parameters and ada0 as the drive with root file-system. Then configure general parameters in /etc/rc.conf:

ctrl-a	ctrl-b
hostname="ctrl-a"	hostname="ctrl-b"
ifconfig_em0="inet 192.168.56.10 netmask 255.255.255.0" # Inter-controller private network	ifconfig_em0="inet 192.168.56.11 netmask 255.255.255.0" # Inter-controller private network
ifconfig_em1="inet 192.168.55.10 netmask 255.255.255.0" # Public network	ifconfig_em1="inet 192.168.55.11 netmask 255.255.255.0" # Public network
# VirtualBox guest additions	# VirtualBox guest additions
vboxguest_enable="YES	vboxguest_enable="YES
vboxservice_enable="YES"	vboxservice_enable="YES"
# iSCSI	# iSCSI
ctld_enable="YES" # target	ctld_enable="YES" # target
iscsid_enable="YES" # initiator	iscsid_enable="YES" # initiator

Now it's important to set iSCSI "disconnection on fail" kernel variable in /etc/sysctl.conf on both systems to enable failover to the next controller in case of disaster:

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

After reboot, according to the dmesg output our shared drives are accessible as ada1, ada2, ada3, ada4 on both machines:

```
root@ctrl-a:/ # dmesg|grep ada
ada0 at ata0 bus 0 scbus0 target 0 lun 0
ada0: <VBOX HARDDISK 1.0> ATA-6 device
ada0: Serial Number VB5f1cd9c5-7f39ae5f
ada0: 33.300MB/s transfers (UDMA2, PIO 65536bytes)
ada0: 20480MB (41943040 512 byte sectors: 16H 63S/T 16383C)
ada0: Previously was known as ad0
ada1 at ahcich0 bus 0 scbus2 target 0 lun 0
ada1: <VBOX HARDDISK 1.0> ATA-6 SATA 2.x device
ada1: Serial Number VBa2a70c86-5e4db960
ada1: 300.000MB/s transfers (SATA 2.x, UDMA6, PIO 8192bytes)
ada1: Command Queueing enabled
ada1: 100MB (204800 512 byte sectors: 16H 63S/T 203C)
ada1: Previously was known as ad4
ada2 at ahcich1 bus 0 scbus3 target 0 lun 0
ada2: <VBOX HARDDISK 1.0> ATA-6 SATA 2.x device
ada2: Serial Number VB6a1e2b6c-fcb1fd23
ada2: 300.000MB/s transfers (SATA 2.x, UDMA6, PIO 8192bytes)
ada2: Command Queueing enabled
ada2: 100MB (204800 512 byte sectors: 16H 63S/T 203C)
ada2: Previously was known as ad6
ada3 at ahcich2 bus 0 scbus4 target 0 lun 0
ada3: <VBOX HARDDISK 1.0> ATA-6 SATA 2.x device
ada3: Serial Number VBad8731fa-8e8050c8
ada3: 300.000MB/s transfers (SATA 2.x, UDMA6, PIO 8192bytes)
ada3: Command Queueing enabled
ada3: 100MB (204800 512 byte sectors: 16H 63S/T 203C)
```

```
ada3: Previously was known as ad8
ada4 at ahcich3 bus 0 scbus5 target 0 lun 0
ada4: <VBOX HARDDISK 1.0> ATA-6 SATA 2.x device
ada4: Serial Number VB0f2bbabf-1b8af7ae
ada4: 300.000MB/s transfers (SATA 2.x, UDMA6, PIO 8192bytes)
ada4: Command Queueing enabled
ada4: 100MB (204800 512 byte sectors: 16H 63S/T 203C)
ada4: Previously was known as ad10
```

Now we can create RAIDs on our shared drives. Actually we may choose any available type (even RAID 0) but as we have four shared drives, lets use GEOM Mirror provider (gmirror) to create two reliable RAID-1 mirrors: one for ctrl-a and another for ctrl-b.

Therefore run:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # gmirror load	root@ctrl-b:/ # gmirror load
root@ctrl-a:/ # gmirror label -v ctrl_a_gm0 /dev/ada1 /dev/ada2	root@ctrl-b:/ # gmirror label -v ctrl_b_gm0 /dev/ada3 /dev/ada4

Note that ada0 is our system drive, so we shouldn't put it under shared RAID configuration.

Then change /boot/loader.conf on both controllers to start gmirror on boot:

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

After reboot we can check if gmirror devices are created:

```
root@ctrl-a:/ # ls -la /dev/mirror
total 0
crw-r-----  1 root  operator  0x5d Mar 15 13:44 ctrl_a_gm0
crw-r-----  1 root  operator  0x61 Mar 15 13:44 ctrl_b_gm0
```

Right the same can be seen on ctrl-b, so I will always skip listings from the other controller to reduce the article size.

Now we can partition the mirrored space. Actually these partitions would be LUNs for our further testing purposes.

Create GPT partition scheme on each mirror:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # gpart create -s GPT /dev/mirror/ctrl_a_gm0	root@ctrl-b:/ # gpart create -s GPT /dev/mirror/ctrl_b_gm0

To simplify the example we will add only one partition to each mirror:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # gpart add -t freebsd-ufs -a 1m /dev/mirror/ctrl_a_gm0	root@ctrl-b:/ # gpart add -t freebsd-ufs -a 1m /dev/mirror/ctrl_b_gm0

Finally we can check the result:

```
root@ctrl-b:/ # ls -la /dev/mirror/
total 0
crw-r-----  1 root  operator  0x5d Mar 15 13:44 ctrl_a_gm0
crw-r-----  1 root  operator  0x60 Mar 15 13:44 ctrl_a_gm0p1
crw-r-----  1 root  operator  0x61 Mar 15 13:44 ctrl_b_gm0
crw-r-----  1 root  operator  0x63 Mar 15 13:44 ctrl_b_gm0p1
```

It's extremely harmful to write data simultaneously from both controllers to the same shared drive. As we don't have any arbitrator yet, we must avoid accessing ctrl_b_gm0 mirror from ctrl-a controller and vice versa: ctrl_a_gm0 mirror from ctrl-b controller. Also we must somehow implement controller redundancy. To achieve both of the goals lets invent this mechanism.

First we have to make links from one controller to another to reach the opposite mirror. We can try to do it with ggate (GEOM gate provider) or by utilizing iSCSI. I chose iSCSI for our testing because it looks more flexible and therefore suitable for our purposes.

At this point we will have two paths on each controller to the opposite controller mirror: one path is local as we can access physical drives directly, another is provided by iSCSI connection. Therefore our second step would be to create gmultipath (GEOM Multipath) device for both of the links.

This construction will help us to redirect I/O from a failed controller to an active one. Nothing seems impossible for now, so lets create this arbitrator construction for our LUN partitions.

We don't need to enable iSCSI -target and -initiator explicitly as we already put appropriate variables to /etc/rc.conf of both controllers. Therefore we will just create iSCSI-target configuration files /etc/ctl.conf:

/etc/ctl.conf on ctrl-a

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 192.168.56.10
}

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

        lun 0 {
                path /dev/mirror/ctrl_a_gm0p1
                size 102760448
        }
}
```

/etc/ctl.conf on ctrl-b

```
portal-group pg0 {
        discovery-auth-group no-authentication
        listen 192.168.56.11
}

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

        lun 0 {
                path /dev/mirror/ctrl_b_gm0p1
                size 102760448
        }
}
```

As you can see, we will use Portal group pg0 for our private inter-controller communication. The size of the LUN 0 is 102760448 bytes. This value is taken from the output of:

```
root@ctrl-a:/ # gpart list mirror/ctrl_a_gm0

Geom name: mirror/ctrl_a_gm0
modified: false
state: OK
fwheads: 32
fwsectors: 9
last: 204765
first: 34
entries: 128
scheme: GPT
Providers:
1. Name: mirror/ctrl_a_gm0p1
   Mediasize: 102760448 (98M)
   Sectorsize: 512
   Stripesize: 0
   Stripeoffset: 1048576
   Mode: r0w0e0
   rawuuid: 5677c154-e917-11e5-944d-080027868477
   rawtype: 516e7cb6-6ecf-11d6-8ff8-00022d09712b
   label: 1
   length: 102760448
   offset: 1048576
   type: freebsd-ufs
   index: 1
   end: 202751
   start: 2048
Consumers:
1. Name: mirror/ctrl_a_gm0
   Mediasize: 104857088 (100M)
   Sectorsize: 512
   Mode: r0w0e0
```

Check Mediasize value of mirror/ctrl_a_gm0p1 provider. The Mediasize of mirror/ctrl_b_gm0 must match the result.

Now we can start iSCSI -target (ctld) on both controllers and try to reach drives:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # /etc/rc.d/ctld start Starting ctld.	root@ctrl-b:/ # /etc/rc.d/ctld start Starting ctld.
root@ctrl-b:/ # iscsictl -A -p 192.168.56.11 -t iqn.2016-01.local.sss.private:target0	root@ctrl-b:/ # iscsictl -A -p 192.168.56.10 -t iqn.2016-01.local.sss.private:target0

After it the dmesg command will show us that a new da0 drive has appeared on each controller:

```
root@ctrl-a:/ # dmesg| tail -6
da0 at iscsi1 bus 0 scbus7 target 0 lun 0
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: 98MB (200704 512 byte sectors: 64H 32S/T 98C)
```

Now lets save this connection definitions in the /etc/iscsi.conf file to refer to it later:

ctrl-a	**ctrl-b**

private {
    TargetAddress = 192.168.56.11
    TargetName = iqn.2016-01.local.sss.private:target0
}
	private {
	    TargetAddress = 192.168.56.10
	    TargetName = iqn.2016-01.local.sss.private:target0
	}

Now that the first part of the arbitrator construction is done, lets provide it with a failover ability:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # gmultipath create CTRL_B_BACK /dev/da0 /dev/mirror/ctrl_b_gm0p1	root@ctrl-b:/ # gmultipath create CTRL_A_BACK /dev/da0 /dev/mirror/ctrl_a_gm0p1

Now pay attention that we use "manual" method for multipathing device creation. In this case multipathing configuration is not stored in the metadata area of the device. And although this information will certainly be lost on reboot, we gain some profit on iSCSI configuration for the client-host.

Check the result:

ctrl-a	**ctrl-b**
root@ctrl-a:/ # dmesg | tail -4 GEOM_MULTIPATH: CTRL_B_BACK created GEOM_MULTIPATH: da0 added to CTRL_B_BACK GEOM_MULTIPATH: da0 is now active path in CTRL_B_BACK GEOM_MULTIPATH: mirror/ctrl_b_gm0p1 added to CTRL_B_BACK	root@ctrl-b:/ # dmesg | tail -4 GEOM_MULTIPATH: CTRL_A_BACK created GEOM_MULTIPATH: da0 added to CTRL_A_BACK GEOM_MULTIPATH: da0 is now active path in CTRL_A_BACK GEOM_MULTIPATH: mirror/ctrl_a_gm0p1 added to CTRL_A_BACK

We have created multipathing with active-passive policy, and active path is /dev/da0 now which means the opposite controller. So the data will always be routed through the opposite controller until it fails. This simple mechanism should prevent data corruption we mention above.

Now we are ready to provide storage to the client host clnt-1. Therefore we have to update iSCSI target configuration file /etc/ctld.conf on controllers with "public" definitions:

ctrl-a	**ctrl-b**

portal-group pg0 {
    discovery-auth-group no-authentication
    listen 192.168.56.10
}
 	
portal-group pg1 {
	portal-group pg0 {
	    discovery-auth-group no-authentication
	    listen 192.168.56.11
	}
	
	portal-group pg1 {

```
        discovery-auth-group no-authentication
        listen 192.168.55.10
}

# Private - inter-node LUN
target iqn.2016-01.local.sss.private:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                path /dev/mirror/ctrl_a_gm0p1
                size 102760448
        }
}

# Public - client access LUNs
target iqn.2016-01.local.sss.public:target0 {
        auth-group no-authentication
        portal-group pg1

        # Direct path to the local LUN
        lun 0 {
                path /dev/mirror/ctrl_a_gm0p1
                size 102760448
        }

        # LUN owned by the opposite controller
        lun 1 {
                path /dev/multipath/CTRL_B_BACK
                size 102760448
        }
}
```

```
        discovery-auth-group no-authentication
        listen 192.168.55.11
}

# Private - inter-node LUN
target iqn.2016-01.local.sss.private:target0 {
        auth-group no-authentication
        portal-group pg0

        lun 0 {
                path /dev/mirror/ctrl_b_gm0p1
                size 102760448
        }
}

# Public - client access LUNs
target iqn.2016-01.local.sss.public:target0 {
        auth-group no-authentication
        portal-group pg1

        # Direct path to the local LUN
       lun 0 {
                path /dev/mirror/ctrl_b_gm0p1
                size 102760448
        }

        # LUN owned by the opposite controller
        lun 1 {
                path /dev/multipath/CTRL_A_BACK
                size 102760448
        }
}
```

We add Port group pg1 for public host connection. Also we export two LUNs from each controller to the client host. Ergo on each controller LUN 0 is provided through so called owner controller, and LUN 1 is the partition from the opposite controller provided through the arbitrator construction.

Now ctld daemon should be forced to re-read its configuration file. On both controllers run:

```
killall -1 ctld
```

Controllers of the storage system are ready now to serve iSCSI requests, therefore we can leave them for a while and configure the client host. We need to add following lines to the /etc/rc.conf file to setup its hostname, network parameters and iSCSI initiator:

```
hostname="clnt-1"

ifconfig_em0="inet 192.168.55.20 netmask 255.255.255.0"

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

```
iscsid_enable="YES"              # SCSI initiator
```

Then set required iSCSI kernel variable via /etc/sysctl.conf:

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

After the reboot we can start our storage related tasks on the client side. Lets access iSCSI targets on both controllers.

For ctrl-a and ctrl-b run:

```
root@clnt-1:/ # iscsictl -A -p 192.168.55.10 -t iqn.2016-01.local.sss.public:target0
root@clnt-1:/ # iscsictl -A -p 192.168.55.11 -t iqn.2016-01.local.sss.public:target0
```

Then check the result:

```
root@clnt-1:/ # iscsictl -L
Target name                          Target portal    State
iqn.2016-01.local.sss.public:target0 192.168.55.10    Connected: da0 da1
iqn.2016-01.local.sss.public:target0 192.168.55.11    Connected: da2 da3
```

Also it's useful to see the dmesg output:

```
da0 at iscsi4 bus 0 scbus2 target 0 lun 0
da0: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da0: Serial Number MYSERIAL   1
da0: 150.000MB/s transfers
da0: Command Queueing enabled
da0: 98MB (200704 512 byte sectors: 64H 32S/T 98C)
da1 at iscsi4 bus 0 scbus2 target 0 lun 1
da1: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da1: Serial Number MYSERIAL   2
da1: 150.000MB/s transfers
da1: Command Queueing enabled
da1: 98MB (200704 512 byte sectors: 64H 32S/T 98C)
da2 at iscsi5 bus 0 scbus3 target 0 lun 0
da2: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da2: Serial Number MYSERIAL   1
da2: 150.000MB/s transfers
da2: Command Queueing enabled
da2: 98MB (200704 512 byte sectors: 64H 32S/T 98C)
da3 at iscsi5 bus 0 scbus3 target 0 lun 1
da3: <FREEBSD CTLDISK 0001> Fixed Direct Access SPC-4 SCSI device
da3: Serial Number MYSERIAL   2
da3: 150.000MB/s transfers
da3: Command Queueing enabled
da3: 98MB (200704 512 byte sectors: 64H 32S/T 98C)
```

Everything looks fine.

Keep in mind that da1 drive on ctrl-a is our arbitrator mechanism which points now to the partition p1 on the RAID-1 ctrl_b_gm0 mirror which actually resides on ctrl-b. Similarly da3 provided by ctrl-b is the arbitrator construction pointing now to the partition p1 on the RAID-1 ctrl_a_gm0 mirror which resides on ctrl-a.

These paths are routed through opposite, non-owner controllers. We also have short, direct paths provided by owner controllers: da0 is ctrl_a_gm0p1 provided by ctrl-a and da2 is ctrl_b_gm0p1 accessible through ctrl-b.

The Figure 1 shows our storage system architecture layout we just created.

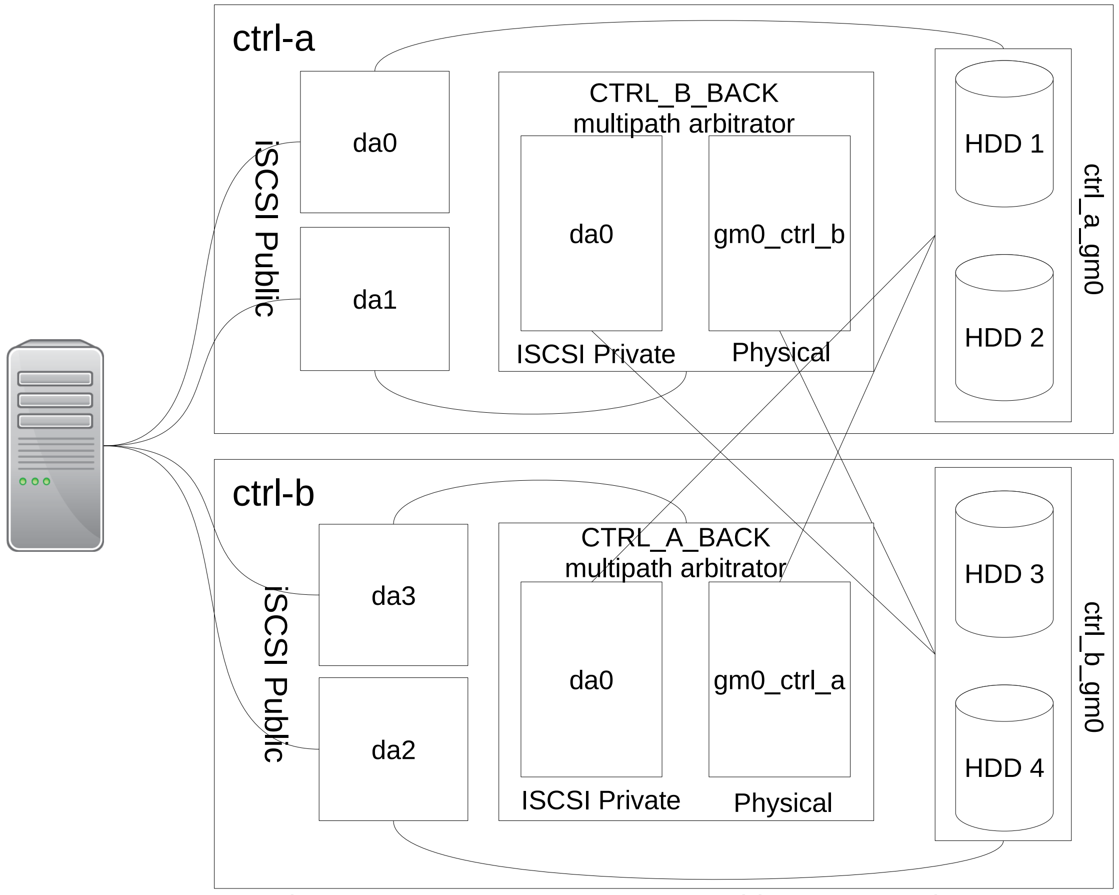


Figure 1. Dual-controller storage architecture overview.

As we have 2 paths for each LUN on our clnt-1 host, we can enable multipathing:

```
root@clnt-1:/ # gmultipath create CTRL_A /dev/da0 /dev/da3
root@clnt-1:/ # gmultipath create CTRL_B /dev/da2 /dev/da1
```

We can check the result with the gmultipath list command or by reading dmesg output:

```
root@clnt-1:/ # dmesg | grep GEOM_MULTIPATH
GEOM_MULTIPATH: CTRL_A created
GEOM_MULTIPATH: da0 added to CTRL_A
GEOM_MULTIPATH: da0 is now active path in CTRL_A
GEOM_MULTIPATH: da3 added to CTRL_A
GEOM_MULTIPATH: CTRL_B created
GEOM_MULTIPATH: da2 added to CTRL_B
GEOM_MULTIPATH: da2 is now active path in CTRL_B
GEOM_MULTIPATH: da1 added to CTRL_B
```

This storage architecture with our simple arbitrator and absence of cache is asymmetric by nature so it has to run in active-passive mode where da0 and da2 are active paths now. Sure we can enable active-active or active-read modes to use all four paths and everything will work without errors. But it will cause potential overheads as the data for every LUN will also be routed through the LAN to non-owner controllers.

Now we can do whatever is needed to use drives on our client. For example, lets label them and create filesystems on the multipathed devices:

```
root@clnt-1:/ # disklabel -Brw /dev/multipath/CTRL_A auto
root@clnt-1:/ # disklabel -Brw /dev/multipath/CTRL_B auto
root@clnt-1:/ # newfs /dev/multipath/CTRL_A
/dev/multipath/CTRL_A: 98.0MB (200704 sectors) block size 32768, fragment size 4096
      using 4 cylinder groups of 24.53MB, 785 blks, 3200 inodes.
super-block backups (for fsck_ffs -b #) at:
 192, 50432, 100672, 150912
root@clnt-1:/ # newfs /dev/multipath/CTRL_B
/dev/multipath/CTRL_B: 98.0MB (200704 sectors) block size 32768, fragment size 4096
      using 4 cylinder groups of 24.53MB, 785 blks, 3200 inodes.
super-block backups (for fsck_ffs -b #) at:
 192, 50432, 100672, 150912
```

Finally create mount points and mount filesystems:

```
root@clnt-1:/ # mkdir -p /storage/d0 /storage/d1
root@clnt-1:/ # mount /dev/multipath/CTRL_A /storage/d0
root@clnt-1:/ # mount /dev/multipath/CTRL_B /storage/d1
```

We are ready now to make various tests and try to break the whole construction to check our architecture. For example, forcibly power-off one of the controllers during a file copy process:

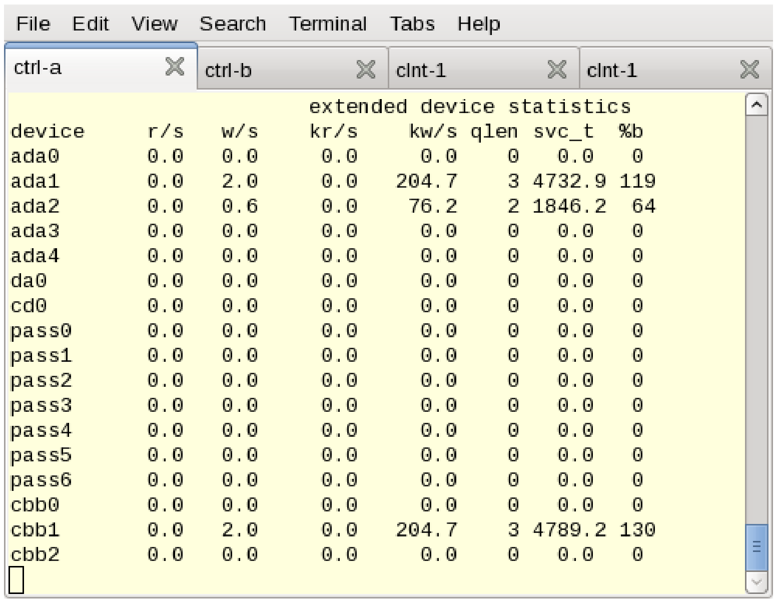
```
root@clnt-1:/ # ls -la ports.tgz
-rw-r--r--  1 root  zmey  59546274 Mar  9 16:22 ports.tgz

root@clnt-1:/ # cp ports.tgz /storage/d0 & cp ports.tgz /storage/d1
[1] 1268
```

On the next Figures we will see performance statistics from each controller and a client. The values are extremely poor as VirtualBox shared drives were intentionally put on slow USB memory stick in order to see the whole process in detail.

Figure 2 shows ctrl-a controller and we can see that ada1 and ada2 drives have workload. This is our ctrl_a_gm0 RAID-1 mirror. Note we don't see any workload on ada3/ada4 as these drives are parts of ctrl_b_gm0 mirror and therefore are governed by ctrl-b.

The working cbb1 device looks like iSCSI target for public LAN clnt-1 connection, and da0, which has zero values of workload, is the link to the LUN on the opposite controller.

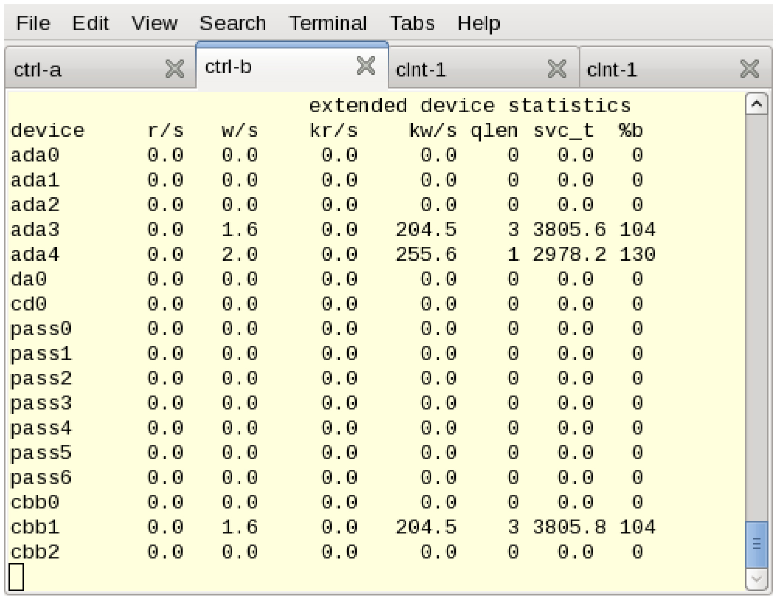
We can state that the traffic goes normally through the owner controller, and our arbitrator works well.



```
                         extended device statistics
device     r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0       0.0   0.0     0.0     0.0    0   0.0   0
ada1       0.0   2.0     0.0   204.7    3 4732.9 119
ada2       0.0   0.6     0.0    76.2    2 1846.2  64
ada3       0.0   0.0     0.0     0.0    0   0.0   0
ada4       0.0   0.0     0.0     0.0    0   0.0   0
da0        0.0   0.0     0.0     0.0    0   0.0   0
cd0        0.0   0.0     0.0     0.0    0   0.0   0
pass0      0.0   0.0     0.0     0.0    0   0.0   0
pass1      0.0   0.0     0.0     0.0    0   0.0   0
pass2      0.0   0.0     0.0     0.0    0   0.0   0
pass3      0.0   0.0     0.0     0.0    0   0.0   0
pass4      0.0   0.0     0.0     0.0    0   0.0   0
pass5      0.0   0.0     0.0     0.0    0   0.0   0
pass6      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
cbb1       0.0   2.0     0.0   204.7    3 4789.2 130
cbb2       0.0   0.0     0.0     0.0    0   0.0   0
```

Figure 2. Normal work of the ctrl-a controller

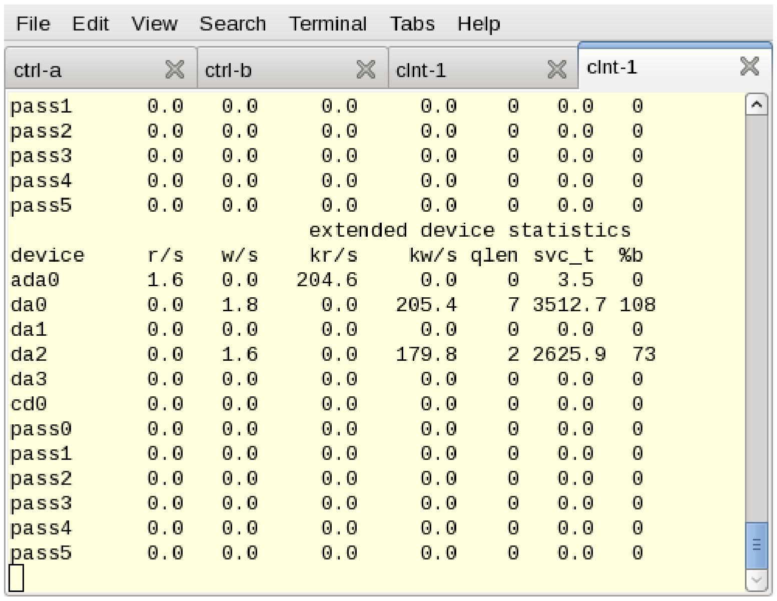
Figure 3 shows the similar picture on ctrl-b the only exception, is that ada3 and ada4 are parts of ctrl_b_gm0 mirror. As we can see the controller ctrl-b works normally and the traffic is directed through the shortest path to the ctrl_b_gm0 mirror.



```
                         extended device statistics
device     r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0       0.0   0.0     0.0     0.0    0   0.0   0
ada1       0.0   0.0     0.0     0.0    0   0.0   0
ada2       0.0   0.0     0.0     0.0    0   0.0   0
ada3       0.0   1.6     0.0   204.5    3 3805.6 104
ada4       0.0   2.0     0.0   255.6    1 2978.2 130
da0        0.0   0.0     0.0     0.0    0   0.0   0
cd0        0.0   0.0     0.0     0.0    0   0.0   0
pass0      0.0   0.0     0.0     0.0    0   0.0   0
pass1      0.0   0.0     0.0     0.0    0   0.0   0
pass2      0.0   0.0     0.0     0.0    0   0.0   0
pass3      0.0   0.0     0.0     0.0    0   0.0   0
pass4      0.0   0.0     0.0     0.0    0   0.0   0
pass5      0.0   0.0     0.0     0.0    0   0.0   0
pass6      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
cbb1       0.0   1.6     0.0   204.5    3 3805.8 104
cbb2       0.0   0.0     0.0     0.0    0   0.0   0
```

Figure 3. Normal work of the ctrl-b controller

Figure 4 reflects the situation on the clnt-1 client. We can see da0 and da2 are utilized. These devices are primary, active and the shortest paths to the backend RAID-1 mirrors drives. Paths da1 and da3 are not loaded according to the active-passive mode of CTRL_A and CTRL_B multipathing devices.

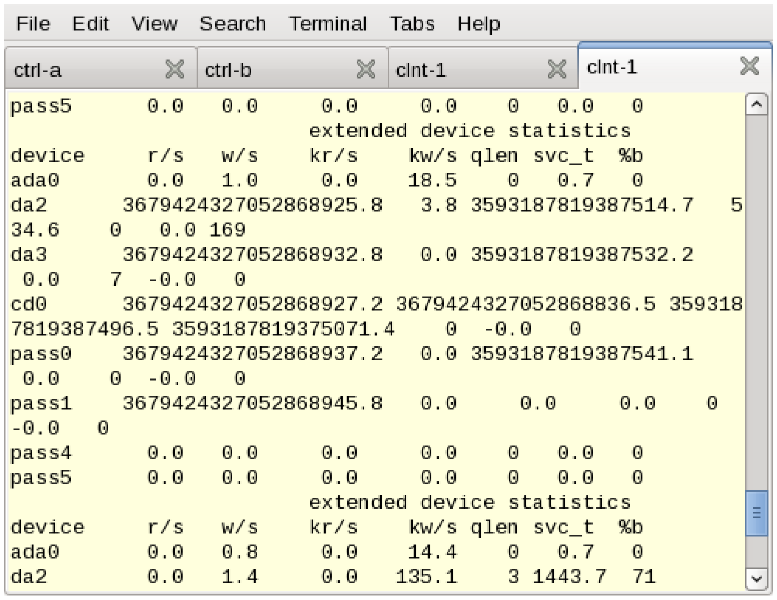


```
pass1       0.0   0.0     0.0     0.0    0   0.0   0
pass2       0.0   0.0     0.0     0.0    0   0.0   0
pass3       0.0   0.0     0.0     0.0    0   0.0   0
pass4       0.0   0.0     0.0     0.0    0   0.0   0
pass5       0.0   0.0     0.0     0.0    0   0.0   0
                        extended device statistics
device      r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0        1.6   0.0   204.6     0.0    0   3.5   0
da0         0.0   1.8     0.0   205.4    7 3512.7 108
da1         0.0   0.0     0.0     0.0    0   0.0   0
da2         0.0   1.6     0.0   179.8    2 2625.9  73
da3         0.0   0.0     0.0     0.0    0   0.0   0
cd0         0.0   0.0     0.0     0.0    0   0.0   0
pass0       0.0   0.0     0.0     0.0    0   0.0   0
pass1       0.0   0.0     0.0     0.0    0   0.0   0
pass2       0.0   0.0     0.0     0.0    0   0.0   0
pass3       0.0   0.0     0.0     0.0    0   0.0   0
pass4       0.0   0.0     0.0     0.0    0   0.0   0
pass5       0.0   0.0     0.0     0.0    0   0.0   0
```

Figure 4. Normal work of the clnt-1.

Now lets fail one of the controllers and check if the system survives it. We can forcible power-off or reset ctrl-a, for example.

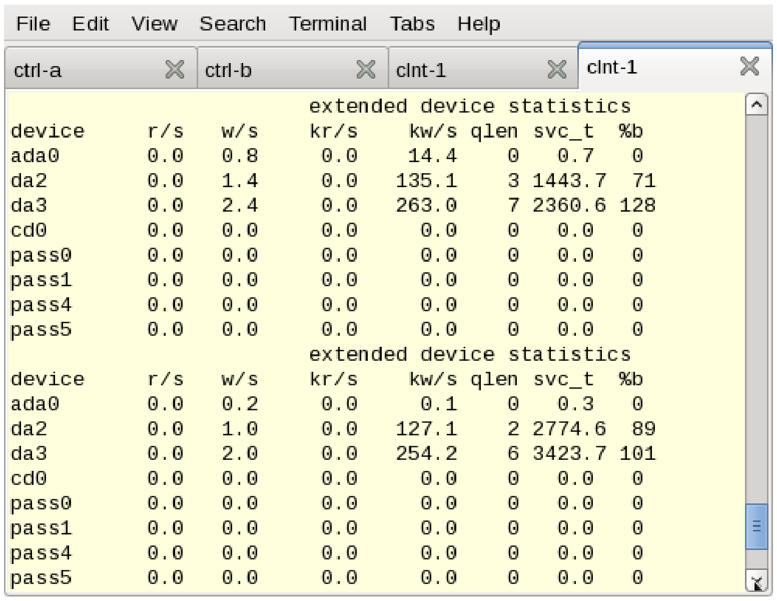
Figure 5 shows the reaction of the client. We can see some mess in the performance statistics resulted from failed paths.



```
pass5       0.0   0.0     0.0     0.0    0   0.0   0
                        extended device statistics
device      r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0        0.0   1.0     0.0    18.5    0   0.7   0
da2       3679424327052868925.8   3.8 3593187819387514.7   5
34.6    0   0.0 169
da3       3679424327052868932.8   0.0 3593187819387532.2
 0.0    7  -0.0   0
cd0       3679424327052868927.2 3679424327052868836.5 359318
7819387496.5 3593187819375071.4    0  -0.0   0
pass0     3679424327052868937.2   0.0 3593187819387541.1
 0.0    0  -0.0   0
pass1     3679424327052868945.8   0.0     0.0     0.0    0
-0.0   0
pass4       0.0   0.0     0.0     0.0    0   0.0   0
pass5       0.0   0.0     0.0     0.0    0   0.0   0
                        extended device statistics
device      r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0        0.0   0.8     0.0    14.4    0   0.7   0
da2         0.0   1.4     0.0   135.1    3 1443.7  71
```

Figure 5. The clnt-1 host immediately after the ctrl-a failure.

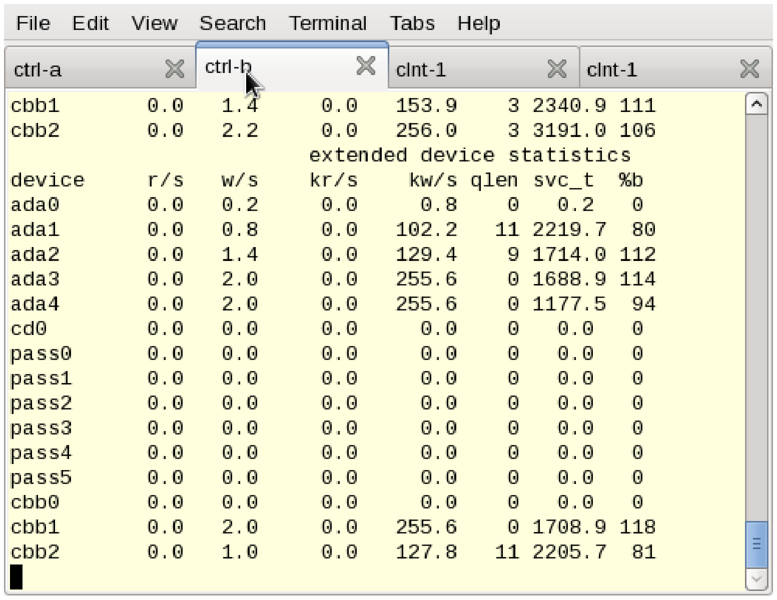
Soon the situation stabilizes. Caused by the multipath policy, workload successfully moved from failed da0 to da3 device. This can be seen on the Figure 6:



```
                      extended device statistics
device     r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0       0.0   0.8     0.0    14.4    0   0.7   0
da2        0.0   1.4     0.0   135.1    3 1443.7  71
da3        0.0   2.4     0.0   263.0    7 2360.6 128
cd0        0.0   0.0     0.0     0.0    0   0.0   0
pass0      0.0   0.0     0.0     0.0    0   0.0   0
pass1      0.0   0.0     0.0     0.0    0   0.0   0
pass4      0.0   0.0     0.0     0.0    0   0.0   0
pass5      0.0   0.0     0.0     0.0    0   0.0   0
                      extended device statistics
device     r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0       0.0   0.2     0.0     0.1    0   0.3   0
da2        0.0   1.0     0.0   127.1    2 2774.6  89
da3        0.0   2.0     0.0   254.2    6 3423.7 101
cd0        0.0   0.0     0.0     0.0    0   0.0   0
pass0      0.0   0.0     0.0     0.0    0   0.0   0
pass1      0.0   0.0     0.0     0.0    0   0.0   0
pass4      0.0   0.0     0.0     0.0    0   0.0   0
pass5      0.0   0.0     0.0     0.0    0   0.0   0
```

Figure 6. The clnt-1 operates now with the ctrl-b only.

Lets see what is going on the ctrl-b. Figure 7 shows us that ctrl-b takes now full workload for both LUNs (cbb1/cbb2). Both mirrors ctrl_a_gm0 (ada1/ada2) and ctrl_b_gm0 (ada3/ada4) are also utilized.



```
cbb1       0.0   1.4     0.0   153.9    3 2340.9 111
cbb2       0.0   2.2     0.0   256.0    3 3191.0 106
                      extended device statistics
device     r/s   w/s    kr/s    kw/s qlen svc_t  %b
ada0       0.0   0.2     0.0     0.8    0   0.2   0
ada1       0.0   0.8     0.0   102.2   11 2219.7  80
ada2       0.0   1.4     0.0   129.4    9 1714.0 112
ada3       0.0   2.0     0.0   255.6    0 1688.9 114
ada4       0.0   2.0     0.0   255.6    0 1177.5  94
cd0        0.0   0.0     0.0     0.0    0   0.0   0
pass0      0.0   0.0     0.0     0.0    0   0.0   0
pass1      0.0   0.0     0.0     0.0    0   0.0   0
pass2      0.0   0.0     0.0     0.0    0   0.0   0
pass3      0.0   0.0     0.0     0.0    0   0.0   0
pass4      0.0   0.0     0.0     0.0    0   0.0   0
pass5      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
cbb1       0.0   2.0     0.0   255.6    0 1708.9 118
cbb2       0.0   1.0     0.0   127.8   11 2205.7  81
```

Figure 7. The ctrl-b workload after the ctrl-a failure.

After files have been copied, we can check if the data were written correctly:

```
root@cln-1:/ # md5 /storage/d0/ports.tgz
MD5 (/storage/d0/ports.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
root@cln-1:/ # md5 /storage/d1/ports.tgz
MD5 (/storage/d1/ports.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
root@cln-1:/ # md5 ports.tgz
MD5 (ports.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
```

MD5 sums are equal on both remote LUNs and on the local filesystem, so everything looks fine. Our simple storage has successfully survived one controller failure and we can say that our mission is completed at this stage.

Instead of creating filesystem and mounting separate partitions on /storage/d0 and /storage/d1 in the example above we could do even more real life storage-like thing: make a stripe of two partitions on the client host:

```
root@clnt-1:/ # gstripe create -v -s 131072 STORAGE /dev/multipath/CTRL_A /dev/multipath/CTRL_B

root@clnt-1:/ # newfs /dev/stripe/STORAGE
/dev/stripe/STORAGE: 196.0MB (401408 sectors) block size 32768, fragment size 4096
        using 4 cylinder groups of 49.03MB, 1569 blks, 6400 inodes.
super-block backups (for fsck_ffs -b #) at:
 192, 100608, 201024, 301440

root@clnt-1:/ # mount /dev/stripe/STORAGE /storage/d0
```

Then copy files:

```
root@clnt-1:/ # cp ports.tgz /storage/d0/ports.tgz & cp ports.tgz /storage/d0/ports.1.tgz
```

Finally power-off one of the controllers and check the result. It will be the same as in the previous experiment:

```
root@clnt-1:/ # md5 ports.tgz
MD5 (ports.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
root@clnt-1:/ # md5 /storage/d0/ports.tgz
MD5 (/storage/d0/ports.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
root@clnt-1:/ # md5 /storage/d0/ports.1.tgz
MD5 (/storage/d0/ports.1.tgz) = 82a5d6a7a3a89b7a5185a543fa6b3a56
root@v4_cln_1:/home/zmey # cp ports.tgz /storage/d0/
```

In other words, our high-available model is working well.

But to bring the controller back into business we need to do a lot of manual work to reconstruct it after the failure: carefully re-enable iSCSI connections between controllers, restore lost multipath devices and set active paths, finally resume iSCSI and multipath connections to the client host.

During my experiments I had to reboot controllers hundreds of times. It was really tedious to do everything manually so I wrote a simple shell-script to restore original storage configuration after clean boot of both controllers, though in case of a failure I still have do everything manually.

This script uses my lightweight expect-like tool called “empty,” to automate interactive login and su operations. If you decide to use this script, don't forget to install “empty” from ports or packages. But beware of the script insecurity as it contains plain text passwords right in the body. Don't do such things in production environments, use SSH with keys instead.

The script must be run from the client host by root:

```
#!/bin/sh

login=storadmin
ctrl_a=192.168.55.10
```

```
ctrl_b=192.168.55.11
passwd=TopSecret
root_passwd=VeryTopSecret

# On controller A
empty -f ssh $login@$ctrl_a
empty -w -t 5 assword "$passwd\n"
empty -s "su\n"
empty -w -t 5 assword "$root_passwd\n"

empty -s "gmultipath create CTRL_B_BACK /dev/mirror/ctrl_b_gm0p1\n"
empty -s "/etc/rc.d/ctld onestart\n"
empty -s "exit\n"
empty -s "exit\n"

sleep 5
# On controller B
empty -f ssh $login@$ctrl_b
empty -w -t 5 assword "$passwd\n"
empty -s "su\n"
empty -w -t 5 assword "$root_passwd\n"

empty -s "gmultipath create CTRL_A_BACK /dev/mirror/ctrl_a_gm0p1\n"
empty -s "/etc/rc.d/ctld onestart\n"
empty -s "iscsictl -A -n private\n"
empty -s "gmultipath add CTRL_A_BACK /dev/da0\n"
empty -s "gmultipath rotate CTRL_A_BACK\n"
empty -s "exit\n"
empty -s "exit\n"

sleep 5
# On controller A
empty -f ssh $login@$ctrl_a
empty -w -t 5 assword "$passwd\n"
empty -s "su\n"
empty -w -t 5 assword "$root_passwd\n"

empty -s "iscsictl -A -n private\n"
empty -s "gmultipath add CTRL_B_BACK /dev/da0\n"
empty -s "gmultipath rotate CTRL_B_BACK\n"
empty -s "exit\n"
empty -s "exit\n"

# On the client. Common part
iscsictl -A -p 192.168.55.10 -t iqn.2016-01.local.sss.public:target0
iscsictl -A -p 192.168.55.11 -t iqn.2016-01.local.sss.public:target0
gmultipath create CTRL_A /dev/da0 /dev/da3
gmultipath create CTRL_B /dev/da2 /dev/da1

# On the client. Test 1
newfs /dev/multipath/CTRL_A
newfs /dev/multipath/CTRL_A
mount /dev/multipath/CTRL_A /storage/d0
mount /dev/multipath/CTRL_B /storage/d1
```

```
# On the client. Test 2
# gstripe create -v -s 131072 STORAGE /dev/multipath/CTRL_A /dev/multipath/CTRL_B
# newfs /dev/stripe/STORAGE
# mount /dev/stripe/STORAGE /storage/d0
```

Real storage systems have serious clustering software for the management and synchronization of controllers, but in our case it will be in the future.

Actually we have a lot of directions for the development from this point. We didn't experiment with ZFS yet, didn't implement mirrored data cache (which is the main part of any modern storage system), we didn't even test our architecture on a physical hardware and Fibre Channel host bus adapters.

Our system reliably works, but honestly this is only a proof of concept which must be seriously tested before being used outside the laboratory.